

中华学习机

紫金Ⅱ系列

APPLE Ⅱ

Zh

朱国江

詹丰兴 等编著

孙建隆

机器语言

高等教育出版社

TP36

二九T/1

中华学习机机器语言

朱国江 詹丰兴 孙建隆等编著



商务出版社

022122

(京) 新登字046号

内 容 简 介

本书系统地介绍了中华学习机机器语言的寻址方式和指令系统，并以程序设计为中心，在编程方法、数学计算、信息处理、图形绘制、软件开发、功能扩展、子程序调用等方面作了详细阐述。书中列举的大量程序实例，不仅在中华学习机上进行过验证，而且也适用于以6502为CPU的APPLE-II、紫金-II等系列机型。

本书适用于从事微型计算机应用、管理的人员和广大青少年及微机爱好者自学，也可作为各种类型培训班的教学参考书。

中华学习机机器语言

朱国江 詹丰兴 孙建隆等编著

责任编辑 杨长新 黄丽荣

电子工业出版社

(北京西郊白石桥路46号)

北京昌平环球科技印刷厂印刷

新华书店总店科技发行所发行 全国各地新华书店经销

*

开本: 787×1092 1/32 印张: 17.5 字数: 453千字

1991年11月第一版 1991年11月第一次印刷

印数: 1—5500 定价: 8.85元

ISBN 7-5029-0675-4/TP·0028

前 言

本书是一本学习和应用6502机器语言程序设计的入门书。虽然，它比任何一种高级语言程序设计无论在内容和方法上，还是在深度和难度上，都要困难和复杂得多，但是，由于机器语言具有执行速度快、占用内存少、可以实现高级语言无法实现的功能等特点，因此学习机器语言就显得十分必要。事实上，任何高级语言的功能，最终必将经过执行机器语言才能实现，而要更清晰地理解高级语言在机器中运行的细节，或者更完满地处理高级语言，读一点机器语言将会受益匪浅。尤其是机器语言在软件开发，图形处理，游戏编写，音乐声响，功能扩展，实时控制等方面具有广阔的应用前景，不了解机器语言将是一大缺陷。

本书就是在这样的指导思想下编写的，作者的目的是多方面的，主要是面向应用和实用，兼顾普及和提高，希望本书有助于广大青少年读者和非计算机专业人员的使用。

本书主要介绍以6502为CPU的中华学习机机器语言程序设计，因而也适用APPLE-Ⅱ及紫金-Ⅱ等系列机。在写法上打破传统的以指令为中心的描述方法，而是以大量短小精炼且富有情趣的程序实例，介绍程序设计的各个方面，通俗易懂，引人入胜。

全书以较大的篇幅详尽介绍6502指令系统和寻址方式，概念准确，叙述清楚。以程序设计为中心，既有原理分析，又有实践经验，并以程序剖析贯穿始终，深入浅出，易于自

学。全书还列举大量典型实例，介绍设计思想，对比编程方法，交待学习要领，提高学习兴趣，循序渐进，丰富多采。考虑到读者的不同层次和不同要求，本书还介绍了软件开发、功能扩展、系统子程序调用等方面的内容，引导读者进一步加深对机器语言的了解，有助于实际编程能力的提高。

全书共分12章，其中1—7章由詹丰兴同志编写，8—11章由朱国江编写，12章由孙建隆同志编写。孙建隆同志为全书程序调试、验证做了大量的工作，马力同志为本书提供了不少有价值的程序，并参加编写了部分章节的内容，宋桂兰同志为全书做了很多文字工作，并整理了附录资料。全书由朱国江同志统一审校。

系列书的编写历时四年有余，字字句句渗透了合作者的心血和奉献，也得到了南京大学大气科学系领导和众多同志的鼓励和关心，得到了邹进上教授、李忠恺教授、周朝辅高级工程师、葛文忠副教授始终如一的支持和帮助，得到了气象出版社、《软件报》社、国营734厂、武汉天向电脑公司、中国中华学习机普及协会、《电子与电脑》杂志社以及南京大学出版社花国良同志的大力协助，在此一并表示感谢。

编者水平有限，书中错误和不妥之处在所难免，敬请广大读者不吝赐教。

南京大学 朱国江

1990年5月

目 录

前言

一、机器语言的一般知识	(1)
1. 机器语言的特点	(1)
2. 数的表示	(3)
3. 指令码的格式	(6)
4. 6502的寄存器	(7)
5. 内存分配	(8)
二、字符的显示和输入	(11)
1. 指令的分类、寻址方式	(11)
2. 文本状态的字符显示	(17)
3. 接受按键	(24)
4. 闪烁及反相字符的显示	(29)
5. 指令讨论	(32)
三、6502监控系统	(41)
1. 程序的输入和显示	(41)
2. 内存的显示和修改	(44)
3. 算术运算	(47)
4. 标志寄存器	(52)
5. 其它监控命令	(56)
四、数值运算	(59)
1. 一字节的加法	(59)
2. 二字节的加法	(63)
3. 多字节的加法	(65)

4. 二字节的减法	(71)
5. 多字节的减法	(73)
6. 乘法运算	(85)
7. 除法运算	(94)
8. 10进制运算	(100)
9. 指令讨论	(104)
五、6502的发音和绘图	(112)
1. 发音原理和发音子程序	(112)
2. 键盘控制发音	(116)
3. 高分辨绘图	(122)
4. 指令讨论	(129)
六、机器语言编程技巧	(131)
1. 程序框图	(132)
2. 转移执行指令	(137)
3. 子程序的使用	(141)
4. 资料的存放和使用	(151)
5. 指令讨论	(155)
七、BASIC语言与机器语言的配合使用	(158)
1. 机器语言的输入和调用	(158)
2. 数据的相互传递	(161)
3. 调用子程序的另一种方法	(166)
4. 一个程序实例	(168)
八、机器语言程序设计	(174)
1. 简单程序设计	(174)
2. 分支程序设计	(183)
3. 循环程序设计	(194)
4. 子程序设计	(214)
5. 堆栈程序设计	(230)
6. 常用监控子程序的调用	(248)

九、数据传送、交换、检索和排序	(281)
1. 数据传送	(281)
2. 数据交换	(292)
3. 数据比较	(303)
4. 检索	(308)
5. 排序	(320)
十、语句及自定义功能扩展	(327)
1. 单个命令或程序段功能键的自定义	(327)
2. 多个命令键的自定义	(329)
3. USR函数及其应用	(336)
4. GOTO带变量和表达式	(341)
5. GOSUB带表达式	(344)
6. RESTORE带行号	(345)
7. 控制反汇编行数	(348)
8. DOS命令的简化	(349)
9. 一行列印16个字节	(353)
10. 一行显示任意个字节	(362)
十一、绘图原理和方法	(364)
1. 低分辨率绘图子程序	(365)
2. 高分辨率绘图子程序	(368)
3. 图形清屏	(371)
4. 图形反相显示	(373)
5. 图形叠加	(376)
6. 图形动画显示	(378)
7. 汇编语言调用造型表	(383)
8. 图形绘制工具软件	(388)
十二、数学运算子程序	(408)
1. 加法程序	(408)
2. 减法程序	(422)

3. 乘法程序	(426)
4. 除法程序	(461)
5. 调用BASIC运算子程序	(484)
附录 1 显示ASCII码表	(520)
附录 2 10进制与16进制转换表	(523)
附录 3 6502 指令助记符	(524)
附录 4 6502 指令码表	(526)
附录 5 6502 操作码表	(539)
附录 6 BASIC函数运算入口地址表	(543)
附录 7 零页地址大全	(543)

一、机器语言的一般知识

为了使读者能更好地学习这本书，本章介绍机器语言的一般知识，包括机器语言的特点、数的表示方法、6502指令码的格式及内存分配等。

1. 机器语言的特点

我们知道，人和计算机进行信息交流，给计算机发送命令需要通过计算机语言。计算机语言可以分为机器语言、汇编语言、高级语言等多种类别。从机器语言到汇编语言和从汇编语言到高级语言的发展过程都是计算机语言发展中的重大突破，它的特点是用降低执行速度和减少用户空间来换取语言的易读、易写、易理解。

毫无疑问，对于大多数用户来说，使用最广又最简便易学的就是高级语言（如BASIC、FORTRAN等），而机器语言则往往用得很少。事实上，机器语言的枯燥难学和没有通用性只是它的一个方面，另一方面，它却能够实现高级语言所不能实现的功能，达到高级语言所无法达到的效果。

这两个方面综合起来才是机器语言的真正特点：

（1）难学、难写、难理解 机器语言程序是由一连串的指令码构成的，这些指令码又由16进制数A—F和0—9组成，它和要实现的功能之间没有任何联系。如A9 10就是一条指令，它的功能是把16进制数# \$ 10（其中符号\$表示其后为16进制数）放入计算机的累加器中，A9是操作码，

10是操作数，即被操作的对象。这条指令相当于 BASIC 中的 $X = 16$ 。显然，它不象高级语言那样直观和容易理解。因此，也难于掌握，难于编写程序。

(2) 没有通用性 由于机器语言是最直接、最原始的语言，它完全依赖于某种特定的计算机系统。因此，用机器语言编写的程序没有通用性。

(3) 需要人为分配内存 机器语言程序和它在运行过程中要用到的所有参数象高级语言一样也放在主机的内存中，但具体放在内存的什么位置，如何合理布局，就需要程序设计者根据机器系统和程序的实际情况来人为确定。这样就不但给编写程序增加了一个棘手的环节，而且要求程序设计者要对机器的内存使用比较了解。

(4) 运行速度最快 由于机器语言是计算机能够识别并直接产生动作的唯一语言，因此它不需要经过任何解释或编译过程，执行速度最快，为高级语言的几百倍。对于速度要求较高的程序，机器语言就能够达到令人满意的效果。

例如，在系统程序和动画程序设计中由于要求执行速度快，高级语言就无法胜任，而只能用机器语言来实现。因此，各种机器的系统程序和游戏程序大都使用机器语言。

(5) 节省内存空间 使用高级语言除了解释程序和编译程序要占用一定的空间外，程序本身也要占用比机器语言程序更多的空间。这是因为解释程序和编译程序对高级语言的源程序在结构方面有一定的要求，这种结构占用了一部分内存区。例如在 BASIC 语言程序中，每一程序行要有一个行号占二个字节，还要一个二字节的地址指针指向下一程序行在内存中的存放地址。这一部分内存存在机器语言程序中就可以完全省去；另外，高级语言程序由于受到解释或编译程

序的限制，变量（包括临时替换用的变量）都一律被安放在变量区中，增加了程序运行所要求的自由空间长度。而机器语言程序的变量可由设计者根据具体情况灵活安排，或者放在设定的变量区中，或者放在暂时不用的零散空间中。

（6）功能更加完善 从机器语言到高级语言的演变除了付出降低运行速度和减少用户空间的代价外，还失去了机器语言的部分功能。可以说，高级语言能够实现的功能机器语言都能够实现，而机器语言能够实现的功能高级语言就不一定能够实现。

（7）可用于设计扩充功能 由于机器语言可以和高级语言一起在内存中并存，其执行速度又快，因而可用来设计扩充功能，来增加系统原有的功能和对高级语言的管理能力。本书后面例举的几个扩充程序可以很好地说明这一点。

2. 数的表示

在日常生活中，我们用得最多的数制是10进制。而在计算机内部，数是用2进制来表示的，因为2进制数只有1和0两个符号，可以代表电路的通断和电位的高低，从而控制计算机的动作。

显然，用2进制表示一个数既不直观又繁琐难记，因而大多数的计算机都有处理16进制数的能力。

本书我们介绍数的2进制、10进制、16进制表示法及其相互间的转换。

（1）2进制与10进制的转换 2进制数只有1和0两个符号，逢2进位。如10进制的13用2进制表示即1101。

把一个2进制数转换成10进制数的方法是：把2进制数的倒数第一位乘以 2^0 ，加上倒数第二位乘以 2^1 ，加上倒数第

三位乘以 2^2 ，……再加上第一位乘以 2^{n-1} （其中 n 是2进制数的位数），最后的结果就是转换后的10进制数。

$$\text{如: } (101)_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

$$= (5)_{10}$$

$$(11010)_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

$$= (26)_{10}$$

从以上例子中还可以看出，一个10进制数可以表示成2的多项式，且各项的系数为1或0，把系数按高次幂到低次幂排列起来，就是这个十进制数的2进制表示形式。

因此，把一个10进制数转换成2进制数的方法是：用10进制数不断除以2，直到不能再除为止，则所有的余数按从末到首的顺序排列起来，就是这个10进制数的2进制表示。

如10进制的97，用2除：

$$2 \overline{) 97} \text{ (1)}$$

$$2 \overline{) 48} \text{ (0)}$$

$$2 \overline{) 24} \text{ (0)}$$

$$2 \overline{) 12} \text{ (0)}$$

$$2 \overline{) 6} \text{ (0)}$$

$$2 \overline{) 3} \text{ (1)}$$

$$2 \overline{) 1} \text{ (1)}$$

0

$$\text{即 } (97)_{10} = (1100001)_2$$

(2) 16进制与10进制的转换 16进制数由0—9，A—F 16个符号组成，逢16进位。

16进制与10进制的转换类似于2进制与10进制的转换：

把16进制数的倒数第一位乘以 16^0 ，加上倒数第二位乘以 16^1 ，……再加上第一位乘以 16^{n-1} （ n 为16进制数的位数），

最后的结果就是这个16进制数的10进制表示。

$$\begin{aligned}\text{如: } (A3B)_{16} &= 10 \times 16^2 + 3 \times 16^1 + 11 \times 16^0 \\ &= (2608)_{10}\end{aligned}$$

注意，16进制数中的符号A、B、C、D、E、F必须换成10、11、12、13、14、15、16后再参加运算。

同样，把一个10进制数不断地除以16，直到不能再除，则所有的余数按从末到首的顺序排列起来就是16进制（大于9的余数要换成A、B、C……等符号表示）。

如10进制的428，用16除：

$$\begin{array}{r} 16 \overline{) 428} \quad (12 \\ \underline{32} \\ 16 \overline{) 26} \quad (10 \\ \underline{16} \\ 16 \overline{) 1} \quad (1 \\ \underline{16} \\ 0 \end{array}$$

$$\text{即 } (428)_{10} = (1AC)_{16}$$

（3）2进制与16进制的转换 任何一个四位长度的2进制数都可以表示成一位的16进制数。

$$\text{如: } (1011)_2 = (B)_{16}$$

因此，把2进制数从右到左按四位一组分开，每一组分别用一位16进制数表示，就转换成了16进制数。

$$\begin{array}{cccc} \text{如: } 1101 & 1010 & 0110 & 1110 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ D & A & 6 & E \end{array}$$

$$\text{所以, } (1101 \ 1010 \ 0110 \ 1110)_2 = (DA6E)_{16}$$

相反，把一个16进制数的每一位用4位2进制数表示，就转换成了2进制数。

$$\begin{array}{cccc} \text{如: } 1 & A & 4 & E \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 0001 & 1010 & 0100 & 1110 \end{array}$$

所以, $(1A4E)_{16} = (0001\ 1010\ 0100\ 1110)_2$

表1.1给出了这三种数制之间的转换关系。

表1.1 数制转换表

10进制	16进制	2 进制	10进制	16进制	2 进制
0	0	0000	11	B	1011
1	1	0001	12	C	1100
2	2	0010	13	D	1101
3	3	0011	14	E	1110
4	4	0100	15	F	1111
5	5	0101	16	10	10000
6	6	0110	17	11	10001
7	7	0111	18	12	10010
8	8	1000	19	13	10011
9	9	1001	20	14	10100
10	A	1010			

3. 指令码的格式

6502是8位微处理机。也就是说, 它的每一个内存单元都可以存放八位长度的2进制数。前面曾经介绍过, 一个4位的2进制数可以用一位的16进制数表示, 8位的2进制数就可以用2位16进制数表示。这2位16进制数就称为一个字节。

6502有一套完整的指令系统, 其指令的一般格式是:

操作码 操作数

即一个指令由操作码和操作数两部分组成。操作码说明指令的功能(取数、存数、运算、转移等), 操作数指明被操作的对象。在6502的指令中, 操作数部分可以是隐含的,

可以是被操作数的本身，也可以是操作数存放的内存地址。
如：

CA是一条指令，占一个字节，它的操作数隐含为X寄存器，功能是把X寄存器中的数减去1；

A9 10是一条指令，占二个字节，它的操作数10就是被操作的对象本身，功能是把十六进制数10放入累加器A中；

AD 20 03也是一条指令，占三个字节，它的操作数20 03是指被操作的对象在内存中的存放地址，功能是把地址0320中的数放入累加器A中。

6502的内存地址可以用4位16进制数表示，其范围是0000—FFFF（即10进制的0—65535）。在4位长度的地址中，左边两位为高位，右边两位为低位。如地址0320的高位是03，低位是20。在机器语言指令中，地址的低位写在前面，高位写在后面。因此，最后一条指令写成AD 20 03。

上述三条指令根据它们各自所占的字节数分别称为一字节指令、二字节指令和三字节指令。

4. 6502的寄存器

前面我们说过，编写机器语言程序必须人为分配内存，这些内存（0000—FFFF）一部分存有计算机本身的系统程序，包括BASIC解释程序和各種管理程序等，用户无法或不能使用这一部分内存来存放和运行程序；另一部分（对64K的机器来说大约是48K）则专门供用户支配使用。内存中的每一个单元都可以存放一个字节的数，即8位2进制数或2位16进制数。

但是，机器指令无法对这些内存单元中的数进行直接处理或运算，还必须通过几个具有特殊功能的单元。这些有特

殊功能的单元就是寄存器。

6502有6个寄存器，它们是：

- A——累加器A；
- X——寄存器X；
- Y——寄存器Y；
- PC——指令计数器；
- P——标志寄存器；
- S——堆栈指示器。

下面先介绍A、X、Y这3个寄存器，其余3个留待后面介绍。

1. 累加器A 这是一个8位长度的寄存器，它具有一般内存单元的所有功能：可以存放和读出数据。另外，它还可以进行算术运算和逻辑运算。因此，6502的许多指令都和累加器A有关。

2. 寄存器X、Y 寄存器X和寄存器Y的长度也是8位，而且有内存单元读写数据的功能。但是，它只能进行加1和减1运算。因此，常常用来存放控制值，控制循环的次数和执行的次数等。

机器语言指令的功能从表面上来看非常简单（但这些功能足以设计出各种复杂的软件），它只能进行算术、逻辑运算和数据的转移等，因此，在编写机器语言程序中时刻离不开上述三个寄存器。

5. 内存分配

本章的最后一节介绍内存分配情况，这也是学习机器语言所必须掌握的知识。在编写一个机器语言程序之前，要确定好它的存放地址，这样，其中的转移、调用以及数据的规

划布局才可以具体落实。但是，机器语言程序并不是任何位置都可以存放的，必须先分清什么地方能放，什么地方不能放。

(1) 内存容量的计算 表示主机的内存容量大小通常有两种方法：一种是用字节数来表示，中华学习机的地址范围是0000—FFFF，即65536（10进制）个字节的内存容量；另一种是用多少个K来表示，1K为1024个字节。因此，65536个字节即64K。

这64K内存当然不可能全部提供给用户使用，其中有一部分已被系统程序所占据，这一部分大约有16K，余下的48K就是用户区。

(2) 内存的分配 中华学习机的内存从用途上可分为二类：一类是随机读写内存区RAM (Random Access Memory)，这类内存区不但可以读出数据，还可以把数据写进去。其范围是0000—BFFF共48K；另一类是只读内存区ROM (Read Only Memory)，这类内存区只能读出数据，而无法写进去。其范围是C000—FFFF共16K。ROM的设置是为了保证系统程序的安全而采取的措施。

内存的分配详见表1.2。

从表1.2中可以看到，0300—03FF、0800—BFFF两段内存区是供用户使用的，机器语言程序就可以存放在这里。根据实际情况，一般长的程序存放在0800—BFFF中，短的程序存放在0300—03FF中。但还必须注意以下几点：

① 0320—032A中存放有高清晰度作图时的若干参数；0399—03EA中存放有一些磁盘操作的子程序及入口程序；03EB—03FF中存放有一些机器语言子程序和键盘指令的入口地址。因此，真正的用户区只有0300—031F的单元；

表1.2 内存分配表

内存范围		功 用
16进制	10进制	
0000—00FF	0—255	零页区, 存放机器运行中的一些参量
0100—01FF	256—511	堆栈区, 存放堆栈数据
0200—02FF	512—767	键盘缓冲区, 存放从键盘输入的字符
0300—03FF	768—1023	用户区
0400—07FF	1024—2047	文本及低清晰度作图第二页存贮区
0800—0BFF	2048—3071	用户区, 文本及低清晰度作图第二页存贮区
0C00—1FFF	3072—8191	用户区
2000—3FFF	8192—16383	高清晰度作图第一页存贮区
4000—5FFF	16384—24575	高清晰度作图第二页存贮区
6000—BFFF	24576—49151	用户区
C000—CFFF	49152—53247	I/O区
D000—FFFF	53248—65535	系统程序区

② 从表1.2中可以看到: 0800—0BFF、2000—3FFF 和4000—5FFF 三段内存既是用户区, 又是屏幕存贮区。如果需要使用文本和低清晰度作图第二页、高清晰度作图第一页或高清晰度作图第二页, 则相应的内存区不能存放机器语言程序或参数;

③ 0800—BFFF的区域也是 BASIC 程序及其变量的存放区, 如果机器语言与BASIC语言程序在内存中并存, 就必须合理规划。

二、字符的显示和输入

学习完第一章的内容后，相信读者已经掌握了6502机器语言的一般知识。这样，在以后的学习中就会少一些疑问，多一层理解。从本章开始到第五章着重介绍6502机器语言的各种指令及其用法。

1. 指令的分类、寻址方式

为了使读者在学习各种指令之前能对6502机器语言指令有一个大概的了解，本节先介绍一下指令的分类及寻址方式。

(1) 指令的分类 6502有56条指令，13种寻址方式。每一条指令可以有多种寻址方式，这样一共有151个操作码。因为一个字节能表示0—255范围内的数，所以6502的操作码都用一个字节来表示。这56条指令按其功能可分为如下十类：

- ① 数据传送指令，12条；
- ② 算术运算指令，8条；
- ③ 逻辑运算指令，3条；
- ④ 比较指令，4条；
- ⑤ 转移指令，13条；
- ⑥ 移位指令，4条；
- ⑦ 标志位赋值指令，7条；
- ⑧ 堆栈操作指令，4条；

⑨ 空操作指令，1条；

⑩ 中断指令，1条。

实际上，也可以把这十类指令粗略地分成三大类，即：
数据传送指令（包括标志位赋值指令、堆栈操作指令）；
运算指令（包括算术运算指令、逻辑运算指令、比较指令和移位指令）；

编程辅助指令（包括转移指令、空操作指令和中断指令）。

（2）寻址方式 6502有56条指令，如果给它们每一条指令规定一个操作码，则56个操作码就可以驱动CPU完成56种动作。但由于上面所述的每一条指令在具体应用时还会有各种不同的情况，因而56个操作码还不能全部解决问题。

例如LDA是一个便于记忆的助记符，相当于一条指令。它的功能是取一个数放到累加器A中。这个数从什么地方取至少可以有以下几种情况：

① 直接取操作码后的这个数到累加器A中（这个数称为立即数）；

② 从一个内存地址中取一个数到累加器A中；

③ 从以寄存器X或Y为基地址（变地址）加上一个变地址（基地址）的内存中取一个数到累加器A中。

显然，这些不同的情况是无法用同一个操作码来识别的。因此，6502中规定了13种寻找操作数或操作数地址的方法，即13种寻址方式。在56条指令中，最多的一条指令有8种寻址方式（8个操作码），最少的只有1种寻址方式（1个操作码）。经过这样组合以后，6502共有151个操作码。

现将13种寻址方式分成10类进行介绍。

① 立即寻址。立即寻址是指操作码后面紧接着操作数

本身的寻址方式。这个操作数就是立即数。

如：A9 FF LDA # \$FF

这是一个立即寻址指令。A9是操作码，表示把其后的立即数FF送到累加器A中。FF为立即数，LDA # \$FF是这个指令的汇编形式，#表示其后为立即数，\$表示其后为16进制数。

由于立即数都是一个字节长度（2位16进制数）的，因此立即寻址指令是二字节指令。

② 绝对寻址。绝对寻址是指操作码后面跟着操作数存放地址的寻址方式，这个地址称为绝对地址，绝对地址中存放的数才是真正的操作数。

如：AD 20 03 LDA \$0320

这是一个绝对寻址指令。AD是操作码，表示把地址\$0320中的数送到累加器A中。\$0320是绝对地址，在机器语言中，地址的低位在前，高位在后。

绝对地址用两个字节表示（即地址的低、高位），因此绝对寻址指令为三字节指令。

③ 零页寻址。零页寻址实际上是绝对寻址的一种特例，它也是指操作码后跟着操作数存放地址的寻址方式。但是，这个存放地址的范围只能是0000—00FF，由于地址的高位是00（称为零页地址），因此在指令中被省略了。

显然，零页寻址可以用绝对寻址来代替（高位用00补），但前者少占用一个字节，且指令的执行速度更快。

如：A5 10 LDA \$10

这是一个零页寻址指令。A5是操作码，表示把地址0010中的数送至累加器A中。如果用绝对寻址方式，就是：

AD 10 00 LDA \$0010

零页寻址指令是二字节指令。

④ 隐含寻址（或寄存器寻址）。隐含寻址或寄存器寻址是指指令在形式上没有操作数，实际上操作数已隐含在某个寄存器中的寻址方式。操作数在哪个寄存器中，由操作码隐含约定。

如：0A ASL A

这是一个隐含寻址指令。0A 是操作码，它表示把累加器 A 中数的各个数元左移一位，右边补零。累加器 A 中的数就是操作数，在指令中没有出现，而是隐含的。

又如：E8 INX

这也是一个隐含寻址指令，操作码 E8 表示对寄存器 X 中的数进行加 1 操作。

隐含寻址指令是一字节指令。

⑤ 相对寻址。绝对寻址是用二个字节直接指出某一内存地址的，相对寻址则是相对于当前地址往前或往后多少个单元来计算地址的，它只能用于条件转移指令中。

设从地址 0300 开始有如下一段程序：

0300-	A2 10	LDX	# \$10
0302-	B5 00	LDA	\$00,X
0304-	9D 00 08	STA	\$0800,X
0307-	CA	DEX	
0308-	D0 F8	BNE	\$0302
030A-	60	RTS	

这段程序的功能是将地址 0001 至 0010 内的数据顺序搬移到地址 0801 至 0810 中。地址 0308 中的 D0 F8 指令就是一个相对寻址指令，它表明当 X 内的值不为零时转向 0302 继续执行，这个地址 0302 就是相对于当前地址来定位的。

相对寻址指令是二字节指令，其第一个字节为操作码，说明约定的条件；第二个字节为相对地址。当条件为真时转向此地址执行，否则继续执行下一条指令。

⑥ 寄存器绝对寻址。 寄存器绝对寻址是指把指令中的绝对地址加上寄存器X或Y中的值构成操作数存放地址的寻址方式。因此，这里所说的寄存器仅指X和Y。

如：BD 1E 03 LDA \$031E,X
和 B9 1F 03 LDA \$031F,Y

第一条指令是寄存器X的绝对寻址指令，它表示把地址值031E和X中的值相加后所指的地址单元中的数送往累加器A；第二条指令是寄存器Y的绝对寻址指令，它表示把地址值031F和Y中的值相加后所指的地址单元中的数送往累加器A。

如果在指令执行前X中的值为A0，Y中的值为20，则第一条指令的操作数地址为 $031E + A0 = 03BE$ ，第二条指令的操作数地址为 $031F + 20 = 033F$ 。

寄存器绝对寻址指令是三字节指令。

⑦ 寄存器零页寻址。 寄存器零页寻址是寄存器绝对寻址的一种特例，它表示把指令中的零页地址加上寄存器X或Y中的值构成操作数存放地址。

如：B5 0F LDA \$0F,X 和
96 00 STX \$00,Y

等都是寄存器零页寻址指令。这种寻址方式的指令是二字节指令。

应该注意，一个零页地址（00—FF）加上X或Y中的值所构成的新地址可能会超出零页（如 $E0 + 45 = 0125$ ）而进入第一页，这样就会破坏系统的堆栈区。因此，当出现这

种情况时（即相加后的地址高位不为0），系统将不考虑其高位的值，使有效地址仍在零页中。如：

94 F0 STY \$F0,X

指令，若X的值为5A，则相加后的地址为014A，但实际执行时该地址被认为004A，高位取零。

⑧ 间接寻址。操作码后跟着一个内存地址，但这个地址不是直接的操作数存放地址，而是存放操作数地址的地址。这样一种寻址方式称为间接寻址。间接寻址指令只有无条件转向一条，操作码是6C。

如：6C 00 03 JMP(\$0300)

操作码6C表示程序执行到这里时无条件转向另一个内存地址执行，但这个内存地址并不是0300本身，而是0300和0301中的数所决定的地址（由于一个内存地址需要两个字节才能表示出来，因此0300和0301中分别存放低、高位）。

如果0300中的数是00，0301中的数是F8，则这一指令使程序转向地址F800执行。显然，间接寻址指令是三字节指令。

⑨ 先变址的间接寻址。这种寻址方式在操作码后给定一个零页地址（一字节），先对这一零页地址进行寄存器X变址（即加上X中的数），再作为间接地址使用。与寄存器零页寻址一样，如果进行X变址后超出零页，则变址后的高位取0。

如：A1 2E LDA(\$2E,X)

其中的括号表示零页地址与X先进行变址操作。设X中的数是F0，则进行X变址后为011E(2E + F0)，高位取0后间接地址为1E，即零页地址1E、1F中分别存放操作数地址的低、高位。这时，若1E中的数是AB，1F中的数是08，则

该指令将把08AB地址单元中的数送至累加器A中。

先变址的间接寻址指令是二字节指令。

⑩ 后变址的间接寻址。 这种寻址方式也是在操作码后给定一个零页地址，先对该地址进行间接寻址找出一个基地址，再进行寄存器Y变址。

如：B1 30 LDA (\$30) , Y

这里的括号将Y放在外面，表示先间接寻址后再变址。若地址30、31中的数分别是40、03，Y中的数是18，则间接寻址后的基地址为0340，加上Y后操作数地址是0358，即将地址0358中的数送至累加器A中。

后变址的间接寻址指令也是二字节指令。

在本节指令的分类及寻址方式中，例举了一些指令的例子，这些指令都将在以后的章节中逐个详细介绍。在此，读者可以不必追求过细的理解。

2. 文本状态的字符显示

在BASIC语言中，通常使用PRINT语句在文本状态的屏幕上显示字符，并使用VTAB、HTAB、PRINT TAB等语句进行屏幕定位。由于在一个程序运行的过程中，需要它显示出运行结果、运行情况、操作提示等信息，因此，这一过程几乎是任何程序都必须具备的。那么，它在机器语言中又如何实现呢？

要说明这个问题，首先必须了解中华学习机的屏幕设置。

(1) 屏幕的设置 读者在使用BASIC语言时，可能碰到过POKE和PEEK语句。POKE语句的功能是把一个0—255之间的10进制数存入到指定的内存单元中；PEEK语

句的功能是读出指定内存单元中的数。

如POKE 1024,192即把10进制数192放入10进制地址单元1024中。执行这个语句时可以看到屏幕的左上角出现一个“A”字符(192是A字符的ASCII*码值)。这说明文本状态的左上角位置就是地址1024。

实际上,中华学习机的显示屏幕正是映象式的,即屏幕上的任何一个位置都是内存中一个地址单元的映象。屏幕可以有文本第一、第二页,低清晰度作图第一、第二页,高清晰度作图第一、第二页等。按照表1.2中所列,文本第一页的地址是0400—07FF(即10进制1024—2047),所以1024就是文本页的第一个位置(左上角)。

这种一一对应的映象在同一行上是连续的(如第0行的第0个位置对应地址0400,第1个位置对应地址0401,……),但行与行之间是不连续的(如地址0427对应第0行的最后一个位置,而地址0428则对应第8行的第一个位置)。参看附录2。

(2) 显示一个字符 要在屏幕上显示一个字符,首先必须查出这个字符的ASCII码值(见附录1)和显示位置对应的内存地址。

如要在第0行第0列上显示字符“*”,先查出“*”的ASCII码值为AA,第0行第0列的对应地址是0400,再使用下面二个指令:

A9 AA	LDA#\$AA	把AA送入累加器A中
8D 00 04	STA\$0400	把累加器A中的数(AA)

* ASCII为美国信息交换标准码

执行这两个指令后，屏幕的相应位置就会出现“*”字符。

为了执行上述指令，需要把它放入一个内存区。这里，我们把它放入0300开始的地址中，并加以完善。如程序2.1。

例程序2.1

```
0300-    20 58 FC      JSR    $FC58
0303-    A9 AA        LDA    # $AA
0305-    8D 00 04      STA    $0400
0308-    60           RTS
```

程序说明：

0300：调用FC58的子程序清洗屏幕；

0303：把ASCII码值AA送入累加器A中；

0305：把累加器A中的数（AA）存入地址0400中；

0308：返回。

程序2.1占0300—0308共九个内存单元，由4条指令构成：

① 20 58 FC JSR \$FC58。子程序调用指令。20是操作码，表示调用一个机器语言子程序，这个子程序的绝对地址用二个字节在操作码后给出，且低位在前，高位在后。这个指令与BASIC中的GOSUB类似，其功能是清洗文本页面（HOME）；

② A9 AA LDA # \$AA。累加器A的立即赋值指令。A9是操作码，表示把其后的立即数AA送入累加器A中。AA就是字符“*”的ASCII码值。从这里我们可以看到，机器语言并不直接接受要显示的字符本身，而是用它的ASCII码值来表示。

③ 8D 00 04 STA \$0400。 内存单元的赋值指令。8D是操作码，表示把累加器A中的当前值(AA)送入地址单元0400中，使屏幕的左上角显示出字符“*”；

④ 60 RTS。返回指令。60是操作码，表示该程序段至此结束，退出运行状态。

把这一程序输入内存并执行就可以实现显示字符的功能。但这种显示方法并不方便，因为它需要查出对应地址，而文本页面占960个地址之多。

因此，我们需要采用另一种方法来显示字符。见程序2.2。

例程序2.2

```
0300-    20  58  FC      JSR    $FC58
0303-    A9  AA          LDA    # $AA
0305-    20  ED  FD      JSR    $FDED
0308-    60              RTS
```

程序说明：

0300：清洗屏幕；

0303：把AA送入累加器A中；

0305：显示累加器A中的字符；

0308：返回。

程序2.2中也用了四条指令来完成程序2.1的功能，它与程序2.1的不同之处在于地址0305中用了子程序调用指令，调用FDED为起始地址的系统子程序。这一子程序的功能是在当前的屏幕位置上显示累加器中的字符，即相当于对累加器中的字符作 PRINT 操作，且显示后光标前进一格，不带换行。

从程序2.2中可以看到，调用FDED显示子程序避免了

查找显示地址。

(3) 显示字符串 知道了如何显示一个字符, 就可以显示出字符串来。最直接的办法是重复使用LDA和STA或LDA和JSR二个指令, 直到把字符串中的所有字符显示完。

例程序2.3

```
0300-    20  58  FC      JSR    $FC58
0303-    A9  C1          LDA    # $C1
0305-    20  ED  FD      JSR    $FDED
0308-    A9  C2          LDA    # $C2
030A-    20  ED  FD      JSR    $FDED
030D-    A9  C3          LDA    # $C3
030F-    20  ED  FD      JSR    $FDED
0312-    60              RTS
```

程序说明:

0300: 清洗屏幕;

0303: 把“A”的ASCII码C1送入累加器A;

0305: 显示字符“A”;

0308: 把“B”的ASCII码C2送入累加器A;

030A: 显示字符“B”;

030D: 把“C”的ASCII码C3送入累加器A;

030F: 显示字符“C”。

程序2.3在屏幕的左上角显示出字符串“ABC”, 实现了字符串显示的功能。但是, 如果要显示的字符串长度较长, 用这种方法就不方便, 程序也会显得很长。因此, 必须用循环来缩短程序的长度, 循环的控制量可以放在寄存器X或Y中。

例程序2.4

```
0300-    20  58  FC      JSR    $FC58
0303-    A2  00          LDX    # $00
```

```

0305-   BD 20 03      LDA    $0320,X
0308-   20 ED FD      JSR    $FDED
030B-   E8            INX
030C-   E0 07         CPX    # $07
030E-   D0 F5         BNE    $0305
0310-   60            RTS
0320-   B6 B5 B0 B2 C3 D0 D5

```

程序说明:

0300: 清洗屏幕;

0303: 寄存器X赋初值00;

0305: 将0320 + X地址中的数送入累加器A;

0308: 显示累加器A中的字符;

030B: 寄存器X中的值加1;

030C: 比较X中的值与07;

030E: 如果不相等则转0305执行;

0320: 字符串“6502CPU”中每一字符的ASCII码。

执行程序2.4后, 屏幕左上角将显示出字符串“6502CPU”。运行框图见图2.1。

在程序2.4中除用了前面介绍过的几个指令外, 还用了下列几个指令:

① A2 00 LDX # \$00。这是一个寄存器X的立即赋值指令。A2是操作码, 表示把其后的立即数00送入寄存器X;

② BD 20 03 LDA \$0320,X。这是一个累加器A的变址寻址指令。BD是操作码, 表示把其后二字节地址加上X的值构成的新地址中的数送入累加器A中。因此, 当X的值等于00时, 送0320中的数B6(字符“6”的ASCII码)到A; X的值等于01时, 送0321中的数B5(字符“5”

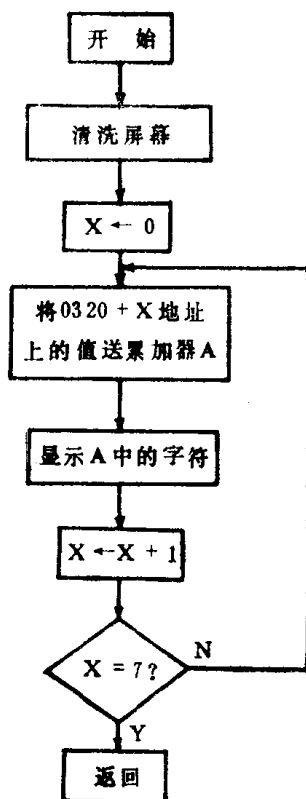


图2.1

的ASCII码) 到A;

③ E8 INX。这是一个隐含寻址指令，操作码 E 8 表示把寄存器X中的值加上 1，以便于控制循环及取用下一个数；

④ E0 07 CPX #07。这是一个寄存器X的比较指令，E0是操作码，表示把X中的数与操作码后的值07进行比较；

⑤ D0 F5 BNE \$0305。这是一个条件转移指

令，D0是操作码，表示当上一步比较 X与07 不相等 时转向 0305执行，否则继续执行0310的指令。

地址0320—0326中存放字符串“6502CPU”每一字符的ASCII码值。

3. 接受 按 键

上面介绍了单个字符和事先存放在内存中的字符串的显示方法。在程序运行过程中，往往还需要接受操作者的输入信息，并将之在屏幕上显示出来。

这个问题可以分为两种情况：一种是程序暂停执行，等待键盘输入；另一种是对键盘进行扫描，判断有无输入。前一种情况在BASIC中通过GET或INPUT 语句实现，后一种情况则通过对键盘接受单元的扫描实现。

(1) 等待输入 在主机的ROM区中有一个GET系统子程序，这个GET子程序的起始地址(入口地址)是FD35，它的功能是暂停程序的执行，等待操作者按下一个键，并将按键字符的ASCII码值送到累加器A中。借助这个子程序，可以设计出一段程序接受按键并显示出来。

例程序2.5

0300-	20	58	FC	JSR	\$FC58
0303-	20	35	FD	JSR	\$FD35
0306-	20	ED	FD	JSR	\$FDED
0309-	60			RTS	

程序说明：

0300：清洗屏幕；

0303：等待按键；

0306：显示所按字符；

0309: 返回

程序2.5用了三条子程序调用指令和一条返回指令。其中20 35 FD就是调用GET子程序的指令，执行这一指令时，屏幕左上角将出现光标，等待按键，按键字符的ASCII码值存放在A中，因此，下一条指令20 ED FD可以直接显示出所按的字符。

当然，也可以不断重复按键和显示这个过程，只要把地址0309中的返回指令60改为4C 0303即可，它的汇编形式是JMP \$0303。

这是一条无条件转向指令，4C是操作码，表示当程序执行到这一指令时无条件转向操作码后二个字节指向的地址（低位在前，高位在后）。

把程序2.5再改进一下：

例程序2.6

0300-	20	58	FC	JSR	\$FC58
0303-	A2	00		LDX	# \$00
0305-	20	35	FD	JSR	\$FD35
0308-	C9	8D		CMP	# \$8D
030A-	F0	0A		BEQ	\$0316
030C-	20	ED	FD	JSR	\$FDED
030F-	9D	20	03	STA	\$0320,X
0312-	E8			INX	
0313-	4C	05	03	JMP	\$0305
0316-	60			RTS	

程序说明：

0300: 清洗屏幕；

0303: 清除寄存器X；

0305: 等待按键；

0308: 是接入回车键吗?

030A: 是, 则转0316结束;

030C: 否则, 显示出所按字符;

030F: 并存入0320 + X单元中;

0312: 寄存器X中的值加1;

0313: 再转0305执行;

0316: 返回。

其执行框图如图2.2。

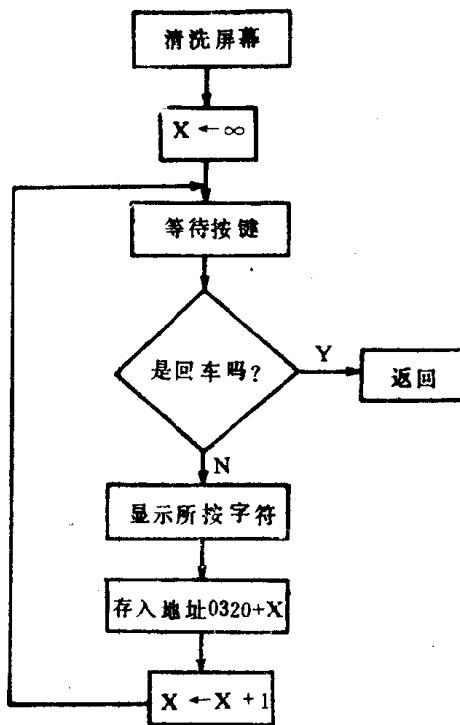


图2.2

(注: 框图中∞应为0)

从图2.2中可以看到这个程序可接受多个字符,并在屏幕的最顶行依次显示出来,直到接入回车键为止。按入字符的个数在寄存器X中,字符依次存入0320开始的地址单元中。

在程序2.6中，使用了三个新的指令：

① C9 8D CMP #8D。这是一个累加器A的比较指令，C9是操作码，表示将累加器A中的数与8D（回车符的ASCII码值）进行比较，判断输入的是不是回车符。

② F0 0A BEQ \$0316。这是一个条件转移指令，F0是操作码，表示上一步比较累加器A中的数等于8D时转向0316执行，否则继续执行030C中的指令。

③ 9D 20 03 STA \$0320,X。在本章第二节中我们曾介绍过指令BD 20 03，这个指令正好与9D 20 03起着相反的作用：前者是把0320+X地址内的数送入累加器A中；后者则是把累加器A中的数存到地址0320+X内。9D是这个指令的操作码，其后的地址是二字节形式的绝对地址。

（2）键盘的扫描输入 等待输入与键盘扫描输入的区别在于前者使程序执行暂停，直到按下一个键；后者对键盘接受单元扫描后，判断有无输入及输入什么字符，并继续执行下一条指令。因此，等待输入常用于必须接受用户输入信息才能执行下一步的程序段中，扫描输入则常用于随机的输入中。

例如，在游戏程序中，为了控制物体的移向和动作，操作者必须按入特定的命令键。这种按键在时间上是随机的。程序必须即时进行检测，如有按键，则按照规定改变移向和动作；否则，程序也不能停顿，应继续使物体保持原来的动作。

我们知道，在BASIC中键盘接收单元和键盘清除单元的10进制地址分别是-16384和-16368，其16进制表示为C000和C010。例程序2.7就是使用这两个地址单元进行扫描输入的例子。

例程序2.7

0310-	AD	00	C0	LDA	\$C000
0313-	C9	80		CMP	# \$80
0315-	90	0A		BCC	\$0321
0317-	C9	9B		CMP	# \$9B
0319-	D0	01		BNE	\$031C
031B-	60			RTS	
031C-	A9	00		LDA	# \$00
031E-	8D	10	C0	STA	\$C010

程序说明:

0310: 把C000中的数送入累加器A;
0313: 比较累加器A中的值与80;
0315: 若A小于80, 则转0321继续执行;
0317: A等于9B (ESC的ASCII码) 吗?
0319: 不等, 转031C继续执行;
031B: 返回;
031C: 把00送入累加器A;
031E: 把A的值送至C010单元。

程序2.7的执行框图见图2.3

在程序2.7中, 首先取出键盘接收单元C000中的数, 并判断是否大于16进制数80。如果小于80, 说明没有按下任何键, 直接转0321继续执行; 反之, 如果大于等于80, 就说明已经按下一个键, 它的按键ASCII码值就在地址C000中。下一步, 判断C000中的数是否等于9B (ESC键的按键ASCII码值), 如果不相等, 则转031C清除键盘接收并继续执行; 如果等于9B, 说明按下了ESC键, 因而执行031B中的返回指令, 使程序退出运行。

这里, 我们又遇到了二个新的指令:

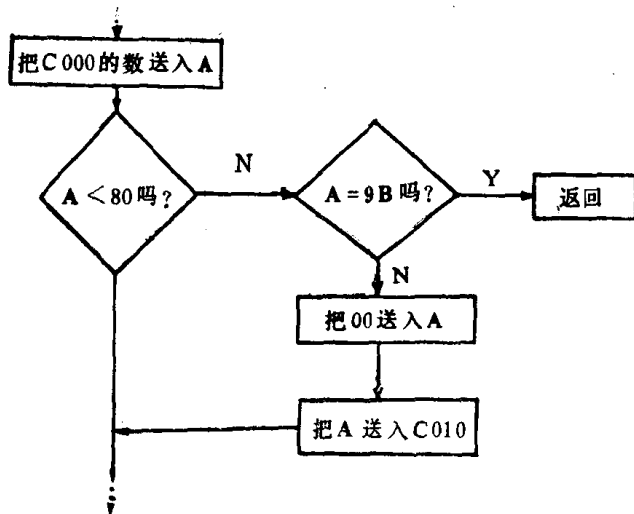


图2.3

① `AD 00 C0 LDA $C000`。这是一个累加器A的绝对寻址指令。AD是操作码,表示把其后二个字节所指的地址单元C000中的数送入累加器A中。

② `90 0A BCC $0321`。这是一个条件转移指令。90是操作码,表示当上一步比较结果A小于80时转向0321执行。地址0321就是通过操作数0A相对寻址得到的。

4. 闪烁及反相字符的显示

为了使屏幕上的提示和说明更加显眼突出,程序中往往需要用闪烁或反相方式显示一些信息。为此,BASIC中设置了FLASH(闪烁)、INVERSE(反相)和NORMAL(正常)三个语句来实现它们之间的转换。

在机器语言中我们可以通过使用不同范围内的ASCII码实现这一功能。参看附录1,其中ASCII码值00—3F的字符是反相字符,40—7F的字符是闪烁字符。这些字符可以

象正常字符一样显示出来。

例如，在程序2.3中我们用正常 ASCII 码值C1、C2、C3显示出字符串“ABC”。同样，可以用反相ASCII码值01、02、03显示出反相字符串“ABC”（见程序2.8），用闪烁ASCII码值41、42、43显示出闪烁字符串“ABC”（见程序2.9）。

例程序2.8

```
0300-   20   58   FC      JSR    $FC58
0303-   A9   01          LDA    # $ 01
0305-   20   ED   FD      JSR    $FDED
0308-   A9   02          LDA    # $ 02
030A-   20   ED   FD      JSR    $FDED
030D-   A9   03          LDA    # $ 03
030F-   20   ED   FD      JSR    $FDED
0312-    60              RTS
```

程序说明：

0303：把数01（反相字符“A”的ASCII码）送入累加器；

0305：显示反相字符“A”；

0308：把数02（反相字符“B”的ASCII码）送入累加器；

030A：显示反相字符“B”；

030D：把数03（反相字符“C”的ASCII码）送入累加器；

030F：显示反相字符“C”。

例程序2.9

```
0300-   20   58   FC      JSR    $FC58
0303-   A9   41          LDA    # $ 41
0305-   20   ED   FD      JSR    $FDED
0308-   A9   42          LDA    # $ 42
030A-   20   ED   FD      JSR    $FDED
030D-   A9   43          LDA    # $ 43
```

```

030F-   20   ED   FD       JSR    $FDED
0312-   60                               RTS

```

程序说明:

0300: 清洗屏幕;
 0303: 把数41(闪烁字符“A”的ASCII码)送入累加器;
 0305: 显示闪烁字符“A”;
 0308: 把数42(闪烁字符“B”的ASCII码)送入累加器;
 030A: 显示闪烁字符“B”;
 030D: 把数43(闪烁字符“C”的ASCII码)送入累加器;
 030F: 显示闪烁字符“C”;
 0312: 返回。

当然, 我们也可以模拟BASIC中的FLASH、INVERSE和NORMAL语句来显示同样的字符。这里, 我们首先要了解: 在ROM中有三个起始地址分别为F280、F277和F273的功能子程序, 可以实现FLASH、INVERSE和NORMAL的功能。使用子程序调用指令进入某种显示方式后, 就能显示出这种方式下的字符。

例程序2.10

```

0300-   20   58   FC       JSR    $FC58
0303-   20   80   F2       JSR    $F280
0306-   A9   C1           LDA    # $C1
0308-   20   ED   FD       JSR    $FDED
030B-   20   77   F2       JSR    $F277
030E-   A9   C2           LDA    # $C2
0310-   20   ED   FD       JSR    $FDED
0313-   20   73   F2       JSR    $F273
0316-   A9   C3           LDA    # $03
0318-   20   ED   FD       JSR    $FDED
031B-   60                               RTS

```

程序说明:

0303: 进入闪烁状态;
0306: 把C1送入累加器 A;
0308: 显示闪烁字符 “A”;
030B: 进入反相状态;
030E: 把C2送入累加器 A;
0310: 显示反相字符 “B”;
0313: 返回正常状态;
0316: 把C3送入累加器 A;
0318: 显示正常字符 “C”。

在程序2.10中,我们仍然用正常的ASCII码值C1、C2和C3,但由于在地址0303中进入了闪烁状态,因此显示出的字符“A”是闪烁方式的;地址030B中进入了反相状态,因此字符“B”是反相方式的;地址0313中返回了正常状态,因而字符“C”是正常方式的。

5. 指令讨论

在本章中我们使用了15个机器语言指令。为了便于读者学习和掌握,这里把15个指令综合在一起进行讨论,并给出每个指令使用的例子。

① LDA: 使用在立即寻址方式下,将操作码后的一字节数存入累加器A中。操作码是A9。

例: A9 FF LDA #\$FF

即将一字节16进制数FF存入累加器A。

② LDX: 使用在立即寻址方式下,将操作码后的一字节数存入寄存器X中。操作码是A2。

例: A2 10 LDX #\$10

即将一字节16进制数10存入寄存器X。

③ INX: 使用在隐含寻址方式下, 将寄存器 X 中的数加 1。操作码是 E8, 没有操作数。

例: E8 INX

由于 X 是一个八位长度的寄存器, 它只能存放 00—FF (即 10 进制 0—255) 范围内的数。因此, 如 X 中原来的数是 FF, 则使用 INX 指令后, X 中的数将变为 00。

2 进制	16 进制
$ \begin{array}{r} 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \\ + \qquad \qquad \qquad 1 \\ \hline 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \end{array} $	$ \begin{array}{r} FF \\ + \quad 1 \\ \hline 1\ 0\ 0 \end{array} $

由算式可以看出: 数 FF 加 1 后结果为 9 位数 (100000000), 最高位为 1, 其余 8 位为 0。这时, 寄存器 X 就只存放右边的低 8 位 (其 16 进制值为 00), 而把最高位 1 丢掉 (实际上, 最高位会存入标志寄存器的进位标志中, 下一章将详细介绍)。

④ LDA: 使用在绝对寻址方式下, 将操作码后的内存地址中的数存入累加器 A。操作码是 AD。

例: AD 00 C0 LDA \$C000

即将地址 C000 中的数存入累加器 A, 地址 C000 保持原来的数不变。

⑤ LDA: 使用在寄存器 X 绝对寻址方式下, 将操作码后的数加上寄存器 X 中的数所指定的内存地址中的数存入累加器 A。操作码是 BD。

例: BD 10 03 LDA \$0310, X

即将 0310 + X 地址中的数存入累加器 A。如 X 中的数为 10, 则将地址 0320 中的数存入累加器 A。

⑥ STA: 使用在绝对寻址方式下, 将累加器 A 中的数存入操作码后给出的内存地址中。操作码是 8D。

例: 8D 10 C0 STA \$C010

即将累加器A中的数存入地址C010中, 而A中保持原来的数不变。

⑦ STA: 使用在寄存器X绝对寻址方式下, 将累加器A中的数存入操作码后的数加上寄存器X中的数所指向的内存地址中。操作码是9D。

例: 9D 00 03 STA \$0300, X

即将累加器A中的数存入地址0300 + X中。如X中的数为2A, 则存入032A中。

⑧ JSR: 使用在绝对寻址方式下, 使程序转向调用机器语言子程序, 子程序的入口地址即操作码后给出的二字节地址。操作码是20。这一指令类似于BASIC中的GOSUB语句, 不过GOSUB后给出的是子程序的入口行号, 而JSR指令给出的是子程序的入口地址。子程序返回后将继续执行JSR后的第一条指令。

例: 20 8E FD JSR \$FD8E

即调用入口地址为FD8E的子程序, 执行回车操作。

⑨ JMP: 使用在绝对寻址方式下, 使程序执行无条件转向操作码后给出的二字节内存地址。操作码是4C。这一指令类似于BASIC中的GOTO语句。

例: 4C 50 03 JMP \$0350

即无条件转向地址0350继续执行。

⑩ RTS: 使用在隐含寻址方式下, 从子程序返回到主程序或从程序执行状态返回到键盘操作状态。操作码是60。

例: 60 RTS

显然, RTS是机器语言子程序的结束标志, 同时使程序执行返回到主程序中JSR指令的后一条指令, 即相当于

BASIC中的RETURN。

设有主程序：

0300-	20	58	FC	JSR	\$FC58
0303-	A2	00		LDX	#\$00
0305-	20	35	FD	JSR	\$FD35
0308-	20	10	03	JSR	\$0310
030B-	E0	14		CPX	#\$14
030D-	D0	F6		BNE	\$0305
030F-	60			RTS	

和子程序：

0310-	20	ED	FD	JSR	\$FDED
0313-	E8			INX	
0314-	60			RTS	

程序说明：

0303：给寄存器X赋初值00；

0305：接受操作者按键；

0308：调用0310子程序；

030B：X等于14吗？

030D：不等，转0305重复上述过程；

0310：显示累加器A中的字符（即刚刚输入的）；

0313：寄存器X的值加1；

0314：子程序返回（到030B）。

主程序在0308上调用子程序0310，当执行到0314中的出口指令时，程序返回到JSR后的第一条指令，即030B。

另外，RTS还具有结束程序运行，返回到键盘操作状态的功能。在上面这个程序中，主程序030F上的RTS指令就有这个功能，它相当于BASIC中的END语句。返回以后，系统处于监控状态（提示符“*”）还是BASIC状态（提

示符“J”)取决于开始是由哪一种状态启动该程序的。

⑪ CMP: 使用在立即寻址方式下, 把立即数与累加器A中的数进行比较, 结果存入标志寄存器中, 供后面的条件判别使用。操作码是C9。

例: C9 8D CMP # \$8D

⑫ CPX: 使用在立即寻址方式下, 把立即数与寄存器X中的值进行比较, 结果存入标志寄存器中, 供以后的条件判别使用。操作码是E0。

例: E0 10 CPX # \$10

⑬ BEQ: 使用在相对寻址方式下, 当上一步的比较结果相等时, 转向指定地址执行。因此, BEQ指令及后续的BNE、等指令往往跟在比较指令CMP、CPX或CPY之后。

例:

```
0320-   C9   10       CMP   # $ 10
0322-   F0   01       BEQ   $ 0325
0324-   E8           INX
0325-   E8           INX
```

本例中地址0322中就是BEQ指令, 它表示如果累加器A中的数与16进制数10相等, 则转向0325执行; 否则, 继续执行0324中的INX指令。BEQ指令的操作码是F0, 其后的操作数(01)说明转向执行的方向(向前或向后)及相对地址。

⑭ BNE: 使用在相对寻址方式下, 当上一步的比较结果不相等时, 转向指定地址执行。因此, BNE指令与BEQ指令的条件正好相反。

例:

```
0320-   C9   10       CMP   # $ 10
```

```

0322-   D0  01      BNE    $ 0325
0324-   E8          INX
0325-   E8          INX

```

本例中地址0322上即BNE 指令,它表示如果累加器A中的数与10不相等,则转向 0325 执行。BNE 指令的操作码为D0,其后的操作数也指明执行的方向及相对地址。

⑮ BCC: 使用在相对寻址方式下,当上一步的运算没有产生进位时,则转向指定地址执行。操作码是90。

例: 90 03 BCC \$0328

在以上15个指令中,最后三个都是相对寻址指令(相当于条件转移)。当条件不成立时继续执行下一条指令,否则转向执行。转向的方向及地址的确定方法是一样的。下面,我们介绍一下通过相对地址如何确定绝对地址,即到底转向哪里执行的问题。

我们知道,相对寻址指令的操作码(即相对地址)是一字节的,其范围是00—FF。因此,可以把它分成两段来表示向前或向后转移。通常由00到7F作为向前转移的相对地址,80到FF作为向后转移的相对地址。这样,另一方面就决定了向前和向后转移的步数最多为128步。

例:

```

0300-   E0  00      CPX    # $00
0302-   D0  05      BNE    $ 0309

```

.....

```

0309-   60          RTS

```

这是一个向前转移的例子。可以看出,向前转移的步数(05)应该从相对寻址指令后的第一个单元(0305)开始计算,即转移0步对应0304,转移1步对应0305,转移2步对

应0306, 转移 3 步对应0307, 转移 4 步对应0308, 转移 5 步对应0309。

又例:

```
0300-   C9  8D           CMP    # $ 8D
0302-   F0  04           BEQ    $ 0308
0304-   E8              INX
0305-   8D  30  03       STA    $ 0330
0308-   60              RTS
```

从F0 04后的下一条指令0304开始对应转移步数为0304 (00)、0305 (01)、0306 (02)、0307 (03)、0308 (04)。因此, 向前转移 4 步即0308。

表2.1 向前转移表

操作数 (16进制)	转移步数 (10进制)	操作数 (16进制)	转移步数 (10进制)
00	0	:	:
01	1	74	116
02	2	75	117
03	3	76	118
04	4	77	119
05	5	78	120
06	6	79	121
07	7	7A	122
08	8	7B	123
09	9	7C	124
0A	10	7D	125
0B	11	7E	126
:	:	7F	127

当操作数在80到FF范围内时, 转移就向后进行。

例:

```
0300-   20  58  FC      JSR    $ FC58
```

```

0303-  E8          INX
0304-  E0  A0      CPX   # $ A0
0306-  D0  FB      BNE   $ 0303
0308-  60          RTS

```

这是一个向后转移的例子。可以看出，向后转移也是从相对寻址指令的下一条指令为 00 进行计算的，即 0308 对应 00，0307 对应 FF，0306 对应 FE，0305 对应 FD，0304 对应 FC，0303 对应 FB。因此，操作数 FB 表明向后转移到地址 0303 执行。

又例：

```

0300-  E8          INX
0301-  9D  00  60  STA   $ 6000,X
0304-  E0  0A      CPX   # $ 0A
0306-  D0  F8      BNE   $ 0300
0308-  60          RTS

```

表2.2 向后转移表

操作数 (16 进制)	转移步数 (10进制)	操作数 (16进制)	转移步数(10进制)
00	0	:	:
FF	1	8B	117
FE	2	8A	118
FD	3	89	119
FC	4	88	120
FB	5	87	121
FA	6	86	122
F9	7	85	123
F8	8	84	124
F7	9	83	125
F6	10	82	126
F5	11	81	127
:	:	80	128

从 BNE 指令的下一条指令 0308 开始对应转移地址为 0308(00)、0307(FF)、0306(FE)、0305(FD)、0304(FC)、0303(FB)、0302(FA)、0301(F9)、0300(F8)。因此，D0 F8即向后转移到0300执行。

三、6502监控系统

上一章介绍了一些简单的机器语言指令，并例举了几个程序实例。读者可能要问，这些程序是怎样输入内存又是怎样调试和执行的呢？针对这个问题，本章着重介绍6502的监控系统。我们知道，在 BASIC 状态下，可以输入、调试和运行 BASIC 语言程序，还可以通过键盘操作进行数值运算及发送命令。

同样，在机器语言状态下也可以实现这些功能。6502的监控系统象 BASIC 解释程序一样由许多功能程序组成，这些程序能实现程序输入、显示、修改、执行等操作。它们也被固化在内存的ROM区中，一开机就能使用。

1. 程序的输入和显示

(1) 监控系统的进入与退出 不管是否引导了DOS系统，只要打开主机，机器就会自动进入 BASIC 状态，其在屏幕上的提示符为“`]`”，提示符之后为光标。光标的出现意味着主机已作好准备等待操作者发送命令。在这种情况下，就可以使它进入监控状态，命令是：

`]` CALL-151↵

当然，可以逐个键入字符，也可以使用键盘保留字 CTRL-SHIFT U，即先按下 CTRL 和 SHIFT 键，再按下 U 键。这时，屏幕上同样出现“CALL-151”字串，主机进入监控状态。

监控状态的提示符是“*”，提示符后为光标。这时，就可以键入各种监控命令了。

从监控状态返回BASIC状态可以用CTRL-C或CTRL-RESET等命令。

(2) 程序的输入 以第二章中的例程序2.6为例，现在将它输入内存。

第一步：进入监控状态：

] CALL-151↵

这时，屏幕上出现“*”提示符。

第二步：打入程序存放的起始地址及冒号(：)：

* 300:

这即表明后面跟着要输入程序的具体内容了。注意，在输入一个16进制数时，数前面的“0”字符可以省略。如地址0300可以直接输300，00FF可以直接输FF，0001可以直接输1。

第三步：紧接着输入程序，即：

0300- 20 58 FC A2 00 20 35 FD↵

0308- C9 8D F0 0A 20 ED FD 9D↵

0310- 20 03 E8 4C 05 03 60↵

这里，字节和字节之间必须用一个空格分开。对于某一个字节而言，如果它的第一个字符是“0”，那么可以省略而只输入第二个字符，如05输入5。可以看出，机器语言程序的输入并不需要用地址对应一个字节或一条指令的方式，而只需要给出起始地址，监控系统就会依照次序把它存入一段连续的内存地址中。因此，整个输入过程比较简便。

第四步：输入完成后按回车键，屏幕上又出现“*”提示符，说明程序已按要求存放好。

应该指出的是，由于键盘缓冲区长度的限制，较长的机

器语言程序不可能一次全部输入。这时，可以把程序分成若干小段，每个小段分别输入。仍以上面的程序为例：

第一步：进入监控状态

] CALL-151↵

第二步：键入起始地址及冒号：

* 300:

第三步：输入第一小段程序，并回车：

* 300: 20 58 FC A2 00 20 35 FD C9 8D F0 0A↵

第四步：接着输入冒号及第二小段程序，并回车：

*: 20 ED FD 9D 20 03 E8 4C 05 03 60↵

这里的冒号没有带地址，表示紧接着上一步的程序连续存放。当然，也可以在冒号之前输入地址：

* 30C: 20 ED FD 9D 20 03 E 8 4 C 05 03 60↵

于是，该程序就被分段输入了内存。同样，对于较长的程序，还可以分成更多的小段，只要保证每小段的输入长度不超过键盘缓冲区的长度就可以了。

(3) 指令的汇编显示 程序输入内存以后，很自然地想到要显示出来检查一下，以便发现错误，这就可以使用汇编显示命令（也可以用后述的内存单元显示命令）。

这个命令的格式是：

* [地址]LL...L↵

它将从给定地址开始，连续显示出 $n \times 20$ 条指令及其汇编形式，其中 n 是命令中 L 的个数。当省略地址时，显示从当前地址开始。

当 n 等于 1 时，这个命令变为：

* [地址]L↵

这就是 L 命令的常用格式，它将显示出从指定地址开始的 20

条指令及其汇编形式。这里所说的指令汇编形式不是操作者输入机器的，而是L命令的执行程序通过解释内存中的指令显示出来的。如：

* 300L ↙

0300-	20	58	FC	JSR	\$FC58
0303-	A2	00		LDX	# \$ 00
0305-	20	35	FD	JSR	\$FD35
0308-	C9	8D		CMP	# \$ 8D
030A-	F0	0A		BEQ	\$ 0316
030C-	20	ED	FD	JSR	\$FDED
030F-	9D	20	03	STA	\$0320, X
0312-	E8			INX	
0313-	4C	05	03	JMP	\$ 0305
0316-	60			RTS	
0317-	00			BRK	
0318-	00			BRK	
0319-	00			BRK	
031A-	00			BRK	
031B-	00			BRK	
031C-	00			BRK	
031D-	00			BRK	
031E-	00			BRK	
031F-	00			BRK	
0320-	00			BRK	

2. 内存的显示和修改

(1) 内存单元内容的显示 第一节中介绍了显示内存指令及其汇编形式的L命令。但有时候，我们只需要查看一个或若干个内存单元的内容，并且不需要汇编形式。这时，

可以使用以下几种命令形式:

① * [地址]↵: 显示指定地址中的内容。如: * 300 ↵
则屏幕显示出:

0300-20

说明300单元中的数为20 (16进制)。

② * • [地址]↵: 显示从当前地址到指定地址的所有内容。如输入:

* • 305 ↵

则屏幕显示出:

0301-58 FC A2 00 20

③ * [地址] • [地址]↵: 显示从起始地址到结束地址的所有单元内容。如输入:

* 300 • 308 ↵

则屏幕显示出:

0300-20 58 FC A2 00 20 35 FD

0308-C9

④ * ↵: 显示当前地址后的单元内容, 一次最多显示8个单元。如输入:

* ↵

则屏幕显示出:

0309-8D F0 0A 20 ED FD 9D

通过以上四个命令可以显示出指定范围内的单元内容, 以便进行校对、修改和必要的处理。

(2) 内存单元内容的修改 在第一节中我们介绍过程序 (即机器语言指令代码) 输入内存的方法, 这实际上就是内存单元内容的修改。因为程序输入内存后必然要覆盖原来的内容, 而代之以新的程序指令。一般地, 修改内存单元内

容可使用以下几种方式的命令:

① * [地址]: [数值]↵: 把给定的16进制数值写入指定的地址单元。这里的数值必须在00—FF范围内。如:

* 03E0: AC↵

检查一下03E0的内容:

* 03E0↵

03E0-AC

说明确实改过了。

② * [地址]: [数值][数值][数值]...[数值]↵: 把给定的若干个16进制数值顺序改入从指定地址开始的连续单元中, 数和数之间必须用空格分开。如:

* 3A0: C1 C2 C3 C4 C5↵

检查一下:

* 3A0: 3A4↵

3A0-C1 C2 C3 C4 C5

说明已经修改了。

③ *: [数值][数值]...[数值]↵: 把给定的若干个16进制数值写入从当前地址开始的连续单元中。当前地址可借助于显示来确定。如:

* 3A0↵ 显示地址3A0中的数

3A0-C1 屏幕显示出3A0中的数是C1

*: D1 D2↵ 把3A0中的数改为D1, 3A1中的数改为

D2

检查一下:

* 3A0: 3A1↵

3A0-D1 D2

3. 算术运算

在BASIC状态下，可以使用?或PRINT命令进行10进制数值的运算。同样，在监控状态下，也可以进行16进制数的立即加减运算。

(1) 16进制的加法 要使两个16进制数相加只要在*提示符下输入这两个数，并用加号连接起来，按回车键就可以了。这时，屏幕上显示等于号及相加后的16进制和值。如：

* 30 + 2C ↵

= 5C

又如：

* F0 + 0A ↵

= FA

* F0 + 5D ↵

= 4D

请注意最后一个例子，F0 (240) 加上5D (93) 的正确结果应该是14D (333)，而显示的结果却丢掉了高位1，变成4D。这是因为6502是8位微处理器，而14D (2进制码 101001101) 有9位，因此结果中丢掉了最高一位。从这个例子中可以看到，立即式的加法只能在和值小于等于FF时进行。

那么，如果和值大于FF时，又如何实现两个数的加法运算呢？事实上，只要记住这个被丢掉的进位数元，最后置于显示结果之前就可以了。

如上例中F0加上5D显示结果为4D，加上进位数元，正确的答案就是14D。又如：

$$* DA + 71 \checkmark$$

$$= 4B$$

加上进位数元结果应是14B。又如：

$$* 5F + EA \checkmark$$

$$= 49$$

加上进位数元结果应是149。

同样，还可以对占用二个字节的数进行相加。以 13F0 与 5AA 相加为例，步骤如下：

第一步：将低位和低位先行相加，记住是否有进位：

$$* F0 + AA \checkmark$$

$$= 9A \quad \text{有进位}$$

第二步：把高位和高位相加：

$$* 13 + 5 \checkmark$$

$$= 18$$

第三步：把高位相加后的和再加上进位：

$$* 18 + 1 \checkmark$$

$$= 19$$

第四步：高位和在前，低位和在后就是最后的结果，即 13F0 加上 5AA 等于 199A。

再看一个例子：54C + 3F0

$$* 4C + F0 \checkmark \quad \text{低位相加}$$

$$= 3C \quad \text{有进位}$$

$$* 5 + 3 \checkmark \quad \text{高位相加}$$

$$= 08 \quad \text{无进位}$$

$$* 8 + 1 \checkmark \quad \text{高位和加进位}$$

$$= 09$$

所以，54C + 3F0 = 93C。这种方法还适用于二字节以上的

数相加:

A93B4F7加上F83D:

* F7 + 3D ↙	最低字节相加
= 34	有进位C1, 得和的最低字节
* B4 + F8 ↙	次低字节相加
= AC	有进位C2
* AC + 1 ↙	加上进位C1
= AD	无进位, 得和的次低字节
* 93 + 00 ↙	高字节相加
= 93	无进位
* 93 + 1 ↙	加上进位C2, 得和的高字节
= 94	
* A + 0 ↙	最高字节相加
= A	得和的最高字节

因此, $A93B4F7 + F83D = A94AD34$ 。

应该说明的是, 立即式的加法每次只能二个数相加。如果把二个以上的数用加号连接起来, 其结果将是最前一个数和最后一个数相加的和, 中间部分被忽略。如:

* 1 + 2 + 3 + 4 ↙
= 05

实际上, 只计算了 $1 + 4 = 5$ 。因此, 这种情况应分步进行:

* 1 + 2 ↙
= 03
* 3 + 3 ↙
= 06
* 6 + 4 ↙

= 0A

(2) 16进制的减法 立即式的减法与加法类似, 即输入二个数, 中间用减号连接。如:

* 5A - 30 ↵

= 2A

* AD - 11 ↵

= 9C

按下回车键后, 屏幕上即显示出等于号和差值。不过, 当被减数小于减数时, 显示结果又会出现类似加法的情况。如:

* 0 - 3 ↵

= FD

0减去3本来应该得到负3, 但负数在计算中无法直接表示。因此, 规定81到FF之间的数为负数(00到80之间的数为正数)。其对应关系如表3.1。

表3.1 负数对应表

16进制数	对应负值	16进制数	对应负值
FF	-1	:	:
FE	-2	87	-79
FD	-3	86	-7A
FC	-4	85	-7B
FB	-5	84	-7C
FA	-6	83	-7D
F9	-7	82	-7E
:	:	81	-7F

回过头去看看第一章中的表1.4, 可以发现, 表1.4和表2.1实质上是一样的。因此, 可以据此把向后转移理解为“负转移”。

那么，为什么 0 减去 3 会得到差值 FD 呢？请看它的相减过程：

2 进制	16 进制	10 进制
00000000	00	0
- 00000011	- 03	- 3
<u>有借位</u> 11111101	<u>有借位</u> FD	- 3

这说明差值 FD 是在有借位的情况下得到的。和 16 进制的加法相类似，我们可以对二字节或二字节以上的数做减法。以 3D4 减去 1F2 为例：

第一步：将低位先相减，记住是否有借位：

$$* D4 - F2 \swarrow$$

$$= E2 \quad \text{有借位}$$

第二步：将高位相减：

$$* 3 - 1 \swarrow$$

$$= 02 \quad \text{无借位}$$

第三步：将高位相减后的差再减去借位：

$$* 2 - 1 \swarrow$$

$$= 01$$

第四步：高位差在前，低位差在后就是最后相减的结果，即 3D4 减去 1F2 等于 1E2。

又如 A37 减去 8B4：

$$* 37 - B4 \swarrow \quad \text{低位相减}$$

$$= 83 \quad \text{有借位}$$

$$* A - 8 \swarrow \quad \text{高位相减}$$

$$= 2 \quad \text{无借位}$$

$$* 2 - 1 \swarrow \quad \text{高位差减去借位}$$

$$= 1$$

所以, $A37 - 8B4 = 183$ 。

4. 标志寄存器

上一节中通过人为的办法来记忆和处理加法产生的进位和减法产生的借法。显然, 这种办法不能使用在机器语言程序中, 因此有必要设置一个专用单元, 来自动记录产生进位、借位等的情况。这个专用单元就是6502的标志寄存器P, 它是一个8位寄存器, 每一位记录一种当前状态:

第0位: 代位标志C (Carry), 记录加减法运算时的进位、借位状态。当加法运算有进位或减法运算无借位时, C等于1; 当加法运算无进位或减法运算有借位时, C等于0;

第1位: 零值标志Z (Zero), 记录运算结果是否为0的状态。当结果为0时, Z等于1; 当结果不为0时, Z等于0;

第2位: 中断标志I (Interrupt), 指明当前是否允许中断。当I为0时, 说明允许中断; 当I为1时, 说明禁止中断;

第3位: 10进位标志D (Decimal), 指明运算的数制方式。当D为1时, 说明采用10进制方式; 当D为0时, 说明采用16进制方式;

第4位: 中断状态标志B (Break), 记录当前的中断状态。当B为1时, 说明中断; 当B为0时, 说明没有中断;

第5位: 扩展标志位, 目前还未使用;

第6位: 溢出标志V (oVerflow), 记录溢出状态。当V为1时, 说明有溢出; 当V为0时, 说明没有溢出;

第7位: 负数标志N (Negative), 指明一个数是否为

负数。当N为1时，为负数；当N为0时，为正数或零。

6502标志寄存器P的示意图如图3.1

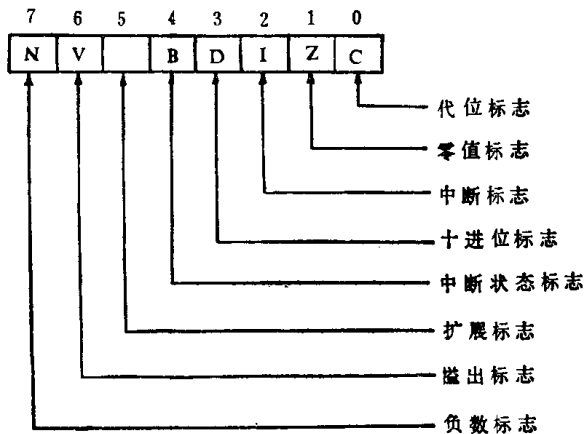


图3.1 标志寄存器

在以上八个标志中，C、Z和N用得较广，条件转移指令就是把这些标志的当前状态作为判别条件的。D标志用来设置数运算的方式：在需要10进制方式时，要先将D设置为1；否则，将D设置为0。

结合第一章的内容，已经介绍了A、X、Y和P四个寄存器，剩下一个堆栈指示器S和一个指令计数器PC。

堆栈指示器S是8位寄存器，它是专门用来指示堆栈指针位置的。在主机的记忆空间中，0100—01FF为堆栈区，用以存放系统运行过程中的一些特殊参量，这些参量从01FF开始逐个往前存放，并且具有“先入后出”的特点。也就是说，当从堆栈区中取数时，最后存入的数总是最先取出，而最先存入的数总是最后取出。每取出一个数，堆栈指示器往后移一个位置。

如图3.2所示，堆栈区中存有四个字节的数（A0、3D、0A、8D），栈指针指向8D，堆栈指示器指向01FC。这时，

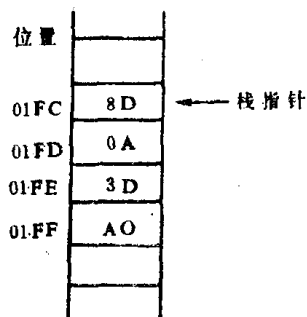


图3.2

再使数FC入栈，则有（见图3.3）：

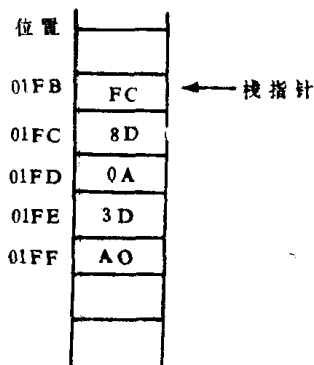


图3.3

（注：图3.2,3.3中的A0应为A0）

堆栈指示器指向01FB。

反之，如果从堆栈区中取数，就应该先取出 01FB 中的数FC（栈指针指向01FB），再取出01FC中的数8D（栈指针指向01FC），依次类推。

现在已经知道，堆栈指示器就是一个始终存有栈顶地址（如上述的01FB、01FC、01FD）的寄存器。但它只有 8 位（即一字节），而内存地址是二字节的，如何存放呢？

从堆栈区的位置看，它处于内存空间的第 1 页（0100—01FF），在二个字节表示的地址中，高位总是01。因此，可

以只考虑其低位，而默认高位为01，这样，就可以用8位的堆栈指示器来存放栈指针了。

在以后的章节中，将会介绍如何用指令使数据进栈和使栈中的数据出栈。

指令计数器PC是6502唯一的16位寄存器，它在程序执行过程中始终存放有下一条将要执行的指令的起始地址。程序在执行完一条指令后，便根据这一地址取出指令执行，同时，指令计数器中又自动存入下一条指令的存放始址。因此，指令计数器是一个存放指令地址的寄存器。内存地址从0000到FFFF是16位的(二字节)，所以指令计数器也必须是16的。

在已经介绍过的六个寄存器中，A、X、Y、P和S五个寄存器可以在监控状态下直接修改它的内容。

第一步：在监控状态下按入 CTRL-E 及回车，屏幕上显示出各寄存器的当前值。如：

* ↵ (CTRL-E不在屏幕上显示)

A = 00 X = 10 Y = 20 P = 0 S = FF

*

第二步：输入冒号及要修改的值。如：

*: A0 BC 30 28 10 ↵

这样，新的内容新会按寄存器排列顺序逐个替换原来的值，使：

A = A0, X = BC, Y = 30, P = 28, S = 10

再次显示寄存器的内容，可以看到修改已经完成了：

* ↵ (CTRL-E)

A = A0 X = B0 Y = 30 P = 28 S = 10

*

又如，把A改为10，X改为20，Y改为30，P改为40，S改为50，操作如下：

```
* ↵ (CTRL-E)
A = A0 X = B0 Y = 30 P = 28 S = 10
*: 10 20 30 40 50 ↵
*
```

5. 其它监控命令

(1) 内存的搬移 内存搬移就是把内存中一段连续区域的内容原封不动地移到另一段连续区域中。因此，必须先知道起始地址、结束地址和目的地址。命令的格式是：

* 目的地址<起始地址·结束地址M↵ 这里，字符“<”、“·”和“M”是命令中规定的，M意即Move。在内存搬移时，两段内存区必须没有重叠，否则会达不到搬移的目的。假设0300—030A中有00—0A共11个数：

```
0300- 00 01 02 03 04 05 06 07
0308- 08 09 0A
```

要把它移至0320开始的内存单元中，只要键入命令：

```
* 0320<0300·030AM↵
```

就可以了。这时屏幕上没有任何反映，说明已经完成了搬移。显示一下0320—032A的内容，可以看到搬移的结果：

```
* 0320·032A↵
0320- 00 01 02 03 04 05 06 07
0328- 08 09 0A
*
```

但0300—030A中的数并没有变，这个搬移实际上是复制。

(2) 内存的比较 内存的比较是指比较两个给定的内存区内容是否相同或哪些单元上不相同。它可以用来验证内存搬移是否成功。

命令的格式是:

* 目的地址<起始地址·结束地址V↵

V意即Verify。当比较结果相同时,屏幕上不显示任何信息;如果比较不相同,则显示出原来的地址和不同的数值。如比较上述经搬移后的两个区域:

* 0320<0300·030AV↵

*

屏幕上没有显示任何信息,说明两段区域内的内容相同,搬移成功。如把单元0305的内容改为FF,再比较结果就不同了:

* 0305: FF↵

* 0320<0300·030AV↵

0305-05(FF)

*

(3) 程序的执行 内存中的机器语言程序或子程序是用G命令来启动执行的。命令的格式是:

* 起始地址G↵

G意即GO。这个命令将把给出的地址直接赋给指令计数器,从而指向程序开始的第一个指令。如:

* 300G↵

(4) 设备的启动 在BASIC状态下有PR#N的命令可以用来启动DOS,或者接通、关闭打印机等。在监控状态下这个命令就是:

N CTRL-P

启动DOS: * 6 ↵ (6 CTRL-P)

接通打印机: * 1 ↵ (1 CTRL-P)

关闭打印机: * 0 ↵ (0 CTRL-P)

上面的数字 6、1、0 键入后要按控制字符CTRL-P。

(5) 设置显示状态 在屏幕上显示信息可以有正常、闪烁、反相三种状态, 监控状态下也可以互相转换。

设置正常显示, 即NORMAL;

* N ↵

设置闪烁显示, 即FLASH;

* F ↵

设置反相显示, 即INVERSE;

* I ↵

本章中我们介绍了6502的监控系统命令, 它可以用来进行程序和数据的输入、显示、修改、16进制数的简单加减法以及其它一些工作。灵活运用这些命令, 可以简化处理流程, 减少工作量。因此, 希望读者要进一步掌握使用。

四、数值运算

在任何一种程序设计语言中，都具备一些数值运算的功能。其中最简单的运算就是加法和减法。在 6502 机器语言中就有若干个指令，能实现这种功能。通过最基本的加减运算，还可以用程序设计出乘除等运算。

一般地说，在机器语言中参加运算的数都是16进制数，但为了增强运算的功能，适应用户的多种要求，6502机器语言还允许对10进制数进行运算，这只要通过设置标志寄存器中的10进位标志D就可以了。

本章中将先通过分析和程序实例介绍一字节、二字节及多字节的16进制数加法、减法、乘法、除法，再介绍10进制数的运算。

1. 一字节的加法

在上一章的监控系统中已经介绍过直接用键盘命令进行单字节加减运算的方法，现在，就介绍用指令和程序来实现这一功能的方法。

设有二个数3 A和22，要求出它们相加的和可以通过两个步骤来完成：

首先，将其中一个数存入累加器A，准备和第二个数相加，

其次，用加指令使累加器中的数和第二个数相加。结果仍保留在累加器A中，即使用指令LDA和ADC。

例程序4.1

```
0300-    20  58  FC      JSR    $FC58
0303-    A9  3A          LDA    # $3A
0305-    18              CLC
0306-    69  22          ADC    # $22
0308-    20  DA  FD      JSR    $FD5A
030B-    60              RTS
```

程序说明,

0300: 清洗屏幕;

0303: 将被加数3A存入累加器A;

0305: 清除进位标志;

0306: 将累加器中的数与22相加;

0308: 显示相加结果;

030B: 返回;

下面在监控状态下输入并执行这个程序:

] CALL-151↵

* 300: 20 58 FC A9 3A 18 69 22 20 DA FD 60↵

* 300G↵

5C

*

显示相加结果5C, 是正确的。

在这个程序中, 使用了二个新的指令:

(1) 18 CLC: 这是一个清除标志寄存器中进位标志的指令 (置标志寄存器中的进位标志 C 为 0)。由于 ADC 指令不但会把指定的两个数进行相加, 而且还要考虑进位标志 (即还要加上进位标志的值), 因此, 在进行加法之前必须先清除进位标志 C, 以免影响结果的正确性。

(2) 69 22 ADC#\$22: 这是一个立即寻址方式下的

加法指令，它把累加器 A 中的数与操作码 69 之后的立即数相加，再加上进位标志，最后结果仍存在累加器 A 中。

在地址 0308 中调用了 FDDA 系统子程序。这个子程序的功能是把累加器 A 中的数以 16 进制的形式显示出来。

现在把地址 0307 中的数 22 改为 FA，再执行一遍，看看结果又是怎么样的：

* 307:FA↙

* 300G↙

34

*

显然，这个结果是错误的，3A 加上 FA 不会等于 34。究其原因，和在监控系统下的直接加法一样，它的正确结果 134 已经超出了八位，累加器 A 无法存贮，以致丢掉了最高位的 1（存入进位标志 C），成为 34。因此，必须改进程序 4.1，使之考虑相加以后的进位标志。

例程序 4.2

0300-	20	58	FC	JSR	\$FC58
0303-	A9	3A		LDA	#\$3A
0305-	18			CLC	
0306-	69	FA		ADC	#\$FA
0308-	8D	20	03	STA	\$0320
030B-	A9	00		LDA	#\$00
030D-	69	00		ADC	#\$00
030F-	20	DA	FD	JSR	\$FDDA
0312-	AD	20	03	LDA	\$0320
0315-	20	DA	FD	JSR	\$FDDA
0318-	60			RTS	

程序说明：

0300: 清洗屏幕;
 0303: 把数3A送入累加器 A;
 0305: 清除进位标志 C;
 0306: 使累加器 A 中的数 (3A) 与 FA 相加;
 0308: 和值存入 0320 单元中;
 030B: 把数 00 送入累加器 A;
 030D: 使累加器 A 中的数 (00) 与 00 相加;
 030F: 显示相加后的和值;
 0312: 取出 0320 单元中的数, 送入累加器 A;
 0315: 显示 A 中的数 (即和的低位)。

输入并执行这个程序:

* 300: 20 58 FC A9 3A 18 69 FA 8D 20 03 A9

00 69 00 20 DA FD AD 20 03 20 DA FD 60 ✓

* 300 G ✓

0134

*

结果是正确的。这里使用了 $00 + 00$ 的技巧。首先, 清除进位标志, 使 3A 和 FA 相加, 结果进位标志中存入最高位 1, 累加器中存有 34。再用 00 加上 00, 使进位标志中的数存入累加器 A。按照从高位到低位的顺序显示出来就是 0134。在 00 加上 00 之前, 没有用 CLC 指令清除进位标志 C。因为这时候, 进位标志中的数也是上一步相加结果的一部分, 是有效的, 不能清除。实际上这一步加法即: $00 + 00 + C = C$, 使 C 标志存入累加器 A。

程序 4.2 适用于任何一字节数相加的运算。如 FA 加上 D5:

* 304: FA ✓ (被加数改为 FA)

* 307: D5 ✓ (加数改为 D5)

* 300G↙

01CF

*

结果为01CF。

2. 二字节的加法

上面我们通程序实例介绍了单字节16进制数的加法，下面继续介绍二字节16进制数的加法。和监控状态下的直接相加类似，二字节数的相加可以先让低位相加，再让高位相加，最后把高低位组合在一起。

以3A20加上507F为例，我们把第一个数的低位20存入0320，高位3A存入0321，第二个数的低位7F存入0322，高位50存入0323，再进行相加。

例程序4.3

0300-	20	58	FC	JSR	\$FC58
0303-	AD	20	03	LDA	\$0320
0306-	18			CLC	
0307-	6D	22	03	ADC	\$0322
030A-	8D	24	03	STA	\$0324
030D-	AD	21	03	LDA	\$0321
0310-	6D	23	03	ADC	\$0323
0313-	8D	25	03	STA	\$0325
0316-	20	DA	FD	JSR	\$FDDA
0319-	AD	24	03	LDA	\$0324
031C-	20	DA	FD	JSR	\$FDDA
031F-	60			RTS	

程序说明，

0300，清洗屏幕；

0303: 取第一个数的低位到 A;
0306: 清除代位标志 C;
0307: 加上第二个数的低位;
030A: 和值存入0324单元;
030D: 取第一个数的高位到 A;
0310: 加上第二个数的高位;
0313: 和值存入0325单元;
0316: 显示结果的高位;
0319: 取结果的低位到 A;
031C: 显示结果的低位。

把这一程序输入内存, 再把0320—0323的数据按前述的顺序输入:

0320: 20 3A 7F 50 ✓

执行这个程序:

* 300G ✓

8A9F

*

显示结果 是8A9F。在这个程序中, 又使用了ADC指令, 但它的操作码是6D。这是一个绝对寻址指令, 它把累加器 A 中的数与操作码6D后的二字节地址所指单元中的数相加, 再加上进位标志, 结果仍存放在累加器 A 中。

程序框图如图4.1。

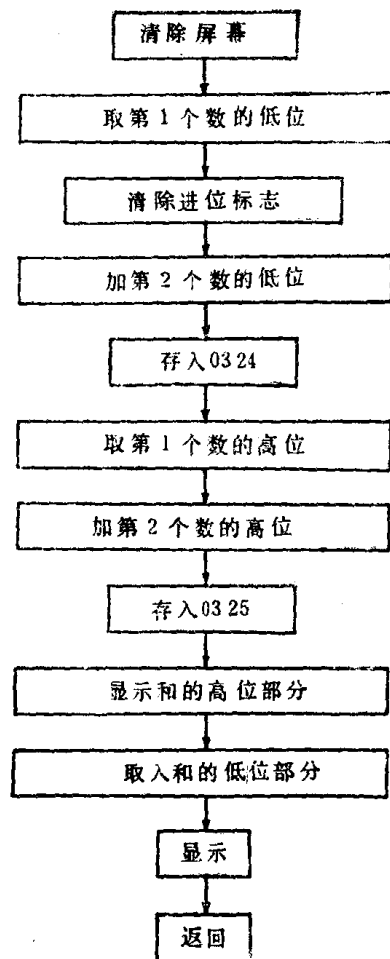


图4.1 程序框图

3. 多字节的加法

多字节相加的方法和二字节相加类似。首先把要相加的二个16进制数从右到左两两一组分成若干个字节，再从低位到高位把对应字节相加起来存入内存单元，最后把各个结果按顺序排列起来就是相加后的和。

下面来分析一个多字节加法运算的通用程序。

例程序4.4

0300-	A2	00		LDX	# \$ 00
0302-	18			CLC	
0303-	BD	30	03	LDA	\$ 0330, X
0306-	7D	40	03	ADC	\$ 0340, X
0309-	9D	40	03	STA	\$ 0340, X
030C-	08			PHP	
030D-	E8			INX	
030E-	EC	2F	03	CPX	\$ 032F
0311-	F0	04		BEQ	\$ 0317
0313-	28			PLP	
0314-	4C	03	03	JMP	\$ 0303
0317-	28			PLP	
0318-	A9	00		LDA	# \$ 00
031A-	69	00		ADC	# \$ 00
031C-	20	DA	FD	JSR	\$ FDDA
031F-	AE	2F	03	LDX	\$ 032F
0322-	CA			DEX	
0323-	BD	40	03	LDA	\$ 0340, X
0326-	20	DA	FD	JSR	\$ FDDA
0329-	EO	00		CPX	# \$ 00
032B-	DO	F5		BNE	\$ 0322
032D-	60			RTS	

程序说明:

0300: X取初值00;

0302: 清除代位标志C;

0303: 取(0330+X)内的数到A;

0306: 加上(0340+X)内的数;

0309: 和值存入 (0340 + X) 单元内;
 030C: 寄存器P进栈;
 030D: 寄存器X加 1;
 030E: X等于032F单元中的值吗?
 0311: 相等, 则转0317执行;
 0313: 否则, P出栈;
 0314: 转0303执行;
 0317: P出栈;
 0318: 取00到累加器A;
 031A: 加上00;
 031C: 显示相加后的值;
 031F: 取032F中的数到X;
 0322: X减去 1;
 0323: 取 (0340 + X) 内的数到A;
 0326: 显示A中的数;
 0329: X等于00吗?
 032B: 不等, 转0322继续显示。

其中用到 5 个新的指令:

(1) 7D 40 03 ADC \$ 0340, X: 这是一个寄存器X的绝对寻址指令, 操作码是7D。它把操作码后的二字节地址加上寄存器X中的数形成一个新的地址, 再把累加器A中的数加上这个新地址中的数, 再加上进位标志, 结果存放在累加器A中。

如7D 40 03, 当X为 2 时, 即把累加器 A 中的数加上 0342 (0340 + X) 单元的数再加进位标志。

(2) EC 2F 03 CPX \$ 032F: 这是一个绝对寻址指令, 它把寄存器X中的数与操作码EC后二字节地址中的数作比较, 比较结果 (相等或不相等) 供下一步条件转移时用。

(3) AE 2F 03 LDX \$032F; 这也是一个绝对寻址指令, 它把操作码AE后二字节地址所指的内存单元中的数存入寄存器X。

(4) 08 PHP: 隐含寻址指令, 它把标志寄存器P推入堆栈区, 使各个标志得到保护。

(5) 28 PLP: 隐含寻址指令, 使堆栈顶部的一个数恢复到标志寄存器P中。

后两个指令一般总是在程序中成对出现。在本例中, 使用PHP指令主要是为了保存上一步相加后的C标志, 以免被比较指令破坏, 影响运算结果。比较以后用PLP恢复原来的C标志。

现在来看看这个程序的框图(见图4.2)。

从框图中可以看到, 这个程序主要有两个部分组成。从0300到0319为运算部分, 031A到032C为结果显示部分。

(1) 运算部分: 先清除屏幕, 使寄存器X赋0值, 再清除进位标志, 进入循环程序段。

在循环开始时, 寄存器X为0, 因此地址0307中的BD 30 03指令将0330中的数取入A, 再加上0340中的数, 结果仍存入0340, 即:

$X = 0: (0330) + (0340) \rightarrow (0340);$

$X = 1: (0331) + (0341) \rightarrow (0341);$

$X = 2: (0332) + (0342) \rightarrow (0342);$

.....

$X = N: (0330 + N) + (0340 + N) \rightarrow (0340 + N)$

每循环一次, X增加1, 这样就可以通过X的改变对下一个字节进行处理。当X的值与032F中的数相等时退出循环, 说明各对应字节已经相加完毕。在地址0316中使用了一个00

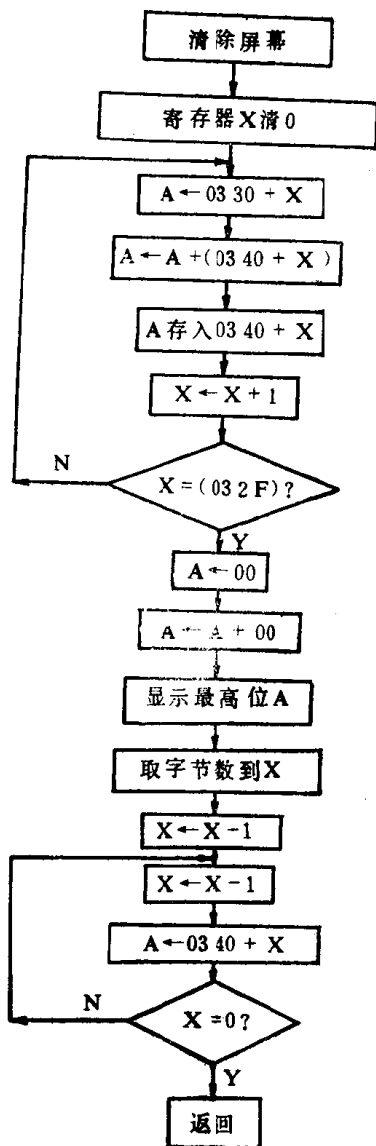


图4.2 程序框图

+00的技巧，以免丢失最高一个字节相加后可能产生的进位标志。

(2) 显示部分：显示部分首先显示出最高位相加后的进位标志，即00加上00的结果（这个结果只能是00或01），再逐个显示相加结果的各个字节。

X = N：显示 (0340 + N)；

.....

X = 2：显示 (0342)；

X = 1：显示 (0341)；

X = 0：显示 (0340)。

最后返回监控状态。

通过以上分析知道，使用这个程序必须先把二个原始数据的字节数存入单元 032F，再从低位到高位把第 1 个数的各个字节放入0330开始的地址中，最后把第 2 个数的各个字节按从低位到高位顺序存入0340开始的地址中。

以33A89加上FA034为例，当程序输入0300开始的内存后，按下列步骤操作：

第一步：确定原始数据的字节数 3（即03、3A、89或0F、A0、34三个字节），放入地址032F：

* 032F：03 ✓

*

第二步：从0330开始输入第 1 个数据（低位在前，高位在后）：

* 0330：89 3A 03 ✓

*

第三步：从0340开始输入第 2 个数据：

* 0340：34 A0 0F ✓

*

第四步：执行程序

* 300G

0012DABD

*

即相加和为0012DABD。

又如A93B4F7加上F83D:

* 032F: 04↙ (取较大数的字节数)

* 0330: F7 B4 93 0A↙

* 0340: 3D F8 00 00↙ (不足四个字节时以00补)

* 300G↙

000A94AD34

*

即结果为A94AD34。

4. 二字节的减法

减法和加法的运算流程大致相似,主要有两个指令不同。一是加法中的清除进位标志指令CLC要改成设置代位标志指令SEC;二是加法指令ADC要改成减法指令SBC。因此,知道了如何实现加法运算,减法运算就很简单了。

实际上,只要对二字节加法运算程序4.3稍作修改就完全适用于减法,即把程序4.3中地址0306的18改为38,把0307和0310的6D改为ED;

0300-	20	58	FC	JSR	\$FC58
0303-	AD	20	03	LDA	\$0320
0306-	38			SEC	
0307-	ED	22	03	SBC	\$0322
030A-	8D	24	03	STA	\$0324
030D-	AD	21	03	LDA	\$0321

0310-	ED	23	03	SBC	\$ 0323
0313-	8D	25	03	STA	\$ 0325
0316-	20	DA	FD	JSR	\$ FDDA
0319-	AD	24	03	LDA	\$ 0324
031C-	20	DA	FD	JSR	\$ FDDA
031F-	60			RTS	

程序说明:

0300: 清洗屏幕;

0303: 取第一个数的低位到 A;

0306: 设置代位标志;

0307: 减去第二个数的低位;

030A: 结果存入单元0324;

030D: 取第一个数的高位到 A;

0310: 减去第二个数的高位;

0313: 结果存入单元0325;

0316: 显示差的高位部分;

0319: 取出差的低位部分;

031C: 显示差的低位部分;

031F: 返回。

设有二个数23A0和1280相减, 把上述程序输入内存后,
再按同样的方法输入这两个数并执行:

* 0320: A0 23 80 12↵

* 0300G↵

1120

*

得相减后的差值为1120。

又如A3F4减去30A1, 操作如下:

* 0320: F4 A3 A1 30↵

* 0300G↵

新的指令:

(1) 38 SEC: 这是一个隐含寻址指令, 具有设置代位标志的功能, 即把标志寄存器中的进位标志C设置为1。

(2) ED 22 03 SBC \$0322: 这是一个绝对寻址指令, 它使累加器A中的数减去操作码ED后二字节地址所指的内存单元中的数, 相减后的结果仍存放在累加器A中。

5. 多字节的减法

多字节的减法也必须按从低位到高位顺序逐个使对应

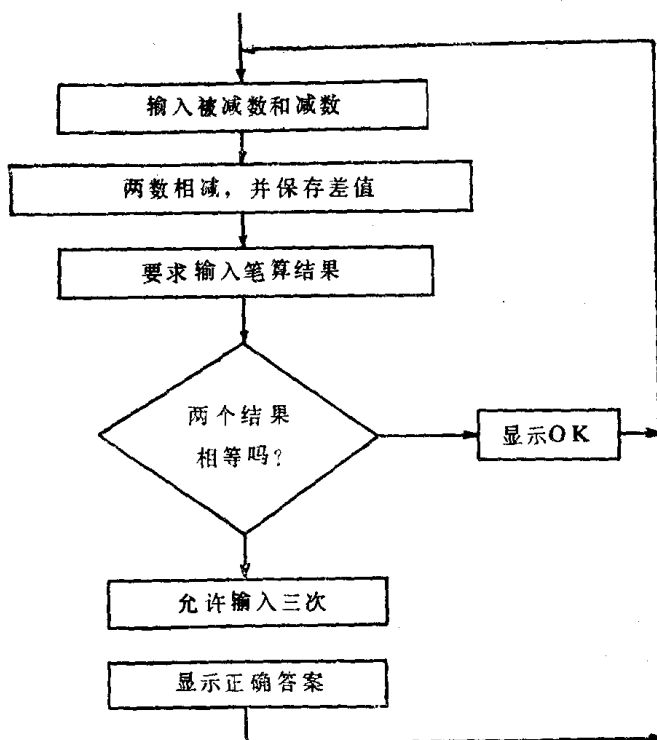


图4.3 执行流程图

字节相减，最后把每个差值排列起来。

下面举一个程序实例来说明它的用法。程序执行的流程如图4.3。

例程序4.5

第一段：输入被减数和减数

0300-	20	58	FC	JSR	\$FC58
0303-	A9	B1		LDA	#\$B1
0305-	20	ED	FD	JSR	\$FDED
0308-	A9	BA		LDA	#\$BA
030A-	20	ED	FD	JSR	\$FDED
030D-	A9	F0		LDA	#\$F0
030F-	85	E0		STA	\$E0
0311-	A9	03		LDA	#\$03
0313-	85	E1		STA	\$E1
0315-	20	93	03	JSR	\$0393
0318-	A9	F8		LDA	#\$F8
031A-	85	E0		STA	\$E0
031C-	A9	03		LDA	#\$03
031E-	85	E1		STA	\$E1
0320-	A9	B2		LDA	#\$B2
0322-	20	ED	FD	JSR	\$FDED
0325-	A9	BA		LDA	#\$BA
0327-	20	ED	FD	JSR	\$FDED
032A-	20	93	03	JSR	\$0393

程序说明：

0300：清洗屏幕；

0303—0307：显示字符“1”；

0308—030C：显示字符“：”；

030D—0314：设定被减数存放间接地址03F0；

0315: 输入被减数并保存;
 0318—031F: 设定减数存放的间接地址03F8;
 0320—0324: 显示字符“2”;
 0325—0329: 显示字符“:”;
 032A: 输入减数并保存。

这一段先清洗屏幕, 显示“1:”, 再把被减数的存放起始地址03F0存入零页单元E0、E1(低、高位), 转向0393请求输入第一个数; 接着显示“2:”, 把减数的存放起始地址03F8存入零页单元E0、E1中, 转向0393请求输入第二个数。

这里, E0、E1中的数据存放地址将在子程序0393中作定址用。

用到的新指令有:

85 E0 STA \$E0: 这是一个零页寻址指令, 它把累加器A中的数存入操作码85后的零页地址中。这个指令相当于8D E0 00 STA \$00E0。由于地址00E0在内存区的第零页, 其高位为00, 因此略去不写, 而把操作码改为85, 特指操作地址在第零页, 以节省内存(三字节变为二字节), 加快执行速度。

第二段: 两数据相减

032D-	84	E2	STY	\$E2
032F-	A0	00	LDY	# \$ 00
0331-	38		SEC	
0332-	B9	F0 03	LDA	\$ 03F0,Y
0335-	F9	F8 03	SBC	\$ 03F8,Y
0338-	99	F8 03	STA	\$ 03F8,Y
033B-	C4	E2	CPY	\$E2
033D-	EA		NOP	

033E-	F0 2E	BEQ	\$ 036E
0340-	C8	INY	
0341-	4C 32 03	JMP	\$ 0332

程序说明:

0332—0337: 使被减数减去减数的对应字节;

0338: 差值存入减数存放位置;

033B: 各字节都相减了吗?

033E: 是, 转036E;

0340: 否则, Y加上1;

0341: 转0332, 对下一字节相减。

这一段程序把两个数对应的字节从低位到高位 逐个 相减, 结果存入03F8开始的地址单元中。

用到的新指令有:

(1) 84 E2 STY \$E2: 零页寻址指令, 把寄存器 Y 中的数存入零页单元00E2中。

(2) A0 00 LDY #00: 寄存器Y立即赋值指令, 把立即数00送入寄存器Y中。

(3) B9 F0 03 LDA \$03F0, Y: 寄存器Y的绝对寻址指令, 使操作码B9后的二字节地址加上寄存器Y中的数形成新的地址, 把这一新地址单元中的数送入累加器A。

(4) F9 F8 03: SBC \$03 F8, Y: 寄存器Y的绝对寻址指令, 使操作码F9后的二字节地址加上寄存器Y中的数形成新的地址, 用累加器A中的数减去新地址单元中的数, 结果仍保留在A中。

(5) 99 F8 03 STA \$03F8, Y: 寄存器y的绝对寻址指令, 使操作码99后的二字节地址加上寄存器Y中的数形成新的地址, 把累加器A中的数存入新的地址单元中。

(6) C4 E2 CPY \$E2: 零页寻址指令, 把寄存器Y

中的数与零页单元 00E2 中的数作比较，比较结果供后面的条件判断使用。

(7) C8 INY: 隐含寻址指令，使寄存器 Y 中的数加上 1，即 $Y \leftarrow Y + 1$ 。

第三段：输入笔算结果，比较正确性

0344-	A9	BF		LDA	#\$BF
0346-	20	ED	FD	JSR	\$FDED
0349-	A9	F0		LDA	#\$F0
034B-	85	E0		STA	\$E0
034D-	A9	03		LDA	#\$03
034F-	85	E1		STA	\$E1
0351-	20	93	03	JSR	\$0393
0354-	A2	00		LDX	#\$00
0356-	BD	F0	03	LDA	\$03F0,X
0359-	DD	F8	03	CMP	\$03F8,X
035C-	F0	0A		BEQ	\$0368
035E-	C8			INY	
035F-	C0	03		CPY	#\$03
0361-	D0	E1		BNE	\$0344
0363-	68			PLA	
0364-	68			PLA	
0365-	4C	83	03	JMP	\$0383
0368-	E8			INX	
0369-	E4	E2		CPX	\$E2
036B-	D0	E9		BNE	\$0356
036D-	60			RTS	
036E-	A0	00		LDY	#\$00
0370-	20	44	03	JSR	\$0344
0373-	A9	CF		LDA	#\$CF

0375-	20	ED	FD	JSR	\$FDED
0378-	A9	CB		LDA	# \$CB
037A-	20	ED	FD	JSR	\$FDED
037D-	20	8E	FD	JSR	\$FD8E
0380-	4C	03	03	JMP	\$0303
0383-	A0	00		LDY	# \$00
0385-	B9	F8	03	LDA	\$03F8,Y
0388-	20	DA	FD	JSR	\$FD8A
038B-	C8			INY	
038C-	C4	E2		CPY	\$E2
038E-	D0	F5		BNE	\$0385
0390-	4C	7D	03	JMP	\$037D

用到的 4 个新指令是：

(1) DD F8 03 CMP \$03F8,Y: 寄存器 Y 的绝对寻址指令, 它使操作码 DD 后的二字节地址与寄存器 Y 中的数相加形成一个新的地址, 把累加器 A 中的数与新地址单元中的数比较, 比较结果供下一步条件判断时使用。

(2) C0 03 CPY # \$03: 立即寻址指令, 将寄存器 Y 与操作码 C0 后的立即数 03 进行比较, 比较结果供下一步条件判断时用。

(3) 68 PLA: 隐含寻址指令, 它把堆栈最顶部的一个数送入累加器 A, 同时栈指针往栈底移动一个单元, 也就是使栈顶的一个数出栈。在上面的程序段中, 连续使用了两个 PLA 指令, 目的是强迫退出子程序 (类似于 BASIC 中的 POP 语句)。

从 0344 开始到 036D 是一段子程序, 0370 上的 JSR 指令调用了这个子程序。一般地说, 调用子程序以后运行无条件转向子程序, 子程序执行完毕后通过 RTS 指令返回到调用

指令的下一条指令。在上面的程序段中，设计了两个子程序出口：一个是当比较结果相同时，通过正常的 RTS 返回；另一个是当输入三次后比较结果仍不相等时，强迫退出子程序，显示正确的结果。这就是两个 PLA 指令起的作用。下面分析一下它的执行原理：

在执行 0370 的 JSR 指令时，指令计数器 PC 中的值是 0370，这时的堆栈区如图 4.4。由于这是一条转子指令，当子程序执行完后要准确返回调用处，因此，系统必须记住返回地址，这一地址就是 $PC + 2$ ，即 0372，它就存放在堆栈区中（如图 4.5）。然后，把操作码 20 后的绝对地址送入 PC，转向执行 0344 的子程序，当遇到 RTS 指令时，又从堆栈区中取出地址 0372，送入 PC，并加上 1，使程序从 0373 开始继续执行。因此，要强迫退出子程序，只要使堆栈顶部二个单元的内容（返回地址）出栈，使堆栈区恢复到调用子程序前的状态就可以了。第一个 PLA 指令使 $nn - 2$ 中的 44 出栈，第二个 PLA 指令使 $nn - 1$ 中的 03 出栈，栈指针恢复到 nn 上。

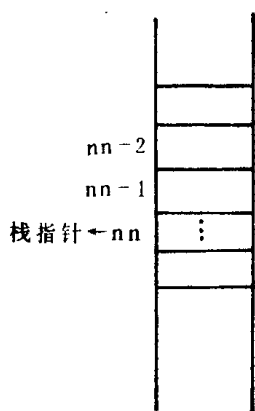


图 4.4 调用子程序前的栈

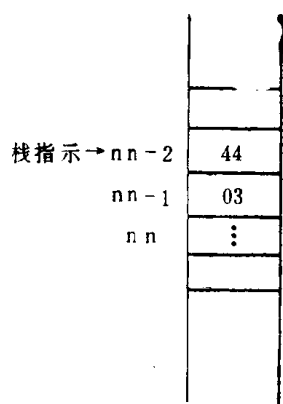


图 4.5 调用子程序后的栈
(注：图中示为针)

(4) E4 E2 CPX \$E2: 零页寻址指令, 把寄存器X中的数与零页单元00E2中的数相比较。

这一程序段先要求输入笔算的结果, 并把它与运算结果相比较。如果相等, 则显示“OK”, 返回程序头; 否则, 要求再次输入, 输入三次后仍不正确时, 屏幕上显示出正确的答案, 并返回程序头。

第四段: 输入数据子程序

0393-	A2	00	LDX	# \$ 00
0395-	A0	00	LDY	# \$ 00
0397-	20	35	FD	JSR \$FD35
039A-	20	ED	FD	JSR \$FDED
039D-	C9	8D		CMP # \$ 80
039F-	F0	07		BEQ \$ 03A8
03A1-	99	00	02	STA \$ 0200,Y
03A4-	C8			INY
03A5-	4C	97	03	JMP \$ 0397
03A8-	98			TYA
03A9-	AA			TAX
03AA-	B9	00	02	LDA \$ 0200,Y
03AD-	C9	C0		CMP # \$ C0
03AF-	30	0E		BMI \$ 03BF
03B1-	29	0F		AND # \$ 0F
03B3-	18			CLC
03B4-	69	09		ADC # \$ 09
03B6-	99	00	02	STA \$ 0200,Y
03B9-	88			DEY
03BA-	D0	EE		BNE \$ 03AA
03BC-	4C	C4	03	JMP \$ 03C4
03BF-	29	0F		AND # \$ 0F

03C1-	4C B6 03	IMP	\$ 03B6
03C4-	CA	DEX	
03C5-	BD 00 02	LDA	\$ 0200,X
03C8-	E0 00	CPX	# \$ 00
03CA-	D0 05	BNE	\$ 03D1
03CC-	91 E0	STA	(\$ E0),Y
03CE-	84 E2	STY	\$ E2
03D0-	60	RTS	
03D1-	CA	DEX	
03D2-	85 E2	STA	\$ E2
03D4-	BD 00 02	LDA	\$ 0200,X
03D7-	0A	ASL	
03D8-	0A	ASL	
03D9-	0A	ASL	
03DA-	0A	ASL	
03DB-	18	CLC	
03DC-	65 E2	ADC	\$ E2
03DE-	91 E0	STA	(\$ E0),Y
03E0-	E0 00	CPX	# \$ 00
03E2-	D0 03	BNE	\$ 03E7
03E4-	84 E2	STY	\$ E2
03E6-	60	RTS	
03E7-	C8	INY	
03E8-	4C C4 03	IMP	\$ 03C4

其中用到的 9 个新指令是：

(1) 98 TYA：隐含寻址指令，把寄存器Y 中的数送入累加器A中，即Y→A。

(2) AA TAX：隐含寻址指令，把累加器A中的数传送到寄存器X中，即A→X。由于在6502的指令系统中，没

有寄存器X和Y之间进行数据传送的指令,因此,常常把累加器A作为中间传送单元。上述二条指令一起使用,实际上就是把寄存器Y中的数传送到寄存器X中,即 $Y \rightarrow X$ ($Y \rightarrow A \rightarrow X$)。

(3) 30 0E BMI \$03BF: 相对寻址指令,当满足标志寄存器中的负数标志为1的条件时,向前转移0E步执行。这里,把BMI指令用在比较指令CMP之后(也可以用在CPX、CPY等指令之后)。因为比较两个数的大小实际上是用第一个数减去第二个数,当差值大于等于0时,置负数标志为0;当差值小于0时,置负数标志为1(这时便满足BMI指令所需的条件,使执行发生转移)。因此,它可以理解为:当累加器A中的数小于C0时,转向03BF执行。

(4) 29 0F AND #0F:立即寻址指令,把累加器A中的数与操作码29后的立即数0F相“与”,结果仍存在A中。用2进制数来说明就是,当两个相“与”的对应位都是1时,该位相“与”的结果仍为1;否则(其中有一个是0或二个都是0),结果为0。如B8与0F相“与”,结果是08:

16进制	2进制
B8	10111000
AND 0F	AND 00001111
08	00001000

实际上,任何一个16进制数与0F相“与”,都将使该数的第一位变为0,如CA与0F相“与”等于0A。

在这一程序段中,03B1和03BF处用了AND指令,其作用是一样的。

(5) 88 DEY: 隐含寻址指令,把寄存器Y中的数减去1。

(6) CA DEX: 隐含寻址指令, 把寄存器 X 中的数减去 1。

(7) 0A ASL: 隐含寻址指令, 把累加器 A 中数的各个数元往左移动一位 (相当于把这个数乘以 2), 最左边移出去的一位送入进位标志中, 右边同时补上一个 0。如:

10 进制	16 进制
ASL $\frac{53}{\text{---}}$	ASL $\frac{35}{\text{---}}$
ASL $\frac{106}{\text{---}}$	ASL $\frac{6A}{\text{---}}$
ASL $\frac{212}{\text{---}}$	ASL $\frac{D4}{\text{---}}$
ASL $\frac{168}{\text{---}}$	ASL $\frac{A8}{\text{---}}$

2 进制
ASL $\frac{00110101}{\text{---}}$
ASL $\frac{01101010}{\text{---}}$
ASL $\frac{11010100}{\text{---}}$
C ← $\frac{1}{\text{---}}$ $\frac{10101000}{\text{---}}$

(8) 65 E2 ADC \$E2: 零页寻址指令, 把累加器 A 中的数与零页单元 00E2 中的数相加, 结果存在 A 中。

(9) 91 E0 STA(\$E0), Y: 后变址的间接寻址指令。在零页单元 E0、E1 中存有一个二字节的绝对地址 (高位在 E1, 低位在 E0 中), 取出这一绝对地址, 并加上寄存器 Y 中的数形成实际地址。这一指令即把累加器 A 中的数存入实际地址中

如地址 E0 中的数为 F0, E1 中的数为 03, Y 中的数为 5 则实际地址为 $03F0 + 5 = 03F5$ 。STA 指令把 A 中的数存入 03F5。

下面对这一段程序作些说明。

首先, 通过 JSR \$FD35 指令接受键盘输入的 16 进制数 (每次接受一个字符的输入, 其 ASCII 码值存放在累加器 A 中), 并存入 0200 开始的内存中。如输入的是 16 进制数

2AC81, 计算机接受的并不是2AC81本身, 而是这些字符的ASCII码值, 即B2、C1、C3、B8、B1, 因此先必须把它转换成02、AC、81三个字节的数。转换分二步进行:

第一步: 把每一个ASCII码值转换成16进制数。如把B2转换成02, C1转成0A。显然, 对于B0到B9之间的数, 只要用AND # \$0F指令使第一位变为0就可以了; 对于C1到C6之间的数也只要用AND # \$0F使第一位变为0, 并加上09就可以了。如C5, 先AND # \$0F变为05, 再加上09变为0E, 即十六进制数本身。

第二步: 把单个的16进制数02、0A、0C、08、01通过组合变为02、AC、81三个数。以08、01组成81为例, 先用ASL指令把08的各个数元左移四位(用四个ASL指令)变为80, 再把80加上01即成为81。

16进制	2进制
ASL 08	ASL 00001000
ASL 10	ASL 00010000
ASL 20	ASL 00100000
ASL 40	ASL 01000000
ASL 80	ASL 10000000

这一程序共占用0300到03 EA的235个内存单元。把它输入计算机, 执行过程如下:

```
*300G
1:340E✓ (输入被减数)
2:2D85✓ (输入减数)
? 0689✓ (输入答案)
OK (正确)
1:A35D1✓ (输入被减数)
2:874D0✓ (输入减数)
```

? 101A0 ↵ (输入答案)
 ? 5034A ↵ (不对, 重新输入)
 ? 39AB1 (不对, 重新输入)
 1C101 (显示正确答案)

6. 乘法运算

在6502的指令系统中, 没有乘法运算一类的指令, 但在程序设计中有时又需要。因此, 可以通过一些间接的方法来实现这种功能。

本节将使用两种不同的方法介绍乘法运算, 即循环加算法和移位算法。通过这两种方法, 读者可以开阔思路, 设计出其它的算法。

(1) 循环加算法 循环加算法具有思路清楚, 直接了当的特点。它的基本思想是被乘数 m 和乘数 n 相乘的积等于 m 个 n (或 n 个 m) 相加的和。如:

$$\begin{array}{c} \text{34个} \\ \hline 34 \times 12 = 12 + 12 \cdots + 12 \text{ 或} \end{array}$$

$$\begin{array}{c} \text{12个} \\ \hline 34 \times 12 = 34 + 34 + \cdots + 34 \end{array}$$

用这种方法可以把乘法运算转换成加法运算, 进而用加法指令来完成。

以6C乘以3D为例, 算法流程如图4.6。

显然, 它是通过3D个6C相加求得它的乘积的。结果的高位 (即每次相加后进位的累加) 存放在零页单元E1中, 低位存放在累加器A中。

程序清单如4.6。

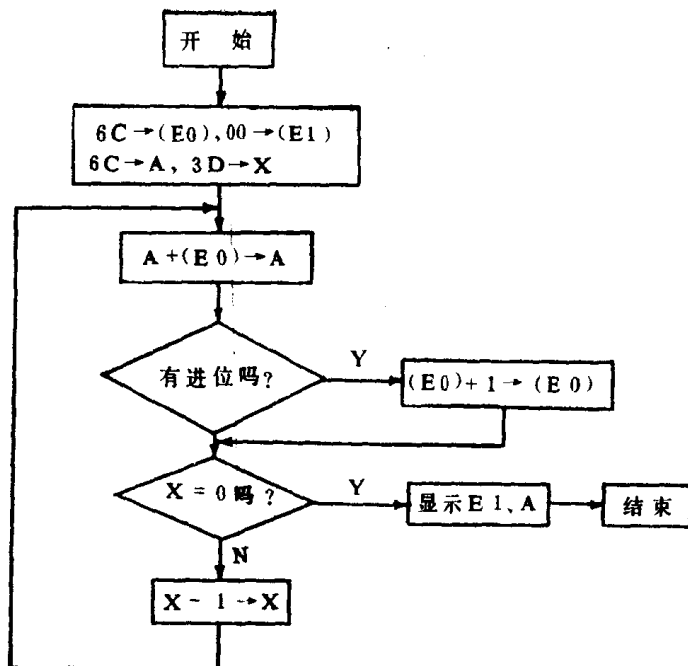


图4.6 算法流程图

例程序4.6

0300-	20	58	FC	JSR	\$FC58
0303-	A2	3D		LDX	#\$3D
0305-	A9	00		LDA	#\$00
0307-	85	E1		STA	\$E1
0309-	A9	6C		LDA	#\$6C
030B-	85	E0		STA	\$E0
030D-	18			CLC	
030E-	65	E0		ADC	\$E0
0310-	90	02		BCC	\$0314
0312-	E6	E1		INC	\$E1
0314-	E0	02		CPX	#\$02

0316-	F0	04	BEQ	\$ 031C
0318-	CA		DEX	
0319-	4C	0D 03	JMP	\$ 030D
031C-	48		PHA	
031D-	A5	E1	LDA	\$ E1
031F-	20	DA FD	JSR	\$ FDDA
0322-	68		PLA	
0323-	20	DA FD	JSR	\$ FDDA
0326-	60		RTS	

程序说明:

0303: 把3D送入寄存器X;
 0305: 把00送入累加器A;
 0307: 把A中的数(00)存入00E1单元;
 0309: 把6C送入累加器A;
 030B: 把A中的数(6C)存入00E0单元;
 030D: 清除代位标志C;
 030E: A(6C)加上00E0中的数;
 0310: 如相加后没有进位, 转0314;
 0312: 00E1中的数加上1;
 0314: X等于2吗?
 0316: 相等则转031C;
 0318: 否则, X减去1;
 0319: 转030D;
 031C: A中的数进栈;
 031D: 取00E1中的数到A;
 031F: 显示A中的数(结果的高位);
 0322: 取堆栈顶部的数到A;
 0323: 显示A中的数(结果的低位);
 0326: 返回。

用到的3个新指令是：

(1) 90 02 BCC \$0314: 相对寻址指令, 当标志寄存器中的代位标志C为0 (即当加法无进位或减法有借位时) 转向0314执行。转移后的地址由操作码90后的一字节相对地址决定。

(2) E6 E1 INC \$E1: 零页寻址指令, 使零页单元00E1中的数加上1。如原来00E1中的数为0A, 则执行INC \$E1后变为0B。

(3) 48 PHA: 隐含寻址指令, 它的功能和前面介绍的PLA指令相反, 即把累加器中的数送入堆栈区的顶部, 同时, 栈指针向上移动一个单元, 即使累加器A中的数进栈。在本例中, 使用PHA指令是为了暂时存放3D个6C相加和的低位。然后, 取出E1中的高位并显示出来, 再用PLA指令把堆栈顶部的数据恢复到累加器A中, 以便调用FDDA子程序显示。

程序对数据6C进行累加计算时, 以寄存器X作为循环控制变量, 其初值设置为3D, 每加一次, X的值减1, 当X减为0时, 退出循环。这样, 就完成了3D个6C的加法运算。

另外, 在加的过程中, 00E0单元存有数6C, 每次把A加上00E0中的6C, 结果仍放在A中。如果相加后有进位, 则使单元00E1的值加1。这种进位的累计和就是最后结果的高位, A是结果的低位。

把这一程序输入内存, 并执行:

* 300G ↙

19BC

*

即3D乘以6C等于19BC。

当然,可以通过改变0304和030A中的两个数来计算任意两个数的乘积。如:

* 0304: 8A ✓

* 030A: F3 ✓

* 300G ✓

82FE

*

即8A乘以F3等于82FE。

同样,对于多字节的两个数相乘,也可以采用循环加的方法,但情况稍微复杂一些。以2AC0乘以381D为例,可以先对低位进行处理,使C0个381D相加(采用二字节相加的方法);再对高位进行处理,使2A×100(这里的100也是16进制数)个381D相加。最后将二部分结果加在一起就是两个数的乘积。

关于多字节乘法问题,本书不作更深入的讨论,有兴趣的读者可以根据同样的思路编写程序。

(2) 移位算法 为了让读者先了解什么是移位算法,先举一个笔算的乘法例子。

10进制

$$\begin{array}{r} 25 \\ \times 37 \\ \hline 175 \\ 75 \\ \hline 925 \end{array}$$

2进制

$$\begin{array}{r} 00011001 \\ \times 00100101 \\ \hline \end{array}$$

被乘数左移 0 位——→00011001

00000000

被乘数左移 2 位——→00011001

00000000

00000000

被乘数左移 5 位——→00011001

00000000

00000000

000001110011101

$$\begin{aligned}\text{而 } (000001110011101)_2 &= (1 \times 2^9 + 1 \times 2^7 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1)_{10} \\ &= (925)_{10}\end{aligned}$$

可见，2 进制算法的结果和10进制一样，是正确的。

从这个 2 进制乘法的例子中可以看到，二个 2 进制数相乘：

① 当乘数出现 0 位时，可以不考虑，而直接进行下一位的运算（如本例中的第 2、4、5、7、8 位为 0）；

② 当乘数出现 1 位时，被乘数的所有数元左移 $n-1$ 位（ n 为乘数中 1 出现的位置，从右到左为第 1—8 位）后，累加起来。

这个累加和就是两个数的乘积。所以，移位算法就是在被乘数的各个数元每次向左移动若干位后累加起来形成两数积的算法。

例程序4.7

0300-	20	58	FC	JSR	\$FC58
0303-	A9	00		LDA	# \$00
0305-	8D	34	03	STA	\$0334
0308-	8D	35	03	STA	\$0335
030B-	A2	08		LDX	# \$08
030D-	0A			ASL	
030E-	2E	35	03	ROL	\$0335
0311-	0E	33	03	ASL	\$0333
0314-	90	09		BCC	\$031F
0316-	18			CLC	
0317-	6D	32	03	ADC	\$0332
031A-	90	03		BCC	\$031F
031C-	EE	35	03	INC	\$0335

031F-	CA		DEX	
0320-	D0	EB	BNE	\$ 030D
0322-	8D	34 03	STA	\$ 0334
0325-	AD	35 03	LDA	\$ 0335
0328-	20	DA FD	JSR	\$ FDDA
032B-	AD	34 03	LDA	\$ 0334
032E-	20	DA FD	JSR	\$ FDDA
0331-	60		RTS	

程序说明:

0300: 清洗屏幕;

0303—030A: 乘积的低、高位存放单元清 0;

030B: 取移位次数 08 到寄存器 X;

030D—0313: 作移位处理;

0316—0319: 加上被乘数;

031A—031E: 相加后如有进位, 则 0335 (乘积高位) 加 1;

0320: X 减 1 后如不等于 0, 转 030D 继续;

0322: 将结果的低位存入 0334;

0325—0330: 显示结果的高、低位。

例程序 4.7 就是一个进行两数相乘运算的简单程序。在程序执行前, 必须把被乘数置入 0332, 乘数置入 0333 单元。最后乘积的高位存放在 0335 中, 而低位存放在 0334 中。

用到的 4 个新指令是:

① 2E 35 03 ROL \$0335: 这是一个绝对寻址指令, 它把操作码 2E 后二字节地址所指单元中的数往左循环一位。

前面曾介绍过一个 ASL 指令, 它的功能是把操作数往左移动一位。那么, 这两个指令区别在什么地方呢? 通过比较可以说明这个问题。

相同点:

• ASL 和 ROL 指令都是把指定数的各个数元往左移动一位，移动方向是一致的；

• 操作数经ASL和ROL运算后，最左边的一个数元（最高位）都会被移入标志寄存器的代位标志 C 中。

不同点：

ASL指令把操作数的各数元左移一位后，最右边一位总是用 0 补充，而ROL指令把操作数的各数元左移一位后，最右边一位用代位标志 C 补充（即代位标志 C 为 1 时，最右边补 1；C 为 0 时，最右边补 0）。

由此可见，如果在执行ROL之前，代位标志 C 为 0，那么它和执行ASL指令的结果是一致的。

可见，对一个数进行连续九次的ROL操作后，仍然恢复

例：

执行前 C 值	ASL		ROL		执行后 C 值
	16进制	2 进制	16进制	2 进制	
	34	00110100	34	00110100	
0	68	01101000	68	01101000	0
0	D0	11010000	D0	11010000	0
0	A0	10100000	A0	10100000	1
1	40	01000000	41	01000001	1
1	80	10000000	83	10000011	0
0	00	00000000	06	00000110	1
1	00	00000000	0D	00001101	0
0	00	00000000	1A	00011010	0
0	00	00000000	34	00110100	0

到原来这个数；而进行连续九次的 ASL 操作后，这个数的所有数元都变为 0。

② 0E 33 03 ASL \$0333: 绝对寻址指令，把操作码 0E 后二字节地址单元中的数左移一位。

③ 6D 32 03 ADC \$0332: 绝对寻址指令，把累加器 A 中的数加上操作码 6D 后二字节地址单元中的数，再加上代位标志 C，结果存放在 A 中。

④ EE 35 03 INC \$0335: 绝对寻址指令，把操作码 EE 后二字节地址单元中的数加 1。

根据前面列举的 2 进制乘法例子，我们必须知道乘数的哪些数元是 1，在这些数元上，对被乘数进行左移累加。在程序 4.7 中，使用了左移的方法 (ASL) 来判别乘数的各个数元，这就是地址 0311 中的 0E 33 03 指令。当 X 内的值从 8 循环到 0 时，这个指令被执行过 8 次，也即乘数的各个数元先后被移到了代位标志 C 中。用 BCC 指令当代位标志为 0 时，跳过中间过程；否则，就将被乘数加入一次。

把这个程序输入内存，并存入被乘数和乘数，再运行：

* 332: 19 ✓

* 333: 25 ✓

* 300G ✓

039D

*

结果 $(039D)_{16} = (925)_{10}$ ，和手算的结果完全一样。改变被乘数和乘数可以计算任二个数的乘积：

* 332: 39 FA ✓

* 300G ✓

37AA

即39乘以FA等于37AA。

7. 除 法 运 算

和乘法一样，除法运算也可以有两种方法，这就是循环减算法和移位算法。下面分别介绍这两种算法。

(1) 循环减算法 先用被除数减去除数，以后用相减的差减去除数，直到差值小于除数为止，则相减的次数就是这两数相除的商，最后留下的差就是余数。这种把除法转换成减法的算法就是循环减的思路。

例：

10进制	16进制	2 进制
86	56	01010110
- 21	- 15	- 00010101
<hr/> 65	<hr/> 41	<hr/> 01000001
- 21	- 15	- 00010101
<hr/> 44	<hr/> 2C	<hr/> 00101100
- 21	- 15	- 00010101
<hr/> 23	<hr/> 17	<hr/> 00010111
- 21	- 15	- 00010101
<hr/> 2	<hr/> 2	<hr/> 00000010

该例中总共相减了4次，最后的差值是2，因此56除以15商是4，余数是2。

从例子中可以看到，循环减算法的思路也是很清楚的。

现在，来编写一个二字节数除以一字节数的机器语言程序。并将被除数的低位存在03A0中，高位存在03A1中，除数存在03A2中，商和余数分别存放在03A3和03A4中，程序框图如图4.7。

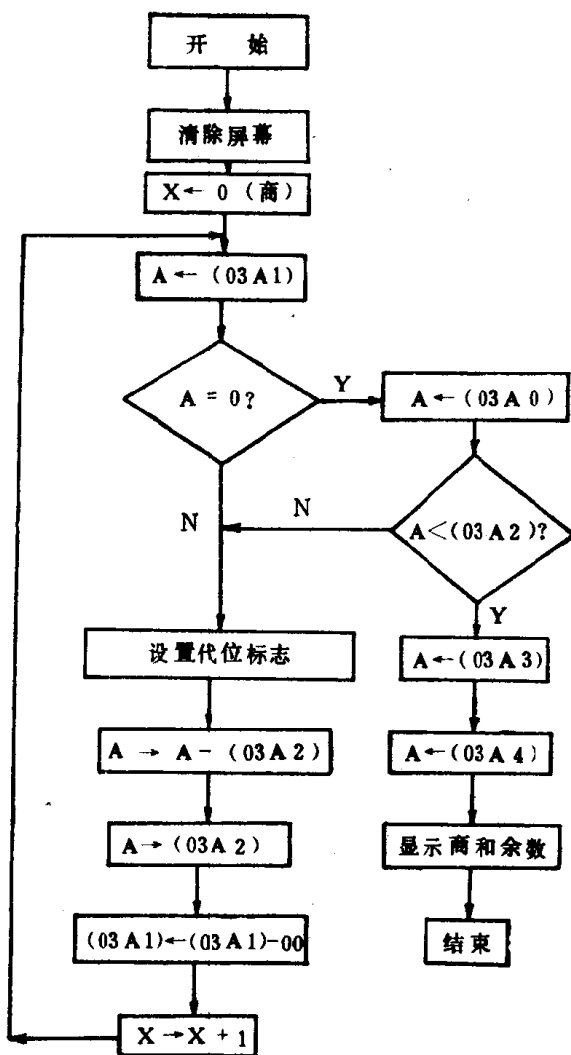


图4.7

例程序4.8

0300-	A2	00	LDX	# \$ 00
0302-	AD	A1	LDA	\$ 03A1

0305-	D0	08		BNE	\$ 030F
0307-	AD	A0	03	LDA	\$ 03A0
030A-	CD	A2	03	CMP	\$ 03A2
030D-	90	16		BCC	\$ 0325
030F-	38			SEC	
0310-	AD	A0	03	LDA	\$ 03A0
0313-	ED	A2	03	SBC	\$ 03A2
0316-	8D	A0	03	STA	\$ 03A0
0319-	AD	A1	03	LDA	\$ 03A1
031C-	E9	00		SBC	≠ \$ 00
031E-	8D	A1	03	STA	\$ 03A1
0321-	E8			INX	
0322-	4C	02	03	JMP	\$ 0302
0325-	8E	A3	03	STX	\$ 03A3
0328-	8D	A4	03	STA	\$ 03A4
032B-	8A			TXA	
032C-	20	DA	FD	JSR	\$ FDDA
032F-	20	8E	FD	JSR	\$ FD8E
0332-	AD	A4	03	LDA	\$ 03A4
0335-	20	DA	FD	JSR	\$ FDDA
0338-	60			RTS	

程序说明:

0300: 寄存器X (存放商值) 清 0;

0302—0306: 被除数高位如不等于 0, 转030F;

0307—030E: 否则判断被除数低位与除数的大小, 如已不够减,
则转出循环;

030F—0320: 否则, 将被除数减去除数;

0321: 商值加上 1;

0322: 转0302重复上述步骤;

0325: 商值存入03A3单元;
 0328: 余数存入03A4单元;
 032B: 寄存器X中的数送入A;
 032C: 显示商值;
 032F: 回车;
 0332: 取03A4中的数(余数)到A;
 0335: 显示余数;
 033B: 返回。

程序中使用了4个新的指令:

① CD A2 03 CMP \$ 03A 2: 绝对寻址指令, 比较累加器A中的数与地址单元03A2中的数的大小。

② E9 00 SBC# \$ 00: 立即寻址指令, 把累加器A中的数减去立即数00, 再减去代位标志C, 结果存放在累加器A中。

③ 8E A3 03 STX \$ 03A3: 绝对寻址指令, 把寄存器X中的数存入操作码8E后的绝对地址单元03A3中。

④ 8A TXA: 隐含寻址指令, 把寄存器X中的数送入累加器A。

在这个程序中, 寄存器X存放相减的次数(商), 当出现一次相减时, X就加上1。

(2) 移位算法 先举一个笔算的例子:

10 进 制

2 进 制

$$\begin{array}{r} 16 \\ 23 \overline{) 385} \\ \underline{23} \\ 155 \\ \underline{138} \\ 17 \end{array}$$

$$\begin{array}{r} 10000 \\ 00010111 \overline{) 000110000001} \\ \underline{00010111} \\ 10001 \end{array}$$

即 $(385)_{10}$ 除以 $(23)_{10}$ 商是 $(16)_{10}$, 余数是 $(17)_{10}$ 。或者

说, $(181)_{10}$ 除以 $(17)_{10}$ 商是 $(10)_{10}$, 余数是 $(11)_{10}$ 。

从这个例子中可以看到, 除法和乘法一样也要进行移位。

假设把被除数的低位存放在 0350, 高位存放在 0352, 最后的商存放在 0353, 余数存放在 0354 中, 则程序框图如图 4.8。

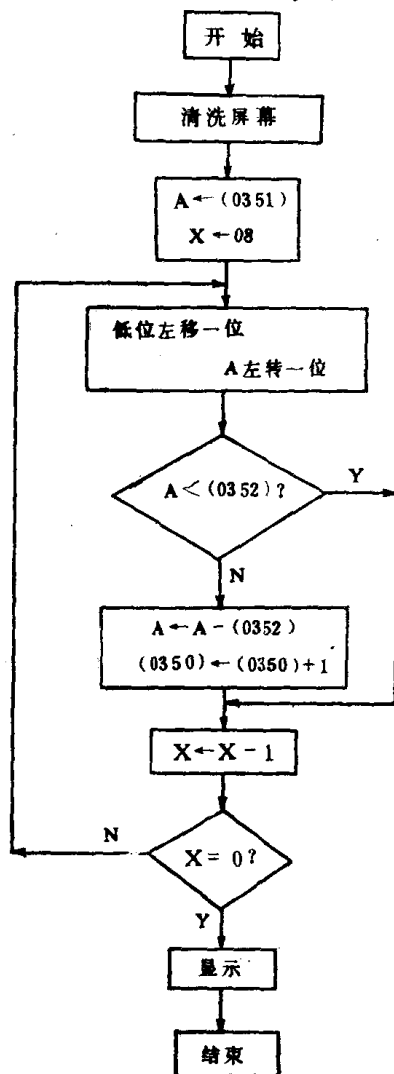


图 4.8

根据这个框图，可以编写出用移位算法来完成除法运算的机器语言程序。

例程序4.9

0300-	EA			NOP	
0301-	A2	08		LDX	# \$ 08
0303-	AD	51	03	LDA	\$ 0351
0306-	0E	50	03	ASL	\$ 0350
0309-	2A			ROL	
030A-	CD	52	03	CMP	\$ 0352
030D-	90	06		BCC	\$ 0315
030F-	ED	52	03	SBC	\$ 0352
0312-	EE	50	03	INC	\$ 0350
0315-	CA			DEX	
0316-	D0	EE		BNE	\$ 0306
0318-	8D	54	03	STA	\$ 0354
031B-	48			PHA	
031C-	AD	50	03	LAD	\$ 0350
031F-	8D	53	03	STA	\$ 0353
0322-	20	DA	FD	JSR	\$ FDDA
0325-	20	8E	FD	JSR	\$ FD8E
0328-	68			PLA	
0329-	20	DA	FD	JSR	\$ FDDA
032C-	60			RTS	

程序说明：

0301：取移位次数08到寄存器X；

0303—0317：移位运算；

031B：将余数推入堆栈；

0322：显示商值

0325：回车；

0328: 取出堆栈顶部的数(余数),

0329: 显示余数。

把程序49输入内存, 设置好被除数和除数, 即可执行:

* 350: 47 03 81 ↵

* 300G ↵

06

41

*

即16进制数0347除以81的商是06, 余数是41。通过改变0350、0351、0352单元中的数, 就可以进行其它数的除法运算。

程序4.9中使用了1个新的指令:

2A ROL: 隐含寻址指令, 把累加器A中数的各个数元往左转动一位, 最右边一位用代位标志C补入。

8. 10进制运算

前面已经说过, 在计算机中, 数是用2进制方式来表示的, 它只有1和0两个元素, 可以用电位的高低或电路的通断表示出来。但2进制数对用户来说却是非常繁琐而不直观的。因此, 一般在人机界面中都采用数的16进制表示法。本章前面的几节所介绍的内容就是16进制数的加、减、乘、除运算。

但另一方面, 我们在日常生活中, 又总是习惯于使用10进制的数, 并需要对10进制数进行处理和运算。这样, 数的处理流程就成为:

10进制数 $\xrightarrow{\text{转换}}$ 16进制数 $\xrightarrow{\text{运算、处理}}$ 16进制结果 $\xrightarrow{\text{转换}}$ 10进制结果

其中有两个环节用于10进制和16进制之间的转换。显然，也是比较繁琐而费时的。为解决这个问题，6502专门设置了一个解决10进制数字运算的指令。使用这个指令，6502就能对10进制数进行运算，并得出10进制表示的结果。

这个指令就是：

F8 SED

它是一个隐含寻址指令，把标志寄存器中的10进制标志D设置为1，使得此后的所有加减运算都以10进制的方式进行（其它指令的执行不受它的影响）。

相应地，还有一个恢复16进制加减运算的指令，即：

D8 CLD

这也是一个隐含寻址指令，把标志寄存器中的10进制标志D清为0，使以后的加减运算恢复到16进制方式。

那么，读者可能要问，前面的16进制运算程序中，从来没有执行过CLD指令，如果在执行前，D标志已经被设置为1，那答案岂不是就不对了吗？实际上，CLD指令在同一个程序执行过程中要从10进制恢复为16进制时是必须使用的。而当一个程序执行完毕返回监控状态时，监控系统就自动地把D标志清为0。因此，在一个程序从监控状态开始执行时，运算方式总是16进制的。

下面各举一个10进制加法和减法运算的例子，来说明这两个指令的功能和用法。

例程序4.10

0300-	F 8	SED	
0301-	A 9 35	LDA	# \$ 35
0303-	18	CLC	
0304-	69 59	ADC	# \$ 59

0306- 20 DA FD JSR \$FDDA

0309- 60 RTS

程序说明:

0300: 设置D标志;

0301: 将立即数35送入累加器A;

0303: 清除C标志;

0304: 使A和立即数59相加;

0305: 显示相加的和;

0309: 返回。

输入并执行这个程序:

* 300 G ↙

94

*

即 $35 + 59 = 94$, 这个10进制加法的答案是正确的。改变0302 (被加数)和0305 (加数)两个单元中的数, 就可以做其它的加法:

* 302 : 37 ↙

* 305 : 44 ↙

* 300 G ↙

81

*

即 $(37)_{10} + (44)_{10} = (81)_{10}$ 。

在前面介绍的16进制加法中, 一个字节的数最大是FF (即10进制255), 而在10进制中, 一个字节的数最大是99。因此, 使用程序4.10时, 必须注意相加后的和要小于等于99。如果两数相加的和大于99, 则最高一位将被丢掉, 而造成运算结果的不完整。

如:

* 302 : 87 ↙

* 305 : 35 ↙

* 300G ↙

22

*

这个答案显然是不完整的。87加上35应是122,因为大于99,最左边的1被移入代位标志。用多字节运算的方法可以解决这个问题。

下面再看一个10进制减法运算的程序。

例程序4.11

0300-	F8		SED
0301-	A9	70	LDA # \$ 70
0303-	38		SEC
0304-	E9	32	SBC # \$ 32
0306-	20	DA FD	JSR \$ FDDA
0309-	20	8E FD	JSR \$ FD8E
030C-	D8		CLD
030D-	A9	70	LDA # \$ 70
030F-	38		SEC
0310-	E9	32	SBC # \$ 32
0312-	20	DA FD	JSR \$ FDDA
0315-	60		RTS

程序说明:

0300: 置10进位标志D为1;

0301—0305: 计算70和32的差值;

0306: 显示差值;

0309: 回车;

030C: 清10进位标志D为0;
030D—0311: 计算70和32的差值;
0312: 显示差值。
* 300G ↙

38

3E

*

即10进制70减去32等于38, 16进制70减去32等于3E。在030C中使用了CLD指令, 目的是使后面的减法运算恢复到16进制方式下进行。

这一节, 就简单介绍这两个程序。读者可以参考16进制的加、减、乘、除运算的思路和程序, 使用SED指令编写出10进制方式下的加、减、乘、除运算程序, 以进一步巩固本章所学的内容。

9. 指令讨论

本章中共学习了45个新的指令, 现在综合进行讨论。

(1) ADC: 使用在立即寻址方式下, 把累加器A中的数加上操作码后的立即数, 再加上代位标志C。操作码是69。

例: 69 05 ADC # \$05

若执行前A为02, 代位标志C为0, 则相加后A为07。

(2) ADC: 使用在零页寻址方式下, 把累加器A中的数加上操作码后零页单元中的数, 再加上代位标志C, 结果存放在A中。操作码是65。

例: 65 E0 ADC \$E0

若执行前A为03, 地址00E0中的数为05, 代位标志C为0, 则执行后A为08。

(3) ADC: 使用在绝对寻址方式下,把累加器A中的数加上操作码后二字节绝对地址中的数,再加上代位标志C,结果仍存放在A中。操作码是6D。

例: 6D 20 03 ADC \$0320

即 $A \leftarrow A + (0320) + C$ 。

(4) ADC: 使用在寄存器X绝对寻址方式下,把操作码后的二字节绝对地址加上寄存器X中的数构成新地址,把累加器A中的数加上新地址单元中的数,再加上代位标志C,结果存在A中。操作码是7D。

例: 7D 20 03 ADC \$0320, X

即 $A \leftarrow A + (0320 + X) + C$ 。

(5) AND: 使用在立即寻址方式下,把累加器A中的数与操作码后的立即数进行“与”运算,结果存放在A中。操作码是29。

例: 29 15 AND #\$15

若执行前A为A7,则该指令执行过程如下:

16进制

	A7
AND	15
<hr/>	
	05

2进制

	10100111
AND	00010101
<hr/>	
	00000101

即: 对应位都为1的相与后仍为1,否则为0。“与”结果存入A。

(6) ASL: 使用在隐含寻址方式下,把累加器A中数各个位元左移一位,右边补0。操作码是0A。

例: 0A ASL

若执行前A为FC,则:

$$\begin{array}{r} 11111100 \\ \text{ASL} \\ \hline 11111000 \end{array}$$

最左边移出去的位 1 进入代位标志中。

(7) ASL: 使用在绝对寻址方式下, 把操作码后二字节地址单元中数的各个位元左移一位, 右边补 0。操作码是 0E。

例: 0E 20 03 ASL \$0320

(8) BCC: 使用在相对寻址方式下, 当标志寄存器中的代位标志为 0 时转移执行。操作码是 90。

例: 90 03 BCC \$0315

即当 C 为 0 时, 转向 0315 执行。

(9) BMI: 使用在相寻址方式下, 当标志寄存器中的负数标志 N 为 1 时发生转移。操作码是 30。

例: 30 03 BMI \$0318

(10) CLC: 使用在隐含寻址方式下, 把代位标志 C 清为 0。操作码是 18。

例: 18 CLC

(11) CLD: 使用在隐含寻址方式下, 把 10 进位标志) 清为 0。操作码是 D8。

例: D8 CLD

(12) CMP: 使用在绝对寻址方式下, 把累加器 A 中的数与操作码后绝对地址中的数进行比较。操作码是 CD。

例: CD 20 03 CMP \$0320

比较结果可能影响 N、Z 二个标志,

① 当 A 小于指定单元中的数时, 负数标志置为 1 (相减为负);

② 当A等于指定单元中的数时，零值标志置为1（相减为0）；

(13) CMP：使用在寄存器X绝对寻址方式下，使操作码后的绝对地址加上X中的数构成一新地址，比较累加器A与新地址单元中的数。操作码是DD。

例：DD 20 03 CMP \$0320, X

(14) CPX：使用在零页寻址方式下，比较寄存器X与操作码后零页地址单元中的数。操作码是E4。

例：E4 00 CPX \$00

(15) CPX：使用在绝对寻址方式下，比较寄存器X与操作码后绝对地址中的数。操作码是EC。

例：EC E0 03 CPX \$03E0

(16) CPY：使用在立即寻址方式下，比较寄存器Y与操作码后的立即数。操作码是C0。

例：C0 10 CPY #\$10

(17) CPY：使用在零页寻址方式下，比较寄存器Y与零页单元中的数。操作码是C4。

例：C4 E1 CPY \$E1

(18) DEX：使用在隐含寻址方式下，将寄存器X中的数减去1。操作码是CA。

例：CA DEX

(19) DEY：使用在隐含寻址方式下，将寄存器Y中的数减去1。操作码是88。

例：88 DEY

(20) INC：使用在零页寻址方式下，将零页单元中的数加上1。操作码是E6。

例：E6 3C INC \$3C

(21) INC: 使用在绝对寻址方式下, 将操作码后绝对地址单元中的数加上 1。操作码是 EE。

例: EE 34 03 INC \$0334

(22) INY: 使用在隐含寻址方式下, 把寄存器 Y 中的数加上 1。操作码是 C8。

例: C8 INY

(23) LDA: 使用在寄存器 Y 绝对寻址方式下, 将操作码后的绝对地址加上寄存器 Y 构成一新地址, 把这一新地址单元中的数送入累加器 A。操作码是 B9。

例: B9 15 03 LDA \$0315, Y

即 $A \leftarrow (0315 + Y)$

(24) LDX: 使用在绝对寻址方式下, 将操作码后二字节地址单元中的数送入寄存器 X。操作码是 AE。

例: AE 10 03 LDX \$0310

(25) LDY: 使用在立即寻址方式下, 将立即数送入寄存器 Y。操作码是 A0。

例: A0 FF LDY #\$FF

(26) PHA: 使用在隐含寻址方式下, 将累加器 A 中的数送入堆栈顶部, 同时栈指针上移一个单元。操作码是 48,

例: 48 PHA

(27) PLA: 使用在隐含寻址方式下, 将堆栈顶部的一个数送入累加器 A, 同时栈指针下移一个单元。操作码是 68。

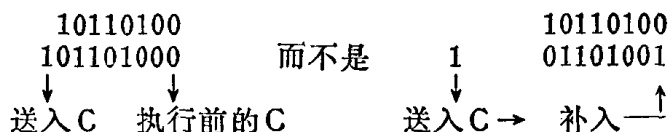
例: 68 PLA

(28) ROL: 使用在隐含寻址方式下, 将累加器 A 中数的各个位元左转一位, 最左边一位进入代位标志 C, 最右边用代位标志 C 补入。操作码是 2A。

例: 2A ROL

应该注意,执行ROL指令补入最右边的代位标志C是执行该指令前的C,而不是执行该指令时由最左边一位送去的C值。

如(执行ROL前代位标志为0):



(29) ROL: 使用在绝对寻址方式下,将绝对地址单元中数的各位元左转一位。操作码是2E。

例: 2E 05 03 ROL \$0305

(30) SBC: 使用在立即寻址方式下,将累加器A中的数减去操作码后的立即数,再减去借位,结果存放在A中。操作码是E9。

例: E9 03 SBC # \$03

即 $A \leftarrow A - 03 - C$

(31) SBC: 使用在绝对寻址方式下,将累加器A中的数减去绝对地址单元中的数,再减去借位,结果存在A中。操作码是ED。

例: ED 0A 03 SBC \$030A

即 $A \leftarrow A - (030A) - C$

(32) SBC: 使用在寄存器Y绝对寻址方式下,把绝对地址加上Y中的数构成一新地址,再将累加器A中的数减去新地址单元中的数,减去借位,结果存放在A中。操作码是F9。

例: F9 0F 03 SBC \$030F, Y

即: $A \leftarrow A - (030F + Y) - C$

(33) SEC: 使用在隐含寻址方式下,将标志寄存器中

的代位标志C设定为1。操作码是38。

例：38 SEC

(34) SED：使用在隐含寻址方式下，将标志寄存器中的10进位标志D设定为1。操作码是F8。

例：F8 SED

(35) STA：使用在零页寻址方式下，将累加器A中的数存入零页单元中。操作码是85。

例：85 00 STA \$00

(36) STA：使用在寄存器Y绝对寻址方式下，将累加器A中的数存入绝对地址加上Y所指向的地址单元中。操作码是99。

例：99 CA 03 STA 03CA, Y

即 $A \rightarrow (03CA + Y)$

(37) STA：使用在寄存器Y的后变址间接寻址方式下，先取出零页单元中存放的绝对地址，将此绝对寻址加上寄存器Y中的值构成操作数地址，将累加器A中的数送入操作数地址。操作码是91。

例：91 00 STA (\$00), Y

若00中存有80，01中存有03，则该指令即：

$A \rightarrow (0380 + Y)$

即Y为0时，A送入地址0380；Y为1时，A送入0381。

(38) STX：使用在绝对寻址方式下，将寄存器X中的数存入操作码后的绝对地址单元中。操作码是8E。

例：8E 0A 03 STX \$030A

(39) STY：使用在零页寻址方式下，将寄存器Y中的数存入零页单元中。操作码是84。

例：84 00 STY \$00

(40) TAX: 使用在隐含寻址方式下, 将累加器 A 中的数送入寄存器 X。操作码是 AA。

例: AA TAX

(41) TYA: 使用在隐含寻址方式下, 将寄存器 Y 中的数送入累加器 A。操作码是 98。

例: 98 TYA

(42) TXA: 使用在隐含寻址方式下, 将寄存器 X 中的数送入累加器 A。操作码是 8A。

例: 8A TXA

(43) PHP: 使用在隐含寻址方式下, 将标志寄存器推入堆栈区。操作码是 08。

例: 08 PHP

(44) PLP: 使用在隐含寻址方式下, 将堆栈顶部的一个数送入标志寄存器 P。操作码是 28。

例: 28 PLP

(45) NOP: 使用在隐含寻址方式下, 当程序执行到 NOP 指令时不发生任何动作。操作码是 EA。

例: EA NOP

五、 6502的发音和绘图

上一章较为详细地介绍了6502的算术运算功能及其实现方法，连同输入输出功能一起，读者已经能够编写出许多具有实用性的机器语言程序了。

但是，作为完整的应用软件，往往还要求具有直观性和趣味性，这就有必要增加一些能够给人以直观感觉和吸引力的功能。本章将要介绍的内容就是6502的发音和绘图功能及其机器语言实现方法，同时也作为对6502机器语言更进一步的探讨。

有了声音和图象，程序就显得更加生动，同时，能比较明显地提高程序的使用效果。特别对游戏、辅助教学一类的软件来说，声音和图象的效果是无法用其它东西来代替的，而对于一些辅助设计软件来说，声音和图象则是必不可少的。

1. 发音原理和发音子程序

在中华学习机的主机箱中有一个专供发音用的喇叭，这个喇叭是由内存单元C030作为触发器推动发音的。从这个单元中取出一个数或者往这个单元中送入一个数，都将在它的地址线上产生高电位和低电位。当产生高电位时，喇叭的放大电路中有电流通过，线圈带动喇叭中圆锥体的底面作向外放大运动，促使底面周围的空气发生振动而发出声音；当产生低电位时，喇叭的放大电路中电流消失，圆锥体的底面作

向里回收运动恢复到原来的位置，也促使底面周围的空气发生振动而发出声音。

因此，根据需要控制触发单元输送出有规律的高低电位，就能使喇叭发出不同音调和频率的声音来。

为了使喇叭发出声音，最简单的办法就是在监控状态下访问地址C030：

* C030 ✓

* (喇叭发出声音)

当然，也可以在程序中使用指令对这个单元进行读写操作：

AD 30 C0 LDA \$C030

或 8D 30 C0 STA \$C030

但这样发出的单音非常轻微而短促，因此必须编写程序来控制触发器（例程序5.1）。这个程序主要控制二个参量：一个是位于0000单元中的数值，它控制发音的频率；一个是位于0001单元中的数值，它控制发音的时间。

例程序5.1

0300-	AD	30	C0	LDA	\$C030
0303-	88			DEY	
0304-	D0	05		BNE	\$030B
0306-	C6	01		DEC	\$01
0308-	D0	01		BNE	\$030B
030A-	60			RTS	
030B-	CA			DEX	
030C-	D0	F5		BNE	\$0303
030E-	A6	00		LDX	\$00
0310-	4C	00	03	JMP	\$0300

其执行框图如图5.1。

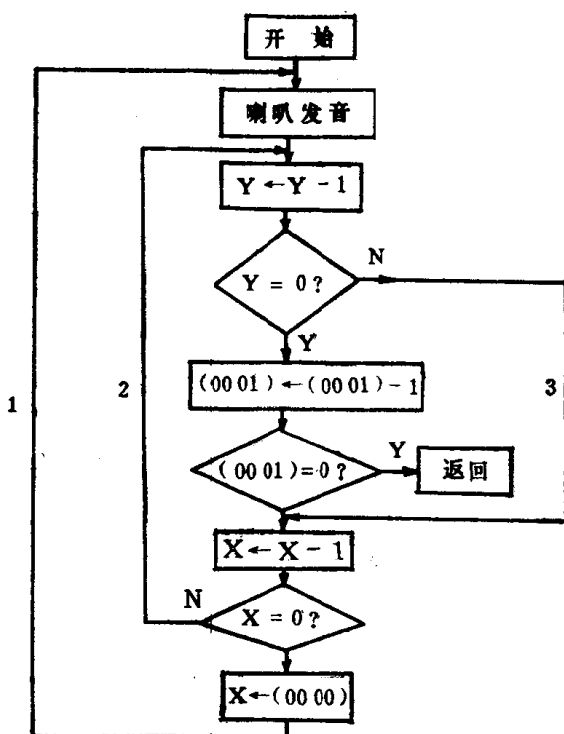


图5.1

从框图中可以看到,整个程序执行有1、2和3 3个循环。循环1每进行一次,喇叭就发音一次。执行时间越长,喇叭两次发音之间的时间间隔越长,即周期长,频率高;执行时间越短,喇叭两次发音之间的时间间隔越短,即周期短,频率低。而循环1是否执行取决于X是否等于0,因此X(即单元0000中的值)越大,就要较长时间才能等于0从而执行循环1;X越小,就可在较短时间内等于0从而执行循环1。所以说,0000中的值控制喇叭的发音频率。

循环2控制发音的次数(也就是整个音节的长短),这

种控制是通过0001单元进行的。当0001中的值较大时，就需要较长时间才能等于0从而退出执行；当0001中的值较小时，就可在较短时间等于0从而退出执行。因此0001单元的值控制音节的长短。

至于循环3只是循环2中的一个过程，它的设置使得循环2从Y为FF进行到Y为00时才对0001是否等于0判断一次。从而保证每一个发音的音节都具有一个最短的基本时间。

程序5.1就是机器语言发音子程序。给定一个音阶值(0000)和一个节拍值(0001)，它就能发出一个需要的音节来。那么，到底如何选择适当的音阶和节拍控制值呢？读者可以在0000和0001中填入不同的值并执行程序5.1进行比较确定。这里，列出一串数据，供读者参考。

音阶	控制值	节拍	控制值
1	C0	十六分音符	1E
2	AC	八分音符	46
3	98	八分音符加附点	6E
4	90	四分音符	A0
5	80	四分音符加附点	FF
6	72		
7	66		
i	60		
0	00		

程序5.1中用到2个新的指令：

(1) C6 01 DEC \$01: 零页寻址指令，把操作码后零页单元中的数减去1。

(2) A6 00 LDX \$00: 零页寻址指令，把操作码

后零页单元中的数送入寄存器X。

2. 键盘控制发音

利用第一节中提供的机器语言发音子程序，我们可以编写出很多发音的实用程序。在这些程序中，主要要做两件事：一是根据事先设计好一串音调和节拍数据或者键盘输入的信息利用循环逐个送入0000和0001单元；二是在循环体当中调用机器语言发音子程序的发音。

作为两个具体的例子，本节中我们介绍一个单音发音程序和一个键盘控制发音的程序。

单音发音程序比较简单，它给出一个音阶表，从这个表中逐个取出音阶，并调用发音子程序。

例程序5.2

0313-	20	58	FC	JSR	\$FC58
0316-	A9	A0		LDA	#\$A0
0318-	85	01		STA	\$01
031A-	A2	00		LDX	#\$00
031C-	BD	2E	03	LDA	\$032E,X
031F-	85	00		STA	\$00
0321-	86	02		STX	\$02
0323-	20	00	03	JSR	\$0300
0326-	A6	02		LDX	\$02
0328-	E8			INX	
0329-	E0	08		CPX	#\$08
032B-	D0	EF		BNE	\$031C
032D-	60			RTS	
032E-	C0	AC			
0330-	98	90	80	72	66 60

程序说明:

0413—0316: 设定单音发音时间为A0

0318: 存入0001

031C: 取得音调值

031F: 存入0000

0321: X暂存到0002中

0323: 调用子程序

032E: 8个音阶值

程序中用了1个新的指令, 即:

86 02 STX \$02: 这是一个零页寻址指令, 它把寄存器X中的数存入零页单元0002中。

程序执行的框图如图5.2。

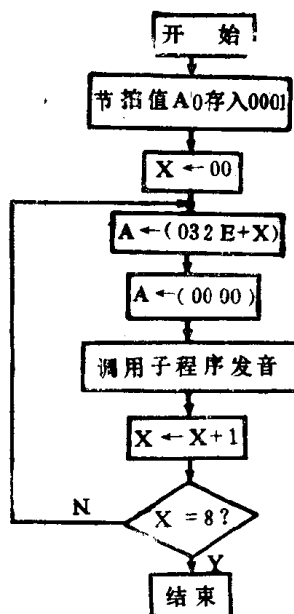


图5.2

把程序5.2连同机器语言发音子程序5.1一起输入内存,

并执行，就能连续发出八个单音来。

请注意程序5.2中使用的一个技巧：象BASIC程序一样，我们把八个音阶值数据置于程序尾端，然后在程序中设置循环，需要时读出这些数据。地址031C中的BD2E03指令就是完成这个功能的，它以寄存器X中的值（00—08）作为标准取数：

当X = 00时，A ← 032E (C0)；

当X = 01时，A ← 032F (AC)；

当X = 02时，A ← 0330 (98)；

.....

下面我们来分析一个键盘控制发音的程序。在这个程序中考虑了第一节列出的九个音阶和五个节拍（读者还可以对它进行扩充，以进一步完善音阶和节拍），把它输入内存并执行，就可以通过按键来演奏乐曲。

例程序5.3

0313-	20	58	FC	JSR	\$FC58
0316-	20	35	FD	JSR	\$FD35
0319-	C9	AF		CMP	# \$AF
031B-	30	F9		BMI	\$0316
031D-	C9	B8		CMP	# \$B8
031F-	30	19		BMI	\$033A
0321-	A0	00		LDY	# \$00
0323-	D9	48	03	CMP	\$0348,Y
0326-	F0	09		BEQ	\$0331
0328-	C8			INY	
0329-	C8			INY	
032A-	C0	0A		CPY	# \$0A

032C-	D0	F5		BNE	\$ 0323
032E-	4C	16	03	JMP	\$ 0316
0331-	C8			INY	
0332-	B9	48	03	LDA	\$ 0348, Y
0335-	85	01		STA	\$ 01
0337-	4C	16	03	JMP	\$ 0316
033A-	29	0F		AND	# \$ 0F
033C-	AA			TAX	
033D-	BD	52	03	LDA	\$ 0352, X
0340-	85	00		STA	\$ 00
0342-	20	00	03	JSR	\$ 0300
0345-	4C	16	03	JMP	\$ 0316
0348-	D1	1E	D7 46 C5 6E D2 A0		
0350-	D4	FF	C0 AC 98 90 80 72		
0358-	66	60	00		

程序说明:

0300: 清洗屏幕;

0316: 接受操作者按键;

0319: 按键ASCII码小于等于AF吗?

031B: 是, 则转0316重新接受;

031D: 按键ASCII码小于等于B8吗?

031F: 是, 则转033A;

0321: 否则, 把00送入寄存器Y;

0323: A与(0348 + Y)相比较;

0326: 相等, 则转0331;

0328: Y加上1;

032A: Y等于0A吗?

032C: 不等, 转0323;

032E: 转0316;

0331: Y加上1;

0332: 取节拍值 (0348 + Y) 到A中;
 0335: 存入0001;
 0337: 转0316;
 033A: 去掉A中数的高位(和0F相“与”);
 033C: 将A中的数送入X;
 033D: 取音阶值 (0352 + X) 到A中;
 0340: 存入0000;
 0342: 调用发音子程序;
 0345: 转0316继续执行;
 0348—0351: 控制键及节拍值;
 0352—035A: 音阶值。

其执行框图如图5.3。

从图中可以看出, 弹奏键盘使喇叭发音主要是0、1、2...、8九个键, 它们分别对应第一节中列出的九个音阶。另外, 为了能够弹出不同的节拍, 我们在程序中设置了Q、W、E、R、T五个控制键, 它们分别对应十六分音符、八分音符、八分音符加附点、四分音符和四分音符加附点。因此, 按入一个控制键, 则其后按入的音符(0—8)将以这个控制键对应的节拍发音, 直至按入另一个控制键为止。

程序5.3也是根据输入信息设置音阶和节拍参数, 并调用发音子程序的例子。其中用到1个新的指令, 即:

D9 48 03 CMP \$0348, Y: 寄存器Y绝对寻址指令, 比较累加器A与绝对地址加Y所指单元中的数。

在地址0321上有一个循环比较的步骤, 使用循环比较目的是节约内存。解释一下它的执行过程, 就是:

当Y = 0时, A与(0348)比较;

当Y = 2时, A与(034A)比较;

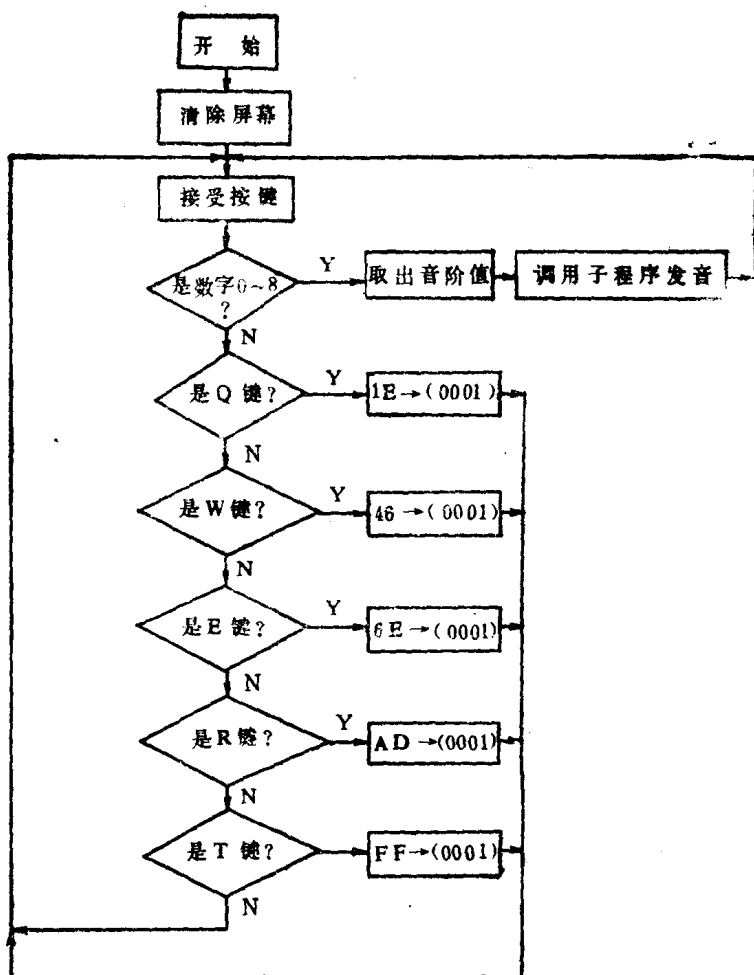


图5.3

当 $Y = 4$ 时, A 与 (034C) 比较;

.....

如果比较相等, 则使 $Y \leftarrow Y + 1$, 把其后的一个数 (节拍值) 取入累加器 A, 并送入 0001。

3. 高分辨绘图

6502除了发音功能外, 还具有较强的绘图功能。绘图屏幕有低分辨和高分辨两种, 低分辨为 $48 \times 40 = 1920$ 个像素, 高分辨为 $192 \times 280 = 53760$ 个像素, 绘图方式有画点画线、造型表和位元表等多种。

关于6502的图形功能以及绘图程序的编制我们已在《中华学习机图形管理》一书中作了较详细的介绍。本书不再重复这些内容, 只提供一个高分辨作图的定位程序, 供读者在编写应用程序时参考。

我们知道, 6502的图形页面是映象式的, 即内存中专门有一段区域存放当前的图形资料。但是, 这个映象区域相对于图象页而言是跳跃式的。请看下面高分辨第一页的例子:

行号	对应内存区
0	\$ 2000- \$ 2027
1	\$ 2400- \$ 2427
2	\$ 2800- \$ 2827
3	\$ 2C00- \$ 2C27
4	\$ 3000- \$ 3027
5	\$ 3400- \$ 3427
6	\$ 3800- \$ 3827
7	\$ 3C00- \$ 3C27
8	\$ 2080- \$ 20A7
9	\$ 2480- \$ 24A7
:	:

显然, 它在内存中并不是按第 0、第 1、第 2 …… 的行

号排列的，而是第64行紧接在第0行后面，第65行紧接在第1行后面……（这就是我们用BLOAD命令调用图象时，其显示过程不是自上而下连续进行，而是跳跃显示的原因）。

6502的这种映象存贮给用户带来不便。例如我们要用给映象区赋值的方法在第5行第8列上画一点，就要通过繁琐的转换和查表找出这一点的对应地址。为了解决这个问题，我们编写了一个机器语言程序来实现坐标值和地址间的自动转换，只要在执行前给出行号就可以得出该行在映象区的首地址（同一行的象素在映象区是连续的，因此只需要算出第0列的地址）。

分析一下高分辨映象区的结构，可以看出：整个192行扫描线可分为三个区域，即第0—63、第64—127、第128—191，相邻区域中的对应地址相差40个单元；每一区域可分为八组，每一组中的对应地址相差128个单元；每一组有八条扫描线，相邻两条扫描线的对应地址相差1024个单元。

因此，给出一个行号H（0—191），它所在的区就是 $X = \text{INT}(H/64)$ ，所在的组就是 $Y = \text{INT}((H - 64 * X)/8)$ ，所在的组内号就是 $Z = H - 64 * X - 8 * Y$ 。而映象地址则是：

$$W = 8192 + 40 * X + 128 * Y + 1024 * Z \quad (\text{第一页})$$

$$\text{或 } W = 16384 + 40 * X + 128 * Y + 1024 * Z \quad (\text{第二页})$$

综合起来，并写成16进制的形式，就是：

$$W = 2000 * P + 28 * X + 80 * Y + 400 * Z$$

其中P是页面号。例程序5.4就是根据这个算法求算地址值的。

例程序5.4

```
0300-    20    58    FC        JSR    $FC58
```


0303-	A9	00	LDA	# \$ 00
0305-	85	02	STA	\$ 02
0307-	A9	20	LDA	# \$ 20
0309-	85	03	STA	\$ 03
030B-	C6	00	DEC	\$ 00
030D-	F0	02	BEQ	\$ 0311
030F-	06	03	ASL	\$ 03
0311-	A5	01	LDA	\$ 01
0313-	4A		LSR	
0314-	4A		LSR	
0315-	4A		LSR	
0316-	4A		LSR	
0317-	4A		LSR	
0318-	4A		LSR	
0319-	85	04	STA	\$ 04
031B-	0A		ASL	
031C-	0A		ASL	
031D-	0A		ASL	
031E-	0A		ASL	
031F-	0A		ASL	
0320-	85	02	STA	\$ 02
0322-	A5	04	LDA	\$ 04
0324-	0A		ASL	
0325-	0A		ASL	
0326-	0A		ASL	
0327-	18		CLC	
0328-	65	02	ADC	\$ 02
032A-	85	02	STA	\$ 02
032C-	0A		ASL	
032D-	0A		ASL	

032E-	0A		ASL	
032F-	85	05	STA	\$ 05
0331-	A5	01	LDA	\$ 01
0333-	38		SEC	
0334-	E5	05	SBC	\$ 05
0336-	85	06	STA	\$ 06
0338-	4A		LSR	
0339-	4A		LSR	
033A-	4A		LSR	
033B-	85	05	STA	\$ 05
033D-	AA		TAX	
033E-	E0	00	CPX	# \$ 00
0340-	F0	16	BEQ	\$ 0358
0342-	E0	01	CPX	# \$ 01
0344-	F0	07	BEQ	\$ 034D
0346-	CA		DEX	
0347-	CA		DEX	
0348-	E6	03	INC	\$ 03
034A-	4C	3E 03	IMP	\$ 033E
034D-	A9	80	LDA	# \$ 80
034F-	18		CLC	
0350-	65	02	ADC	\$ 02
0352-	90	02	BCC	\$ 0356
0354-	E6	03	INC	\$ 03
0356-	85	02	STA	\$ 02
0358-	A5	05	LDA	\$ 05
035A-	0A		ASL	
035B-	0A		ASL	
035C-	0A		ASL	
035D-	85	04	STA	\$ 04

035F-	A5	06	LDA	\$ 06
0361-	38		SEC	
0362-	E5	04	SBC	\$ 04
0364-	A8		TAY	
0365-	F0	0A	BEQ	\$ 0371
0367-	A9	04	LDA	# \$ 04
0369-	18		CLC	
036A-	65	03	ADC	\$ 03
036C-	85	03	STA	\$ 03
036E-	88		DEY	
036F-	D0	F6	BNE	\$ 0367
0371-	A5	03	LDA	\$ 03
0373-	20	DA FD	JSR	\$ FDDA
0376-	A5	02	LDA	\$ 02
0378-	20	DA FD	JSR	\$ FDDA
037B-	60		RTS	

程序中用到 5 个新的指令：

(1) 06 03 ASL \$ 03; 零页寻址指令，把零页单元中数的各个数元左移一位，右边补 0。

(2) A 5 01 LDA \$ 01; 零页寻址指令，把零页单元 0001 中的数送入累加器 A。

(3) 4 A LSR; 隐含寻址指令，把累加器 A 中数的各个数元向右移动一位，左边补 0。

LSR 指令和前述的 ASL、ROL 指令以及后面要介绍的 ROR 指令都是移位指令：

ASL 指令把位元往左移动一位，最右边补 0；

ROL 指令把位元往左转动一位，最右边用 C 标志补入；

LSR 指令把位元往右移动一位，最左边用 0 补；

ROR指令把位元往右转动一位，最左边用C标志补入。

例：

11010011	11010011	11010011
ASL	ROL	LSR
<u>10100110</u>	<u>10100110</u>	<u>01101001</u>
ASL	ROL	LSR
<u>01001100</u>	<u>01001101</u>	<u>00110100</u>
11010011		
ROR		
<u>01101001</u>		
ROR		
<u>10110100</u>		

(4) E 5 05 SBC \$ 05: 零页寻址指令,把累加器A中的数减去零页单元0005中的数,再减去代位标志,结果仍保留在A中。

(5) A 8 TAY: 隐含寻址指令,把累加器A中的数传入寄存器Y中。

程序5.4的执行框图如图5.4。

在执行该程序之前,必须把图象页的页面(01或02)送入地址0000,行数(00—BF)送入地址0001。如:

* 0000:01 A0↙

* 300G↙

2250

* 0000:02 8A↙

* 300G↙

48D0

*

即高分辨作图第1页第A0行的映象区首地址是2250(16

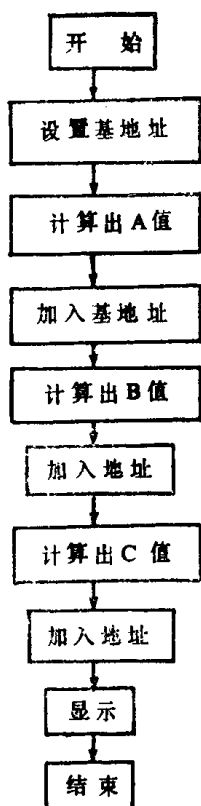


图5.4

进制)，高分辨作图第2页第8A行的映象区首地址是48D0。

在例程序5.4的乘法运算中，我们使用了乘数和被乘数之一为256的约数或倍数的技巧。如10进制的128是256的一半，因此，每两个Y可以凑成一个256，即地址低位不变，高位加1；又如1024是256的四倍，因此一个Z有4个256，即地址低位不变，高位加4。至于10进制的40乘以X我们则把它拆成32乘以X加上8乘以X，32乘以X即做五次X乘以2，8乘以X即做三次X乘以2。而X乘以2可以用ASL

指令来完成，这样就使得乘法得以简化。

4. 指令讨论

本章我们介绍了10个新的指令，现综合讨论如下。

(1) ASL: 使用在零页寻址方式下，把操作码后零页单元中数的各个位元向左移动一位，最右边补0。操作码是06。

例: 06 02 ASL \$02

(2) CMP: 使用在寄存器Y绝对寻址方式下，比较累加器A与绝对地址加上Y所指单元中的数。操作码是D9。

例: D9 A3 03 CMP \$03A3, Y

即比较A与(03A3 + Y)中的数。

(3) DEC: 使用在零页寻址方式下，把操作码后零页地址单元中的数减去1。操作码是C6。

例: C6 01 DEC \$01

4. INC: 使用在绝对寻址方式下，把操作码后地址单元中的数加上1。操作码是EE。

例: EE 10 03 INC \$0310

若0310中原来为FF(一个字节能表示的最大数)，则执行INC后变为00，且零值标志Z被置为1。

(5) LDA: 使用在零页寻址方式下，把操作码后零页地址单元中的数送入累加器A。操作码是A5。

例: A5 05 LDA \$05

(6) LDX: 使用在零页寻址方式下，把操作码后零页地址单元中的数送入寄存器X。操作码是A6。

例: A6 03 LDX \$03

(7) LSR: 使用在隐含寻址方式下，把累加器A中数

的各个位元向右移动一位。操作码是4A。

例：4A LSR

该指令使位元右移一位后，左边一位用 0 补上，原来最右边的一位进入代位标志 C。

(8) SBC：使用在零页寻址方式下，把累加器 A 中的数减去零页单元中的数，再减去代位标志，结果仍存在 A 中。操作码是 E5。

例：E5 E0 SBC \$E0

(9) STX：使用在零页寻址方式下，把寄存器 X 中的数存入操作码后的零页地址单元中。操作码是 86。

例：86 01 STX \$01

(10) TAY：使用在隐含寻址方式下，把累加器 A 中的数传入寄存器 Y。操作码是 A8。

例：A8 TAY

六、机器语言编程技巧

近年来在国内得到广泛重视和应用的微型计算机，无论在科学计算、信息处理，还是在自动控制、辅助设计、人工智能等各个方面都具有在此之前使用的计算工具所无法比拟的强大功能，但这些功能的实现依赖于程序设计者编制的各种各样的应用程序。因此，程序设计的好坏是发挥计算机效益和功能的基本的重要的条件。

一般来说，一个好的应用程序除了能够较好地完成预先规划好的各种功能外，还必须具有以下几个特点：

(1) 容易掌握和使用。一个应用程序可能是程序设计人员自己使用，也可能提供给别人使用，这些人甚至包括非计算机专业或者不大熟悉计算机的各种管理人员。这就要求程序要易于掌握和使用。首先人机界面要清晰，需要输入什么资料，得到什么结果，要使人一目了然。其次，数据和程序的存放、结构要经过合理布局，既要考虑其自身，又要考虑系统的内存结构，避免因此而出现种种问题，影响使用效果。

(2) 程序编写精炼，执行速度快。这个特点和程序要易于维护之间存在一定的矛盾。我们认为，处理这个矛盾要根据程序使用的具体情况。而最基本的要求是程序编写不能拖泥带水，执行速度要能够满足使用的要求。

(3) 程序要易于维护。易于维护即指程序要容易阅读和修改。通常这个特点要以一定的内存和降低执行速度为代

价，因此，也应根据具体情况考虑。例如，在内存足够、执行速度能达到使用要求的情况下，就不必为了节省一个或若干个单元而使程序变得难以理解。

（4）具有一定的可扩充性。一个程序在投入使用以后功能不可能是尽善尽美的，还需要在维护过程中不断加以完善，这就要求它具有可扩充性。在机器语言程序中，这个特点主要体现在内存空间和子程序的巧妙安排上。

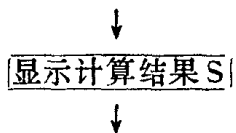
总之，要能够编写出好的应用程序，就要靠多设计、多编写，在实践中不断总结经验、积累经验。本章我们将就6502机器语言编程中的一些基本手段和技巧作一些介绍。

1. 程序框图

在前面几章的内容中，我们已经举过许多程序框图的例子。从中可以看到，机器语言的程序框图和 BASIC 程序框图一样不但能使编程过程清晰明了，避免编程的盲目性和走弯路，还可以很好地帮助他人阅读程序，了解程序的各种功能，掌握使用方法。

一般来说，根据不同的需要程序框图可以有粗细之分。对于一些比较长的应用程序来说，可以事先设计出粗框图，标明功能模块的设置、资料的输入输出和算法要点等内容；而对于一些有较多条件、非条件转移、子程序调用等复杂流程的程序块来说，则有必要事先设计出较为详细的框图，以具体指导程序的编写；保持清晰的思路，避免转移、调用中容易出现混乱。

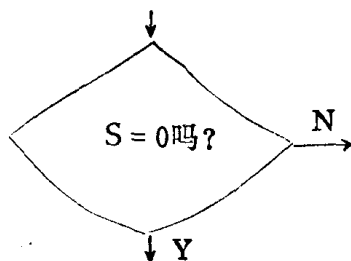
程序框图主要有三种图块组成：一种是长方形框，表示要完成某种功能或者操作。如：



它有一个入口和一个流出口，是程序执行中的一个过程。

第二种是菱形框，表示某种条件的判断。

如：



它一般有一个入口和二个流出口。当条件满足时，从一个出口去执行后面的操作；条件不满足时，从另一个出口去执行后面的操作。

第三种是流线，即上、下、左、右等方向的箭头，表示程序执行从一个地方完成后再走向另一个地方。

除此之外，在框图比较复杂而难以连线时，可以在一个地方画上一个圆圈，里面写上一个字母，再在另一个地方同样画一个圆圈，里面写上同一个字母，以表示它们之间相连接。

以一个大炮打飞机的游戏程序为例，我们可以画出它的框图如图6.1。

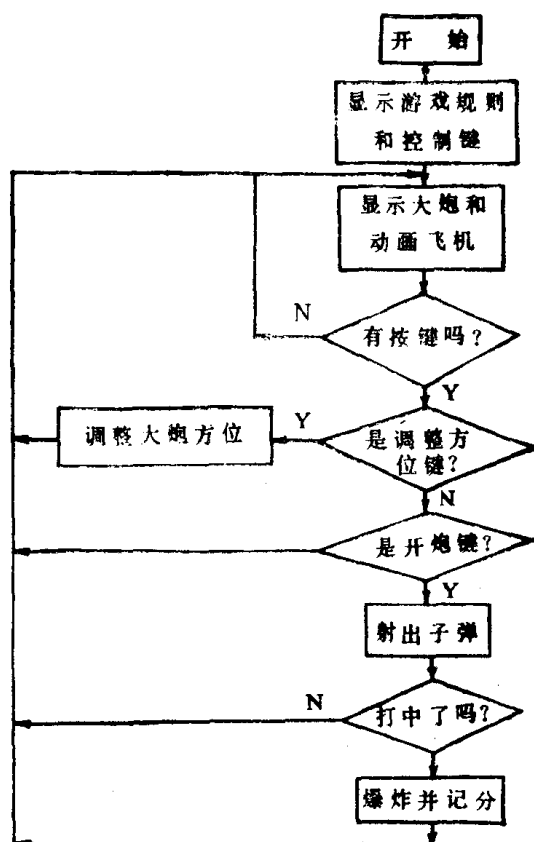


图6.1

框图内的每一步仍包含有较多的步骤，还可以画出更细的框图。如射出子弹一步就可以再详细一步，以直接指导编程。如图6.2。

上面两个框图都是叙述性的，每一个框中都用文字来说明要完成的功能或要进行的操作。在这个基础上编程，还必须另外规划内存的使用、资料的存放等。因此，根据需要，

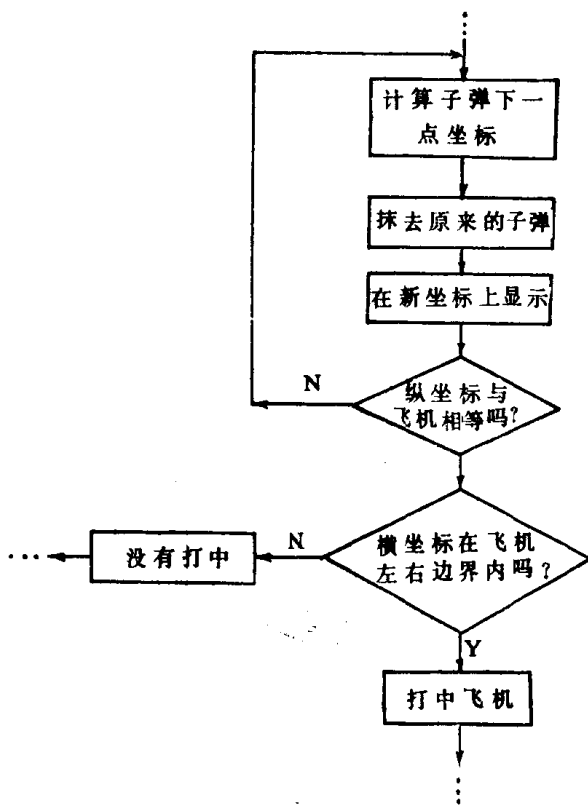


图6.2

我们还可以绘制用符号表示的框图，它更接近于程序编写的过程。

例如要编写一个 $1 + 2 + 3 + \dots + FF$ 的程序，可以先画出框图6.3。

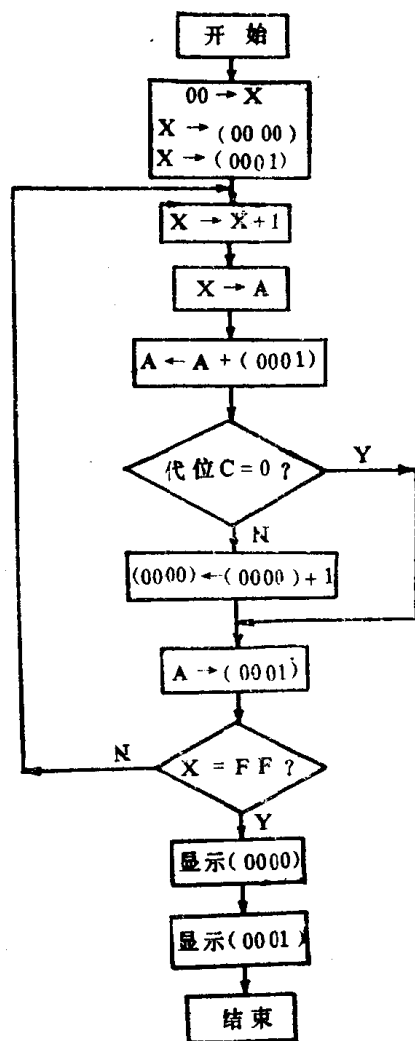


图6.3

按照这个框图，我们很容易就可以编出相应的程序来。

例程序6.1

0300-	A2 00	LDX	# \$ 00
0302-	86 00	STX	\$ 00
0304-	86 01	STX	\$ 01
0306-	E8	INX	
0307-	8A	TXA	
0308-	18	CLC	
0309-	65 01	ADC	\$ 01
030B-	90 02	BCC	\$ 030F
030D-	E6 00	INC	\$ 00
030F-	85 01	STA	\$ 01
0311-	E0 FF	CPX	# \$ FF
0313-	D0 F1	BNE	\$ 0306
0315-	A5 00	LDA	\$ 00
0317-	20 DA FD	JSR	\$ FDDA
031A-	A5 01	LDA	\$ 01
031C-	20 DA FD	JSR	\$ FDDA
031F-	60	RTS	

*0300G✓

7F80

*

结果表明，和等于7F80。

2. 转移执行指令

如同 BASIC 语言的转向语言一样，6502 机器语言中也有转移执行指令，并且也包括条件转移指令和无条件转移指令两种。不同的是，BASIC 中的转向语句指明转向执行的行号，因此在解释执行过程中，必须通过对程序区的扫描找到该行号所在的内存地址，从而把控制交给所转向的语句，这样就在一定程度上浪费了时间，降低了执行速度；而机器

语言中的转移指令直接给出了转移执行的地址，只要把这个地址送入指令计数器，就完成了转移（如果没有转移，则下一条指令的存放地址同样要送入指令计数器），因此不会影响执行速度。就这个意义上说，我们在编写机器语言程序时可以放心地去使用各种转移指令。

（1）无条件转移指令 无条件转移指令共有五条，即：JMP、JSR、RTS、RTI和BRK。

① JMP指令：JMP指令有绝对寻址和间接寻址两种方式，它们的操作码分别是4C和6C。绝对寻址方式下的JMP指令在操作码后给出转向执行的绝对地址。当程序执行遇到操作码4C后，就把其后两个字节的绝对地址送入指令计数器PC，并转向该地址执行。

如果在0380地址上使执行转移到0320，只需使用指令：

0380—4C 20 03 JMP \$0320

间接寻址方式下的JMP指令在操作码6C后也存有一个二字节绝对地址，但这个地址并不是真正要转向执行的地址，而是存放转向执行地址的内存单元地址。如指令：

6C 50 08 JMP(\$0850)

执行这一指令时，并不是要转向0850，而是要从0850、0851二个单元中取出要转移执行的地址值，再转向这个地址。若这时0850中存有FA，0851中存有08，则应该转移到08FA去执行。因此，这一间接寻址指令的操作码后存放的是间接地址。

② BRK指令：BRK指令是一个强迫中断指令，它的操作码是00，其功能类似于BASIC中的STOP语句。当程序执行遇到BRK指令时，就自动转向执行系统的中断处理程序，并在保护程序现场的情况下退出执行状态。

因此, BRK 指令常常被用于对程序的调试, 即在程序被强迫中断后查询各寄存器和有关单元的当前值, 以此判断程序执行是否正确。查询寄存器和内存单元的方法已在前面的监控命令中作过介绍。

执行BRK指令时, 标志寄存器中的中断标志B将被设置为1。

③ RTI指令: RTI指令是一个中断返回指令, 它的操作码是40。这个指令被用在中断处理程序中, 一旦执行遇到BRK中断时, 便转去执行中断处理程序, 而遇到RTI指令后, 又返回到原中断处的下一条指令继续执行。因此, BRK和RTI指令也类似于一对子程序调用和返回的指令。

(2) 条件转移指令 从本书后面的6502机器语言指令表中可以看到, 有很多指令在执行过程中对标志寄存器的一个或若干个标志会产生影响, 这种影响是依据指令的执行结果而重新设置标志位的。例如, DEX 指令使X中的值减1, 如减1以后X中的值变为0, 则零值标志Z被置为1, 否则, Z标志被置为0。

条件转移指令就是针对各标志位的值(0或1)来设置条件的。这类指令共有八条, 两两构成一对来判断C、Z、V、N四个标志。

① BCC和BCS指令: 这一对指令对代位标志C设置条件:

BCC转移的条件是代位标志C等于0, 操作码是90;

BCS转移的条件是代位标志C等于1, 操作码是B0。

② BNE和BEQ指令: 这一对指令对零值标志Z设置条件:

BNE 转移的条件是零值标志Z等于0, 操作码是D0;

BEQ 转移的条件是零值标志Z等于1,操作码是F0。

③ BVC和BVS指令:这一对指令对溢出标志V设置条件:

BVC 转移的条件是溢出标志V等于0,操作码是50;

BVS 转移的条件是溢出标志V等于1,操作码是70。

④ BPL和BMI指令:这一对指令对负数标志N设置条件:

BPL 转移的条件是负数标志N等于0,操作码是10;

BMI 转移的条件是负数标志N等于1,操作码是30。

以上8个条件转移指令的长度都是二字节,其中第一字节是操作码,第二字节是相对地址。相对地址和转移步数(包括转移方向)之间的对应关系参照第二章的最后一节。

顺便指出,条件转移指令在编程中会经常使用到。有一些应用程序甚至用条件转移指令来代替无条件转移指令。

例如下面这段程序:

0300-	A2 01	LDX	# \$ 01
0302-	FE 56 08	INC	\$ 0856, X
0305-	E0 10	CPX	# \$ 10
0307-	F0 05	BEQ	\$ 030E
0309-	F6 E0	INC	\$ E0, X
030B	E8	INX	
030C	D0 F4	BNE	\$ 0302

在0305中对X的值进行判断,如果等于10则转出这一程序段。因此,在这一程序段中,X的值只能在01—10之间,或者说,030C中的当X不等于0时转向0302执行的条件是永远成立的。当程序执行到030C时必然要转向0302。这样一种转向一般来说总是用JMP指令,而在这一程序段中却用了

条件转换来代替。

原因主要有两个方面：一是条件转移指令是二字节的，比无条件转移指令节约一个字节的内存（JMP指令是三字节长度的）；二是条件转移指令具有再定址的能力，比之无条件转移指令更灵活方便。如果由于重新规划或调整内存分配，需要把原来存放在A区的程序移到B区内执行，则程序中条件转移指令的再定址能力使之在移动之后仍可正常运行，而无条件转移指令由于直接给出了绝对地址，移动后必须人为再定址。

上面列出的程序段中有二个新的指令：

① FE 56 08 INC \$0856, X：这是一个寄存器X的绝对寻址指令，它把操作码FE后二字节绝对地址加上X所指内存单元中的数加上1。

② F6 E0 INC \$E0, X：寄存器X的零页寻址指令，把操作码F6后零页地址值加上X所指内存单元中的数加上1。

3. 子程序的使用

和BASIC语言程序一样，当一个机器语言程序中多次使用到同一个操作或处理过程时，为了节省内存，使程序结构更加清晰、精炼，并避免重复输入，可以把这个过程单独设置为具有相对独立功能的程序段，这个程序段就称为子程序。当程序中需要这个过程时，就用了JSR指令使子程序执行一次，称为子程序调用。子程序执行结束后，通过RTS指令返回到调用处的下一条指令。

调用和返回指令在前面已经作过介绍，它们是JSR和RTS。

20 ED FD JSR \$FDED

60 RTS

为了说明子程序的用法，下面举一个具体的例子。在这个例子中，要求程序完成 $47 + 5A$ 、 $29 + 8D$ 二组数的加法运算，并给出如下的显示格式：

***** $47 + 5A =$ *****

***** $29 + 8D =$ *****

例程序6.2

0300-	20	58	FC	JSR	\$FC58
0303-	20	4F	03	JSR	\$034F
0306-	A9	47		LDA	# \$47
0308-	18			CLC	
0309-	69	5A		ADC	# \$5A
030B-	48			PHA	
030C-	A9	47		LDA	# \$47
030E-	20	DA	FD	JSR	\$FDDA
0311-	A9	AB		LDA	# \$AB
0313-	20	ED	FD	JSR	\$FDED
0316-	A9	5A		LDA	# \$5A
0318-	20	DA	FD	JSR	\$FDDA
031B-	A9	BD		LDA	# \$BD
031D-	20	ED	FD	JSR	\$FDED
0320-	68			PLA	
0321-	20	DA	FD	JSR	\$FDDA
0324-	20	4F	03	JSR	\$034F
0327-	20	8E	FD	JSR	\$FD8E
032A-	20	4F	03	JSR	\$034F
032D-	A9	29		LDA	# \$29
032F-	18			CLC	

0330-	69 8D	ADC	# \$ 8D
0332-	48	PHA	
0333-	A9 29	LDA	# \$ 29
0335-	20 DA FD	JSR	\$ FDDA
0338-	A9 AB	LDA	# \$ AB
033A-	20 ED FD	JSR	\$ FDED
033D-	A9 8D	LDA	# \$ 8D
033F-	20 DA FD	JSR	\$ FDDA
0342-	A9 BD	LDA	# \$ BD
0344-	20 ED FD	JSR	\$ FDED
0347-	68	PLA	
0348-	20 DA FD	JSR	\$ FDDA
034B-	20 4F 03	JSR	\$ 034F
034E-	60	RTS	
034F-	A2 00	LDA	# \$ 00
0351-	A9 AA	LDA	# \$ AA
0353-	20 ED FD	JSR	\$ FDED
0356-	E8	INX	
0357-	E0 0A	CPX	# \$ 0A
0359-	D0- F8	BNE	\$ 0353
035B-	60	RTS	

程序说明:

0303: 调用子程序显示10个“*”，

0306—030A: 使47和5A相加;

030B: 和值推入堆栈;

030C—0310: 显示16进制数47;

0311—0315: 显示“+”号;

0316—031A: 显示16进制数5A;

031B—031F: 显示“=”号;

0320: 取出堆栈顶部的和值;

0321—0323: 显示和值;
 0324—0329: 显示10个“*”, 并回车;
 032A—034D: 对数值29和8D重复上述过程;
 034E: 返回
 034F: X赋初值00;
 0351: 把AA (“*”的ASCII码) 送入A;
 0353: 显示“*”;
 0356: X加上1;
 0357: X等于0A (已显示完10个“*”) 吗?
 0359: 不是, 转0353继续显示;
 035B: 否则, 子程序结束返回。

显然, 034F到035B为一子程序, 其功能是显示出10个“*”。主程序中四次调用过它, 避免了重复, 节约了内存。实际上, 这个程序还可以进一步精炼: 主程序中两组数的相加和显示过程相当相似, 如果把它们也设置成一个子程序, 就会显得更加清晰了。但两次相加和显示的数据不同, 如何设置呢? 这里, 可以通过先设置好数据, 传入子程序, 子程序取出数据作相应处理的办法来解决。

例程序6.3

0300-	20	58	FC	JSR	\$FC58
0303-	A0	47		LDY	# \$47
0305-	A9	5A		LDA	# \$5A
0307-	20	12	03	JSR	\$0312
030A-	A0	29		LDY	# \$29
030C-	A9	8D		LDA	# \$8D
030E-	20	12	03	JSR	\$0312
0311-	60			RTS	
0312-	48			PHA	
0313-	20	3B	03	JSR	\$033B

0316-	84	00		STY	\$ 00
0318-	68			PLA	
0319-	AA			TAX	
031A-	18			CLC	
031B-	65	00		ADC	\$ 00
031D-	48			PHA	
031E-	98			TYA	
031F-	20	DA	FD	JSR	\$ FDDA
0322-	A9	AB		LDA	#\$ AB
0324-	20	ED	FD	JSR	\$ FDED
0327-	8A			TXA	
0328-	20	DA	FD	JSR	\$ FDDA
032B-	A9	BD		LDA	#\$ BD
032D-	20	ED	FD	JSR	\$ FDED
0330-	68			PLA	
0331-	20	DA	FD	JSR	\$ FDDA
0334-	20	3B	03	JSR	\$ 033B
0337-	20	8E	FD	JSR	\$ FD8E
033A-	60			RTS	
033B-	A2	00		LDX	#\$ 00
033D-	A9	AA		LDA	#\$ AA
033F-	20	ED	FD	JSR	\$ FDED
0342-	E8			INX	
0343-	E0	0A		CPX	#\$ 0A
0345-	D0	F8		BNE	\$ 033F
0347-	60			RTS	

程序6.3占用0300—0347共71个内存单元，比程序6.2少占20个单元，而且主程序非常简洁，就是给A和Y两次赋值并调用0312的子程序。因此，子程序设置的好坏与程序的质

量有很大关系。

注意，在程序6.3中还有另一个子程序，其起始地址是033B，它在第一个子程序的0313和0334中被两次调用过。这种在一个子程序中又调用另一个子程序的情况称为子程序嵌套。

在使用子程序时必须注意以下几个问题：

① 子程序可以多层嵌套，即一个子程序调用另一个子程序，另一个子程序又再调用别的子程序，但嵌套的层数最多不得超过124层。

我们知道，当一个程序（包括子程序）调用一个子程序时，返回地址将被存入堆栈区。因此，每多一层子程序调用，就要在堆栈区中多占两个单元（一个绝对地址占两个内存单元），由于堆栈区长度的限制，子程序嵌套就不能超过一定的层次。

```
0300-    20 20 03      JSR    $ 0320
```

.....

```
0320-    20 40 03      JSR    $ 0340
```

.....

假设这个程序执行前堆栈区是空的（如图6.4(a)），则执行JSR \$0320后堆栈区变为(b)，再执行JSR \$0340后堆栈区变为(c)，即每次嵌一层子程序用掉两个单元。

在实际的程序中，由于其它一些指令对堆栈区的使用，因此，子程序嵌套的层数达不到124层。

② 一般来说，一个子程序必须由JSR指令调用，由RTS指令返回。之所以说一般情况，是因为在某些特殊情况下，一个子程序可以不经过JSR指令而使用，也可以不经RTS就返回。

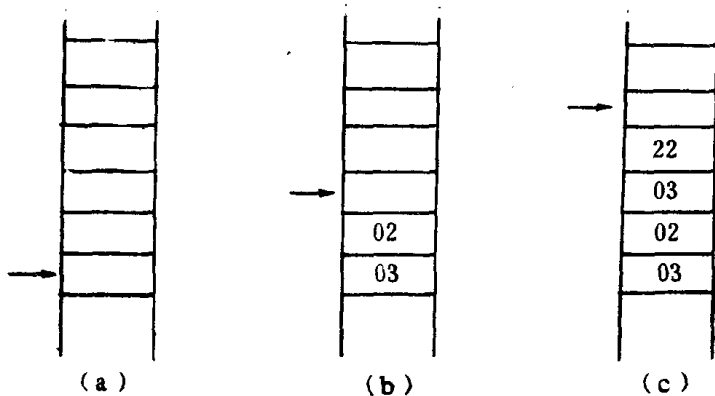


图6.4

例如下面这段程序：

```

0300-   A9  A3           LDA   # $ A3
0302-   20  20  03      JSR   $ 0320
0305-   20  8E  FD      JSR   $ FD8E
0308-   A9  AA           LDA   # $ AA
030A-   4C  20  03      JMP   $ 0320
030D-   .....
0320-   A2  00           LDX   # $ 00
0322-   20  ED  FD      JSR   $ FDED
0325-   E8              INX
0326-   E0  0A           CPX   # $ 0A
0328-   D0  F8           BNE   $ 0322
032A-   C9  AA           CMP   # $ AA
032C-   F0  01           BEQ   $ 032F
032E-   50              RTS
032F-   4C  0D  03      JMP   $ 030D

```

0320到0331为一个子程序，它的功能是连续显示十个累加器A中的字符（“#”或“*”），然后对累加器A中的ASCII码值进行判断：如果是AA（即字符“*”），则执行

032F中的JMP指令使程序无条件转移到030D;否则,在032E上通过RTS指令正常返回。

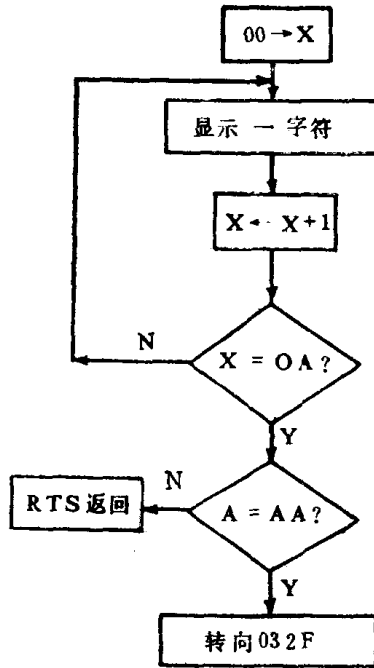


图6.5

在主程序的0302处JSR指令正常调用了这个子程序,因此首先显示出十个“*”,在对累加器值进行判别时,由于值为A3,不等于AA,因此执行032E中的RTS指令,正常返回到0305继续执行。第二次使用这个子程序是030A中的JMP指令,这是一个无条件转移指令,而不是子程序调用指令。转移后也先显示出十个“*”字符(ASCII码为AA),在判别累加器A的值时,因等于AA,所以转向032F执行无条件转移。这就是说,这次整个子程序的执行没有经过RTS

指令，而是把它当成主程序的一部分来使用。

第二种情况是不经过JSR而进入子程序，而由RTS返回，这在一般情况下是不允许的。请看下面的例子：

```
0350-   A9   03           LDA   # $ 03
0352-   48               PHA
0353-   A9   58           LDA   # $ 58
0355-   48               PHA
0356-   4C   5A   03      JMP    $ 035 A
0359-   00               BRK
035 A-   A9   30           LDA   # $ 30
035 C-   18               CLC
035 D-   75   01           ADC    $ 01,X
035 F-   60               RTS
```

其中75 01 ADC \$01, X是一个新的指令，它使用在寄存器X的零页寻址方式下，将累加器A中的数加上零页单元加X所指地址中的数，再加上代位标志C，结果存放在A中。即 $A \leftarrow A + (0001 + X) + C$ 。

在这一段程序中，035A到035F为一个实现加法的子程序。0356中用JMP指令转向该子程序，并将通过RTS指令准确返回到0359上。之所以能这样调用子程序，主要是因为JMP指令之前已对堆栈区作过了处理。我们知道，JSR指令的操作码是20，当程序执行遇到操作码20时，取出指令计数器PC中的值加上2送入堆栈区，再将操作码20后的绝对地址送入PC，转向执行被调用的子程序。在子程序执行完毕遇到RTS时，原来存入堆栈区的地址值被送入PC，并加上1，使之指向调用指令的下一条指令。

鉴于上述道理，这个程序段首先把03和58送入堆栈区

(如图6.6(a)变为(b))。在执行JMP指令时,地址值035A时,地址值035A被送入PC,进而转向子程序起始地址执行。完成加法运算遇到RTS时,堆栈区中的0358送入PC,然后PC加上1,使之指向0359,完成子程序的调用和返回动作。

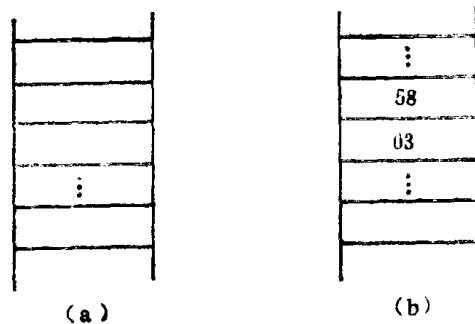


图6.6

因此,这个程序段实际上是在JMP之前设置堆栈区,模拟JSR指令来调用子程序的。

第三种情况是不通过RTS指令而强迫退出子程序。这种情况前面已经举过例子,它是通过在子程序中退栈恢复堆栈区来实现强迫退出的。

例:

0330-	20	34	03	JSR	\$ 0334
0333-	00			BRK	
0334-	E6	00		INC	\$ 00
0336-	F0	01		BEQ	\$ 0339
0338-	60			RTS	
0339-	68			PLA	
033A-	68			PLA	
033B-	4C	46	03	JMP	\$ 0346

0334开始的子程序使零页单元0000的值加1，然后判断是否等于00；如是则转向0339强迫退出子程序；否则正常返回。这里，PLA退栈的两个数就是在执行JSR指令时进栈的32和03。

③ JSR指令和RTS指令必须成对出现（除了上面列出的几种特殊例子）。这里所谓的成对并不是要求形式上要一个JSR对一个RTS，而是要求对一个固定的执行流程来说必须成对出现。

0385-	20	89	03	JSR	\$ 0389
0388-	00			BRK	
0389-	E8			INX	
038A-	F0	01		BEQ	\$ 038D
038C-	60			RTS	
038D-	C8			INY	
038E-	60			RTS	

这里一个JSR指令形式上有两个RTS指令对应，即子程序有两个出口。但从执行流程来看是一一对应的：0385处调用子程序0389，X加上1，如果不等于0，则由038C处返回；0385处调用子程序0389，X加上1，如果等于0，则Y加上1，由038E处返回。

4. 资料的存放和使用

(1) 资料的存放 在BASIC中有LET、INPUT或GET、DATA等三种资料获取和存放的方式，在机器语言中也有类似的这三种方式，它们是立即赋值、键盘输入和资料表。

① 立即赋值。前面已经介绍过LDA、LDX、LDY等

指令，这些指令在立即寻址方式下的功能就是将操作码后的数值存入相应的寄存器中，也就是立即赋值。它的数据直接存放在指令之中。这种方式比较适用于数据量小又不需要反复使用的情况。

例如，指定要完成60和8A的加法，就只要使用：

A9 60 LDA # \$60

18 CLC

69 8A ADC # \$8A

三个指令，显得比较简明扼要。如果把这种数据用键盘输入或者存在内存单元中，程序就显得不直观，而且占用内存更多。

② 键盘输入。键盘输入使用在数据具有随机性，必须由操作者在执行过程中输入的情况。例如，在游戏程序中一般有多个控制键决定物体的移动、移动方向、进攻及其它动作，是否按键，按哪一个控制键都是随机的而且必须由操作者通过键盘输入。

例程序6.4

0300- 20 58 FC JSR \$FC58

0303- 20 35 FD JSR \$FD35

0306- C9 8D CMP # \$8D

0308- F0 06 BEQ \$0310

030A- 20 ED FD JSR \$FDED

030D- 4C 03 03 JMP \$0303

0310- 60 RTS

例程序6.4每次允许操作者按入一个键，并判断是否为回车键，如是则退出执行，返回监控状态，否则显示这个字符，并重复上述过程。这里，要显示的字符是随机的，必须通过

键盘输入。

③资料表。这是一种在程序设计中使用较多的方法，它把程序执行时要用的一些资料集中存放在一段内存区域中，供随时取出使用。它比较适合于一些数据量较大或需要反复使用的情况，特别是一些重要的参数，存放在资料表中不但存取灵活，还便于调整修改，以保证程序的相对稳定。

例程序6.5

```
0360-   A2  00           LDX    # $ 00
0362-   BD  6 E   03     LDA    $ 036E,X
0365-   20  ED  FD     JSR     $ FDED
0368-   E8           INX
0369-   E0  1 F           CPX    # $ 1F
036B-   D0  F5           BNE     $ 0362
036D-   60           RTS
036E-   D0  D2
0370-   C5  D3  D3  A0  C1  CE  D9  A0
0378-   CB  C5  D9  A0  D4  CF  A0  C3
0380-   CF  CE  D4  C9  CE  D5  C5  AE
0388-   AE  AE  AE  AE  AE  00
```

例程序 6.5 中 036E 之后为一资料表，它是提示信息“PRESS ANY KEY TO CONTINUE……”中每一个字符的ASCII码值。把它单独设置成一个表的形式，保证了程序和资料间的相对独立性，也便于修改。

总之，选用哪一种获取和存放数据的方式要根据程序的具体情况而定，原则是要方便、精炼、易于修改。

(2) 资料的使用 这里所说的资料使用是指如何把资料表中的数据取出来，又如何把数据存入资料表。实际上，这就联系到了存取资料指令的寻址方式问题。以最常用的

LDA、STA指令为例，分别有八种和7种寻址方式。

LDA 指令的八种寻址方式是：

寻址方式	操作码	指令例子
立即寻址	A9	A9 01 LDA # \$01
零页寻址	A5	A5 E0 LDA \$E0
X零页寻址	B5	B5 E0 LDA \$E0,X
绝对寻址	AD	AD C0 03 LDA \$03C0
X绝对寻址	BD	BD 56 08 LDA \$0856,X
Y绝对寻址	B9	B9 56 08 LDA \$0856,Y
先变址间接寻址	A1	A1 2E LDA (\$2E,X)
后变址间接寻址	B1	B1 2E LDA (\$2E),Y

其中：

B5 E0 LDA \$E0, X即把00E0 + X所指内存单元中的数送入累加器A；

A1 2E LDA(\$2E,X)即把2E + X、2E + X + 01两个零页单元中存放的绝对地址单元中的数送入累加器A。如X为00，则间接地址为002E、002F，这两个单元中分别存放有操作数地址的低位和高位。

B1 2E LDA(\$2E), Y 即把2E、2F中存放的地址值 + Y所指的地址单元中的数送入累加器A。如2E中为30，2F中为08，Y为05，则该指令使0835 (0830 + 05) 中的数送入A。

STA指令的七种寻址方式是：

寻址方式	操作码	指令例子
零页寻址	85	85 05 STA \$05
X零页寻址	95	95 00 STA \$00,X
绝对寻址	8D	8DA5 10 STA \$10A5

X绝对寻址	9D	9D 30 20	STA \$2030,X
Y绝对寻址	99	99 30 03	STA \$0330,Y
先变址间接寻址	81	81 03	STA (\$03,X)
后变址间接寻址	91	91 05	STA (\$05,)Y

其中:

95 00 STA \$00,X把累加器A中的数存入00 + X 的单元内;

81 03 STA(\$03,X) 把累加器A中的数存入03 + X、03 + X + 01两个地址单元中所存放的绝对地址中。如X为01, 0004中存有FA, 0005中存有09, 则该指令即 $A \rightarrow (09FA)$ 。

5. 指令讨论

本章共介绍了15个新的指令, 现在加以综合讨论。

(1) ADC: 使用在寄存器X零页寻址方式下, 把累加器A中的数加上零页地址 + X所指单元中的数, 再加上代位标志C, 结果存放在A中, 操作码是75。

例: 75 E0 ADC \$E0,X

即 $A \leftarrow A + (00E0 + X) + C$ 。

(2) BCS: 使用在相对寻址方式下, 当代位标志C为1时发生转移。操作码是B0。

例: B0 03 BCS \$0308

(3) BPL: 使用在相对寻址方式下, 当负值标志为0时发生转移。操作码是10。

例: 10 03 BPL \$3012

(4) BRK: 使用在隐含寻址方式下, 强迫中止程序的执行。操作码是00。

例: 00 BRK

(5) BVC: 使用在相对寻址方式下, 当溢出标志V为0时发生转移。操作码是50。

例: 50 05 BVC \$1203

(6) BVS: 使用在相对寻址方式下, 当溢出标志V为1时发生转移。操作码是70。

例: 70 01 BVS \$0912

(7) INC: 使用在寄存器X零页寻址方式下, 将操作码后零页地址+X所指单元中的数加上1。操作码是F6。

例: F6 E0 INC \$E0,X

(8) INC: 使用在寄存器X绝对寻址方式下, 将操作码后绝对地址+X所指单元中的数加上1。操作码是FE。

例: FE 17 03 INC \$0317,X

(9) JMP: 使用在间接寻址方式下, 使程序执行向间接地址中的数指向的内存单元转移。操作码是6C。

例: 6C 01 00 JMP (\$0001)

如果0001中存有58, 0002中存有03, 则该指令使程序执行转向0358。

(10) LDA: 使用在寄存器X零页寻址方式下, 把零页地址加上X所指内存中的数送入累加器A。操作码是B5。

例: B5 2E LDA \$2E,X

(11) LDA: 使用在先变址的间接寻址方式下, 使操作码后的二字节地址加上X构成间接地址, 再把间接地址中存放的数所指内存单元中的数送入累加器A。操作码是A1。

例: A1 03 LDA (\$03,X)

(12) LDA: 使用在后变址的间接寻址方式下, 取出操作码后间接地址单元中的数加上Y构成绝对地址, 把绝对地址中的数送入累加器A。操作码是B1。

例: B1 30 LDA(\$30),Y

(13) RTI: 使用在隐含寻址方式下,使控制从中断处理程序中返回到原中断处。操作码是40。

例: 40 RTI

(14) STA: 使用在寄存器X零页寻址方式下,把累加器A中的数存入零页地址加上X所指的单元中。操作码是95。

例: 95 02 STA \$02,X

(15) STA: 使用在先变址的间接寻址方式下,使操作码后的零页地址加上X构成间接地址,把累加器A中的数存入间接地址中的数所指的单元中。操作码是81。

例: 81 00 STA(\$00,X)

七、BASIC语言与机器语言的配合使用

众所周知，BASIC语言的简便、易学与机器语言的繁琐、难懂，BASIC语言的执行速度慢、功能受到限制与机器语言的执行速度快、功能强大是两种语言之间互为补充的特点。如果在软件开发过程中能够集这两种语言的优点而避开它们的缺陷，将会取得很好的效果。在很多的软件中，为了达到这个目的，而把两种语言配合起来使用。

大致思路是：一般情况下采用 BASIC 语言编程，在需要快速执行的部分或BASIC难以实现的功能上则编制机器语言子程序，以便在BASIC中调用，达到设计的目的。

本章所要介绍的内容就是BASIC语言对机器语言程序的调用以及它们之间的数据传递等。

1. 机器语言的输入和调用

我们已经知道了如何在监控状态下把16进制表示的机器语言程序输入内存，现在介绍利用 BASIC 输入机器语言程序的方法。这里面实际上包含了两个问题：一个是用什么语句或命令输入；另一个是用什么方式输入。

第一个问题比较简单，这就是使用POKE语句。用这个语句可以把一个 0—255之间的10进制数值存入RAM区的任一地址单元中。它的格式是：

POKE地址值, 数值

用多个POKE语句或者把POKE语句置入循环体中, 就可以轻而易举地把机器语言程序输入内存。问题是POKE中的数值必须是10进制的, 而通常编写机器语言程序是用十六进制方式的。这就需要一个数制转换的过程。

例如有下面一段机器语言子程序:

```
0300-   A5  00      LDA   $ 00
0302-   18          CLC
0303-   65  01      ADC   $ 01
0305-   85  02      STA   $ 02
0307-   60          RTS
```

它使0000和0001两个单元中的数相加, 和存入地址 0002。为了使它能通过POKE语句输入内存, 先要把它转换成10进制数放在DATA语句中, 再POKE到内存中并调用。

```
1000   FOR   I=0 TO 7
1010   READ  A
1020   POKE  768+I,A
1030   NEXT I
1040   CALL  768
1100   DATA 165,0,24,101,1,133,2,96
```

这种人工转换的方法显然是不方便的。为此, 可以插入二行BASIC程序, 以实现自动转换, 改进后的程序如下:

```
1000   FOR   I=0 TO 7
1010   READ  A$
1020   X$=LEFT$(A$,1);Y$=RIGHT$(A$,1);X=ASC(X$);Y=ASC(Y$)
1030   A=16*(X-48)*(X<58)+
```

```

16 * (X - 55) * (X > 64) + (Y
- 48) * (Y < 58) + (Y - 55) *
(Y > 64)
1040 POKE 768 + I, A
1050 NEXT I
1060 CALL 768
1070 END
1100 DATA A5,00,18,65,01,85,02,60

```

这样，就只要把机器语言程序直接置于DATA语句中，程序就能自动将之转换成10进制方式置入内存。如果觉得这样仍不够方便，当然也可以直接在监控状态下输入机器语言子程序，再返回BASIC状态，这时BASIC程序中就只要用CALL语句直接调用就可以了。另一种方法是把子程序保存在软盘上，需要时用BLOAD命令调出并执行。

BASIC使用机器语言子程序必须注意以下几个问题：

(1) 子程序必须存放在适当的位置。首先不能存放在系统本身要使用的一些区域内，如第0页、第1页、第2页等，以免破坏系统程序或系统参数；其次，不能和BASIC程序发生冲突，包括不能存入BASIC程序区、变量区及其它要用的区域。否则，将破坏BASIC程序或影响它的正常运行。例如，BASIC程序中使用了高分辨作图页面，则存入子程序后将破坏已画好的图象。

一般来说，比较短小的子程序可以存放在0300—03FF内，这是一个安全区域；比较长的子程序可以存放在BASIC程序尾或变量表后。

(2) 子程序必须有进出口，这个进出口就是CALL语句调用的地址；必须有流出口，以便子程序执行完毕后能返回

BASIC继续执行。而且出口处的最后一个指令只能是RTS。当子程序执行遇到RTS指令时就返回到CALL的后续语句去继续执行。否则，无法返回BASIC，更不能继续执行。

```
0300-   AD 10 03   LDA   $0310
0303-   20 ED FD   JSR   $FDED
0306-   4C DA FD   JMP   $FDDA
```

这个子程序从形式上看没有出口，也没有RTS指令，但这种情况是允许的，它可以为BASIC程序调用。原因是它的最后一个指令JMP \$FDDA将控制交给了系统子程序FDDA。这是一个显示累加器A中16进制数的完整子程序，它的出口处为RTS指令，当遇到这个RTS指令时，同样能返回执行BASIC程序。

运行下面这个BASIC程序可以看到结果：

```
1000   POKE   784,193
1010   CALL   768
1020   PRINT
1030   PRINT  "OK"
1040   END
RUN↵
C1
OK
```

(3) 调用机器语言子程序前要设置好一些子程序中使用的变量；子程序在返回之前也要保存好运算结果。

2. 数据的相互传递

一般来说，一个BASIC语言程序要调用机器语言子程序，调用之前都要传递一些变量，供子程序使用。例如要显示的字符ASCII码、要运算的数据、要绘制的图表参数

等。

这些数据不能通过BASIC的变量形式来传递，而只能用POKE语句送入安全的内存区域。例如要显示出字符“*”，可以把它的ASCII码值170送入0300（即10进制768）：

POKE 768,170

然后，子程序又从这个单元中取出数据，调用系统功能显示出来。

例程序7.1

```
1000  REM    PROGRAM
1010  INPUT  'FIRST DATA,',A
1020  INPUT  'SECOND DATA,',B
1030  POKE  0,A,POKE  1,B
1040  CALL   768
1050  PRINT,PRINT
1060  GOTO  1010
```

例程序7.2

0300-	DB	CLD	
0301-	A5 00	LDA	\$ 00
0303-	C5 01	CMP	\$ 01
0305-	30 0C	BMI	\$ 0313
0307-	38	SEC	
0308-	E5 01	SBC	\$ 01
030A-	48	PHA	
030B-	A9 AD	LDA	# \$ AD
030D-	20 ED FD	JSR	\$ FDED
0310-	4C 1C 03	JMP	\$ 031C
0313-	18	CLC	
0314-	65 01	ADC	\$ 01

0316-	48		PHA
0317-	A9 AB		LDA # \$AB
0319-	20 ED FD		JSR \$FDED
031C-	68		PLA
031D-	20 DA FD		JSR \$FDDA
0320-	60		RTS

其中C5 01 CMP \$01 是一个新的指令，它使用在零页寻址方式下，把累加器A中的数与操作码后零页单元中的

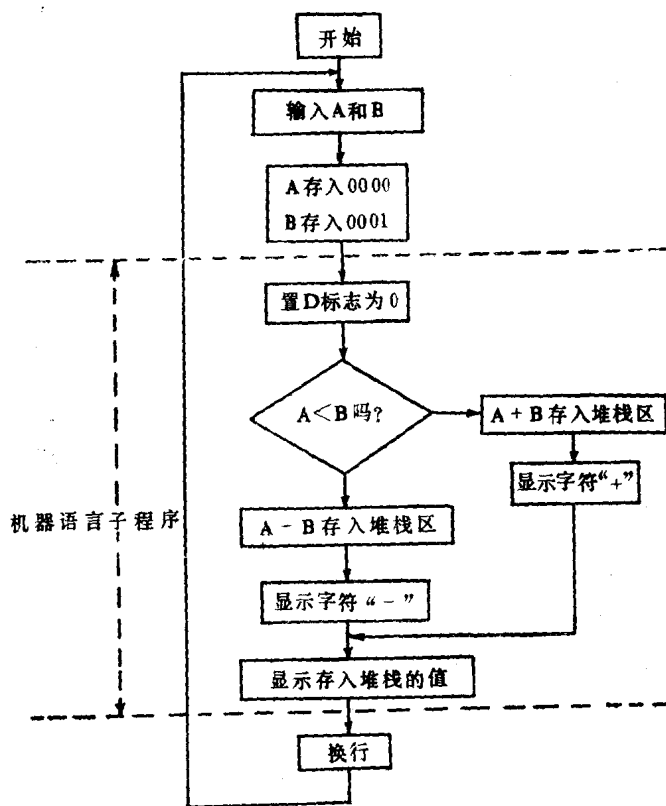


图7.1

数进行比较。

把程序7.1和程序7.2放在一起考虑，则其运行情况如图7.1。

可以看出，首先BASIC程序段要求输入A、B两个0—255之间的整数，并把它们分别存入零页地址单元0000和0001。然后，子程序设置了10进制运算方式，并判断A、B的大小：如果 $A > B$ 则计算 $A - B$ 的值，显示“-”及差值；否则计算 $A + B$ 的值，显示“+”及和值。返回BASIC后继续重复上述过程。

```
JRUN↵  
FIRST DATA:39↵  
SECOND DATA:21↵  
- 18  
FIRST DATA:18↵  
SECOND DATA:23↵  
+ 3B  
.....
```

从这个程序中可以看到，内存区域是这两种不同的程序语言都可以进行数据存取的地方，可以沟通它们之间的联系，达到用机器语言子程序实现特殊功能的目的。

同样地，当机器语言子程序的运算和处理结果需要返回到BASIC时，也必须通过内存单元作为中间媒介来传送。

例程序7.3

```
1000 REM PROGRAM  
1010 PRINT"ENTER DATA STRING..."  
1020 CALL 768  
1030 X$ = " "  
1040 FOR I=0 TO PEEK (788) - 1
```

```

1050   X$ = X$ + CHR$ (PEEK (789 +
      I))
1060   NEXT I
1070   PRINT X$
1080   END

```

例程序7.4

```

0300-   A2  00           LDX   #$00
0302-   20  35  FD      JSR   $FD35
0305-   C9  8D           CMP   #$8D
0307-   F0  07           BEQ   $0310
0309-   9D  15  03      STA   $0315,X
030C-   E8              INX
030D-   4C  02  03      JMP   $0302
0310-   8E  14  03      STX   $0314
0313-   60              RTS

```

例程序7.4中等待操作者按键，并将之存入0315 开始的地址中，直至按入回车键为止。把按入的字符个数存入0314，返回BASIC。在例程序7.3中，首先调用这个机器语言子程序，取得操作者输入的字符串，再通过PEEK语言取出该字符串显示出来。

这种通过内存单元进行数据传递的方法比较常用。当然，为了节省内存，加快运行速度，还可以在数据传递时加入一些技巧。例如，机器语言子程序接收或处理后的结果可以按规定设计成变量直接置入BASIC程序的变量表中。这样，子程序结束返回BASIC时，不需要经过任何处理就可以使用这个变量。

3. 调用子程序的另一种方法

CALL是BASIC调用机器语言子程序的常用语句，但这个语句本身不能进行参数的传递，必须在使用之前将数据存入地址单元，并在子程序中加入相应的取数指令来完成传递。为此，BASIC中还设置了另一个调用机器语言子程序的函数，即USR函数。

它的格式是USR (X)，其中X可以是常数、变量或表达式，它就是传递到子程序中的基本数据。USR函数的返回值就是机器语言子程序对X进行处理或运算的结果。

USR函数的执行步骤如下：

(1) 取出变量X的值，送入主浮点累加器（即9D—A3共7个连续单元）；

(2) 转入执行0A—0C的机器语言指令；

(3) 从主浮点累加器中取出数值，并作为USR函数的返回值。

因此，事先在0A—0C中设置一条转向机器语言子程序的指令，再使用USR函数就能够完成子程序特定的数据处理

表7.1

函数名	地 址	函数名	地 址
ABS	EBAF	RND	EFAE
ATN	F09E	SGN	EB90
COS	EFEA	SIN	EFF1
EXP	EF09	SQR	EE8D
INT	EC23	TAN	F034
LOG	E941		

功能。

表7.1是系统ROM区中一些BASIC运算函数解释 执行程序的入口地址。

使用USR函数可以很方便地调用这些系统子程序，完成相应的数据运算功能。

例程序7.5

```
1000  DIM A(10)
1010  POKE 10,76,POKE 11,65,POKE
      12,233
1020  FOR I=1 TO 10
1030  A(I) = USR(I)
1040  PRINT I,A(I)
1050  NEXT I
1060  END
```

]RUN

1	0
2	.693147181
3	1.09861229
4	1.38629436
5	1.60943791
6	1.79175947
7	1.94591015
8	2.07944154
9	2.19722458
10	2.30258509

例程序7.5中,1000行定义了一维数组A;1010行在地址0A—0C(即10进制10—12)中存入转移指令4C 41 E9,即把4C(76)存入10,41(65)存入11,E9(233)存入12。

以便使用USR函数时能转向入口地址E941进行对数运算，1030行对1—10的每一个I进行LOG (I) 运算，并将结果存入A (I)；1040行显示I及LOG (I) 的值。从运算结果可以看到，这种处理是系统允许的。实际上，这里的USR (I) 等同于LOG (I)。

其它函数功能的调用与此相类似。

不仅如此，知道了USR函数的功能及执行原理后，还可以自己设计各种功能子程序，并在BASIC中用USR调用，以实现多种功能。

除了CALL语句和USR函数可以调用机器语言子程序外，BASIC中的&命令也可以调用机器语言子程序，其功能类似于USR函数。这个命令的处理入口是03F5，即在03F5中设置一个JMP指令，就可以在BASIC中用&指令调用。关于&命令的详细功能和用法，本书不再赘述，请参考有关BASIC书籍。

4. 一个程序实例

本节介绍一个BASIC程序调用机器语言子程序的实例。

这个程序的功能是对100个0—99的10进制数进行快速排序：

每输入一个数，就调用一次子程序进行排序，并将排序结果从大到小在屏幕上立即显示出来。程序框图如图7.2。

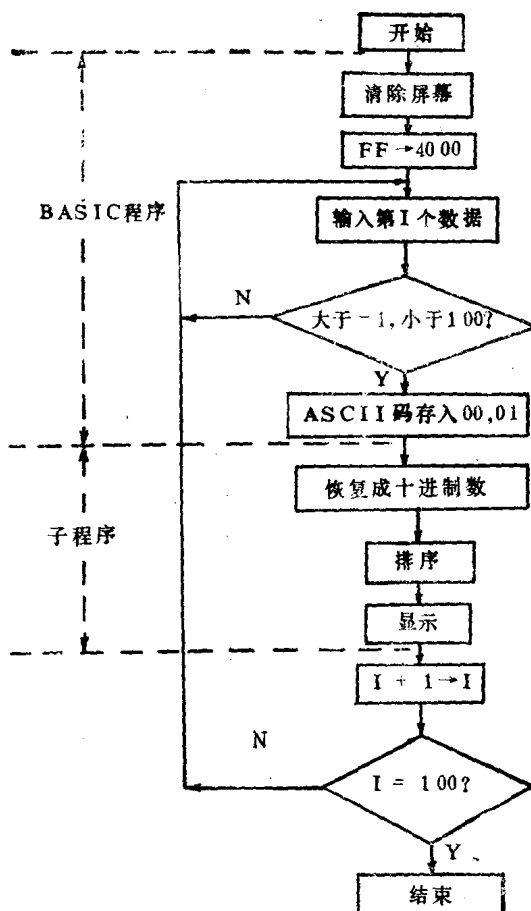


图7.2

例程序7.6

```

1000 HOME,POKE 16384,255,FOR I
      =1 TO 100
1010 VTAB 3, HTAB 1,PRINT "NO.",

```

```

      I, ' ', ' ',
1020  INPUT " ", A$, IF LEN(A$) =
      1 THEN A$ = " 0 " + A$
1030  X = ASC(A$), Y = ASC(MID$
      (A$, 2))
1040  POKE 0, X, POKE 1, Y, POKE 2, I
      , CALL 768
1050  NEXT I
1060  END

```

例程序7.7

0300-	A5	00	LDA	\$ 00
0302-	29	0F	AND	# \$ 0F
0304-	0A		ASL	
0305-	0A		ASL	
0306-	0A		ASL	
0307-	0A		ASL	
0308-	85	00	STA	\$ 00
030A-	A5	01	LDA	\$ 01
030C-	29	0F	AND	# \$ 0F
030E-	F8		SED	
030F-	18		CLC	
0310-	65	00	ADC	\$ 00
0312-	85	01	STA	\$ 01
0314-	A4	02	LDY	\$ 02
0316-	88		DEY	
0317-	D9	00 40	CMP	\$ 4000, Y
031A-	B0	FA	BCS	\$ 0316
031C-	84	00	STY	\$ 00
031E-	A4	02	LDY	\$ 02

0320-	88			DEY	
0321-	C4	00		CPY	\$ 00
0323-	F0	0B		BEQ	\$ 0330
0325-	B9	00	40	LDA	\$ 4000,Y
0328-	C8			INY	
0329-	99	00	40	STA	\$ 4000,Y
032C-	88			DEY	
032D-	4C	20	03	JMP	\$ 0320
0330-	C8			INY	
0331-	A5	01		LDA	\$ 01
0333-	99	00	40	STA	\$ 4000,Y
0336-	A0	00		LDY	# \$ 00
0338-	B9	01	40	LDA	\$ 4001,Y
033B-	20	DA	FD	JSR	\$ FDDA
033E-	A9	A0		LDA	# \$ A0
0340-	20	ED	FD	JSR	\$ FDED
0343-	C8			INY	
0344-	C4	02		CPY	\$ 02
0346-	D0	F0		BNE	\$ 0338
0348-	D8			CLD	
0349-	60			RTS	

机器语言子程序中用到 1 个新的指令：

A4 02 LDY \$02：零页寻址指令，把操作码后零页地址单元中的数送入寄存器Y。

在这个程序实例中，我们希望输入 0—99 的 10 进制数，排序以后显示出来的也是 10 进制数。因此，必须在存入地址单元前对输入的数进行变换。

例如输入一个 10 进制数 76，直接用 POKE 0,76 命令，则存入 0000 的将是 16 进制数 4C ($4 \times 16 + 12 = 76$)。这样，

在排序以后用FDDA系统子程序显示出来的也是4C，而不是76。为了解决这个问题，程序中把76当成一个字符串，并分成“7”和“6”两个字符，将其ASCII码值55、54分别存入0000和0001单元。这样，机器语言子程序调出的值就是16进制数37、36。将37各数元左移4次，变成70：

2 进制	16进制
00110111	37
ASL	ASL
01101110	6E
ASL	ASL
11011100	DC
ASL	ASL
10111000	B8
ASL	ASL
01110000	70

再将36与0F相“与”，得到06：

2 进制	16进制
00110110	36
AND 00001111	AND 0F
00000110	06

再将70与06进行10进制加法即得原始输入数据76。将76进行排序后：调用FDDA系统子程序显示出来。

将例程序7.6、7.7分别输入内存，可以看到运行结果。

]RUN↵

No.1:23↵

23

No.2:63↵

63 23

No.3:47↵

63 47 23

No.4:79✓

79 63 47 23

No.5:37✓

79 63 47 37 23

No.6:56✓

79 63 56 47 37 23

No.7:29✓

79 63 56 47 37 29 23

No.8:81✓

81 79 63 56 47 37 29 23

.....

在本章中，只用到一个新的指令：即：

LDY：使用在零页寻址方式下，将操作码后零页地址单元中的数送入寄存器Y。操作码是A4。

例：A4 E0 LDY \$E0

八、机器语言程序设计

本章介绍机器语言程序设计的方法和技巧，包括简单程序设计，分支程序设计，循环程序设计，子程序设计，堆栈程序设计及调用监控子程序等六个方面的内容。

虽然我们分节介绍上面的六个方面内容，但实际上，一个实用程序常常是六个方面的有机结合。

简单程序是复杂程序的基础，也是初学者程序设计入门的必经之路。分支程序是循环程序的基础，用以解决各种各样的分支转移问题。循环程序完成大量的在处理形式上完全相同的重复性计算工作，是机器语言程序设计的核心和精华。子程序用以处理在功能上具有一定独立性，能够完成相同计算和操作，并被经常调用的程序段，用以简化设计并使结构清晰。堆栈是计算机信息处理中的重要概念，是保护和恢复现场的主要手段。监控子程序简短独立、功能完整、结构合理、简炼灵活，在程序设计中，合理调用能起到事半功倍的效果。

1. 简单程序设计

简单程序一般是指整个程序自始至终按指令顺序逐条执行，而在执行过程中不会出现分支或转移，因而又称直线程序。

简单程序设计方法虽然比较单一，但包含的内容还是比较丰富的，即包括数据输入、进行计算、输出结果、结束停

机等四个方面内容。也就是说它包含了任何一个程序设计中
所不可缺少的每一部分，反映了程序设计的结构和全貌。

简单程序仅能完成一些简单操作，一般不单独使用。而
且，任何一个实际程序绝非单纯由简单程序组成，但初学编
程从简单程序入手实为必要，因为它是构成复杂程序的基础，
又是初学者程序设计入门的必经之道。

现在，举一些简单程序的设计实例。

例 1：数据块转移

将起始地址 \$2000—\$2004 的 5 个字节内容转移到 \$4000
—\$4004 单元中去。

由于转移的数据较少，问题十分简单，直接写出程序
8.1。

程序 8.1:

1000-	AD	00	20	LDA	\$2000
1003-	8D	00	40	STA	\$4000
1006-	AD	01	20	LDA	\$2001
1009-	8D	01	40	STA	\$4001
100C-	AD	02	20	LDA	\$2002
100F-	8D	02	40	STA	\$4002
1012-	AD	03	20	LDA	\$2003
1015-	8D	03	40	STA	\$4003
1018-	AD	04	20	LDA	\$2004
101B-	8D	04	40	STA	\$4004
101E-	60			RTS	

运行前: 2000—0A 0B 0C 0D 0E

4000—00 00 00 00 00

运行后: *1000G✓

*4000·4004✓

程序 8.1 用的是取数和送数两条指令，反复进行同样的操作共 5 次。显然，若要传送 250 个字节的数据，则程序将要写上 500 个语句，这不仅没有必要，而且几乎也是不可能的，因此，对于大量数据的传送，直接程序有很大的局限性。对于少量数据的传送，程序 8.1 既简单又实用，复杂程序中的数据传送常采用类似的结构。

例 2：数据交换

将 \$06 单元和 \$08 单元中的内容互换。设原来 (06) = AA, (08) = BB。交换后应有 (06) = BB, (08) = AA。

对这类简单问题设计方法很多，这里借助于一个暂存单元 \$07，做中间寄存器来互换。见程序 8.2。

程序 8.2:

```

1000-   A5  06      LDA    $06
1002-   85  07      STA    $07
1004-   A5  08      LDA    $08
1006-   85  06      STA    $06
1008-   A5  07      LDA    $07
100A-   85  08      STA    $08
100C-   60          RTS

```

运行前: * 6: AA ↙

* 8: BB ↙

运行后: * 1000G ↙

* 6 ↙

0006—BB

* 8 ↙

0008—AA

分析程序 8.2 时，应紧紧抓住这样一个事实：数据从源

地址传送到目的地址，目的地址的内容被源地址内容更新，而源地址内容不变。这也是分析一切传送指令的要领。

例 3：求反码

设原码为 \$6A，设计一个程序求出它的反码。结果为 \$95。

6502指令系统中没有求反码的指令，但可以将一个 8 位 2 进制数与 \$FF 进行异或处理，异或结果即为该 8 位 2 进制数的反。

$$\begin{array}{r}
 \text{即: } 6A = 01101010 \\
 \oplus FF = 11111111 \\
 \hline
 95 \quad 10010101
 \end{array}$$

故有程序 8.3。

程序 8.3:

1000-	A9	6A	LDA	# \$6A
1002-	49	FF	EOR	# \$FF
1004-	20	DA FD	JSR	\$FDDA
1007-	60		RTS	

运行: *1000G↙

*95

例 4：求补码

设原码为 \$09，设计一个程序求出它的补码。结果为 \$F7。

6502指令系统中也没有求补码的指令，但可根据原码取反加 1 就是补码的定义，容易写出程序 8.4。

程序 8.4:

1000-	A9	09	LDA	# \$09
1002-	49	FF	EOR	# \$FF
1004-	D8		CLD	

1005-	18		CLC
1006-	69	01	ADC #\$01
1008-	20	DA FD	JSR \$FDDA
100B-	60		RTS

运行: *1000G↙

*F7

例 5: 拆字

拆字又叫字的分离或字的分解。

设原字为A3 (10100011), 将其分成两段, 每段四位, 高4位放入\$6001单元中低4位位置上, 低4位放入\$6000单元低4位位置上, 并使\$6001和\$6000两个单元的高4位全部清零。

即: (6000) = 00000011 = 03

(6001) = 00001010 = 0A

解本题的思路可分为二步: 第一步将原字A3取出来, 屏蔽高4位 (AND #\$0F), 留下低4位 (0011), 存入\$6000单元中; 第二步将原字A3再一次调出来, 用4次逻辑右移一位令LSR, 这样原高四位的值A (1010) 全部移入低4位, 而高4位变0, 再存入\$6001单元。故有程序8.5。

程序8.5:

1000-	A9	A3	LDA	#\$A3
1002-	29	0F	AND	#\$0F
1004-	8D	00 60	STA	\$6000
1007-	A9	A3	LDA	#\$A3
1009-	4A		LSR	
100A-	4A		LSR	
100B-	4A		LSR	
100C-	4A		LSR	

```

100D-    8D 01 60    STA    $6001
1010-    60          RTS

运行: *1000G↙
      *6000↙
      *6000—03
      *6001↙
      *6001—0A

```

例 6：组字

组字又称语句组合，字组合。

例如，将 \$01 和 \$02 单元中的低 4 位组成一个字，放在 \$03 单元中。其中 \$01 单元的低 4 位放入 \$03 单元高 4 位；\$02 单元的低 4 位放入 \$03 单元低 4 位。

即：(01) = 6A

(02) = 43

(03) = A3

设计本题程序的要领是先拆字后组字，见程序 8.6。

程序 8.6

```

0300-    A9 6A    LDA    # $6A
0302-    85 01    STA    $01
0304-    A9 43    LDA    # $43
0306-    85 02    STA    $02
0308-    A5 02    LDA    $02
030A-    29 0F    AND    # $0F
030C-    85 03    STA    $03
030E-    A5 01    LDA    $01
0310-    0A       ASL
0311-    0A       ASL
0312-    0A       ASL
0313-    0A       ASL

```


0314-	85	04	STA	\$ 04
0316-	18		CLC	
0317-	A5	03	LDA	\$ 03
0319-	65	04	ADC	\$ 04
031B-	85	03	STA	\$ 03
031D-	60		RTS	

程序说明:

0300—0302: (01) = 6A

0304—0306: (02) = 43

0308: 取43

030A: 屏蔽高四位

030C: 存数 (03) = 03

030E: 取6A

0310—0313: 左移四位

0314: 存数 (04) = A0

0316: 清进位位 0 → C

0317: 取03

0319: 加A0

031B: 存结果A3

031D: 结束

非常有趣的是, 程序8.6中的ADC \$04指令, 可以换成下面三条指令中的两条:

AND \$04

ORA \$04

EOR \$04

而不影响程序的正确性。

值得指出的是, 程序8.6还可以进一步化简, 完全可以不用\$04单元暂存, 因为在执行第七条指令STA \$03后, \$03单元已存好03的值; 而在执行四条左移一位指令ASL后,

\$01单元的值为A0。因此，可以将\$01单元的内容和\$03单元中的内容进行组合。方法是修改\$0314单元开始的内容。

如：

```
0314-A5 03 LDA $03
0316-05 01 ORA $01
0318-85 03 STA $03
031A-60     RTS
```

或：

```
0314-A5 03 LDA $03
0316-45 01 EOR $01
0318-85 03 STA $03
031A-60     RTS
```

程序8.6还可以进一步简化，可以省去\$01—\$04的4个零页地址的选用，同样完成字的组合任务，见程序8.7。

程序8.7：

```
1000-    A9 43          LDA    #$43
1002-    29 0F          AND    #$0F
1004-    85 06          STA    $06
1006-    A9 6A          LDA    #$6A
1008-    0A             ASL
1009-    0A             ASL
100A-    0A             ASL
100B-    0A             ASL
100C-    18             CLC
100D-    65 06          ADC    $06
100F-    20 DA FD       JSR    $FDDA
1012-    60             RTS
```

运行：*1000G✓

*A3

总之，一道题目的解法是多种多样的，学习用不同方法编程，可以训练思维，扩大思路，学会方法，熟习指令，提高编制程序的能力和技巧。

例 7：求平均值

为简单起见，仅求两个数 \$3A 和 \$22 的平均值。结果应为 \$2E。见程序 8.8。

程序 8.8:

```
1000- 20 58 FC    JSR    $FC58
1003-  A9 3 A      LDA    # $3 A
1005-  18          CLC
1006-  69 22      ADC    # $22
1008-  4 A        LSR
1009-  20 DA FD   JSR    $FD DA
100C - 60          RTS
```

程序 8.8 在使用上有很多不足之处：

- ① 不能进行 N 个数的累加和求平均；
- ② 和数只能限于一个 8 位数，不能有进位；
- ③ 只适用于无符号数或正数，LSR 相当于 $\div 2$ 。

因此，本例和本节中的其它几个例子，处理的都是一些最简单问题。而对一些比较复杂的问题，如：N 个数据的累加和，大量数据的传送，多个数据的交换，判断奇偶数，确定正负数，输入输出码的转换，字符串的比较，多个数据块的连续搬移等等诸如此类的问题，简单程序设计是不能胜任的。故有必要介绍其它一些程序设计方法，以补不足。但不管怎样，简单程序设计方法，仍是一切初学者程序设计入门的必然阶梯，因为它处理的问题简单、直接，用的指令少，没有分支、转向和循环、直观、形象，易于初学者接受。通过上

述实例分析，我们初步勾画了程序设计的部份面貌，也为程序设计方法和技巧抛出了引子，希望起到开阔视野、启迪思维的作用。

2. 分支程序设计

在程序设计中，除极简单的问题外，一般都比较复杂，解决问题的程序常常带有分支，即根据不同条件，使程序转向不同的地方执行。因此，我们十分需要一种具有判断能力的控制转移手段，以便对程序进行有效的控制。而计算机则按给定的条件进行分析、比较、判断后决定程序的走向，这就是分支程序设计。

6502指令系统中为我们提供了众多的转移指令，这种指令使程序在执行过程中出现分支，其重要性在于它们使计算机具有某种判断功能，可以根据具体条件满足与否，来决定是进行分支转移还是继续执行下一条指令。

我们可以根据问题的性质，选择不同的转移指令，控制程序的流向，以解决各种各样的分支问题。必须着重指出，灵活而准确地使用转移指令，特别是条件转移指令，是高质量汇编语言程序设计不可缺少的部份，而且使用的技巧性很强。

现举若干实例，作为分支程序设计入门的引导。

例 1：数的比较

一数放在 \$06 单元，一数放在 \$07 单元，比较两者是否相同。相同给出 00 标志，不同给出 FF 标志。

对于数或子字符串的比较是否相同的问题，最好用异或指令 EOR，它的功能是将存贮器同累加器内容异或，并将结果送累加器，即 $A \vee M \rightarrow A$ 。见程序 8.9。

程序8.9:

```

1000-   A5  06           LDA    $ 06
1002-   45  07           EOR    $ 07
1004-   D0  06           BNE    $ 100C
1006-   85  08           STA    $ 08
1008-   20  DA  FD       JSR    $ FDDA
100B-   60              RTS
100C-   A9  FF           LDA    # $ FF
100E-   85  09           STA    $ 09
1010-   20  DA  FD       JSR    $ FDDA
1013-   60              RTS

```

运行前: * 6:AA↙

* 7:FF↙

运行后: * 1000G↙

* FF

运行前: * 6:AA↙

* 7:AA↙

运行后: * 1000G↙

* 00

程序8.9中,在执行异或指令EOR后,有两种情况:两数相同,结果为0;两数不同,结果非0。因此,标志寄存器P中的零标志Z保留了执行EOR指令后结果的情况,前者Z=1,后者Z=0,所以,选用BNE指令也是顺理成章的,它是根据Z标志是0还是1,发生转移或者继续执行下一条指令。两数相同,结果为0,Z=1,执行下一条指令,0→(08),显示(A)=0的结果并停机;两数相异,结果非0,Z=0,转移到\$100C地址,取不同标志FF→(A),FF→(09),显示(A)中内容FF并结束。

例 2：判断奇偶数

在\$06单元中存放一个数B，若B是奇数，给出01标志；若B是偶数，给出00标志。

任何一个数（正数、负数或零），将其展开成一个8位2进制数时，第0位不是“1”就是“0”。若原数是偶数或零时，则第0位一定是“0”；若原数是奇数时，则第0位一定是“1”。这个事实告诉我们，为了判断数的奇偶性，可以改为判断第0位是“1”还是“0”。

如果我们将一个数先右移一位，再左移一位，则第0位一定是“0”。

这样，我们就可以根据运算前后的值相同与否，判定数的奇偶性，结果相同为偶数，结果相异为奇数。故有程序8.10。

程序8.10:

```
1000-   A5  06           LDA    $ 06
1002-   4A              LSR
1003-   0A              ASL
1004-   C5  06           CMP    $ 06
1006-   D0  05           BNE    $ 100D
1008-   A9  00           LDA    # $ 00
100A-   4C  0F  10       JMP    $ 100F
100D-   A9  01           LDA    # $ 01
100F-   20  DA  FD       JSR    $ FDDA
1012-   60              RTS
```

运行前：* 6 :C7↙ 输入一个奇数

运行后：* 1000G↙

* 01 给出奇数标志

运行前：* 6 :34↙ 输入一个偶数

运行后: * 1000G↙

* 00 给出偶数标志

例 3: 找两数中较大者

两数分别存入\$6000和\$6001单元中, 将大数找出并放入\$6002单元中。

取第一个数, 与第二个数比较, 若第一个数大, 则存入\$6002单元, 反之, 第二个数大, 将第二个数调进累加器A, 再用STA \$6002指令, 存入\$6002单元。见程序 8.11。

程序8.11:

```
1000-   AD  00  60      LDA    $ 6000
1003-   CD  01  60      CMP    $ 6001
1006-   B0  03          BCS    $ 100B
1008-   AD  01  60      LDA    $ 6001
100B-   8D  02  60      STA    $ 6002
100E-   60              RTS
```

运行前: * 6000:4F 3F

运行后: * 1000G↙

* 6002↙

* 6002-4F

运行前: * 6000:6D 80

运行后: * 1000G↙

* 6002↙

* 6002-80

这段程序的思想是用CMP指令来比较两个数, 然后利用BCS指令进行程序分支。在比较指令之后配合条件转移指令, 是分支程序设计中经常使用的一种方法。

应该指出的是, 在选用CMP指令时, 要注意:

(1) CMP指令是累加器A的内容同存贮器M的内容

比较，这里的比较实际上是做一次减法操作，即 $A - M$ 。

(2) CMP 指令同减法指令 SBC 区别之一是，进位标志 \overline{C} (表示 C 标志取反) 不参加减法运算，前者 $A - M$ ，后者 $A - M - \overline{C}$ 。

(3) CMP 指令同减法指令 SBC 的另一个区别是，减法运算结果不送入累加器 A，即 CMP 执行后不改变 A 的内容，前者 $A - M$ ，后 $A - M - \overline{C} \rightarrow A$ 。

(4) 执行 CMP 指令后，影响 P 寄存器中的标志位 C、Z、N，尤其是影响标志位 C。若 $A \geq M$ ，表示够减无借位，C 置 1；若 $A < M$ ，表示不够减有借位，C 置 0。因此，可由 C 的状态判断 A 同 M 中两数谁大谁小。

因此，本例在选用 CMP 指令后，自然配合使用 BCS 指令，因为 BCS 指令是根据 P 寄存器中 C 的状态决定分支转移或继续执行下一条指令，即 $C = 1$ 时转移， $C = 0$ 时继续。

例 4：确定正负数

一个数（正数或负数）存放在 \$06 单元中，设计一个程序能判断该数是正数还是负数，并给出正数标志 00、负数标志 01。

由于本例中只判断正数或负数性质，而且仅限于对一个数的判别，因此，选用 BPL 指令是合适的，因为它以 N 作为判别标准， $N = 0$ ，表示正数，执行分支动作； $N = 1$ ，表示负数，则执行下一条指令。N 标志的状态可由执行 LDA 指令后，P 标志寄存器中的 N 标志位取得，因为 LDA 指令对符号位 N 及零标志位 Z 皆有影响。故有程序 8.12。

程序 8.12:

1000~	A5 06	LDA	\$ 06
1002~	10 05	BPL	\$ 1009


```

1004-   A9  01           LDA   # $01
1006-   4C  0B  10      JMP    $100B
1009-   A9  00           LDA   # $00
100B-   85  07           STA   $07
100D-   60              RTS

```

运行前: * 6:7D 输入一个正数

运行后: * 1000G ↙

* 7 ↙

* 0007-00 给出正数标志

运行前: * 6:96 输入一个负数

运行后: * 1000G ↙

* 7 ↙

* 0007-01 给出负数标志

程序8.12中安排了一条无条件转移指令JMP,它是编程人员事先安排好的,它控制程序没有任何条件地实现转移,即将程序由通常的顺序执行状态转向另一个指定的目标地址,这和条件转移指令的执行情况不同,后者则是在满足某种条件的情况下,才使程序转移,即根据P寄存器中各标志位状况来确定程序是否转移。

例5: 输入码转换

如从键盘上按下一位10进制数,将其转换成16进制数。

10进制数的代码是0—9,程序的设计必须保证小于0的和大于9的键码都不要,而且还应保证按任何字符和运算符号都不应出错,能自动返回等待正确输入;同时,也应安排一个控制键,例如回车键,让其程序自动停机。因此,本程序的设计是包含多个条件语句的分支程序设计。见程序8.13。

程序8.13:

1000-	20	1B	FD	JSR	\$FD1B
1003-	C9	8D		CMP	#\$8D
1005-	F0	11		BEQ	\$1018
1007-	C9	B0		CMP	#\$B0
1009-	90	F5		BCC	\$1000
100B-	C9	BA		CMP	#\$BA
100D-	B0	F1		BCS	\$1000
100F-	20	DA	FD	JSR	\$FDDA
1012-	20	8E	FD	JSR	\$FD8E
1015-	4C	00	10	JMP	\$1000
1018-	60			RTS	

程序8.13运行后，等待操作者从键盘键入输入的数码，若在0至9之间，则显示B0—B9（0—9对应的ASCII码），否则按任何字符、符号都无效转向\$1000开始重新接受按键输入，若要停机，则按RETURN（它的ASCII码是8D）键。程序中用了三个监控子程序，\$FD1B：等待键盘输入，\$FDDA：输出一个字符子程序，\$FD8E：换行。

例6：16进制数转换成ASCII码

计算机键盘与屏幕显示器都采用ASCII码表示字符。一个16进制数字可以用代码0—9或者用字母A—F表示。这两段内容在正常方式的屏幕显示时，对应的ASCII码是B0—B9及C1—C6。一个需要显示的数字必须先换成对应的ASCII码，然后再送入显示器。

要完成16进制数向ASCII的值转换，其方法是：如果属于0—9数字，则只要将它们加上B0；如果属于A—F字母，则需将它们加上B7。所以，16进制数字转换成ASCII码的程序，实际上是一个分支转移程序：即判断输入码属于0—9还是A—F，是前者加B0，是后者加B7。故有程序8.14。

程序8.14:

1000-	A5	06	LDA	\$06
1002-	C9	0A	CMP	# \$0A
1004-	90	03	BCC	\$1009
1006-	18		CLC	
1007-	69	07	ADC	# \$07
1009-	69	B0	ADC	# \$B0
100B-	85	07	STA	\$07
100D-	20	DA FD	JSR	\$FDDA
1010-	60		RTS	

运行前: * 6 :F✓

运行后: * 1000G✓

* C6

或: * 1000G✓

* 7✓

* 0007-C6

运行前: * 6 :05✓

运行后: * 1000G✓

* B5

或: * 1000G✓

* 7✓

* 0007-B5

本题也可用另法求解,其思路是先把数字0的ASCII码B0加到所有待转换的16进制数上去,然后与BA比较,以判断是否属于0—9,再决定是否加07。故有程序8.15。

程序8.15:

1000-	A5	06	LDA	\$06
1002-	29	0F	AND	# \$0F
1004-	09	B0	ORA	# \$B0

1006-	C9	BA	CMP	# \$ BA
1008-	90	02	BCC	\$ 100C
100A-	69	06	ADC	# \$ 06
100C-	85	07	STA	\$ 07
100E-	20	DA FD	JSR	\$ FDDA
1011-	60		RTS	

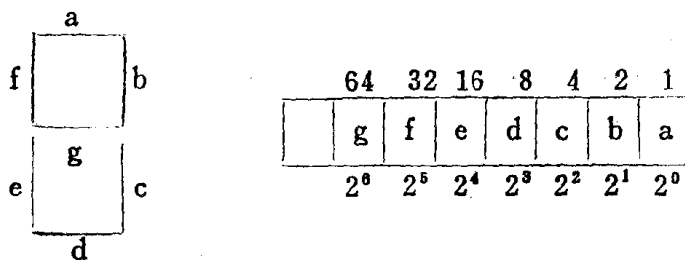
程序8.15的运行方法同程序8.14。

例7：10进制数转换成7段代码

7段代码是7段发光二极管显示器显示字符所采用的代码，对共阴极7段显示器来说，所用7段代码显示10进制数字的对应关系为：

数字	代码	数字	代码
0	3F	5	6D
1	06	6	7D
2	5B	7	07
3	4F	8	7F
4	66	9	6F

7段代码的安排是：



在一个字节中构成7段代码的方法是：最高位总是0，其余7位由低位向高位顺序为a、b、c、d、e、f、g各段代码（1亮，0不亮）。

例如，当 a、b、c、d、g 均为 1 时，则显示 10 进制数 3，即：

$$\begin{aligned}
 & 2^0 \times 1 + 2^1 \times 1 + 2^2 \times 1 + 2^3 \times 1 + 2^4 \times 0 + 2^5 \times 0 + 2^6 \times 1 \\
 &= 1 + 2 + 4 + 8 + 0 + 0 + 64 \\
 &= (79)_{10} \\
 &= (4F)_{16}
 \end{aligned}$$

所以，要求编一个程序，输入 0—9 这 10 个 10 进制数中的任一代码，显示出相应的 16 进制数。

程序可以这样安排，在 \$6000 中存放一个 10 进制数，比较是否在 0—9 之间，是的，取对应的 16 进制代码；不是的，显示 0。故有程序 8.16。

程序 8.16:

```

1000-   A9  00           LDA   # $00
1002-   AE  00  60       LDX   $ 6000
1005-   E0  0A           CPX   # $0A
1007-   B0  03           BCS   $ 100C
1009-   BD  00  70       LDA   $ 7000,X
100C-   8D  01  60       STA   $ 6001
100F-   20  DA  FD       LSR   $FD0A
1012-   60              RTS

```

程序 8.16 运行前应先设置：

7000- 3F 06 5B 4F 66 6D 7D 67

7008-07 7F 6F

* 6000: 5 ✓

* 1000G ✓

* 6D

本题可以进一步简化，只要在 \$06 单元中放置 0—9 当

中的任一个10进制数码，运行程序8.17后，立即得到结果。

程序8.17:

```
1000-   A4  06           LDY    $ 06
1002-   B9  00  70      LDA    $ 7000,Y
1005-   20  DA  FD      JSR    $ FDDA
1008-   60              RTS
```

程序8.17 非常简捷，运行前应设置 \$06 单元和 \$7000—\$7009单元中的值。

如 (06) = 03

7000:3F 06 5B 4F 66 6D 7D 07 7F 6F
* 1000G ↙

显示: 4F

以上我们通过几个实例，介绍了分支程序设计的一些问题。可以看出，分支程序解决和处理的问题，要比简单程序广泛和多样，在设计方法和应用技巧上，又比简单程序丰富和灵活。

应该着重指出的是，学习和掌握分支程序设计有一定的难度，其困难所在是如何选用条件转移指令，以及什么条件转移和转向到那里去，这需要经常编程和大量调试。但有一点是至关重要的，必须牢牢掌握标志寄存器 P 的各个标志位是 0 还是 1 的状况。因为标志寄存器 P 中保留了每条指令执行后的情况，如执行后结果为全零或非 0；是正数或负数；有进位或无进位；有溢出或无溢出等等，而这些情况即为判断条件分支的重要依据，也是学习本节内容的要领和程序设计的重点。

还有一点要说及的，仅仅掌握了简单程序、分支程序的设计方法，还不能解决名目繁多形式各异的实际问题，特别

是不能有效地解决大量的在处理形式上又完全相同的重复性计算问题。因此，循环程序的设计方法和技巧，应该作为重点来介绍，请看下节内容。

3. 循环程序设计

在科学技术和生活领域里，常常遇到一些需要大量重复计算的课题，人们所关心的是如何减少一些不必要的重复性工作，以节省存贮单元和提高计算效率。

完成大量的在处理形式上完全相同的重复性工作，这就是循环。在程序中，对某一段内容反复执行多次，这就是循环程序。而在一个循环程序中，包含另一个或几个循环，则称为循环的嵌套或多重循环。

循环程序的结构包括以下几个部分：

(1) 初始化部分：它建立计数器，地址寄存器（指针）和有关变量的起始值。如累加器清零，源数据块首地址、数据块长度、内外循环计数器、循环次数等置初值。

(2) 处理部分：在这部分进行实际的数据处理，这是循环的工作部分，它完成重复执行程序段的内容。

(3) 循环控制部分：它检查每循环一次是否结束，并为下一轮循环做好修正计数器和指针的准备。

(4) 结束部分：它分析、存放或显示结果，结束停机。

循环程序对于处理数据块的传送问题比较方便，而采用变址寻址方式处理动态数据块和多个数据块则较为理想，因为变址寻址方式允许用改变变址寄存器内容的方法来改变存储器的实际地址。为此，程序必须在每一轮处理后将变址寄存器内容加1，使它指向数据块的下一个元素，这样就使下

一轮对下一个存贮单元的数据执行相同的操作，因而可用同一个程序段来处理 256 个字节长度的数据块（因变址寄存器是 8 位字长）。

另外，我们将从下面所举的实例中看到，在循环程序中大量的包含了程序分支内容，因为每循环执行一次后是继续重新循环呢？还是不再循环而往下执行新的程序段呢？这里就存在着控制循环的程序分支，以根据某种条件满足与否决定程序的走向，控制语句的执行路径。因此，我们可以看到分支和循环的概念是不同的，对于一些单纯的分支程序，其中每条指令可能只执行一次而不进行循环，而循环程序的处理部分则在程序执行过程中将被执行许多次。同时，我们也看到，单纯分支不包括循环，循环却一定包含分支，它们既有区别又有联系，而 6502 指令系统中的比较指令和条件转移指令，为设计分支程序和循环程序提供了方便。

例 1：求数据的累加和

计算一串 2 进制数的和，串数的长度放在 \$06 单元，而这串数从 \$6001 单元开始存放。将和数放在 \$07 单元，设和数是一个 8 位 2 进制数，这样可以不考虑进位。见程序 8.18。

程序 8.18:

1000-	20	58	FC	JSR	\$FC58
1003-	A9	00		LDA	# \$00
1005-	AA			TAX	
1006-	D8			CLD	
1007-	18			CLC	
1008-	7D	01	60	ADC	\$6001,X
100B-	E8			INX	
100C-	E4	06		CPX	\$06

100E-	D0	F7	BNE	\$ 1007
1010-	85	07	STA	\$ 07
1012-	A5	07	LDA	\$ 07
1014-	20	DA FD	JSR	\$ FDDA
1017-	60		RTS	

程序说明:

1000: 清屏

1003: 和置初值

1005: 计数器置初值

1006: 2 进制求和, 清10进制标志位

1007: 清进位位C

1008: 和 = 和 + 数据

100B: 计数器加 1

100C: 次数够了吗?

100E: 不够, 继续循环

1010: 够了, 送结果

1012: 取结果→A

1014: 显示结果

1017: 结束

程序8.18设计要点:

(1) 初始化: 和数放累加器A中, 初始值为0, 寄存器X当计数器, 未加前为0, 清10进制标志和清进位位标志。

(2) 循环: 取数相加, 变址计数器加1指向下一个地址的数据, 判断数据是否加完, 不是继续循环求和, 是的送结果。

(3) 结束: 从\$07单元取结果, 调显示字符(16进制)子程序, 送现行输出设备上, 结束。

运行前: * 6:05 设串长度为 5

* 6001: 12 34 1D 56 20 5个数据

运行后: * 1000G ↙

* D9

或 * 1000G ↙

* 7 ↙

* 0007-D9

关于程序8.18的几点说明:

(1) 本程序原则上可以完成N个数的重复相加, 数据个数放\$06单元, 并可灵活更改。

(2) N个数的累加和不能超过\$FF (255), 否则结果不正确。

(3) 是一个单循环, 数据的累加是从前向后进行的。

(4) 循环中包括分支, 主要由执行比较指令 CPX \$06 后, P寄存器中的零标志Z值, 决定BNE指令是否转移。而循环的次数由指令1NX, CPX, BNE共同决定。

对N个数据的累加和, 也可以如下编制, 见程序8.19。

程序8.19:

1000-	20	58	FC	JSR	\$FC58
1003-	A9	00		LDA	# \$00
1005-	AE	00	60	LDX	\$6000
1008-	D8			CLD	
1009-	18			CLC	
100A-	7D	00	60	ADC	\$6000,X
100D-	CA			DEX	
100E-	D0	F9		BNE	\$1009
1010-	8D	30	10	STA	\$1030
1013-	AD	30	10	LDA	\$1030

```
1016-    20    DA    FD        JSR    $FDDA
```

```
1019-    60                                RTS
```

运行前: *6000:05

*6001:12 34 1D 56 20

运行后: *1000G↙

*D9

*1030↙

*D9

程序8.18和8.19的异同点:

(1) 共同点,都是用单一循环编程,完成和数不超过8位二进制数的N个数累加。

(2) 主要区别,设计思想不同,程序8.18求和时是从前面往后面加的,而程序8.19求和时则是从后面向前面加。

此外,我们还看到在选用ADC指令时,其操作数部份的基地址用的不同,程序8.18中用的是\$6001,程序8.19中用的是\$6000,这是两种程序不同算法和字符串长度选用的不同单元决定的。这是编写程序时应该注意的重要细节。

最后,我们还应指出,这两个程序在编制结构上虽然具有一定的灵活性(可以进行N个数的累加和),但通用性不够(和数不能超过一个8位二进制数)。解决思路是设法保存进位的值或采用多字节加法程序。

例2: 找最大值

在一段资料中寻找最大的元素。设此段资料个数放在\$1A单元,\$1B—\$1E单元中存放资料,要求结果放在\$1F单元。并设资料中的数皆为无符号的8位二进制数。见程序8.20。

程序8.20

```
1000-    A6    1A        LDX    $1A
```

1002-	A9	00	LDA	# \$ 00
1004-	D5	1A	CMP	\$ 1A, X
1006-	B0	02	BCS	\$ 100A
1008-	B5	1A	LDA	\$ 1A, X
100A-	CA		DEX	
100B-	D0	F7	BNE	\$ 1004
100D-	85	1F	STA	\$ 1F
100F-	60		RTS	

程序说明:

1000: 计数器初值为数的个数

1002: 设最大值 = 0

1004: 下一个数 > 最大值吗?

1006: 否, 维持 A 中最大值不变

1008: 是, 用该数代替最大值

100A: X-1

100B: 所有数都比较完了? 否, 循环

100D: 是, 有最大值

100F: 结束

运行前: * 1A: 04 05 FF 15 E3

运行后: * 1000G ↙

* 1F ↙

* 001F—RF

程序8.20虽然较短, 但因用了比较指令和两条条件转移指令, 使程序阅读有些困难, 今作一些简单解释。

开始, X 寄存器作为计数器用应置初值(1A)→X, 即 X = 04, 接着在累加器A中放了一个立即数00, 表示找数开始前假想的一个最大值为 0, 执行第三条指令 CMP \$ 1A, X时, 表示(A) - (1A + X) = (A) - (1E) = 0 - E3, 显然不够减, P寄存器中的C标志置 0, 执行第四条指令 BCS时,

由于执行前一条指令后 $C = 0$ ，所以不发生转移而执行下一条指令 $LDA \$1A, X$ ，即将 $(1A + X) \rightarrow A$ ， $(1E) \rightarrow A$ ， $(A) = E3$ ，可见这时已用 $E3$ 代替原来假想的最大值 0 。执行第六条指令 $X - 1 \rightarrow X$ ，只要 X 不为 0 ，则执行 BNE 指令后循环上去转至 $\$1004$ ，由于此时 $X = 03$ ，故再次执行 CMP 指令， $(A) - (1A + X) = (A) - (1D) = E3 - 15$ ，够减， $C = 1$ ，又执行 BCS 指令，因此时 $C = 1$ ，故发生转移，到 $\$100A$ ，注意此时 A 中仍为 $E3$ ，维持最大值不变。 X 又减 1 ，为 02 ，不为 0 ， $Z = 0$ ，又循环上去。执行 CMP 指令， $(A) - (1A + X) = (A) - (1C) = E3 - FF$ ，不够减， $C = 0$ ，执行 BCS 指令时由于 $C = 0$ 执行下一条指令 $LDA \$1A, X$ ，即将 $(1A + X) \rightarrow A$ ， $(1A + 02) \rightarrow A$ ， $(1C) = A$ ， $(A) = FF$ ，可见这时已用更大值 FF 代替次大值 $E3$ 。……，下面的过程重复执行上述类似的步骤，每一次循环后都将更大的数放入 A 中，不断循环，直至比较完毕所有的数，最后将最大值（真正的）放入 $\$1F$ 中，结束。

综上所述，分析程序时应注意：

(1) 什么时候循环，循环多少次，以及何时退出循环，都由实现分支的条件语句控制着，循环中包含着分支。

(2) 比较指令 CMP 表示 $A - M$ ，够减 $C = 1$ ，不够减 $C = 0$ ； BCS 指令在 $C = 1$ 时转移， $C = 0$ 时执行下一条指令； BNE 指令表示结果不为 $0 (Z = 0)$ 时转移，而结果为 0 时 ($Z = 1$)继续执行下一条指令。记住这些对看懂程序大有好处。

例3：确定负数的个数

已知数据块的长度放在 $\$6000$ 单元中，而数据块本身从存储单元 $\$6001$ 开始存放，要求将其中负数的个数放在 $\$07$ 单元。

如：6000: 06 68 F2 87 30 59 2A

因\$6002和\$6003两个单元的值均为负数，所以程序设计运行后的结果应有：(07)=02。

分析

(1) 确定一个数据块中负数的个数，就是查找数据块中数的最高位为1的个数。

(2) 由于本例中只有正数和负数两种，而没有零值，因此，选用BPL指令将是合适的，因为它以N标志作为判别标准，若 $N=0$ ，表示正数，执行分支动作；若 $N=1$ ，表示负数，则执行下一条指令。

(3) 由于本例中有6个数（为简单起见只选了6个数，原则上可以有N个数， $N < 256$ ），必须逐一比较、检查才能确定那些是负数，那些是正数，而这种重复性的比较显然用循环的方法处理。

(4) 在选用BPL指令之前，选用取数指令LDA\$6001，X也是顺理成章，一是因为只有取出一个数来，才好判别正负数；二是LDA指令执行后对符号位N及零标志Z皆有影响；三是采用绝对X变址寻址方式的LDA指令，只要改变X值，即可改变指向下一个数据的地址。

(5) 为了控制循环的次数，必须设置计数器，例如选用X寄存器作为计数器。为了统计负数的个数，也应设置一个负数计数器，例如选用Y寄存器。

根据上述分析，容易写出源程序8.21。

程序8.21:

1000-	A2	00	LDX	# \$ 00
1002-	A0	00	LDY	# \$ 00
1004-	BD	01 60	LDA	\$ 6001,X
1007-	10	01	BPL	\$ 100A

1009-	C8		INY	
100A-	E8		INX	
100B-	EC	00 60	CPX	\$ 6000
100E-	D0	F4	BNE	\$ 1004
1010-	84	07	STY	\$ 07
1012-	98		TYA	
1013-	20	DA FD	JSR	\$ FDDA
1016-	60		RTS	

程序说明:

1000: 计数器置初值

1002: 负数计数器置初值

1004: 取一个数

1007: 是负数吗? 否, 转

1009: 是, 负数个数加 1

100 A: 计数器加 1

100 B: 所有数都检查完了吗?

100 E: 否, 继续取数检查

1010: 是, 存负数个数

1012: Y→A

1013: 显示

1016: 结束

运行前: * 6000: 06 68 F2 87 30 59 2A

运行后: * 1000G↙

* 02

或: * 1000G↙

* 07↙

* 0007-02

讨论: 若要确定正数的个数, 如何编程。

这个问题同确定负数的个数相类似, 可以仿照程序8.21

的模式编程。但最为简捷的方法是只要改动一条指令，你能指出来吗？

在思考之前，最好先不要看下面的结果。

事实上，只要将程序8.21中的BPL指令改为BMI指令。因为，这两条指令虽然都以N标志作为判别的依据，但两者转移的条件是不同的。BMI指令是当 $N = 1$ 时，表示负数转移；而当 $N = 0$ 时，表示正数继续执行下一条指令。

实际改动时，只要改动程序8.21中\$1007单元的值，即将10改为30，这是因为两条指令的操作码不同，代表的物理意义相异。

当然，经改动后，Y寄存器中存放的是正数的个数，显示出来的也是正数的个数。

例4：确定正数、负数和0的个数

设数据块长度为06，要求这6个数中逐一确定正数、负数和零的个数。

分析：

(1) 这个问题相对于仅仅判定一段数据中负数的个数或正数的个数来说，要复杂一些。因为，当读取一个数据时，要做几种判断。

(2) 可以选用BNE指令，先判断是否为0，若是0，则记录0的个数；若不是0，再用BPL指令，以判别正、负数。在用BPL指令判别正、负数时，是负数立即存其个数；不是负数跳转并存放正数个数。

(3) 为了统计数据块中正数、负数和零的个数，必须设置三个存贮单元，如负数、零、正数的个数分别存放在\$10，\$11和\$12单元中，而在未统计各自个数前，应使这三个单元为0，这就是程序的初始化条件。

(4) 在每一个数的性质判定以后, 要做两件事, 一是存每一个数的个数, 这可以通过存贮单元内容加 1 来解决, 如用 INC \$10, INC \$11 或 INC \$12; 二是让计数器加 1, 并判所有的数是否处理完, 这可以用 1NX 和 CPX \$03 两条指令完成, 其中 \$03 单元为存放数据块的长度, X 寄存器为计数器, 当所有数都判别完以后结束, 而只要有一个数未判完, 即应循环下去再判。初始化时也应设置 X = 0。

(5) 每判定一个数的性质以后, 都要比较是否已是最后一个数, 这可以用一个共同的子程序来实现, 这样做可以减少相同操作的编程, 使程序简化。关于子程序的设计我们将在下一节介绍。数据放在 \$04—\$09 单元中。

由上分析, 不难写出源程序 8.22。

程序 8.22:

1000-	A9	00	LDA	# \$ 00
1002-	85	10	STA	\$ 10
1004-	85	11	STA	\$ 11
1006-	85	12	STA	\$ 12
1008-	A2	00	LDX	# \$ 00
100A-	B5	04	[LDA	\$ 04,X
100C-	D0	06	BNE	\$ 1014
100E-	E6	11	INC	\$ 11
1010-	20	1E	JSR	\$ 101E
1013-	60		RTS	
1014-	10	06	BPL	\$ 101C
1016-	E6	10	INC	\$ 10
1018-	20	1E	JSR	\$ 101E
101B-	60		RTS	
101C-	E6	12	INC	\$ 12

101E-	E8	INX	
101F-	E4 03	CPX	\$ 03
1021-	D0 E7	BNE	\$ 100A
1023-	60	RTS	

程序说明:

1000: 初始化, 累加器, 正、负、零的个数存贮单元, 计数器 置0

100 A: 取一个数

100 C: 非 0, 转

100 E: 是 0, 0 的个数加 1

1010: 转子程序

1013: 返回

1014: 不是负数, 转

1016: 是负数, 负数个数加 1

1018: 转子程序

101 B: 返回

101 C: 正数个数加 1

101 E: 计数器加 1

101 F: 所有数都比较完吗?

1021: 否, 循环

1023: 是, 结束

运行前: *03: 06 68 F2 87 00 59 2A

运行后: *1000G✓

*10.12✓

*0010—02 01 03

程序8.22编制中有几点技巧:

(1) 仅用一条指令LDA # \$00, 给三个存贮单元\$10, \$11, \$12赋初值 0, 而不是用三条LDA # \$00指令, 这样做程序清晰, 又节省内存。

(2) 调用\$101E—\$1023单元的子程序并正确返回。

因为在判断一个数的性质以后，计数器指针加1，以便正确指向下一个数据；同时，就程序的整体而言，还必须判别是否比较完全体数据，因此，编制一段具有共同功能的子程序实为必要，目的已如前所述。

(3) 注意了主程序和子程序之间的正确衔接。调用子程序后必须注意正确返回，这不仅要在\$1023单元放置一条由子程序返回到主程序断点处的RTS指令；而且对本程序来说，\$1013和\$101B两个单元也必须放置RTS指令，否则程序不能正确运行。

例5：数据插入

假定存贮单元\$6000中的内容6B，尚未出现在序列中，则把该单元内容增加到此序列中。已知序列的长度存放在\$6001单元，而序列本身从\$6002单元开始存放。

示范题：(6000) = 6B

(6001) = 04

(6002) = 37

(6003) = 61

(6004) = 38

(6005) = 1D

结果：(6001) = 05

(6006) = 6B

该单元(6B)被附加到序列中，因原序列中没有此单元，这个序列的长度增加1。

见程序8.23。

程序8.23：

1000-	AD	00	60	LDA	\$ 6000
1003-	AC	01	60	LDY	\$ 6001

1006-	D9 01 60	CMP	\$ 6001,Y
1009-	F0 0F	BEQ	\$ 101A
100B-	88	DEY	
100C-	D0 F8	BNE	\$ 1006
100E-	EE 01 60	INC	\$ 6001
1011-	AC 01 60	LDY	\$ 6001
1014-	99 01 60	STA	\$ 6001,Y
1017-	20 DA FD	JSR	\$ FDDA
101A-	60	RTS	

程序说明:

1000: 取欲插入的数

1003: 取序列长度

1006: 比较, 相同吗?

1009: 是, 转 \$ 101A

100B: 否, $Y-1 \rightarrow Y$

100C: 所有数都比较完吗? 否, 转

100E: 是, 序列长度加 1

1011: 放增加后的长度

1014: 取增加的数

1017: 显示

101A: 结束

运行前: *6000: 6B 04 37 61 38 1D

运行后: *1000G ↙

*6B

*6000. 6006 ↙

*6000-6B 05 37 61 38 1D 6B

程序8.23编制要点:

(1) 取欲插入的数, 与序列中的其它数相比较, 如果序列中原来就有和插入的数相同的数则结束。

(2) 取欲插入的数, 与序列中的数逐一比较, 只要有一个数不同, 就将长度减 1, 只要未比较完全部序列的数, 就循环再比, 只到比较完全部数据, 原数据序列长度加 1 并显示插入的结果。

说明: 程序 8.23 也可采用绝对 X 变址寻址方式安排 CMP 指令和 STA 指令, 同时改 LDY \$6001 指令, 为 LDX \$6001 指令, 即程序中所有有 Y 的地方, 改成 X, 并改正相应指令的操作码, 也能正确运行。

例 6: 单个数据块的传送

将存放在 \$6000—\$60FF 地址单元内的 256 个数据顺序传送到 \$7000—\$70FF 单元中去。

分析: 本题虽然数据量较大 (256 个), 但设计思想比较简单, 取一个数, 送一个数, 计数器加 1, 指向下一个数据地址, 再取, 再送, 只要没有取完送完, 就继续循环, 反之结束。故有程序 8.24。

程序 8.24:

1000-	A2	00	LDX	# \$ 00
1002-	BD	00 60	LDA	\$ 6000,X
1005-	9D	00 70	STA	\$ 7000,X
1008-	E8		INX	
1009-	D0	F7	BNE	\$ 1002
100B-	60		RTS	

程序说明:

1000: 计数器置初值

1002: 取数

1005: 送数

1008: 计数器加 1

1009: 送完了吗? 否, 转

100 B: 是的, 结束

如果我们将题目改一下, 将 \$6000—\$60FF 单元中的数据块按相反顺序传送到 \$7000—\$70FF 单元中去, 程序 又是如何编制呢?

这个题目的解题方法很多, 今摘其一如程序 8.25。

程序 8.25:

1000-	A2	00	LDX	# \$ 00
1002-	A0	FF	LDY	# \$ FF
1004-	BD	00 60	LDA	\$ 6000,X
1007-	99	00 70	STA	\$ 7000,Y
100A-	E8		INX	
100B-	88		DEY	
100C-	D0	F6	BNE	\$ 1004
100E-	60		RTS	

程序说明:

1000: 源数据计数器初值

1002: 目的数据计数器初值

1004: 取源数据

1007: 送入目的地址

100 A: 源计数器加 1

100 B: 目的计数器减 1

100 C: 次数不够, 循环

100 E: 次数够了, 结束

程序中用了三条取数指令和一条存数指令, 并选用了两个变址寄存器 X 和 Y, 作为源数据块计数器和目的数据块计数器用。应该指出的是, 变址寄存器 X(或 Y)都是八位寄存器, 在编程中常被当作一个计数器来用。它们可以由指令控制而被置成一个常数, 并能方便地用加 1 (INX, INY),

减1(DEX, DEY)、比较(CPX, CPY)操作来修改和测试其内容, 从而使得程序能够方便灵活地处理数据块, 修改地址指针, 控制循环次数等。而在本程序8.25中由于要处理的数据块按逆序传送, 因而一个变址寄存器就显得不够用, 所以既用了变址寄存器X, 又用了变址寄存器Y。

例7: 多个数据块的传送

将\$6000—\$600F单元的第一个数据块送到\$7000—\$700F单元中, 将\$6100—\$610F单元的第二个数据块传送到\$7100—\$710F单元中, 将\$6200—\$620F单元中的第三个数据块传送到\$7200—\$700F单元中去。

分析: 这是三个数据块的传送问题, 由于是顺序搬迁, 而且数据块长度相同(都是16个), 使得问题求解不至于过分复杂。数据块可以一个一个送, 送一个数据块就是一个循环, 当送完一个数据块后只要改变一些地址指针, 就可以再送另一个数据块, 直到全部送完。因此, 一个循环程序是处理不了的, 所以本题应是一个多重循环问题。

我们先给出源程序清单, 见程序8.26, 然后看一下运行结果, 最后再对程序编制思想作一说明。

程序8.26:

1000-	A2	00	LDX	# \$00
1002-	A0	10	LDY	# \$10
1004-	A1	1A	LDA	(\$1A,X)
1006-	81	FA	STA	(\$FA,X)
1008-	F6	1A	INC	\$1A,X
100A-	F6	FA	INC	\$FA,X
100C-	88		DEY	
100D-	D0	F5	BNE	\$1004
100F-	E0	04	CPX	# \$04

1011-	10	05	BEQ	\$1018
1013-	E8		INX	
1014-	E8		INX	
1015-	4C	02 10	JMP	\$1002
1018-	60		RTS	

程序说明:

1000: 变址计数器X置

1002: 数据块长度送Y寄存器

1004: 传送一个源数据到目的地址中

1006: 第一次将(6000) → 7000

1008: 修改源地址, 使之增1

100A: 修改目的地址, 使之增1

100C: 一个数据块传送完了吗?

100D: 未送回, 循环

100F: 三个数据块传送完了吗?

1011: 是, 结束

1013: 否, 将X增2, 为取下一个数据块作准备

1015: 转入下一个数据块

1018: 结束

运行前: *1A: 00 60 00 61 00 62

将三个源数据块首地址送入\$1A—\$1F单元

*FA: 00 70 00 71 00 72

将三个目的块首地址送入\$FA—\$FF单元

*6000.600F

6000- 01 01 01 01 01 01 01 01

6008- 01 01 01 01 01 01 01 01

*6100.610F

6100- 02 02 02 02 02 02 02 02

6108- 02 02 02 02 02 02 02 02

*6200.620F

6200- 03 03 03 03 03 03 03 03

6208- 03 03 03 03 03 03 03 03

将第 1, 2, 3 个源数据块的数据分别送入 \$6000—\$600F, \$6100—\$610F, \$6200—\$620F 单元中。

运行后: * 1000 G ↙

* 7000.7000F

7000- 01 01 01 01 01 01 01 01

7008- 01 01 01 01 01 01 01 01

* 7100.710F

7100- 02 02 02 02 02 02 02 02

7108- 02 02 02 02 02 02 02 02

* 7200.720F

7200- 03 03 03 03 03 03 03 03

7208- 03 03 03 03 03 03 03 03

完成了正确传送。

程序 8.26 编制思想说明:

(1) 源数据块的首地址放在 \$1A—\$1F 单元, 目的数据块的首地址放在 \$FA—\$FF 单元, 这些单元都是采用的零页地址单元。这样处理比用多组 LDA 和 STA 指令设置地址单元要简捷得多, 同时也使初始化部分省去了不少语句。

(2) 程序中的第 1, 2 两条指令, 是安排了两个计数器, 其中 X 变址寄存器作为修改源地址和目的地址的指针计数器, 为读取下一个数据作准备, 同时又兼作三个数据块是否全部送完的一个标志, 配合 CPX # \$04 来处理。Y 变址寄存器则作为传送数据块长度的计数器, 和 DEY, BNE 指令配合, 以便决定一个数据块的所有数据是否全部送完。

(3) 程序中有两个循环, 第一个是从 \$1002—\$1015 的

外循环，第二个是从\$1004—\$100E的内循环。它们完成的任务是不同的，内循环主要完成一个数据块（16个数据）的传送工作，而外循环则是完成三个数据块的传送工作。

（4）循环程序中包含了分支转移，在内循环中，控制循环次数，判断一个数据块的所有数据是否全部送完，主要由条件转移指令BNE决定的（是否分支转移）。而在外循环中判断三个数据块是否全部送完则主要由条件转移指令BEQ决定的（送完结束，未送完修改地址指针再循环）。

（5）程序中数据块的传送，主要选用了LDA(\$1A, X)和STA(\$FA, X)这两条指令，它们都是采用先变址(X)间接寻址的寻址方式，其优点是，使得多个数据块的处理变得简单、容易。一般来说，处理多个数据块的传送问题，用先变址间接寻址方式的指令比较理想。当然也不是绝对的，本章的下面几节内容还将介绍多个数据块的传送方法，那时，我们还可以看到不少的编程方法和技巧。

关于循环程序的设计，我们暂介绍到这里。现在，对本节的内容作一简单小结。

（1）循环程序的任务，在于完成大量的在处理形式上完全相同的重复性的计算工作。

（2）循环程序的结构，包括初始化、处理、循环控制、结束四个部份。各部份有机结合，协同动作，缺一不可。

（3）循环程序的应用，它几乎能解决名目繁多，形式各异的各种课题，是机器语言程序设计的核心和精华。

（4）循环包括分支，分支控制循环，两者互相联系又相互制约。灵活而准确地使用比较指令，特别是条件转移指令，是高质量设计循环程序所必不可少的。

（5）循环程序的嵌套，是指循环内还可以嵌套循环，

一个循环内可以包含几个循环，但应注意内外循环的层次要清楚，内循环允许并列几个小循环，但不允许交叉。

4. 子程序设计

在实际程序的编制过程中，常常遇到相同的计算或操作，或者，在一个程序中要多次处理重复计算的问题。例如代码转换、数据传送、图形处理、函数运算等等。

可以把这些相同的部份编制成一个独立的程序段，当遇到相同的计算或操作时，就转入这个程序段，从而省去对这些相同部份重新编制程序的工作。

我们把用来表示那些在功能上具有一定独立性、能够完成一定计算和操作并在程序的不同部位要被经常调用的程序段称作子程序。而把调用子程序的程序称为主程序。

大多数程序都是由一个主程序和几个子程序组成。在程序的适当位置安排调用点，就可以转向子程序，这称为转子。而当子程序执行之后再返回到主程序，这称为返主。转子指令JSR和返回指令RTS，可以实现主程序对子程序的正确调用和子程序到主程序的自动返回。因此，我们也可以说称以JSR指令调用，以RTS指令返回的程序段为子程序。

转子指令JSR也是一种无条件转移指令，但它们有区别。无条件转移指令控制程序转出以后就不再返回了；而转子指令控制程序转向子程序以后，当子程序执行完毕时还要返回主程序被打断处（叫做断点），即返回到调用它的指令的下一条指令去执行程序。保证子程序正确返回的是返回指令RTS。

为了准确地完成返回“断点”的任务，调用子程序的指令必须是能保留断点的指令；而子程序最后执行的必须是将

断点送入PC寄存器（程序计数器）的指令。6502的程序转移指令虽然都能使程序转入一个新的程序段操作，但没有保护断点的功能。只有JSR转子指令具有保护断点的能力，它将断点地址-1后送入堆栈S保存再转向子程序，而RTS返主指令正好完成自堆栈S中取一个地址，加1后赋给PC寄存器的操作，从而使子程序返回到主程序原来的断点处继续执行。由此可见，JSR和RTS的作用是转移程序的控制权，先转到子程序，再返回主程序。

上述概念较多，但很重要。否则就不能保证主程序对子程序的正确调用；也不能保证子程序到主程序的自动返回。而且对于处理多个子程序的嵌套来说，对于主程序和子程序的合理衔接问题，在程序设计中都将碰到麻烦。

让我们通过一些实例，来介绍子程序设计吧！读者也许会从这些实例中掌握子程序设计的方法和技巧。

例1：数据块转移

试用子程序改写简单程序设计中〔例1〕数据块转移程序。即要求将从\$2000开始的5个连续字节的内容转移到\$4000开始的5个连续字节内。

分析：

（1）整个程序设计可以分成一个主程序和一个子程序两个部份。

（2）主程序结构十分简单，首先将转移的个数（可以根据题意要求任意定，本例是5个）安排在累加器A中，然后调用子程序，结束。一共只要安排三条指令。

（3）子程序的任务是完成N个数据块的传送。取数、送数采用Y寄存器的绝对变址寻址方式指令，通过修改Y寄存器中的值（放了数据长度），达到修改地址指针的目的，并

用BPL指令控制数据是否取、送完。为了完成N个数据的传送，采用循环的方法。特别注意子程序的结束处不应忘记放一条返回指令RTS：

由上分析，容易写出程序8.27。

程序8.27：

1000-	A9	05	LDA	# \$ 05	} 主程序
1002-	20	06 10	JSR	\$ 1006	
1005-	60	RTS			
1006-	A8		TAY		} 子程序
1007-	88		DEY		
1008-	B9	00 20	LDA	\$ 2000,Y	
100B-	99	00 40	STA	\$ 4000,Y	
100E-	88		DEY		
100F-	10	F7	BPL	\$ 1008	
1011-	60		RTS		

运行前：* 2000: 01 02 03 04 05

运行后：* 1000G↙

* 4000.4004↙

* 4000—01 02 03 04 05

比较本程序8.27和简单程序设计中的〔例1〕程序8.1，可以看出：

(1) 程序8.27有更大的灵活性。因为只要改变数据块长度（即改变LDA # \$nn中的立即数），即可完成N个数（不超过FF）的数据块传送问题，而无需改动源程序。

(2) 程序8.27简捷和通用。不难看出程序8.1，若处理250个字节的数据块长度的传送，至少要写上500个语句（500条指令），而程序8.27除改变数据块长度外，源程序不加一条指令。

由此可知，程序编制中应注意简捷以节省内存，灵活以方便修改，通用以完成各种类似问题的处理。所有这些都是程序设计的质量要求。

例 2：确定正数的个数

数据长度放在\$6000单元，数据从\$6001开始存放，要求统计出数据块中正数的个数。

这个问题，我们在循环程序设计〔例 3〕中，作过类似的介绍，那里是确定负数的个数，因而有关程序设计思想的分析，大同小异，不再重复。但我们这里采用子程序方法设计，请注意结构和指令选用上的差别。见程序8.28。

程序8.28:

1000-	AD	00	60	LDA	\$ 6000	} 主程序
1003-	A0	00		LDY	# \$ 00	
1005-	20	09	10	JSR	\$ 1009	
1008-	60			RTS		
1009-	AA			TAX		
100A-	BD	00	60	LDA	\$ 6000,X	} 子程序
100D-	30	01		BMI	\$ 1010	
100F-	C8			INY		
1010-	CA			DEX		
1011-	D0	F7		BNE	\$ 100A	
1013-	84	06		STY	\$ 06	
1015-	98			TYA		
1016-	20	DA	FD	JSR	\$ FDDA	
1019-	60			RTS		

运行前: * 6000: 06 68 F2 87 30 59 2A

运行后: * 1000G ↗

* 04

或: * 1000G ↗

* 6 /

* 0006—04

例 3：找最大值

找一个无符号数据块中的最大值，数据块的长度放在\$6000单元，数据块起始地址为\$6001，最大值存于\$08单元。

求最大值问题，我们在循环程序的设计中（见例 2 程序 8.20）作过介绍，但是程序 8.20 虽有简短易懂的特点，可通用性不好，数据量一大即不能正确运行。故这里再介绍一个找最大值的程序 8.29。

程序 8.29:

1000-	A9	01	LDA	# \$ 01
1002-	85	06	STA	\$ 06
1004-	A9	60	LDA	# \$ 60
1006-	85	07	STA	\$ 07
1008-	AC	00 60	LDY	\$ 6000
100B-	20	11 10	JSR	\$ 1011
100E-	85	08	STA	\$ 08
1010-	60		RTS	
1011-	A9	00	LDA	# \$ 00
1013-	88		DEY	
1014-	08		PHP	
1015-	D1	06	CMP	(\$ 06),Y
1017-	B0	02	BCS	\$ 101B
1019-	B1	06	LDA	(\$ 06),Y
101B-	28		PLP	
101C-	D0	F5	BNE	\$ 1013
101E-	60		RTS	

程序说明:

1000: 把数据块首地址送入07和06单元
 1008: 数据块长度存Y寄存器
 100B: 转子程序
 100E: 存结果
 1010: 结束
 1011: 设最大值 = 0
 1013: Y - 1
 1014: 标志寄存器进栈, 保护Z标志
 1015: 下一个数 > 最大值吗?
 1017: 否, 转, 保持最大值不变
 1019: 是, 换最大值 → A
 101B: 标志寄存器出栈, 判Y - 1是否为0
 101C: 否, 循环
 101E: 是, 返回主程序
 运行前: *6000: 0A↙
 *6001: 75 44 A0 BF 54 C4
 8F DD 20 21↙源数据
 运行后: *1000G↙
 *8↙
 *0008—DD 找出最大值

现在, 我们介绍程序8.29的设计思想:

(1) 主程序放在\$1000—\$1010单元, 子程序放在\$1011—\$101E单元。在主程序中, 数据块首地址放在一个特定的地方, 例如, 放在\$06、\$07单元中, 前者存数据块首地址低8位, 后者放数据块首地址高8位, 而将数据块长度放在Y寄存器中。然后调用子程序, 存结果结束。子程序实际上是一个完成“找最大值”这个操作的独立程序段, 任务就是找最大值。

(2) 在子程序中, 把所得最大值结果放在 A 中, 这是为了主程序、子程序的衔接所做的规定, 有了这种规定, 对于另一个起始地址, 另一个长度的数据块, 找最大值时只需仿照此例, 把起始地址送入 07 和 06 单元, 把长度送入 Y 寄存器, 然后转入“找最大值”子程序, 即可在累加器 A 中得到最大值, 而无需对子程序作任何改动或修正。使这个子程序具有了通用性。

例 4: 搬家子程序

这里介绍一个实用的搬家子程序, 它可以方便地实现数据块的搬移任务。无论对单个数据块或是多个数据块; 无论对少量数据还是大量数据; 无论对数据记录还是图形信息都可以方便地实现搬移。如程序 8.30 所示。

程序 8.30:

1000-	A0	00	LDY	# \$ 00
1002-	A9	00	LDA	# \$ 00
1004-	85	3C	STA	\$ 3C
1006-	A9	60	LDA	# \$ 60
1008-	85	3D	STA	\$ 3D
100A-	A9	10	LDA	# \$ 10
100C-	85	3E	STA	\$ 3E
100E-	A9	60	LDA	# \$ 60
1010-	85	3F	STA	\$ 3F
1012-	A9	00	LDA	# \$ 00
1014-	85	42	STA	\$ 42
1016-	A9	70	LDA	# \$ 70
1018-	85	43	STA	\$ 43
101A-	20	2C	FE JSR	\$ FE2C
101D-	60		RTS	

将源数据块起始地址放在\$3C和\$3D单元，源数据块结束地址放在\$3E和\$3F单元，数据块的目的地址放在\$42和\$43单元，都是低位地址在前，高位地址在后，然后调用监控中的搬家子程序（入口地址为\$FE2C），即可完成单个数据块的搬移任务。但一定不要忘记应用本程序8.30时，注意将#00值放入Y寄存器中。

本例是将\$6000—\$6010单元的数据搬移至\$7000—\$7010单元中去。

对于任意单个数据块的搬移，只要改变\$3C—\$3F及\$42—\$43单元中的内容。

例如，将高分辨率第1页的图形搬至高分辨第2页图形区，即将\$2000—\$3FFF单元内容搬至\$4000—\$5FFF单元中去，只要改动以下单元值：

1007—20

100B—FF

100F—3F

1017—40

请注意这种搬移的信息量是很大的，它搬移了8192个存储单元的信息。而且其搬迁速度也是很快的，几乎瞬间完成。这是由于机器语言执行速度特别快的特点决定的。

对于多个数据块的搬移，仅仅用上述子程序8.30还不够，因为多个数据块的源地址有几处，搬移到的目的地址也有几个，从那几个地方搬，结果又放在何处，必须随时调正地址指针。因此，我们可以想像，编一个主程序，设置好调正的地址指针，然后不断调用上述子程序8.30，使之完成多个数据块的搬移任务，请看下面实例。

例5：三个数据块的搬移

将\$6000—\$600F, \$6100—\$610F, \$6200—\$620F单元的内容分别顺序搬移到\$7000—\$700F, \$7100—\$710F, \$7200—\$720F的连续单元中去。

这个问题, 我们在循环程序设计[例7]中作过介绍(见程序8.26), 现在我们用调用子程序8.30的方法处理。

先给出源程序8.31, 并看一下运行结果, 然后对程序作一说明。

程序8.31:

1000-	A9	60	LDA	# \$ 60	
1002-	8D	2C	10	STA	\$ 102C
1005-	8D	34	10	STA	\$ 1034
1008-	A9	70	LDA	# \$ 70	
100A-	8D	3C	10	STA	\$ 103C
100D-	20	25	10	JSR	\$ 1025
1010-	EE	2C	10	INC	\$ 102C
1013-	EE	34	10	INC	\$ 1034
1016-	EE	3C	10	INC	\$ 103C
1019-	20	25	10	JSR	\$ 1025
101C-	EE	2C	10	INC	\$ 102C
101F-	EE	34	10	INC	\$ 1034
1022-	EE	3C	10	INC	\$ 103C
1025-	A0	00	LDY	# \$ 00	
1027-	A9	00	LDA	# \$ 00	
1029-	85	3C	STA	\$ 3C	
102B-	A9	62	LDA	# \$ 62	
102D-	85	3D	STA	\$ 3D	
102F-	A9	10	LDA	# \$ 10	
1031-	85	3E	STA	\$ 3E	
1033-	A9	62	LDA	# \$ 62	

1035-	85	3F	STA	\$3F
1037-	A9	00	LDA	# \$ 00
1039-	85	42	STA	\$42
103B-	A9	72	LDA	# \$ 72
103D-	85	43	STA	\$43
103F-	20	2C FE	JSR	\$FE2C
1042-	60		RTS	

运行前:

*6000.600F

6000- 01 02 03 04 05 06 07 08

6008- 09 0A 0B 0C 0D 0E 0F 10

*6100.610F

6100- 09 0A 0B 0C 0D 0E 0F 10

6108- 01 02 03 04 05 06 07 08

*

*6200.620F

6200- 2F 2E 2D 2C 2B 2A 29 28

6208- 27 26 25 24 23 22 21 03

运行后:

*1000G

*7000.700F

7000- 01 02 03 04 05 06 07 08

7008- 09 0A 0B 0C 0D 0E 0F 10

*7100.710F

7100- 09 0A 0B 0C 0D 0E 0F 10

7108- 01 02 03 04 05 06 07 08

*7200.720F

7200- 2F 2E 2D 2C 2B 2A 29 28

7208- 27 26 25 24 23 22 21 03

程序8.31说明:

(1) \$1025—\$1042就是本节〔例4〕中介绍的搬家子程序(\$1000—\$101D), 它们的指令用的完全一样, 只是个别地址的内容不同, 存贮空间相异而已。其功能已如前所述, 是一个独立的完成单个数据块搬家的子程序。

(2) \$1000—\$1024是主程序, 它完成第一个数据块首地址、末地址、目标地址的确定和搬迁任务(\$1000—\$100F), 又完成第二、三两个数据块地址指针的调正和搬移任务(分别为\$1010—\$1010—\$101B, \$101C—\$1042)。其中\$1000—\$100F程序段对整个程序的正确运行至关重要, 否则重复运行本程序将发生错误。

(3) 本程序执行过程共分三段: ①\$1000—\$100F—\$1025—\$1042; ②\$1010—\$101B—\$1025—\$1042; ③\$101C—\$1042。可以看出是一个主程序几次重复调用子程序的结构, 最后一次省去了转子指令JSR, 而直接和子程序连成一片, 从而完成了主程序和子程序的正确衔接, 这是编制程序时应予注意的。

例6: 多个子程序的调用

在子程序的设计中, 大多数由一个主程序和几个子程序组成, 每一个子程序完成的功能不一样, 它们之间又具有相互的独立性。以本节〔例5〕为例, 程序8.31中修改地址指针使内存单元加1的功能出现二处, 可以改成一个独立的子程序, 从而使程序8.31变成一个主程序调用二个子程序的结构, 见程序8.32。

程序8.32:

1000-	A9	60		LDA	# \$60
1002-	8D	20	10	STA	\$1020

1005-	8D	28	10	STA	\$ 1028
1008-	A9	70		LDA	# \$ 70
100A-	8D	30	10	STA	\$ 1030
100D-	20	19	10	JSR	\$ 1019
1010-	20	37	10	JSR	\$ 1037
1013-	20	19	10	JSR	\$ 1019
1016-	20	37	10	JSR	\$ 1037
1019-	A0	00		LDY	# \$ 00
101B-	A9	00		LDA	# \$ 00
101D-	85	3C		STA	\$ 3C
101F-	A9	62		LDA	# \$ 62
1021-	85	3D		STA	\$ 3D
1023-	A9	10		LDA	# \$ 10
1025-	85	3E		STA	\$ 3E
1027-	A9	62		LDA	# \$ 62
1029-	85	3F		STA	\$ 3F
102B-	A9	00		LDA	# \$ 00
102D-	85	42		STA	\$ 42
102F-	A9	72		LDA	# \$ 72
1031-	85	43		STA	\$ 43
1033-	20	2C	FE	JSR	\$ FE2C
1036-	60			RTS	
1037-	EE	20	10	INC	\$ 1020
103A-	EE	28	10	INC	\$ 1028
103D-	EE	30	10	INC	\$ 1030
1040-	60			RTS	

程序8.32说明:

(1) 从程序8.32看出,共有四次调用子程序(见\$100D—\$1018),调用的是两个子程序(\$1019—\$1036和\$1037

—\$1040)。

(2) 程序8.32中, 有子程序嵌套的情况。例如, 当主程序工作时, 首先转入子程序 1 (\$1019—\$1036), 而子程序 1 工作时又转入子程序 2 (\$1033—\$1036), 当程序 2 (调用监控搬家子程序) 工作完毕后返回子程序 1, 子程序执行完毕后再返回主程序。在这个过程中, 子程序 1 相对于子程序 2 而言就处在主程序的地位, 这就叫做子程序 2 嵌套。

例 7: 输入码转换

在键盘上按下三次 10 进制数, 将其转换成 16 进制数。

如: 按键 254 (10 进制数)

显示 FE (16 进制数)

按键 128 (10 进制数)

显示 80 (16 进制数)

分析:

关于从键盘上按下一位 10 进制数, 将其转换成 16 进制数的问题, 我们曾在分支程序设计一节中作过介绍, 见程序 8.13。

现在的问题是, 要从键盘上按下三位 10 进制数, 将其转换成 16 进制数, 除了可以调用前面的程序 8.13 (它的任务是把 10 进制数转换成 2 进制数) 外, 还应多次调用, 以完成百位数、十位数、个位数均转换成 2 进制数的问题, 同时, 还要有另外一个子程序, 它完成乘 10 加数的功能, 并且不止一次调用。

源程序如 8.33 所示。

程序 8.33:

1000~	20	15	10	JSR	\$ 1015
1003~	85	06		STA	\$ 06

1005-	20	15	10	JSR	\$ 1015
1008-	20	23	10	JSR	\$ 1023
100B-	20	15	10	JSR	\$ 1015
100E-	20	23	10	JSR	\$ 1023
1011-	20	DA	FD	JSR	\$ FDDA
1014-	60			RTS	
1015-	20	1B	FD	JSR	\$ FD1B
1018-	C9	B0		CMP	# \$ B0
101A-	90	F9		BCC	#1015
101C-	C9	BA		CMP	# \$ BA
101E-	B0	F5		BCS	\$ 1015
1020-	29	0F		AND	# \$ 0F
1022-	60			RTS	
1023-	85	07		STA	\$ 07
1025-	A5	06		LDA	\$ 06
1027-	0A			ASL	
1028-	0A			ASL	
1029-	65	06		ADC	\$ 06
102B-	0A			ASL	
102C-	65	07		ADC	\$ 07
102E-	85	06		STA	\$ 06
1030-	60			RTS	

程序说明:

1000: 转子, 取百位数转成 2 进制在 A

1003: 存 06 单元作为中间结果

1005: 转子, 取拾位数转成 2 进制在 A

1008: 转乘 10 加数子程序, 中间结果有 (06)

100B: 转子, 取个位数转 2 进制在 A

100E: 转乘 10 加数子程序, 中间结果在 (06)

1011: 显示结束
 1015—1022: 10进制数转2进制数子程序
 1023: 暂存十位, 个位数
 1025: 调中间结果
 1027: 得2 A
 1028: 得4 A
 1029: 得5 A
 102B: 得10 A
 102C: 乘10加数
 102E: 存结果到工作单元
 1030: 返回
 运行: *1000G ↙
 *254 ↙
 *F E
 *1000G ↙
 *120 ↙
 *80

程序8.33结构说明:

(1) \$1015—\$1022是一个子程序, 它完成10进制数转换成2进制数的工作, 由于本程序要处理百位数、十位数、个位数转换成2进制的工作, 所以三次调用了这个子程序。

(2) \$1023—\$1030是另一个子程序, 它完成乘10加数的工作, 共要调用两次。

(3) 所以本程序的两个子程序构成两重子程序结构。

例8: 比较两个字符串是否相同

比较两个ASCII字符串, 看其是否相同。字符串长度在存贮单元6000中, 两个字符串的起始地址各为6100和6200, 如果这两个字符串相同, 将00放入6001单元, 反之不同放FF

标志。

示范题:

a) (6000) = 03	字符串长度
(6100) = 43	"C"
(6101) = 41	"A"
(6102) = 54	"T"
(6200) = 44	"D"
(6201) = 41	"A"
(6202) = 54	"T"

结果: (6001) = FF

b) (6000) = 03	
(6100) = 43	"C"
(6101) = 41	"A"
(6102) = 54	"T"
(6200) = 43	"C"
(6201) = 41	"A"
(6202) = 54	"T"

结果: (6001) = 00

源程序如8.34所示。

程序8.34:

1000-	AC	00	60	LDY	\$ 6000
1003-	20	0A	10	JSR	\$ 100A
1006-	8D	01	60	STA	\$ 6001
1009-	60			RTS	
100A-	A2	FF		LDX	# \$ FF
100C-	88			DEY	
100D-	B1	06		LDA	(\$ 06), Y
100F-	D1	08		CMP	(\$ 08), Y
1011-	D0	05		BNE	\$ 1018

1013-	98	TYA
1014-	D0 F6	BNE \$100C
1016-	A2 00	LDX # \$00
1018-	8A	TXA
1019-	60	RTS

程序8.34说明:

(1) 主程序见\$1000—\$1009,字符串长度放Y寄存器,转字符串比较子程序,送结果,结束。

(2) 子程序见\$100A—\$1019,首先置字符串不同标志FF→X,Y-1,逐字比较两字符串,若不相同返回;若相同,看每个字符是否都比较完,是的取00标志→X,返回,否则再循环上去。

(3) 在运行本程序前,应将两字符串起始地址存入06,07单元和08,09单元,即*6:00 61 00 62,而在6100和6200开始存放字符串的ASCII码,然后1000G↵,即可看到结果。

(4) 本程序在比较两个字符串时,是采用由后向前比的。

有关子程序的设计,我们就介绍到这里。从以上所举实例,可以看出在实际程序设计中,有一些程序需要反复使用,为了方便操作,简化设计,我们常采用子程序的方法,使那些在功能上一样,处理问题相同,并且有一定独立性的重复操作部份编在一起,只要注意正确调用和返回,主程序与子程序衔接完整即可。

5. 堆栈程序设计

堆栈是计算机信息处理中的一个重要概念,它实际上是

在随机存储器RAM中开辟的某个特定的存储区域，用于对一些重要数据的暂存和保护断点地址。这个存储区域有这样的特点：数据可以连续存入，断点地址可以得到保护，不会发生冲掉已存入数据和断点地址的情况，并且严格按照先存入的数据后取出来的规则工作。这种存储结构就叫做堆栈(Stack)。

中华学习机规定堆栈设置在第1页位置上(第1页是指地址码高字节为01的存储区域，通常又称零页地址)，即只能把\$0100—\$01FF这256个单元用作堆栈。由于堆栈中的数据存取是按照“先进后出，后进先出”的顺序进行的，因而还必须有一个“指示器”，随时指出栈顶地址的变化，指出栈区中存入的数据已经堆到了那里，或者说，其作用是指示堆栈中允许存取操作的当前地址。并且必须具有在数据出栈时有自动加1的功能，而在数据进栈时有自动减1的功能，这个指示器叫做堆栈指针寄存器，简称栈指示器，记作S。它是一个8位寄存器，即S表示堆栈地址的低8位，而高8位固定为01。

栈的操作只有两种：一是压入(PUSH)，一是弹出(POP)，每压入(或弹出)一个内容的字节，栈指针S就减1(或加1)，微处理器就是根据S指出的位置(栈顶地址)去存取数据，这样，就保证了“先进后出”的原则去转子程序，保护断点，恢复断点，保护现场及恢复现场。

顺带说一句，堆栈在使用过程中，栈指针S总是指向栈顶的一个空单元，而S的初值通常设定在栈底位置01FF处。

关于堆栈的几条指令简述如下：

(1) 累加器进栈指令PHA—— $A \rightarrow MS, S-1 \rightarrow S$

(2) 累加器出栈指令PLA—— $S+1 \rightarrow S, MS \rightarrow A$

(3) 标志寄存器进栈指令PHP—— $P \rightarrow MS, S-1 \rightarrow S$

(4) 标志寄存器出栈指令PLP—— $S+1 \rightarrow S, MS \rightarrow P$

它们各自的操作码分别为：48,68,08,28。

例1：数据块的倒序传送

例如，将\$6001—\$6010单元的16个数据，按相反次序传送到\$7001—\$7010的连续单元中去。程序如8.35所示。

程序8.35：

1000-	A2 01	LDX	# \$ 01
1002-	A0 10	LDY	# \$ 10
1004-	BD 00 60	LDA	\$ 6000,X
1007-	48	PHA	
1008-	E8	INX	
1009-	88	DEY	
100A-	D0 F8	BNE	\$ 1004
100C-	A2 01	LDX	# \$ 01
100E-	68	PLA	
100F-	9D 00 70	STA	\$ 7000,X
1012-	E8	INX	
1013-	E0 11	CPX	# \$ 11
1015-	D0 F7	BNE	\$ 100E
1017-	60	RTS	

运行前：

* 6001,6010

6001- 01 02 03 04 05 06 07

6008- 08 09 0A 0B 0C 0D 0E 0F

6010- 10

运行后：

* 1000G

*7001,7010

7001- 10 0F 0E 0D 0C 0B 0A

7008- 09 08 07 06 05 04 03 02

7010- 01

程序8.35的几点说明:

(1) 初始化条件: 变址指针存于X寄存器, 开始为01→X, 数据长度计数器Y。

(2) 这是一个双循环结构的程序设计。第一个循环, \$1004—\$100B, 第二个循环\$100E—\$1017。前者取一个数, 进栈保存, $X+1 \rightarrow X$, $Y-1 \rightarrow Y$, 只要16个数未取完, 就循环上去(\$1004), 不断取数, 不断进栈保存, 当全部源数据取完, 进栈保存好后, 转入第二个循环。后者, 把第一个循环在栈中保存的数, 逐一取出来, 放入目的地址(开始\$7001), 用X作计数器, 并控制循环次数, 一直到所有数(\$10=16)全部顺次出栈, 并在目的地址一一保存后结束。

(3) 本例是学习堆栈概念, 存取原则的一个典型实例。若用中华学习机单步执行命令S, 就可以看到数据如何一个一个地进栈, 又如何执行后进先出的原则一个一个地出栈的。

(4) 若取本题为255个数据传送, 也是非常方便的, 你有最简便的方法吗?事实上, 只要将\$1003单元改为FF, \$1014单元改为00, 即可。

例2: N个数据的交换

例如, 将\$6000—600F单元的内容和\$7000—\$700F单元的内容交换。见程序8.36。

程序8.36:

1000-	A2 00	LDX	# \$ 00
1002-	BD 00 60	LDA	\$ 6000,X
1005-	48	PHA	
1006-	BD 00 70	LDA	\$ 7000,X
1009-	9D 00 60	STA	\$ 6000,X
100C-	68	PLA	
100D-	9D 00 70	STA	\$ 7000,X
1010-	E8	INX	
1011-	E0 10	CPX	# \$ 10
1013-	D0 ED	BNE	\$ 1002
1015-	60	RTS	

运行前:

* 6000.600F

6000- 50 51 52 53 54 55 56 57

6008- 58 59 5A 5B 5C 5D 5E 5F

* 7000.700F

7000- 12 13 14 15 16 17 18 19

7008- 1A 1B 1C 1D 1E 1F 20 21

运行后:

* 1000G

* 6000.600F

6000- 12 13 14 15 16 17 18 19

6008- 1A 1B 1C 1D 1E 1F 20 21

* 7000.700F

7000- 50 51 52 53 54 55 56 57

7008- 58 59 5A 5B 5C 5D 5E 5F

几点说明:

(1) 程序8.36非常简短, 并且只用了一重循环, 就可以完成16个数据的互换。事实上本程序通用性也好, 只要将

C PX # \$10 中的立即数改成任意数 N ($N \leq FF$)，即可完成 N 个数据的互换。不仅如此，源地址（本例是6000），目的地址（本例是7000），也可以任意改变，从而可以完成从任何一个区域向另外任何一个区域的数据交换。

(2) 程序8.36中数据的交换，实际上是借助了一个中间存贮区域作为数据交换的暂存空间，而这个暂存空间不是别的，正好是堆栈。其思路是，取一个数，进栈保存，调另一个数，存入原来数的地方（交换），数据出栈，再放数又一次交换，指针加1，只要未将全部数据交换后，即循环上去，直到结束。

(3) 请注意，本程序8.36只能用PHA，PLA指令，而不能用PHP，PLP指令。为什么？请读者自行分析。

例3：存寄存器程序

编制程序必须使用寄存器，这是众所周知的事。有时候需要保存寄存器中原有的值，以便在程序的其它部份继续使用，这就需要将寄存器的值送到某些单元保存，这一任务通常被称“保护现场”。例如，将寄存器A、X、Y、P及S的值分别保存在\$45、\$46、\$47、\$48、\$49五个零页地址，程序如下：

```
ORG $FF4A
```

程序由\$FF4A开始存放。

我们用单步执行命令S来看下面的实例：

```
* FF4AS
```

```
FF4A-85 45 STA $45
```

```
A=00 X=01 Y=30 P=70 S=01
```

这是进入\$FF4A单元开始的随机状况，A、X、Y、P及S已经存贮的值。

* 45

0045—00

说明执行STA \$45指令后,累加器A中的内容00,已经送入\$45单元,即 (A) → \$45

* S

FF4C—86 46 STX \$46

A = 00 X = 01 Y = 30 P = 70 S = 01

* 46

0046—01

在执行STX \$46指令后,X寄存器的内容01送入了\$46单元,即 (X) → \$46

* S

FF4E—84 47 STY \$47

A = 00 X = 01 Y = 30 P = 70 S = 01

* 47

0047—30

Y寄存器之值30送入\$47,即 (Y) → \$47

* S

FF50—08 PHP

A = 00 X = 01 Y = 30 P = 70 S = 00

这时P寄存器的状况进栈 (P) → 堆栈,栈指示器S - 1 → S,所以此时S = 00。

* S

FF51—68 PLA

A = 70 X = 01 Y = 30 P = 70 S = 01

执行PLA指令后,刚才进栈保存的P值(70)出栈送入累加器A中,所以此时A = 70,由于执行了一次出栈指令,

$S + 1 \rightarrow S$ ，所以此时 $S = 01$ 。

* S

FF52—85 48 STA \$48

A = 70 X = 01 Y = 30 P = 70 S = 01

* 48

0048—70

执行STA \$48指令后，累加器A中的值70送入\$48单元，实际上就是P寄存器值70存入\$48单元，即 $(P) \rightarrow \$48$

可见，存状态标志寄存器P的值是经过累加器A中转的（倒换）。即

PHP ; $(P) \rightarrow$ 堆栈

PLA ; (堆栈) \rightarrow A

STA \$48 ; $(P) \rightarrow \$48$

* S

FF54—BA TSX

A = 70 X = 03 Y = 30 P = 70 S = 01

这时栈指示器S的值01，送入X寄存器，即 $(S) \rightarrow X$ ，注意原来X中的值发生变化（由01 \rightarrow 03）。

* S

FF55—84 69 STX \$49

A = 70 X = 03 Y = 30 P = 70 S = 01

* 49

0049—01

执行STX \$49以后，\$49单元中存放了 $S = 01$ 的值，即 $(S) \rightarrow \$49$ 。

从这里可以看出堆栈指针寄存器S中的值要经过变址寄存器X中转的。即

TSX , (S) → X

STX \$49; (S) → \$49

* S

FF57—D8 CLD

A = 70 X = 03 Y = 30 P = 70 S = 01

清10进制工作方式

* S

FF58—60 RTS

A = 70 X = 03 Y = 30 P = 70 S = 01

返回。

注：存寄存器程序我们直接引用3监控中（\$FF4A—\$FF58一段子程序，从中可以看出，A、X、Y、P、S原来的值00、01、30、70、01已经分别存入\$45、\$46、\$47、\$48、\$49各零页地址单元中。

例4：恢复寄存器程序

在完成某项任务之后，有时需要恢复寄存器在执行本项任务之前的值，这就要求设计一个与上述程序（存寄存器程序）相反的程序，这一程序被称为“恢复现场”。例如，已知A、X、Y、P及S的值分别为70、03、30、70、01，执行从\$FF3F—\$FF49的程序段，即可恢复寄存器A、X、Y及P的值，但不能恢复堆栈指针。

ORG \$FF3F

程序由\$FF3F开始存放。

* FF3FS

FF3F—A5 48 LDA \$48

A = 70 X = 03 Y = 30 P = 70 S = 03

* 48

0048—70

取原存的P寄存器值。

* S

FF41—48 PHA

A=70 X=03 Y=30 P=70 S=02

(P) →堆栈, S-1→S

* S

FF42—A5 45 LDA \$45

A=70 X=03 Y=30 P=70 S=02

* 45

0045—70

恢复累加器的值。

* S

FF44—A6 46 LDX \$46

A=70 X=03 Y=30 P=70 S=02

* 46

0046—03

恢复X寄存器的值。

* S

FF46—A4 47 LDY \$47

A=70 X=03 Y=30 P=70 S=02

* 47

0047—30

恢复Y寄存器的值。

* S

FF48—28 PLP

A=70 X=03 Y=30 P=70 S=03

恢复状态标志。 $S + 1 \rightarrow S$ 。

* S

FF49—60 RTS

A = 70 X = 03 Y = 30 P = 70 S = 03

返回。

本程序与“存寄存器程序”配合使用，因此，它们可以在同一个大程序中，这里不需要对存贮器再作重复的定义。

例5：保护和恢复现场的方法

前面一节在谈到子程序设计时，曾指出当主程序转入子程序时，若主程序中已经用到的寄存器（指X、Y、A、P）在子程序中也要用到，而且当子程序返回主程序后，主程序还要继续用这些寄存器。这时，为了防止寄存器内容被冲掉，就有一个保护现场和恢复现场的问题。即：在转子程序前应首先把这些寄存器的状态存入堆栈中保存起来（保护现场），然后执行子程序的主体部份，最后再返回；而当返回主程序之前，还要从堆栈中取出它们，以恢复寄存器原内容（恢复现场）。

下面的程序指出了保护现场和恢复现场的方法。

PHP , P进栈, 保留P

PHA , A进栈, 保留A

TXA , X→A

PHA , X进栈, 保留X

TYA , Y→A

PHA , Y进栈, 保留Y

⋮

PLA , 最后进栈的Y先出栈, 进入A

TAY , A→Y, 恢复Y
 PLA , X出栈, X→A
 TAX , A→X, 恢复X
 PLA , A出栈, 恢复A
 PLP , P出栈, 恢复P

⋮

由此程序可以看出：

(1) 保留现场时各寄存器进栈的顺序和恢复现场时各寄存器出栈的顺序恰好相反。这是因为各寄存器状况在栈中保存又从栈中出来，必须按先进后出的原则处理，这是堆栈的结构决定的。

(2) 堆栈指令中有P和A的进栈指令和出栈指令。因此，P寄存器和A累加器需要进栈时直接使用PHP和PHA指令；而且出栈时也可直接用PLP和PLA指令。

(3) 6502指令中没有X和Y寄存器的进出栈指令，因此它们的进栈出栈，需要经累加器A中转。即：X和Y寄存器要进栈时，必须先用TXA或TYA指令，把进栈的内容发送到累加器A中，然后再使用PHA指令使X或Y进栈；出栈时，则先用PLA指令，将Y或X的内容送入累加器A中（出栈），然后再用TAX指令恢复X，或用TYA指令恢复Y。

例6：标志寄存器P进出栈的一个实例

关于求最大值的问题，我们已在循环程序设计（例2，程序8.20）和子程序设计（例3，程序8.29）中作过介绍，其中程序8.29中就是一个标志寄存器P进出栈的实例，不妨将它重新写出，并分析一下它保护的是什么现场，有什么用，恢复的是什么内容，应该在什么时候用。

程序8.29的主程序：

1000-	A9 01	LDA	# \$ 01
1002-	85 06	STA	\$ 06
1004-	A9 60	LDA	# \$ 60
1006-	85 07	STA	\$ 07
1008-	AC 00 60	LDY	\$ 6000
100B-	20 11 10	JSR	\$ 1011
100E-	85 08	STA	\$ 08
1010-	60	RTS	

前面 4 条指令，是把源数据块首地址送入\$07和\$06单元，第 5 条指令将数据块长度存入 Y 寄存器，以后是转子程序，待子程序执行完毕后存结果再结束。

程序 8.29 的子程序：

1011-	A9 00	LDA	# \$ 00
1013-	88	DEY	
1014-	08	PHP	
1015-	D1 06	CMP	(\$ 06), Y
1017-	B0 02	BCS	\$ 101B
1019-	B1 06	LDA	(\$ 06), Y
101B-	28	PLP	
101C-	D0 F5	BNE	\$ 1013
101E-	60	RTS	

这段子程序的任务是找最大值。我们主要看 \$1013—\$101D 这段循环，而特别注意循环程序中标志寄存器 P 进栈指令 PHP 和出栈指令 PLP 的安排。循环前 A 中的内容为 00，这是一个假想的最大值（也可以取数据块中的第一个数据）。

执行 DEY 指令使 $Y - 1 \rightarrow Y$ ，立即执行 PHP 指令，这样就使 P 进栈，从而保存了零标志 Z，如果执行 DEY 指令后结果为 0，则 Z 被置 1；反之 Z 被置为 0。这就为以后数据块长度

是否被比较完留下了一个判断标志。当执行CMP (\$06)，Y指令后，判断下一个数是否大于最大值，若是的，换最大值放入A，不是则保留最大值不变，但不管是与否，最后都转向执行PLP指令，这条指令是P出栈，给出Z标志，再用BNE判别是否Z = 0，是0，返主程序（通过\$101E RTS指令），非0继续转向\$1013处循环再比。然后Y - 1 → Y，再使P进栈，每次均保留P的现场，一直到A是最大数，使P出栈，恢复现场，并当P中的Z标志位为1时，结果为0（所有数据均比较完），结束子程序操作，返回主程序的断点地址\$100E，存结果结束停机。

例7：输出码转换程序

将MPU的A中8位2进制转换成3位10进制数，并在显示器上显示（送显示器的字符必须用ASCII码）。

示范题：a) A = FF (16进制数)

结果：显示255 (10进制数)

b) A = 80 (16进制数)

结果：显示128 (10进制数)

c) A = 64 (16进制数)

结果：显示100 (10进制数)

我们先给出完成本题的程序8.37，然后再给予说明。

程序8.37：

1000-	A2 00	LDX	# \$00
1002-	C9 64	CMP	# \$64
1004-	90 06	BCC	\$ 100C
1006-	E9 64	SBC	# \$64
1008-	E8	INX	
1009-	4C 02 10	JMP	\$ 1002

100C-	20	24	10	JSR	\$ 1024
100F-	A2	00		LDX	# \$ 00
1011-	C9	0A		CMP	# \$ 0A
1013-	90	06		BCC	\$ 101B
1015-	E9	0A		SBC	# \$ 0A
1017-	E8			INX	
1018-	4C	11	10	JMP	\$ 1011
101B-	20	24	10	JSR	\$ 1024
101E-	09	B0		ORA	# \$ B0
1020-	20	F0	FD	JSR	\$ FDF0
1023-	60			RTS	
1024-	48			PHA	
1025-	8A			TXA	
1026-	09	B0		ORA	# \$ B0
1028-	20	F0	FD	JSR	\$ FDF0
102B-	68			PLA	
102C-	60			RTS	
102D-	A9	FF		LDA	# \$ FF
102F-	20	00	10	JSR	\$ 1000
1032-	20	48	F9	JSR	\$ F948
1035-	A9	80		LDA	# \$ 80
1037-	20	00	10	JSR	\$ 1000
103A-	20	48	F9	JSR	\$ F948
103D-	A9	64		LDA	# \$ 64
103F-	20	00	10	JSR	\$ 1000
1042-	60			RTS	

运行: * 102DG↙

显示: 255 128 100

程序8.37说明:

(1) 本程序结构比较复杂,但可以分为三大段: \$1000—\$1023为子程序 1, \$1024—\$102C为子程序 2, \$102D—\$1042为主程序。在主程序中,既有调子程序 1 的操作,又有调监控中子程序(入口地址\$F948)的操作。而子程序 1 中又包含调子程序 2 的操作,这是一个二重子程序的嵌套结构。而从主程序看,则是一个主程序调三个子程序的复杂操作。

(2) 主程序中,首先将A赋值FF(即最初输入的16进制数),然后转子程序 1,完成16进制数到3位10进制数的转换,结果为255,再调\$F948的监控子程序,它的功能空一格。然后输入第2个16进制数80,转子程序 1,完成\$80向3位10进制的转换,结果为128,再次调\$F948子程序空一格,最后输入第3个16进制数64,转子程序 1,得10进制数100,主程序结束。由此可见,主程序是为了不断送数到累加器A中,完成三个16进制数的赋值工作。

(3) 子程序 1 的结构比较复杂,我们用框图来说明。

子程序 1 框图:

(4) 子程序 1 在执行过程中还要调用子程序 2,从这个角度上说,子程序 1 又相当于是子程序 2 的主程序,这样的主从关系应该明确。这里,分析一下子程序 2,它是在数据换成ASCII码之前,首先把累加器A的内容(实际上是余数)进栈保护(保护现场),所以返主以后(此时应理解返回子程序 1,子程序 1 为主程序),主程序(即子程序 1)还要再接着使用这个余数,因此,A中的内容不能由子程序(子程序 2)的执行而丢失。同时,子程序 2 也要用A来存放待显示的数,所以就需要先把A保护进栈,然后再执行子程序 2 的内容。此外,还应注意,在子程序 2 内容执行完毕

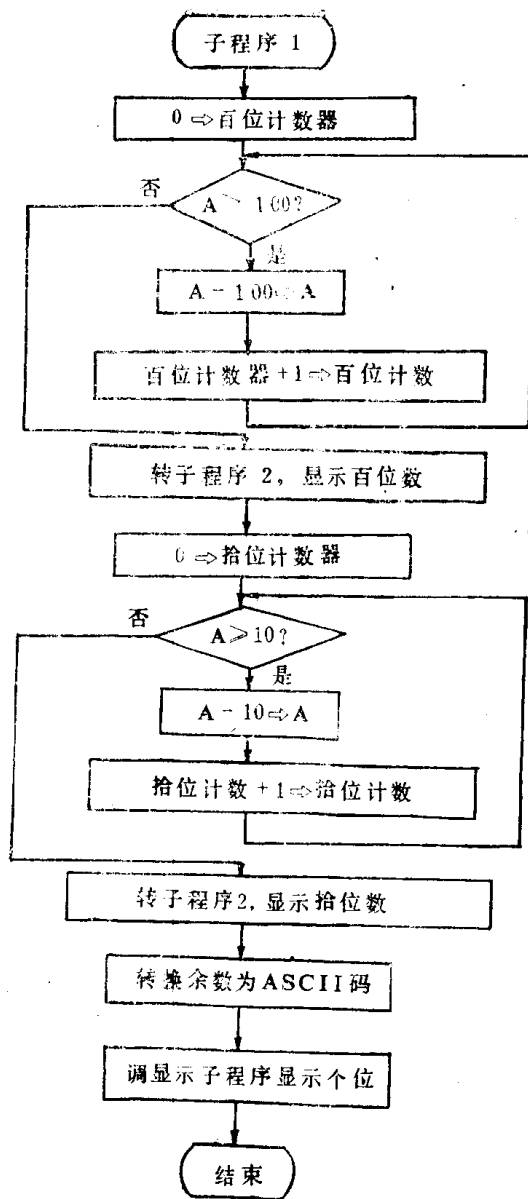


图8.1

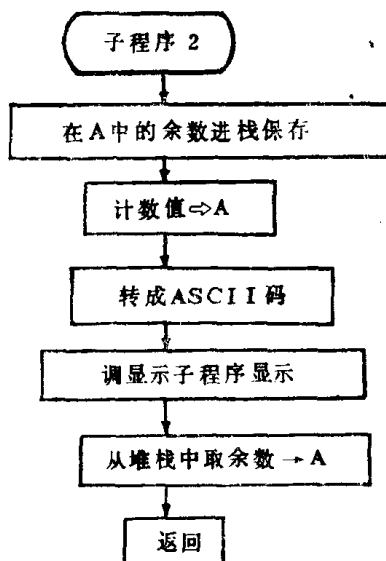


图8.2

返主（子程序 1）之前，必须用出栈指令使恢复 A（余数出栈恢复的内容（恢复现场），以返主后供主程序（子程序 1）继续使用。

（5）作为一般情况，如果主程序和子程序使用的寄存器中，发生冲突的不只是 A，还有 X、Y、P 也发生冲突，那么在子程序中还要对 X、Y、P 的状况加上保护和恢复。因此，转入子程序时应当首先分析一下是否需要保护现场，若需要，那么应注意是哪几个寄存器需要保护，以及在什么时候退回保护。只有这样仔细的加以分析和妥善的处理，才能使程序得以正确执行。本题只有 A 的情况，没有 X、Y、P 的情况。

关于堆栈程序的设计，我们就介绍到这里。总之，要掌握堆栈的概念、结构、先进后出原则；掌握断点概念、保护和恢复现场的方法；掌握子程序和堆栈之间的关系；掌握 A、

X、Y、P进栈出栈的指令和方法。这是学习 本节 的重点所在。

6. 常用监控子程序的调用

中华学习机系统程序中，有很多可供用户调用的系统子程序，其中最有用的要算监控中的子程序，这些子程序不仅列出了程序入口地址、名称、操作功能，而且还列出了子程序所需数据并预置在那个寄存器以及子程序执行过程中使用和变更了哪些寄存器。如果我们能充分利用这些子程序，这对于减少用户编写程序的工作量和节约内存空间，都有实际意义。不仅如此，这些子程序简短独立，功能完整，结构合理，简炼灵活，如能合理巧用，往往起到事半功倍的作用，从而使程序清晰易懂，并能提高程序的执行速度。

本节介绍常用监控子程序及其使用实例。

例 1：将26个英文大写字母A—Z输出给打印机。

输出一个字符子程序的入口地址为\$FDED，该子程序既可以将字符输出至屏幕，也可以将字符输出给打印机。若输出给打印机，则应先将设备码C100存入\$36和\$37中。\$36单元存放设备码低字节\$00；\$37单元存放设备码高字节\$C1。然后调用入口为\$FDED的输出字符子程序，这样就可以把累加器A中存放的字符送往所指定的外部设备（本例是打印机）。当该子程序执行完毕后，又会回到刚才调用它的程序。源程序见8.38。

程序8.38：

0300-	A9 00	LDA	# \$ 00
0302-	85 36	STA	\$ 36
0304-	A9 C1	LDA	# \$ C1

0306-	85	37	STA	\$ 37
0308-	A9	C1	LDA	# \$ C1
030A-	20	ED FD	JSR	\$ FDED
030D-	18		CLC	
030E-	69	01	ADC	# \$ 01
0310-	C9	DB	CMP	# \$ DB
0312-	D0	F6	BNE	\$ 030A
0314-	60		RTS	

程序说明:

0300: }
 0302: } 设置送往打印
 0304: } 机的设备码
 0306: }

0308: 取字符“A”的ASCII码

030A: 显示

030D: 清进位

030E: 指向下一个

0310: 打完了吗? “Z”的ASCII码为DA

0312: 没有, 再打

0314: 打完结束

例 2: 在屏幕上连续显示15个星号: “*”

开机后, 通常\$36和\$37的内容为F0和FD。\$FDF0就是一个设备码, 它的功能是将输出字符送至显示器, 因此, 不需要重新设置设备码, 而直接调用\$FDF0的子程序, 即可将累加器A中的字符输出到屏幕。源程序见8.39:

程序8.39:

0300-	A2	0F	LDX	# \$ 0F
0302-	A9	AA	LDA	# \$ AA
0304-	20	F0 FD	JSR	\$ FDF0
0307-	CA		DEX	

```

0308-   D0   F8           BNE    $ 002
030A-   60           RTS

```

程序说明:

0300: 置初值15
 0302: 调 “*” 的ASCII码
 0304: 显示
 0307: $X - 1 \rightarrow X$
 0308: 未减完, 再循环显示
 030A: 减完, 结束

例 3: 用反相显示方式在屏幕上第一列显示16个 “+”

号

置反相显示方式的子程序入口地址为\$FE80, 该子程序置屏幕为白底黑字方式。

置正相显示方式的子程序入口地址为\$FE84, 该子程序可将原反相显示方式转为正相显示方式, 即置屏幕为黑底白字方式。

源程序见8.40。

程序8.40:

```

0300-   20   80   FE       JSR    $FE80
0303-   A0   10           LDY    # $10
0305-   20   8E   FD       JSR    $FD8E
0308-   A9   AB           LDA    # $AB
030A-   20   F0   FD       JSR    $FDF0
030D-   88           DEY
030E-   D0   F5           BNE    $0305
0310-   20   84   FE       JSR    $FE84
0313-   60           RTS

```

程序说明:

0300: 设置反相显示方式

0303: 置初值16
 0305: 回车换行
 0308: “+”号的ASCII码
 030A: 显示
 030D: 次数减1
 030E: 未完,继续
 0310: 恢复正常显示方式
 0313: 结束

程序8.40中用了调用\$FD8E的子程序,该子程序将回车换行的控制字符送往现行输出设备(本例是屏幕),以保证本题“+”号在一列上显示而不是一行上显示。

例4: 将\$03F0—\$03FF单元中的两位16进制数显示出来。

输出两位16进制字符子程序的入口地址是\$FDDA,该子程序的功能是将累加器A中的内容,按两位16进制数的形式输出到现行输出设备。

源程序见8.41。

程序8.41:

0300-	A9	F0	LDA	# \$F0
0302-	85	03	STA	\$03
0304-	A9	03	LDA	# \$03
0306-	85	04	STA	\$04
0308-	A0	00	LDY	# \$00
030A-	B1	03	LDA	(\$03),Y
030C-	20	DA FD	JSR	\$FDDA
030F-	A9	A0	LDA	# \$A0
0311-	20	F0 FD	JSR	\$FDF0
0314-	C8		INY	
0315-	C0	10	CPY	# \$10

0317- D0 F1 BNE \$ 030A

0319- 60 RTS

程序说明:

0300—0306 (04) (03) = 03F0

0308: 变址计数 = 0

030A: 第一次调 \$ 03F0内容→A

030C: 显示两位16进制数

030F: 空格 “ ” 的ASCII码

0311: 显示

0314: 计数器 + 1

0315: 16个字节内容都送完吗?

0317: 否, 继续循环

0319: 是, 结束

程序8.41中取数指令LDA(\$03), Y是采用的后变址Y间接寻址方式, 它常用于动态数据块处理, 即数据块在存储器中存放的位置是可以变化的, 只要把数据块首地址置入这种寻址方式所选用的零页地址中即可。

例5: 已知在\$03F8—\$03FF单元中, 存放四个转移地址, 现要求将它们打印出来。

输出四位16进制数的子程序入口地址为\$F941, 它的功能是将寄存器X和累加器A中的内容按四位16进制数的形式输出到现行输出设备。

调用\$F941之前, 必须将高字节放入累加器A中, 低字节放在X寄存器中。

源程序见8.42。

程序8.42:

0300- A0 02 LDY # \$ 02

0302- AD F9 03 LDA \$ 03F9

0305-	A E	F 8	03	LDX	\$ 03F8
0308-	20	41	F 9	JSR	\$ F941
030 B-	A 9	A 0		LDA	# \$ A0
030 D-	20	F 0	F D	JSR	\$ FDF0
0310-	B 9	F 9	03	LDA	\$ 03F9,Y
0313-	B E	F 8	03	LDX	\$ 03F8,Y
0316-	C 8			INY	
0317-	C 8			INY	
0318-	C 0	0 A		CPY	# \$ 0A
031 A-	D 0	E C		BNE	\$ 0308
031 C-	60			RTS	

程序说明:

0300: 变址计数器 = 2

0302: 设置源数据块首址

0308: 显示四位16进制数

030B: 空一格

0310: 第一次取 \$ 03FA—\$ 03FB内容

0316: 计数器加2

0318: 数据块有无送完?

031A: 没有, 继续

031C: 是的, 结束

LDA \$03F9,Y和LDX \$03F8, Y均采用绝对Y变址寻址方式。

例6: 把\$03F0—\$03FF中各单元内容按两位16进制数显示出来, 要求每两个数之间空三个空格。

\$F948是把三个空格字符送往现行输出设备的子程序入口地址。

源程序见8.43。

程序8.43:

0300-	A0	00		LDY	# \$ 00
0302-	B9	F0	03	LDA	\$ 03F0,Y
0305-	20	DA	FD	JSR	\$ FDDA
0308-	20	48	F9	JSR	\$ F948
030B-	C8			INY	
030C-	C0	10		CPY	# \$ 10
030E-	D0	F2		BNE	\$ 0302
0310-	60			RTS	

程序说明:

0300: 初始化计数器

0302: 调 (\$ 03F0 + Y) 的内容

0305: 显示两位16进制数

0308: 空三格

030B: 计数器加 1

030C: 送完吗?

030E: 否, 再送

0310: 是, 结束

在监控子程序中, 还有一个可以输出X个空格的子程序, 其入口地址为\$F94A, 但调用该子程序前, 必须将空格数设置在寄存器X中, 然后才能安排JSR \$F94A指令。

例 7: 不断按键取字符送显示屏显示, 若按下空格键时则结束。

取一按键的值 (取一个输入字符) 的子程序入口地址为\$FD0C, 该子程序可以接受从键盘输入的一个字符。

见程序8.44。

程序8.44:

0300-	20	0C	FD	JSR	\$ FD0C
0308-	20	F0	F0	JSR	\$ FDF0
0306-	C9	A0		CMP	# \$ A0

```

0308-    D0    F6                BNE    $ 0300
030 A-    60                RTS

```

程序说明:

0300: 取一按键

0303: 显示

0306: 是一空格吗?

0308: 不是, 继续

030A是, 停机

读取键盘数据还有一个子程序,其入口地址为 \$FD 1B, 这是一个键盘输入子程序, 它能等待按键输入, 当有键按下时, 该子程序就将该键值送往屏幕光标当前位置和累加器A中。

例 8: 编制几个不同延时时间的延迟程序

在监控程序中,有一个延时程序,其入口地址为\$FCA8, 可以直接调用。

程序8.45、8.46、8.47、8.48、8.49是用不同方法编制的几个延时程序。

程序8.45:

```

0300-    A2    40                LDX    # $ 40
0302-    A9    FF                LDA    # $ FF
0304-    20    A8    FC        JSR    $ FCA8
0307-    CA                DEX
0308-    D0    F8                BNE    $ 0302
030 A-    60                RTS

```

程序说明:

0300: 计数器置初值

0302: 累加器置初值

0304: 调监控延迟子程序

0307: $X - 1 \rightarrow X$

0308: 只要X不为零、循环

030A: $X = 0$, 结束

程序8.46:

```
0300-   A9  FF      LDA    # $FF
0302-   38          SEC
0303-   E9  01      SBC    # $01
0305-   D0  FB      BNE    $0302
0307-   60          RTS
```

程序说明:

0300: 置延时常数

0302: 置进位标志 $C = 1$

0303: $A - 01 - \bar{C}$

0305: 若不为0, 循环

0307: 是0, 结束

程序8.47:

```
0300-   A0  10      LDY    # $10
0302-   A2  FF      LDX    # $FF
0304-   CA          DEX
0305-   D0  FD      BNE    $0304
0307-   C8          INY
0308-   D0  F3      BNE    $0302
030A-   60          RTS
```

程序说明:

0300: 外循环计数器初值

0302: 内循环计数器初值

0304: $X - 1 \rightarrow X$, 为0否

0305: X不为0, 继续内循环

0307: $X = 0$, 外循环次数 $Y + 1$

0308: $Y \neq 0$, 继续外循环

030A: $Y = 0$, 结束

程序8.48:

0300-	A2	10	LDX	# \$ 10
0302-	A0	FF	LDY	# \$ FF
0304-	88		DEY	
0305-	D0	FD	BNE	\$ 0304
0307-	E8		INX	
0308-	D0	F8	BNE	\$ 0302
030A-	60		RTS	

程序说明:

0300: 外循环计数器初值

0302: 内循环计数器初值

0304: $Y - 1 \rightarrow Y$

0305: $Y \neq 0$, 继续内循环

0307: $Y = 0$, 外循环次数 $X + 1$

0308: $X \neq 0$, 继续外循环

030A: $X = 0$, 结束

程序8.49:

FCA8-	38		SEC	
FCA9-	48		PHA	
FCAA-	E9	01	SBC	# \$ 01
FCAC-	D0	FC	BNE	\$ FCAA
FCAE-	68		PLA	
FCAF-	E9	01	SBC	# \$ 01
FCB1-	D0	F6	BNE	\$ FCA9
FCB3-	60		RTS	

程序说明:

FCAB: 置进位标志 $C = 1$

FCA9: 累加器A的内容进入堆栈

FCAA: $(A) - 1 - \bar{C} \rightarrow A$

FCAC: 直减至 $(A) = 0$ 为止, 否则循环

FCAE: A的内容出栈

FCAF: $(A) - 1 \rightarrow A$

FCB1: 循环至 $(A) = 0$, 否则循环

FCB3: $(A) = 0$, 结束

几点说明:

(1) 程序8.45是一个延时程序, 但其中又调用了监控中的延时程序\$FCA8。

(2) 程序8.46中用了SEC指令, 这是6502指令群中唯一的减法指令, 使用前必须用SEC指令将C标志位置为1, 否则影响运算结果。

(3) 8.47、8.48其程序设计思想完全一样, 都是用双重循环控制循环次数的办法达到延时的目的。但内、外循环计数器选用的不同, 因而暂存器X (或Y) 内容增、减一的指令也不同, 使用时应注意搭配。

(4) 8.49是监控中延迟子程序, 采用了堆栈处理方法, 调用前最好先设置累加器A中的初值, 然后调用。如:

```
0300- A9 20      LDA  #$20
0302- 20 A8 FC JSR   $FCA8
0305- 60          RTS
```

(5) 延时时间的长短, 可以人为地设置, 它可以放在累加器A中, 也可以放在寄存器X或寄存器Y中。

例9: 显示字符的各种方法

为了在屏幕上显示字符或汉字 (一个或多个), 可以有不同的方法。而字符的显示方式也可以多样 (黑底白字的正

常显示、白底黑字的反相显示、白底黑字的闪动显示)。字符可以在固定的位置上显示,也可以在屏幕上用“开窗口”的方法,设定一块显示区显示。

(1) 单个字符的正常显示。例如欲显示一个“A”字符,取正常方式,在文本状态第一页的起始位置上,可用程序8.50:

程序8.50:

```
0300-    20    58    FC        JSR        $FC58
0303-    A9    C1                LDA        # $C1
0305-    8D    00    04        STA        $0400
0308-    60                                RTS
```

程序说明:

0300: 清屏

0303: 把C1送入累加器A中

0305: 把A中的数存入地址0400

0308: 返回

其中C1是字符“A”正常显示方式的16进制表示的ASCII码。\$0400是屏幕文本第一次第一个位置的映像地址。在监控状态下300G↵,即可看到屏幕左上方显示一个“A”字符。

(2) 调用输出一个字符的监控子程序,显示单个字符。正常显示“A”字符,见程序8.51。

程序8.51

```
0300-    20    58    FC        JSR        $FC58
0303-    A9    C1                LDA        # $C1
0305-    20    ED    FD        JSR        $FDED
0308-    60                                RTS
```

输出一个字符的监控子程序入口地址是\$FDED,调用

方法机器码为20 ED FD，其汇编形式为JSR \$FDED，该子程序的功能是将累加器A中的字符（本例是A）送往输出设备。运行方法同上，或在] 状态下执行CALL 768↵

（3）连续显示几个反相字符。例如，反相显示“BASIC”五个字符。

方法是反复调用LDA和JSR这两个指令，见程序8.52。

程序8.52:

```

0300-    20    58    FC    JSR    $FC58
0303-    A9    02          LDA    # $02
0305-    20    ED    FD    JSR    $FDED
0308-    A9    01          LDA    # $01
030A-    20    ED    FD    JSR    $FDED
030D-    A9    13          LDA    # $13
030F-    20    ED    FD    JSR    $FDED
0312-    A9    09          LDA    # $09
0314-    20    ED    FD    JSR    $FDED
0317-    A9    03          LDA    # $03
0319-    20    ED    FD    JSR    $FDED
031C-    60          RTS

```

*300G↵后，在屏幕左上方自左至右反相显示“BASIC”五个字符。# \$02, # \$01, # \$13, # \$09, # \$03分别为字符B, A, S, I, C 的反相显示方式的ASCII码（详见附录1）

（4）用循环的方法显示（闪烁）字符串。例如，闪烁显示“6502 CPU”这七个字符。见程序8.53。

程序8.53:

```

0300-    20    58    FC    JSR    $FC58
0303-    A2    00          LDX    # $00
0305-    BD    11    03    LDA    $0311,X

```

0308-	20	ED	FD	JSR	\$FDED
030B-	E8			INX	
030C-	E0	07		CPX	# \$ 07
030E-	D0	F 5		BNE	\$ 0305
0310-	60			RTS	
0311-	76	75		ROR	\$ 75,X
0313-	70	72			
0315-	43				
0316-	50	55			

程序说明:

0300: 清屏

0303: 寄存器X置初值00

0305: 将(0311 + X)的数送入累加器A

0308: 显示累加器A中的字符

030B: $X + 1 \rightarrow X$

030C: 比较X的值是否与07相等

030E: 如果不等转0305

0310: 相等则返回

0311} \$ 0311开始存放6502

0316} CPU的闪动方式的ASCII码

程序8.53的设计思想如下:

首先初始化X寄存器, 置初值为00, 然后取 $0311 + X = 0311$ 地址的值76 (字符6的闪动显示的ASCII码) 至累加器A, 接着显示这个字符 (调\$FDED输出一个字符的子程序), 让寄存器 $X + 1$, 判断是否与07相符 (6502 CPU共7个字符), 不相符再循环上去 (到\$0305), 重复上述过程, 不断取字符的ASCII码值又不断显示; 若相等则结束。

(5) 用无条件转向语句JNP转向一个地址, 而这个地址开始的内容是反复调用LDA和JSR指令。例如, 反相显

示 “BASIC” 五个字符，见程序8.54。

程序8.54:

0300-	20	58	FC	JSR	\$FC58
0303-	4C	06	03	JMP	\$0306
0306-	A9	02		LDA	#\$02
0308-	20	ED	FD	JSR	\$FDED
030B-	A9	01		LDA	#\$01
030D-	20	ED	FD	JSR	\$FDED
0310-	A9	13		LDA	#\$13
0313-	20	ED	FD	JSR	\$FDED
0315-	A9	09		LDA	#\$09
0317-	20	ED	FD	JSR	\$FDED
031A-	A9	03		LDA	#\$03
031C-	20	ED	FD	JSR	\$FDED
031F-	60			RTS	

程序8.54和程序8.52十分类似，仅仅比程序8.52多了一条指令JMP \$0306，这是一个绝对寻址指令，如操作码为4C，若用6C（间接寻址）则错。另外注意返回指令RTS的操作码60，必须放在\$031F中，放其它值则错。

（6）连续显示若干个相同字符。例如，连续显示20个“*”，可用程序8.55。

程序8.55:

0300-	20	58	FC	JSR	\$FC58
0303-	A2	00		LDX	#\$00
0305-	A9	AA		LDA	#\$AA
0307-	20	ED	FD	JSR	\$FDED
030A-	E8			INX	
030B-	E0	14		CPX	#\$14
030D-	D0	F8		BNE	\$0307

程序8.55采用单重循环反覆相减的方法,控制显示“*”的次数。循环之前清屏、计数器置0、取“*”的ASCII码AA,循环开始显示“*”,计数器加1,只要不是20次(\$14),继续循环显示,直到计满20次跳出循环结束运行。

(7) 间隔显示一串不同字符。例如,间隔显示: * = * = * = * = * = *, 参见程序8.56。

程序8.56:

```

0300-    20    58    FC        JSR        $FC58
0303-    A0    00                LDY        # $00
0305-    A9    AA                LDA        # $AA
0307-    20    ED    FD        JSR        $FD58
030A-    A9    BD                LDA        # $BD
030C-    20    ED    FD        JSR        $FD58
030F-    C8                INY
0310-    C0    05                CPY        # $05
0312-    D0    F1                BNE        $0305
0314-    A9    AA                LDA        # $AA
0316-    20    ED    FD        JSR        $FD58
0319-    60                RTS

```

程序8.56设计方法和8.55大同小异,只是8.56中多调一个显示字符“=”(ASCII码为BD),循环次数为5次,每次既显示一个“*”号,又显示一个“=”号,当循环完成后,再调显一次“*”,只有这样,才和题意要求符合。由于两个程序选用的计数器不同(前者X,后者为Y),因而操作码也有区别。

(8) 连续显示全部ASCII码对应的字符。例如,要求

以正常方式显示附录一中从@到?的字符。见程序8.57。

程序8.57:

0300-	20	58	FC	JSR	\$FC58
0303-	A9	C0		LDA	#\$C0
0305-	20	ED	FD	JSR	\$FDED
0308-	18			CLC	
0309-	69	01		ADC	#\$01
030B-	C9	DF		CMP	#\$DF
030D-	D0	F6		BNE	\$0305
030F-	20	ED	FD	JSR	\$FDED
0312-	A9	A0		LDA	#\$A0
0314-	20	ED	FD	JSR	\$FDED
0317-	18			CLC	
0318-	69	01		ADC	#\$01
031A-	C9	C0		CMP	#\$C0
031C-	D0	F6		BNE	\$0314
031E-	60			RTS	

程序说明:

0300: 清屏

0303: @的代码送入累加器A

0305: 调用输出一个字符子程序

0308: 清进位

0309: 累加器内容加1

030B: 累加器内是DF吗? “一”的代码是DF

030D: 不是, 跳转\$0305再继续循环

030F: 是的, 显示“一”

0312: } 同上分析, 只不过开始放空格的ASCII码, 循环累加中

031E: } 和C0比较。BF是“?”的ASCII码

程序8.57编制中应注意的问题两点:

(1) 由于正常显示@—? 字符的 ASCII 码 (16进制) 不连续, 即C0—DF, A0—BF (连续时DF后加1为E0), 所以程序分两段编写, 一段从\$0300—\$0311 (显示@—); 一段从\$0312—\$031E (显示! —?)。

(2) \$030C单元应放DF, \$031B单元应放C0, 否则显示有错。

若从“!”开始显示到“—”结束, 即按“!”# \$ ……? @ABC…^—□”显示, 可用更为简单的程序, 见程序8.58。

程序8.58:

```

0300-    20    58    FC        JSR    $FC58
0303-    A9    A0                LDA    # $A0
0305-    20    ED    FD        JSR    $FDED
0308-    18                CLC
0309-    69    01                ADC    # $01
030B-    C9    E0                CMP    # $E0
030D-    D0    F6                BNE    $0305
030F-    60                RTS

```

值得指出的程序8.58通用性好, 事实上, 由于反相显示和闪烁显示ASCII码 (16进制表示) 按数值顺序递加, 只要改动程序8.58中的\$0304为\$00, \$030C为\$40即可顺序反相显示所有对应ASCII码的字符 (见程序8.59); 改动8.58程序中\$0304内容为\$40, \$030C内容为\$80, 则可顺序闪烁显示所有对应ASCII码的字符 (见程序8.60)。

程序8.59:

```

0300-    20    58    FC        JSR    $FC58
0303-    A9    00                LDA    # $00

```

```

0305-   20   ED   FD       JSR   $FDED
0308-   18               CLC
0309-   69   01       ADC   # $ 01
030B-   C9   40       CMP   # $ 40
030D-   D0   F6       BNE   $ 0305
030F-   60               RTS

```

程序8.60:

```

0300-   20   58   FC       JSR   $FC58
0303-   A9   40       LDA   # $ 40
0305-   20   ED   FD       JSR   $FDFD
0308-   18               CLC
0309-   69   01       ADC   # $ 01
030B-   C9   80       CMP   # $ 80
030D-   D0   F6       BNE   $ 0305
030F-   60               RTS

```

(9) 显示一个方框。如在屏幕上显示一个如下方框，要求充满整个屏幕。

```

* = * = * = * = *
=
*
=
*
=
*
=
* = * = * = * = *

```

这个问题表面上看比较简单，但实际编程还较复杂，先给出设计好的程序，见8.61，然后再作一些技巧上的说明。

程序8.61:

```

0300-   20   58   FC       JSR   $FC58

```

0303-	A0	00		LDY	# \$ 00
0305-	A9	AA		LDA	# \$ AA
0307-	20	ED	FD	JSR	\$ FDED
030A-	A9	BD		LDA	# \$ BD
030C-	20	ED	FD	JSR	\$ FDED
030F-	C8			INY	
0310-	C0	13		CPY	# \$ 13
0312-	D0	F1		BNE	\$ 0305
0314-	A9	AA		LDA	# \$ AA
0316-	20	ED	FD	JSR	\$ FDED
0319-	A0	00		LDY	# \$ 00
031B-	20	5C	03	JSR	\$ 035C
031E-	C8			INY	
031F-	C0	0A		CPY	# \$ 0A
0321-	D0	F8		BNE	\$ 031B
0323-	20	62	FC	JSR	\$ FC62
0326-	A9	BD		LDA	# \$ BD
0328-	20	ED	FD	JSR	\$ FDED
032B-	A2	25		LDX	# \$ 25
032D-	20	4A	F9	JSR	\$ E94A
0330-	A9	BD		LDA	# \$ BD
0332-	20	ED	FD	JSR	\$ FDED
0335-	20	62	FC	JSR	\$ FC62
0338-	A0	00		LDY	# \$ 00
033A-	A9	AA		LDA	# \$ AA
033C-	20	ED	FD	JSR	\$ FDED
033F-	A9	BD		LDA	# \$ BD
0341-	20	ED	FD	JSR	\$ FDED

程序8.61共分六段:

(1) \$0300—\$0318: 水平显示19对 * 一, 最后再补上一个 *

(2) \$0319—\$0322: 垂直显示10组 = * = 字符, *

中间空37格

(3) \$0323—\$0337: 下移一行, 显示 = = , 中间空37格, 再下移一行

(4) \$0338—\$034D: 同\$0300—\$0318

(5) \$034E—\$035B: 延时一段时间, 其中又调用监控延时子程序

(6) \$035C—\$0380: 垂直显示一组 = * = 字符, *

中间空37格

程序8.61共用了5个监控中机器语言子程序:

(1) \$FC58: 清屏

(2) \$FDED: 输出一个字符

(3) \$FC62: 回车换行

(4) \$FCA8: 延时

(5) \$F94A: 屏幕上输出若干个空格, 空格数在X寄存器中设定

程序8.61主要用了调用机器语言子程序的技巧, 这包括两方面的内容, 一是调用监控中机器语言子程序, 实现诸如清洗屏幕、输出字符、控制延时、光标下移、输出空格等功能; 二是调用自编的机器语言子程序, 如 \$035C—\$0380单元中存放的子程序, 它实现一组 = 字符的显示, 而用\$0319—\$0322程序段对其实现次数控制, 其主要指令是JSR \$035C。

(10) 显示一个题头。在边框里 (例如上面方框中) 放

上一些说明性文字（或资料），就设计成了题头。边框设计可以用程序8.61，或者根据8.61的设计思想，绘成其它图案。现在问题的关键是如何设计一个存有文字字符的资料表，并把它放在边框内的适应位置。

为了直接引用程序8.61设计的边框，我们可以设想再编一个独立的存放资料表的程序，为简单计假设资料表就是一行，即PRESS RETURN TO CONTINUE，见程序8.62。

程序8.62:

```

1000-  A0  00          LDY  # $ 00
1002-  B9  0E  10      LDA  $ 100E,Y
1005-  20  ED  FD      JSR  $ FDED
1008-  C8              INY
1009-  C0  26          CPY  # $ 26
100B-  D0  F5          BNE  $ 1002
100D-  60              RTS
100E-  A0  A0
1010-  A0  A0  A0  A0  50  52  45  53
1018-  53  60  52  45  54  55  52  4E
1020-  60  54  4F  60  43  4F  4E  54
1028-  49  4E  54  55  45  60  61  A0
1030-  A0  A0  A0  A0

```

程序8.62的设计思想和结构安排均和程序8.53相类似，只是用的指令不尽相同，这里不再重述。从\$1000E开始到\$1033止，存放的是PRESS RETURN TO CONTINUE！对应的闪烁显示方式的ASCII字符。运行时在监控下用1000G↵，在BASIC状态下用CALL 4096↵。

有了边框和资料表，就可以设计一个题头，这实际上就

是将程序8.61和程序8.62合起来，并加改适当的控制段和修改个别字节的内容，见程序8.63。

程序8.63:

0300-	20	58	FC	JSR	\$FC58
0303-	A0	00		LDY	#\$00
0305-	A9	AA		LDA	#\$AA
0307-	20	ED	FD	JSR	\$FDED
030A-	A9	BD		LDA	#\$BD
030C-	20	ED	FD	JSR	\$FDED
030F-	C8			INY	
0310-	C0	13		CPY	#\$13
0312-	D0	F1		BNE	\$0305
0314-	A9	AA		LDA	#\$AA
0316-	20	ED	FD	JSR	\$FDED
0319-	A0	00		LDY	#\$00
031B-	20	5C	03	JSR	\$035C
031E-	C8			INY	
031F-	C0	0A		CPY	#\$0A
0321-	D0	F8		BNE	\$031B
0323-	20	62	FC	JSR	\$FC62
0326-	A9	BD		LDA	#\$BD
0328-	20	ED	FD	JSR	\$FDED
032B-	A2	25		LDX	#\$25
032D-	20	4A	F9	JSR	\$F94A
0330-	A9	BD		LDA	#\$BD
0332-	20	ED	FD	JSR	\$FDED
0335-	20	62	FC	JSR	\$FC62
0338-	A0	00		LDY	#\$00
033A-	A9	AA		LDA	#\$AA

033 C -	20	ED	FD	JSR	\$ FDED
033 F -	A9	BD		LDA	# \$ BD
0341-	20	ED	FD	JSR	\$ FDED
0344-	C8			INY	
0345-	C0	13		CPY	# \$ 13
0347-	D0	F1		BNE	\$ 033A
0349-	A9	AA		LDA	# \$ AA
034B-	20	ED	FD	JSR	\$ FDED
034 E -	A2	20		LDX	# \$ 20
0350-	A9	10		LDA	# \$ 10
0352-	20	AB	FC	JSR	\$ FCAB
0355-	CA			DEX	
0356-	D0	F8		BNE	\$ 0350
0358-	20	AE	03	JSR	\$ 03AE
035 B -	60			RTS	
035 C -	20	62	FC	JSR	\$ FC62
035 F -	A9	BD		LDA	# \$ BD
0361-	20	ED	FD	JSR	\$ FDED
0364-	A2	25		LDX	# \$ 25
0366-	20	4 A	F9	JSR	\$ F94A
0369-	A9	BD		LDA	# \$ BD
036B-	20	ED	FD	JSR	\$ FDED
036 E -	20	62	FC	JSR	\$ FC62
0371-	A9	AA		LDA	# \$ AA
0373-	20	ED	FD	JSR	\$ FDED
0376-	A2	25		LDX	# \$ 25
0378-	20	4 A	F9	JSR	\$ F94A
037 B -	A9	AA		LDA	# \$ AA
037 D -	20	ED	FD	JSR	\$ FDED
0380-	60			RTS	

1000-	A0	00			LDY	# \$ 00
1002-	B9	0E	10		LDA	\$ 100E,Y
1005-	20	ED	FD		JSR	\$ FDED
1008-	C8				INY	
1009-	C0	26			CPY	# \$ 26
100B-	C0	F5			BNE	\$ 1002
100D-	60				RTS	
100E-	AA	A0				
1010-	A0	A0	A0	A0	50	52 45 53
1018-	53	60	52	45	54	55 52 4E
1020-	60	54	4F	60	43	4F 4E 54
1028-	49	4E	54	55	45	60 61 A0
1030-	A0	A0	A0	A0		
1034-	20	FB	F4		JSR	\$ F4FB
1037-	20	ED	FD		JSR	\$ FDED
103A-	20	FB	F4		JSR	\$ F4FB
103D-	20	1A	FC		JSR	\$ FC1A
1040-	20	1A	FC		JSR	\$ FC1A
1043-	20	1A	FC		JSR	\$ FC1A
1046-	20	00	10		JSR	\$ 1000
1049-	20	66	FC		JSR	\$ FC66
104C-	60				RTS	

\$0300—\$0380: 为主要程序段, 显示一个边框

\$1000—\$1033: 显示文字资料, 其中 \$100E—\$1033为对应文字资料的ASCII码字符

\$1034—\$104C: 为使文字资料放置在边框中的适当位置而设置的控制程序, 其中 \$F4FB是光标右移一格的子程序, \$FC1A是光标上移一行的子程序。

(11) 汉字字符的显示。在中华学习机 (APPLE-II

及其兼容机不具备)内,有一个重要的子程序CSWA,它的功能是完成汉字或ASCII字符的输出,其入口地址为\$C32B。

对CSWA子程序的调用,必须首先要求在累加器A中存放汉字或字符的显示码以及显示控制命令码。而对于汉字则一定要用三字节的中华学习机内码,同时需要分三次调用才能输出一个汉字。下面提供一个实例,见程序8.64。

程序8.64:

```

1000-   A9  0D           LDA   #$0D
1002-   20  2B  C3       JSR   $C32B
1005-   A2  00           LDX   #$00
1007-   BD  18  10       LDA   $1018,X
100A-   48               PHA
100B-   20  2B  C3       JSR   $C32B
100E-   E8               INX
100F-   68               PLA
1010-   10  F5           BPL   $1007
1012-   A9  0D           LDA   #$0D
1014-   20  2B  C3       JSR   $C32B
1017-   60               RTS
1018-   7F  55  4F  7F  39  27  7F  50
1020-   24  7F  4E  2E  7F  39  79  20
1028-   43  4F  4D  50  55  54  45  D2

```

程序8.64的运行方法比较特殊,因为程序中有显示汉字的安排,所以必须在中华学习机中文状态下,并且处于BASIC状态才能运行,即]CALL 4096↵

屏幕显示:

中华学习机 COMPUTER

(12) 文本状态窗口控制显示。中华学习机在文本状态下可以显示各种ASCII字符,正常的显示屏幕被设定为24行40列。为了能在屏幕的任意位置上显示字符,可以采用“开窗口”的方法,所谓屏幕“开窗口”,就是在屏幕上任意设定一块显示区,而在这个区域以外的屏幕不发生变化。窗口的设置由用户向四个特殊0页单元置数确定:

\$20 单元存放窗口左极限列数 (\$0—\$27)

\$21 单元存放窗口的宽度 (\$1—\$28)

\$22 单元存放窗口顶行行数 (\$0—\$18)

\$23 单元存放窗口底行行数 (\$0—\$18)

正常情况下\$20—\$23四个单元中的值分别为\$0, \$28, \$0, \$18, 这可以从监控下查看得到,即: * 20.23 ↙

* 0020—00 28 00 18

在BASIC状态下可以用POKE指令设定参数,例如在\$20单元中放\$0A:

POKE 32, 10

而用机器语言设定时,则可用:

A9 0A LDA # \$0A

85 20 STA \$20

例如要在屏幕中央反相显示“6502CPU”,可用程序8.65。

程序8.65:

0300-	20	58	FC	JSR	\$FC58
0303-	A2	00		LDX	# \$00
0305-	BD	11	03	LDA	\$0311,X
0308-	20	ED	FD	JSR	\$FDED
030B-	E8			INX	

030C-	E0	07		CPX	井\$07
030E-	D0	F5		BNE	\$0305
0310-	60			RTS	
0311-	76	75		ROR	\$75,X
0313-	70	72		BVS	\$0387
0315-	43			???	
0316-	50	55		BVC	\$036D
0318-	20	58	FC	JSR	\$FC58
031B-	A9	13		LDA	井\$13
031D-	85	20		STA	\$20
031F-	A9	09		LDA	井\$09
0321-	85	22		STA	\$22
0323-	20	00	03	JSR	\$0300
0326-	60			RTS	

程序8.65中\$0300—\$0317单元和程序8.53完全一样，反相显示“6502 CPU”七个字符。\$031B—\$031E是设定窗口左极限，其值为\$13（取屏幕左、右边界的中间值），\$031F—\$0322是设定窗顶行数，其值为\$09（取屏幕上、下边界的中间值）。

（13）中华学习机区位码转换成内码的方法。中华学习机的区位码和学习机内码是不同的，它们之间不存在一一对应的关系。

例如，区位码01—05，学习机内码为\$1D—\$21，两种码值之间相差\$1C；区位码06—14，学习机内码为\$23—\$2B，两种码之间相差\$1D；区位码15—27，学习机内码为\$2D—\$39，两种码之间相差\$1E；区位码28—94，学习机内码为\$3B—\$7D，两种码值相差\$1F。

由此可知，学习机内码从\$1D—\$7D中，不出现\$22，

\$2C, \$3A这三个内码。而这三个内码正好是引号”、逗号,、冒号:所对应的代码。学习机内码所以不采用上述代码,是为了避免引起字符串或程序混乱,例如,在DOS操作系统下,从软盘读取数据时,需要用逗号(,)和冒号(:)的代码来区分变量和语句。换句话说,这里的区位码不能和国标中的代码一样,而是经过转换来的。

根据以上分析,可知中华学习机CEC—I的内码(又称汉字内码),是将区位码修正而后得到的。修正的规律是将区(位)码先加\$1C,若结果 \geq \$22则加1;若又 \geq \$2C再加1;若又 \geq \$3A,则需再加1。掌握了这些修正规律,不难编写一个机器语言程序,实现区位码向学习机内码的转换。

见程序8.66。

程序8.66:

1000-	A0	00	LDY	# \$ 00
1002-	A9	00	LDA	# \$ 00
1004-	B9	60 10	LDA	\$ 1060,Y
1007-	18		CLC	
1008-	69	1C	ADC	# \$ 1C
100A-	C9	22	CMP	# \$ 22
100C-	10	06	BPL	\$ 1014
100E-	20	37 10	JSR	\$ 1037
1011-	4C	31 10	JMP	\$ 1031
1014-	69	00	ADC	# \$ 00
1016-	C9	2C	CMP	# \$ 2C
1018-	10	06	BPL	\$ 1020
101A-	20	37 10	JSR	\$ 1037
101D-	4C	31 10	JMP	\$ 1031
1020-	69	00	ADC	# \$ 00

1022-	C9	3 A		CMP	#\$3 A
1024-	10	06		BPL	\$102 C
1026-	20	37	10	JSR	\$1037
1029-	4 C	31	10	JMP	\$1031
102 C -	69	00		ADC	#\$00
102 E -	20	37	10	JSR	\$1037
1031-	C8			INY	
1032-	C0	5 E		CPY	#\$5 E
1034-	D0	CE		BNE	\$1004
1036-	60			RTS	
1037-	20	DA	FD	JSR	\$FDDA
103 A -	A9	A0		LDA	#\$A0
103 C -	20	F0	FD	JSR	\$FDF0
103 F -	60			RTS	

\$1060—\$10BE中放的是区(位)码(16进制)，见下

表

1060-	01	02	03	04	05	06	07	08
1068-	09	0A	0B	0C	0D	0E	0F	10
1070-	11	12	13	14	15	16	17	18
1078-	19	1A	1B	1C	1D	1E	1F	20
1080-	21	22	23	24	25	26	27	28
1088-	29	2A	2B	2C	2D	2E	2F	30
1090-	31	32	33	34	35	36	37	38
1098-	39	3A	3B	3C	3D	3E	3F	40
10A0-	41	42	43	44	45	46	47	48
10A8-	49	4A	4B	4C	4D	4E	4F	50
10B0-	51	52	53	54	55	56	57	58
10B8-	59	5A	5B	5C	5D	5E		

如\$1060: 01相当于区(位)码1, \$1069: 0A相当于

区(位)码10, ……,\$10BD; 5E相当于区(位码) 94, 其余类推。

运行结果(学习机内码)

1D 1E……20 21 23 24……2A 2B 2D 2E…
 …38 39 3B 3C……7A 7B 7C 7D

对应的区(位码)为1, 2, 3, ……,\$91, 92, 93, 94

例10: 数据块搬家

将\$6000—\$60FF的内容搬至\$7000—\$70FF中去。见程序8.67。

程序8.67:

300-	A0	00	LDY	#\$00
302-	A9	00	LDA	#\$00
304-	85	3C	STA	\$3C
306-	A9	60	LDA	#\$60
308-	85	3D	STA	\$3D
30A-	A9	FF	LDA	#\$FF
30C-	85	3E	STA	\$3E
30E-	A9	60	LDA	#\$60
310-	85	3F	STA	\$3F
312-	A9	00	LDA	#\$00
314-	85	42	STA	\$42
316-	A9	70	SDA	#\$70
318-	85	43	STA	\$43
31A-	20	2C FE	JSR	\$FE2C
31D-	60		RTS	

这里利用了监控中搬家子程序, 只需将源数据首地址和末地址放入\$3C—\$3F中, 而把目的地址放在\$42—\$43中,

调用 \$FE2C 子程序一次搬完。在调用前必须设置 LDY # \$00。

例11：地址指针增 1 程序

A1H和A1L、A2H和A2L、A4H是A4L的内容分别表示三个地址。要求A1和A4增 1，并根据A1比A2大或小来消除或设置进位

设\$3C：存放A1低字节A1L

\$3D：存放A1高字节A1H

\$3E：存放A2低字节A2L

\$3F：存放A2高字节A2H

\$42：存放A4低字节A4L

\$43：存放A4高字节A4H

程序如下：

\$FCB4-	E 6 42	1NC \$ 42； A4低字节加1
\$FCB6-	D 0 02	BNE \$ FCBA； 加1后不为 0 则不处 理高字节
\$FCB8-	E 6 43	1NC \$ 43； 否则， A4高字节加1
\$FCBA-	A 5 3C	LDA \$ 3C； 取A1低字节
\$FCBC-	C 5 3E	CMP \$ 3E； (A1L) ≥ (A2L)？
\$FCBE-	A 5 3D	LDA \$ 3D； 取A1高字节
\$FCC0-	E 5 3F	SBC \$ 3F； A1 ≥ A2则进位位置1， 否则置 0
\$FCC2-	E 6 3C	1NC \$ 3C； A1低字节加1
\$FCC4-	D 0 02	BNE \$ FCC8； 加1后不为 0 则不处 理高字节
\$FCC6-	E 6 3D	1NC \$ 3D； 否则A1高字节加1
\$FCC8-	60	RTS； 结束

从上述程序看出，程序中几乎全部用了零页寻址方式。

运行程序前应将A1, A2, A4的地址分别置入 \$3C, \$3D, \$3E, \$3F, \$42, \$43单元中。

例12: 搬家程序

A1H和A1L、A2H和A2L、A4H和A4L中的内容分别表示三个地址并设它们为A1, A2及A4。要求将以A1为首地址, 以A2为末地址的代码搬至以A4为首地址的存贮区中去。如果 $A2 \leq A1$ 则只搬动一个字节。

A1, A2及A4的地址存贮单元同〔例11〕。(即放在\$3C—\$3F, \$42—\$43单元中)

程序如下:

\$FE2C-	B1	3C	LDA(\$3C),Y, 取一个数
\$FE2E-	91	42	STA(\$42),Y, 存一个数
\$FE30-	20	B4 FC	JSR \$FCB4, A1,A4均指向 下一个数, 并 测试是否搬完
\$FE33-	90	F7	BCC \$FE2C, 未搬完, 继续
\$FE35-	60		RTS, 搬家程序结束

本程序中调用了〔例11〕中的地址指针增1程序, 进入本程序前应使Y寄存器清0。值得提出的是, 在监控状态下移动一段存贮器单元的内容就是用的搬家程序, 即所谓执行监控状态下的M命令。即:

* <新地址> <原首地址> · <原末地址> M ↙

关于调用监控子程序的实例, 我们就介绍到这里。必须指出的是监控子程序远远不只是上面的内容, 还有许多子程序, 我们没有列出, 有兴趣的读者可查阅有关资料, 另外, 本书的其它章节也有一些调用监控子程序的程序, 请注意对照学习。

九、数据传送、交换、 检索和排序

数据传送、交换、数据分类（又称排序）和数据检索（也称搜索、查找）是计算机数据（信息）处理中最常见的几项工作，也是计算机应用中最主要的功能。本章通过不少实例，介绍数据传送、比较、交换、检索、排序的概念和实用程序，以求对信息处理有较为深入的了解。

1. 数据传送

数的传送是计算机最基本、最经常、大量的操作。例如，在进行算术和逻辑运算时，总要有操作数；又如，将一块存贮区的内容搬至另一块存贮区，总是和数打交道。事实上，考察和分析各种各样程序，可以看到数的传送（取数、送数、求和、存数）指令往往是最多的，在整个程序中占有很大的比重。在程序设计中，数的传送是否灵活，传送速度快还是慢，选用什么样的传送指令，对整个程序的执行都起重要作用。而在一些非数值操作（数据块移动、表的检索、各种排序、代码转换）中，也大量涉及到数的存取问题，至于在图形处理（图形移动、合并、剪辑、动画显示）方面，更需要处理数的传送操作，因此，学习数据块传送要很好掌握取数和存数指令，这样才能很好地设计出高质量的程序来。

本节主要通过若干程序设计实例，进一步加深对存取数据指令了解，为分析和研究更复杂的程序打下坚实的基础。

(1) 单个数据块的传送 例如：将存放在\$6001~\$60FF地址单元的数据块按相反顺序传送到\$7001~\$70FF地址中去。

方法：见程序9.1。

程序9.1:

0300-	A2 01	LDX	# \$ 01
0302-	A0 FF	LDY	# \$ FF
0304-	BD 00 60	LDA	\$ 6000,X
0307-	99 00 70	STA	\$ 7000,Y
030A-	E8	INX	
030B-	88	DEY	
030C-	D0 F6	BNE	\$ 0304
030E-	60	RTS	

程序说明:

0300: 源数据计数器初值

0302: 目的数据块计数器初值

0304: 取源数据

0307: 送入目的地址

030A: 源计数器 + 1

030B: 目的计数器 - 1

030C: 次数不够, 循环

030E: 结束

程序9.1中用了三条取数指令和一条存数指令。变址寄存器X、Y，在编程中常常被当作一个计数器来用。它们可以由指令控制而被置成一个常数，并能方便地用加1、减1、

比较操作来修改和测试其内容，使得程序能够灵活方便地处理数据块、表格等问题。由于本程序要同时处理两个以上的数据块的传送，因而一个变址寄存器就显得不够用，所以既用了变址寄存器X，又用了变址寄存器Y。

方法2：见程序9.2。

程序9.2:

0300-	A0	01	LDY	# \$ 01	
0302-	A2	FF	LDX	# \$ FF	
0304-	B9	00	60	LDA	\$ 6000,Y
0307-	9D	00	70	STA	\$ 7000,X
030A-	C8		INY		
030B-	CA		DEX		
030C-	D0	F6	BNE	\$ 0304	
030E-	60		RTS		

程序9.2和程序9.1，在设计思想和结构安排上几乎完全一样，但用的指令和寄存器有所区别，表面上看仅仅是X换成Y，Y换成X，但指令助记符的操作码有很大区别，这一点对初学来说，应注意这些细微的差别。

方法3：见程序9.3。

程序9.3

0300-	A0	01	LDY	# \$ 01	
0302-	A2	FF	LDX	# \$ FF	
0304-	BD	00	60	LDA	\$ 6000,X
0307-	99	00	70	STA	\$ 7000,Y
030A-	C8		INY		
030B-	CA		DEX		
030C-	D0	F6	BNE	\$ 0304	
030E-	60		RTS		

程序9.3和程序9.1、9.2, 在设计思想上不同, 程序9.1和9.2均是按 (6001)→(70FF), (6002)→(70FE), …… (60FF) → (7001) 顺序将数据块搬家, 而程序9.3则是按 (60FF) → (7001), (60FE)→(7002), …… (6001) → (70FF) 次序迁移。由此可见, 不同的设计思想, 就有不同的程序安排。当然这三个程序, 就其本质上来说并没有什么差异。

方法4: 见程序9.4。

程序9.4:

0300-	A2 01	LDX	# \$01
0302-	A0 FF	LDY	# \$FF
0304-	A9 00	LDA	# \$00
0306-	85 06	STA	\$06
0308-	A9 60	LDA	# \$60
030A-	85 07	STA	\$07
030C-	B1 06	LDA	(\$06), Y
030E-	9D 00 70	STA	\$7000, X
0311-	E8	INX	
0312-	88	DEY	
0313-	D0 F7	BNE	\$030C
0315-	60	RTS	

程序说明:

0300: }
0302: } 初始化

0304: }
030A: } 把源数据块首地址送入07和06单元, 即 (07) (06) = 6000

030C: }
030E: } 第一次将 (60FF) → (7001)

0311: $X + 1 \rightarrow X$

0312: $Y - 1 \rightarrow Y$

0313: 不为0继续循环

0315: 为0, 返回

程序9.4内\$0304—\$030B是将源数据块首地址送入零页单元\$07, \$06中, 这种方法在程序设计中常常采用, 地址变了, 只要将改变的地址送入\$07, \$06单元, 一定程度上保证了程序的通用性。指令LDA (\$06), Y是采用的后变址(Y)间接寻址方式, 指令STA \$7000, X是采用的绝对X变址寻址方式, 执行这两条指令就可以把一个地址的内容送到另一个地址中去。说明, 同一个题目, 寻址方式不一样, 编制的程序也不同。

方法5: 见G1。

\$6001—\$60FF 源数据

反序传送\$7001—\$70FF 中

G1:

```

                ORG  $0300
START          LDX  # $01
                LDY  # $FF
LOOP1          LDA  $6000, X
                PHA
                INX
                DEY
                BNE  LOOP1
                LDX  # $01
LOOP2          PLA
                STA  $7000, X
```

```

INX
CPX  # $00
BNE  LOOP2
RTS

```

(2) 多个数据块的传送 例如：将存在\$6000—\$6010单元中第一个数据块送到\$7000—\$7010单元中，将存在\$6100—\$6110单元中的第二个数据块传送到\$7100—\$7110单元中，将存在\$6200—6210单元中的第三个数据块传送到\$7200—\$7210单元中。

这是个三个数据块的传送问题，我们也采用几种方法求解，由于是顺序搬迁，而且数据块长度相同（都是17个），使得问题求解不至于太复杂。

方法 1：见程序9.5。

程序9.5：

0300-	A0	00	LDY	# \$ 00	} ①(3D)(3C) = (6000) (3F)(3E) = (6010) (43)(42) = (7000)
0302-	A9	00	LDA	# \$ 00	
0304-	85	3C	STA	\$ 3 C	
0306-	A9	60	LDA	# \$ 60	
0308-	85	3D	STA	\$ 3 D	
030A-	A9	10	LDA	# \$ 10	
030C	85	3E	STA	\$ 3E	
030E-	A9	60	LDA	# \$ 60	
0310-	85	3F	STA	\$ 3F	
0312-	A9	00	LDA	# \$ 00	
0314-	85	42	STA	\$ 42	
0316-	A9	70	LDA	# \$ 70	
0318-	85	43	STA	\$ 43	
031A-	20	2C	FE JSR	\$ FE2C	

⋮	⋮	
	LDA	# \$ 61
⋮	⋮	⋮
	LDA	# \$ 61
⋮	⋮	⋮
	LDA	# \$ 71
⋮	⋮	⋮
	JSR	\$ FE2C
⋮	⋮	⋮
	LDA	# \$ 62
⋮	⋮	⋮
	LDA	# \$ 62
⋮	⋮	⋮
	LDA	# \$ 72
⋮	⋮	⋮
	JSR	\$ FE2C
⋮	⋮	⋮
	RTS	

② (3D) (3C) =
 (6100)
 (3F) (3E) =
 (6110)
 (43) (42) =
 (7100)

③ (3D) (3C) =
 (6200)
 (3F) (3E) =
 (6210)
 (43) (42) =
 (7200)

程序9.5有几点说明:

(1) 这是一个取数、送数、调子程序的直线程序, 共重复三次。

(2) \$FE2C是监控中搬家子程序的入口地址, 程序中三次调用, 目的是直接完成从一个存贮区到另一个存贮区的数据块移动。

(3) 为了配合调用\$FE2C开始的子程序, 调用前必须使变址寄存器Y为零。

(4) 本程序的结束部份不要忘记执行一次RTS指令。

(5) ①、②、③部份的程序段, 几乎相同, 只是需要改变的是地址, 例如②部份中用61、71分别代替①中的60、

70, ③部份中用62、72分别代替①中的60、70。

(6) 整个程序结构明确, 但过分冗长, 优点是直接调用监控中搬家子程序。

方法2: 见程序9.6。

程序9.6:

0300-	A2	00	LDX	# \$ 00
0302-	A0	11	LDY	# \$ 11
0304-	A1	1A	LDA	(\$ 1A,X)
0306-	81	FA	STA	(\$ FA,X)
0308-	F6	1A	INC	\$ 1A,X
030A-	F6	FA	INC	\$ FA,X
030C-	88		DEY	
030D-	D0	F5	BNE	\$ 0304
030F-	E0	04	CPX	# \$ 04
0311-	F0	05	BEQ	\$ 0318
0313-	E8		INX	
0314-	E8		INX	
0315-	4C	02 03	JMP	\$ 0302
0318-	60		RTS	

程序说明:

0300: 变址计数器X置0

0302: 数据块长度送Y计数器

0304: 第一次取 (6000) → A

0306: A → (7000)

0308: 修改源地址, 使之增1, 第一次使 (1A) = 00 + 1

030A: 修改目的地址, 使之增1, 第一次使 (FA) = 00 + 1

030C: Y - 1 → Y, 判断一个数据块是否送完

030D: 一个数据未送完, 再送

030F: 三个数据块都传送完了吗?

0311: 是, 转结束

0313: 否, 将 $X + 1$, 再加1,

0314: 为取下一个数据块作准备

0315: 转下一个数据块

0318: 结束

001A- 00 60 00 61 00 62

00FA- 00 70 00 71 00 72

程序9.6有两点说明:

(1) 源数据块的首地址放在\$1A—\$1F单元, 目的数据块的首地址放在\$FA—\$FF单元, 这些单元都采用零页地址单元, 这样处理比程序9.5设置源数据块和目的数据块地址要简捷得多。

(2) 使用了LDA (\$1A, X) 和STA (\$FA, X) 两条指令, 它们都是采用先变址(X)间接寻址的寻址方式, 其优点是使得多个数据块的处理变得简单、容易。一般来说处理多个数据块的传送问题, 用先变址间接寻址方式比较理想。

方法3: 见程序9.7。

程序9.7

1000-	A2	00	LDX	# \$ 00
1002-	BD	00 60	LDA	\$ 6000, X
1005-	9D	00 70	STA	\$ 7000, X
1008-	E8		INX	
1009-	E0	0 A	CPX	# \$ 0A
100B-	D0	F5	BNE	\$ 1002
100D-	60		RTS	
100E-	A9	60	LDA	# \$ 60
1010-	8D	04 10	STA	\$ 1004

1013-	A9	70		LDA	# \$70
1015-	8D	07	10	STA	\$1007
1018-	20	00	10	JSR	\$1000
101B-	EE	04	10	INC	\$1004
101E-	EE	07	10	INC	\$1007
1021-	20	00	10	JSR	\$1000
1024-	EE	04	10	INC	\$1004
1027-	EE	07	10	INC	\$1007
102A-	20	00	10	JSR	\$1000
102D-	60			RTS	

运行前:

```
* 6000: A  A  A  A  A  A  A  A  A  A
* 6100: C  C  C  C  C  C  C  C  C  C
* 6200: E  E  E  E  E  E  E  E  E  E
```

运行后:

```
* 100EG
* 7000. 7009
7000- 0A 0A 0A 0A 0A 0A 0A 0A
7008- 0A 0A
* 7100. 7109
7100- 0C 0C 0C 0C 0C 0C 0C 0C
7108- 0C 0C
* 7200. 7209
7200- 0E 0E 0E 0E 0E 0E 0E 0E
7208- 0E 0E
```

程序9.7几点说明:

(1) 本程序采用多次调用子程序的结构编程。主程序放在\$100E—\$102D单元,子程序放在\$1000—\$100D单元。

(2) 主程序的任务有三个,开始设置第一个数据块首

地址（见\$100E—\$1017），修改第二个第三个数据块的首地址指针（见\$101B—\$1020和\$1024—\$1029），三次调子程序（均采用JSR \$1000方式）。

（3）子程序完成单个数据块传送任务，也是本程序的核心程序段。它采用循环结构编程，取一个数，立即送一个数，计数器加1，只要一个数据块个数未送完再重复上去，送完通过\$100D的RTS指令返回主程序。

（4）程序编制中是先安排子程序再安排主程序，这和一般程序设计中先安排主程序再安排子程序的方法不同，但效果一样。因此，本程序的运行方式特殊，必须先运行主程序即*100EG✓。

方法4，见程序9.8。

程序9.8:

1000-	A2	00	LDX	# \$ 00
1002-	BD	00 63	LDA	\$ 6300,X
1005-	9D	00 73	STA	\$ 7300,X
1008-	E8		INX	
1009-	E0	0 A	CPX	# \$ 0A
100B-	D0	F5	BNE	\$ 1002
100D-	60		RTS	
100E-	A9	60	LDA	# \$ 60
1010-	8D	04 10	STA	\$ 1004
1013-	A9	70	LDA	# \$ 70
1015-	8D	07 10	STA	\$ 1007
1018-	20	28 10	JSR	\$ 1028
101B-	20	22 10	JSR	\$ 1022
101E-	20	22 10	JSR	\$ 1022
1021-	60		RTS	

1022-	EE	04	10	INC	\$ 1004
1025-	EE	07	10	INC	\$ 1007
1028-	20	00	10	JSR	\$ 1000
102B-	60			RTS	

运行实例和方法同程序F9。

程序9.8几点说明：

(1) 程序9.8也是采用调子程序的方法编程。主程序安排在\$100E—\$102B单元，子程序安排在\$1000—\$100D单元。这个子程序和程序F9.7的子程序几乎完全一样，任务也相同，完成单个数据块的传送工作。

(2) 主程序则采用子程序嵌套结构，因为在主程序的\$1022—\$102B单元也安排了一个子程序，而在这个子程序(\$1022—\$102B)中又安排了调用\$1000开始的子程序工作。这个结构和程序9.7是不同的，程序9.7结构是一个主程序多次调用子程序。

(3) 子程序(\$1022—\$102B)的任务完成地址指针的调整工作，以及调用单个数据块传送的子程序(\$1000—\$100D)。

关于多个数据块的传送方法很多，其它几种方法参见机器语言程序中循环程序设计和子程序设计中的内容。

2. 数据交换

数据交换是数据处理的一项重要内容，是排序的主要基础，现通过几个实例，介绍数据交换的方法和技巧。

例1：已知XX和YY为两个16进制数，前者存贮在\$00单元中，后者存贮在\$01单元中，今要求交换它们的存贮次序。

设: $XX = A0$, $YY = B0$

交换前: $(00) = A0$, $(01) = B0$

交换后: $(00) = B0$, $(01) = A0$

方法1: 利用堆栈暂存, 见程序9.9。

程序9.9:

```
0300-   A5  00      LDA   $00
0302-   48          PHA
0303-   A5  01      LDA   $01
0305-   85  00      STA   $00
0307-   68          PLA
0308-   85  01      STA   $01
030A-   60          RTS
```

程序说明:

0300: $A0 \rightarrow A$

0302: $A0$ 入栈

0303: $B0 \rightarrow A$

0305: $B0 \rightarrow (00)$

0307: $A0$ 出栈

0308: $A0 \rightarrow (01)$

030A: 结束

方法2: 使用X、Y寄存器暂存, 见程序9.10。

程序9.10:

```
0300-   A6  00      LDX   $00
0302-   A4  01      LDY   $01
0304-   84  00      STY   $00
0306-   86  01      STX   $01
0308-   60          RTS
```

程序说明:

0300: $A0 \rightarrow X$

0302: B0→Y
 0304: B0→(00)
 0306: A0→(01)
 0308: 结束

比较程序9.10和9.9, 前者比较简捷, 不仅占用内存少, 而且周期短, 因为进栈指令PHA执行周期为3, 而出栈指令PLA执行周期为4, 综合这两方面因素, 程序9.10执行速度快。

方法3: 传送指令互换, 见程序9.11。

程序9.11:

```
0300-    A5  00      LDA    $ 00
0302-    AA          TAX
0303-    A5  01      LDA    $ 01
0305-    A8          TAY
0306-    86  01      STX    $ 01
0308-    84  00      STY    $ 00
030 A-    60          RTS
```

程序说明:

0300: A0→A
 0302: A0→X
 0303: B0→A
 0305: B0→Y
 0306: A0→(01)
 0308: B0→(00)

不难分析程序9.11虽和程序9.9占有相同的内存字节, 但因9.11执行周期短, 运行速度比9.9快。而从程序的简洁和易读性、占用内存少和执行速度来看, 程序9.10为佳。

应该特别指出的是, 通过本例的分析, 切不可造成一个

影响，堆栈用处不大，事实上读者可以阅读本书其它章节的内容，将会发现堆栈不仅是计算机软件中的一个非常重要的概念，而且有着多方面的应用。

例2：有两个数X、Y，若 $X > Y$ ，请交换其存贮单元，并给出交换标志00，若 $X < Y$ ，则不交换，并给出不交换标志FF。

设数X放在\$00单元中，数Y放在\$01单元中，标志由\$03单元给出。见程序9.12。

程序9.12:

0300-	A5	00	LDA	\$ 00
0302-	C5	01	CMP	\$ 01
0304-	B0	05	BCS	\$ 030B
0306-	A0	FF	LDY	# \$ FF
0308-	84	03	STY	\$ 03
030 A	60		RTS	
030 B-	48		PHA	
030 C -	A5	01	LDA	\$ 01
030 E -	85	00	STA	\$ 00
0310-	68		PLA	
0311-	85	01	STA	\$ 01
0313-	A0	00	LDY	# \$ 00
0315-	84	03	STY	\$ 03
0317-	60		RTS	

程序说明:

0300: 取(00)的数X

0303: $X > Y$?

0304: 是, 跳转 \$ 030 B

0306: 否, 不交换, 取标志

0306: 放标志

030A: 结束

030B: 暂存X值

030C: 取Y

030E: $Y \rightarrow (00)$, 交换

0310: X值退栈

0311: $X \rightarrow (01)$, 交换

0313: 取交换标志

0315: 放标志

0317: 结束

程序9.12结构清楚, 注释明了。其基本思想是, 取一个数X, 和Y比较, 若 $X > Y$, 交换并给出交换标志00; 若 $X < Y$, 则不交换并给出不交换标志FF。

分析中注意CMP指令是指累加器A的内容减存贮器M中的内容, 若 $A \geq M$, 表示够减, 置 $C = 1$; 反之 $A < M$, 表示不够减, 置 $C = 0$ 。而BCS指令是在 $C = 1$ 时执行分支动作, 而在 $C = 0$ 时继续执行下一条指令。一般程序设计中这两条指令配合在一起使用。

在交换程序段\$030B—\$0317中, 省去了一条指令, 您能指出来吗?

省去的指令是LDA \$00, 它可以放在PHA指令之前, 为什么能省呢? 请考虑。

例3: 将两组16位数交换。

如交换前: (01) (02) = 3A 4B

(03) (04) = 5C 6D

交换后: (01) (02) = 5C 6D

(03) (04) = 3A 4B

方法 4：利用堆栈暂存，见程序 9.13。

程序 9.13:

0300-	A5	01	LDA	\$ 01
0302-	48		PHA	
0303-	A5	02	LDA	\$ 02
0305-	48		PHA	
0306-	A5	03	LDA	\$ 03
0308-	85	01	STA	\$ 01
030A-	A5	04	LDA	\$ 04
030C-	85	02	STA	\$ 02
030E-	68		PLA	
030F-	85	04	STA	\$ 04
0311-	68		PLA	
0312-	85	03	STA	\$ 03
0314-	60		RTS	

程序说明:

0300: (01) = 3A → A

0302: 3A 入栈

0303: (02) = 4B → A

0305: 4B 入栈

0306: (03) = 5C → A

0308: 5C → (01)

030A: (04) = 6D → A

030C: 6D → (02)

030E: 4B 出栈

030F: 4B → (04)

0311: 3A 出栈

0312: 3A → (03)

0313: 结束

分析程序9.13时, 请注意堆栈是严格遵从“后进先出”的规则处理数据的存取的。因此, 第一次出栈的是 4 B (后进先出), 第二次出栈的是 3 A (先进后出)。

方法 5: 使用 X、Y 暂存器, 见程序 9.14。

程序 9.14:

0300-	A6	01	LDX	\$ 01
0302-	A4	02	LDY	\$ 02
0304-	A5	03	LDA	\$ 03
0306-	85	01	STA	\$ 01
0308-	A5	04	LDA	\$ 04
030 A -	85	02	STA	\$ 02
030 C -	86	03	STX	\$ 03
030 E -	84	04	STY	\$ 04
0310-	60		RTS	

程序说明:

0300: (01) = 3 A → X

0302: (02) = 4 B → Y

0304: (03) = 5 C → A

0306: 5 C → (01)

0308: (04) = 6 D → A

030 A: 6 D → (02)

030 C: 3 A → (03)

030 E: 4 B → (04)

0301: 结束

在程序 9.14 中, 我们再次用了 X、Y 寄存器暂存, 从而比程序 9.13 更简短, 又一次体会到利用 X、Y 寄存器的灵活和方便。但是, 对于处理更多个数据, 或一个长数据块来说, X、Y 寄存器的暂存空间太小, 就无能为力了, 这时只有改用

堆栈暂存，因为它可以压进大量的数据，也可随时弹出来。
请看例4。

例4：将\$6000—\$6063存放的100个数和\$7000—\$7063存放的数对换。

设计本题程序的要点是：安排三个循环

(1) FIRST：将\$6000单元开始存放的100个数压进堆栈暂存；

(2) SECOND：将\$7000单元开始存放的100个数存进\$6000单元开始的内存中去；

(3) THIRD：将堆栈中存放的100个数送进\$7000单元开始的100个内存中去。

方法1：见程序9.15。

程序9.15：

```
                ORG    $ 300
START  LDX    # $ 00
FIRST  LDA    $ 6000,X
        PHA
        INX
        CPX    # $ 64
        BNE    FIRST
        LDX    # $ 00
SECOND LDA    $ 7000,X
        STA    $ 6000,X
        INX
        CPX    # $ 64
        BNE    SECOND
        LDX    # $ 63
THIRD  PLA
```



```

STA    $7000,X
DEX
CPX    # $FF
BNE    THIRD
RTS

```

程序9.15仅给出汇编形式，请读者查阅附录，写成机器码并上机验证。阅读本程序时，请说明第三个循环开始初始化条件为什么安排LDX # \$63 指令，而不用 LDX # \$00 指令？最后一条CPX指令，为什么用CPX # \$FF，而不用CPX # \$00。

方法2：程序9.15解题虽然清楚，但过于冗长稍加分析不难发现还可以进一步优化，例如用一重循环编程，参见程序9.16。

程序9.16：

```

                ORG    $300
                LDX    # $00
LOOP           LDA    $6000, X
                PHA
                LDA    $7000, X
                STA    $6000, X
                PLA
                STA    $7000, X
                INX
                CPX    # $64
                BNE    LOOP
                RTS

```

对应的源程序为：

```

0300~    A2    00            LDX    # $00

```

0302-	BD 00 60	LDA	\$ 6000,X
0305-	48	PHA	
0306-	BD 00 70	LDA	\$ 7000,X
0309-	9D 00 60	STA	\$ 6000,X
033C-	68	PLA	
030D-	9D 00 70	STA	\$ 7000,X
0310-	E8	INX	
0311-	E0 64	CPX	# \$ 64
0313-	D0 ED	BNE	\$ 0302
0315-	60	RTS	

例 5：用交换的方法找最大数

已知一列数，其数据长度放在\$6000单元中，从\$6001单元开始存放数列，最大值找出来后放在\$6001单元中。

示范题：(6000) = 05

(6001) = 12

(6002) = 03

(6003) = 15

(6004) = 08

(6005) = 47

结果：(6001) = 47

见程序9.17。

程序9.17:

	LDX	# \$ 00	
	LDA	\$ 6001	; 取一个数
T 0	CMP	\$ 6002,X	; 小于另一个数?
	BCC	T 1	; 是, 转 T 1
	INX		; 否, X + 1 → X
	CPX	\$ 6000	; 所有数都比较完?

	BNE	T0		;	否, 转T0
	RTS			;	是, 结束
T1	LDA	\$6002, X		;	取大数
	STA	\$6001		;	放数
	INX			;	$X + 1 \rightarrow X$
	CPX	\$6000		;	所有数都比较完?
	BNE	T0		;	否, 转T0
	RTS			;	是, 结束

程序9.17正确性是随机的,就是说有可能对,也可能错。

例如,在\$6006单元放置一个比数列中最大数还小的数,结果正确;反之在\$6006单元中放置一个比数列中最大的数还大的数,结果错误。这是因为,本题中取一个数和另外4个数比较,比较次数为4,可是程序中CPX \$6000指令,\$6000的长度为5,这就出现了差错。

另外,程序9.17中有重复运用相同的指令,可以改为调子程序的方法解决。

应该特别指出的,本程序的思路正是多个无规则数排成有规律的数(顺序增加或减小)的基础,即所谓排序。关于排序问题的深入讨论将放在本章稍后部份叙述。

为使程序9.17能正确运行,现改成9.18,程序9.18也只是一个参考程序,事实上还有其它多种方法,有兴趣的读者可以自行改动。

程序9.18:

0300-	A2	00		LDX	# \$00
0302-	AD	01	60	LDA	\$6001
0305-	DD	02	60	CMP	\$6002, X
0308-	90	06		BCC	\$0310
030A-	E8			INX	

030 B-	E0	04	CPX	# \$ 04
030 D-	D0	F6	BNE	\$ 0305
030 F-	60		RTS	
0310-	BD	02 60	LDA	\$ 6002,X
0313-	8D	01 60	STA	\$ 6001
0316-	20	0A 03	JSR	\$ 030A
0319-	60		RTS	

3. 数 据 比 较

搜索就是查找,又称检索数据(信息),检索在计算机应用中是一种十分常用和重要的功能,它可以从计算机存贮的大量数据(信息)中迅速而准确地查找出某个需要的信息。而比较又是检索的基础,所以我们先从比较的实例出发,弄清比较的概念,然后再介绍检索的方法。

例 1: 比较两个操作数是否相同,若相同给出 00 标志;否则给出 FF 标志。

有人给出程序 9.19, 请验证一下程序的正确性。

如① (00) = 4A ② (00) = 4A ③ (00) = 4B
 (01) = 4A (01) = 4B (01) = 4A

若两单元中操作数相同, 00 标志放在 (02) 单元中; 不同 FF 标志放在 (03) 单元中。

程序 9.19:

0300-	A5	00	LDA	\$ 00
0302-	C5	01	CMP	\$ 01
0304-	B0	08	BCS	\$ 030E
0306-	A9	FF	LDA	# \$ FF
0308-	85	03	STA	\$ 03

```

030A- 20 ED FD      JSR    $FDED
030D- 60              RTS
030E- A9 00          LDA    # $00
0310- 85 02          STA    $02
0312- 20 ED FD      JSR    $FDED
0315- 60              RTS

```

程序9.19, 对第①、②两种情况判断是正确的, 但对第③种情况判断却是错误的, 请读者自行核实, 找出出错原因并改正之。

对于数或字符串的比较是否相同问题, 最好用异或指令EOR, 它的功能是将存储器同累加器内容异或, 并将结果送累加器, 即 $A \vee M \rightarrow A$ 。

对于本题可用程序9.20。

程序9.20:

```

0300- A5 00          LDA    $00
0302- 45 01          EOR    $01
0304- D0 06          BNE    $030C
0306- 85 02          STA    $02
0308- 20 ED FD      JSR    $FDED
030B- 60              RTS
030C- A9 FF          LDA    # $FF
030E- 85 03          STA    $03
0310- 20 ED FD      JSR    $FDED
0313- 60              RTS

```

程序说明:

0300: 取一个数

0302: 和另一个数比较, 相同吗?

0304: 否, 转 \$030C

0306: 是, $00 \rightarrow (02)$

0308: 显示
 030B: 结束
 030C: 取不同标志
 030E: FF → (03)
 0310: 显示
 0313: 结束

程序9.20无论对输入的两个操作数相同或不同（包括是有符号数还是无符号数）均能正确处理。

例2：检查两个字节的操作数是否完全一样

例如，① { (6001) = 4 B
 (6002) = 2 A
 (6003) = 4 B
 (6004) = 2 C
 不同 (02) = FF

② { (6001) = 4 B
 (6002) = 2 A
 (6003) = 4 B
 (6004) = 2 A
 相同 (01) = 00

程序如9.21所示：

程序9.21:

0300-	A2	00	LDX	# \$ 00
0302-	BD	01 60	LDA	\$ 6001,X
0305-	5D	03 60	EOR	\$ 6003,X
0308-	D0	0B	BNE	\$ 0315
030A-	E8		INX	
030B-	E0	02	CPX	# \$ 02
030D-	D0	F3	BNE	\$ 0302
030F-	85	01	STA	\$ 01
0311-	20	ED FD	JSR	\$ FDED
0314-	60		RTS	
0315-	A9	FF	LDA	# \$ FF
0317-	85	02	STA	\$ 02

```

0319-    20    ED    FD    JSR    $FDED
031C-    60                      RTS

```

程序说明:

```

0300: 00→x
0302: 取一个数
0305: 相同吗?
0308: 否, 转
030A: 是X+1→X
030B: 都比较完吗?
030D: 否, 转
030F: 是, 取00标志
0311: 显示
0314: 结束
0315: 取不同标志FF
0317: FF→(02)
0319: 显示
031C: 结束

```

程序9.21中, 首先是低位字节内容相比, 若相同计数器X加1, 再比高字节内容, 直到比较完所有数, 若有一个字节内容不同, 即不再比较, 并给出不同标志FF, 结束。

注: 程序9.19、9.20和9.21中, 用了JSR \$FDED 数令, 这是调用监控中字符显示子程序, 两数或两字节操作数相同时, 屏幕上显示@字符, 反之显示*符号。

例3: 比较两个字符串是否相同。字符串长度放在\$00单元中, 两个字符串的起始地址各为\$6000和\$6100, 如果这两个字符串相同, 将\$01单元清0, 否则将\$01单元置为FF。

```

例如, ①(00) = 03          ②(00) = 03
      (6000) = C3          (6000) = C 3

```

(6001) = D 0	(6001) = D 0
(6002) = D 5	(6002) = D 5
(6100) = C 3	(6100) = B 3
(6101) = D 0	(6101) = D 0
(6102) = D 5	(6102) = D 5
结果: (01) = 00	结果: (01) = FF

程序见9.22。

程序9.22:

0300-	A2	00	LDX	# \$ 00
0302-	A0	FF	LDY	# \$ FF
0304-	BD	00 60	LDA	\$ 6000,X
0307-	DD	00 61	CMP	\$ 6100,X
030A-	D0	07	BNE	\$ 0313
030C-	E8		INX	
030D-	E4	00	CPX	\$ 00
030F-	D0	F3	BNE	\$ 0304
0311-	A0	00	LDY	# \$ 00
0313-	84	01	STY	\$ 01
0315-	60		RTS	

程序说明:

0300: 计数器初值00→X
 0302: 不同标志FF→Y
 0304: 取一个字符
 0307: 和另一字符比较
 030A: 不同, 转 \$ 0313
 030C: 相同, X + 1 → X
 030D: 字符串比较完?
 030F: 否, 转 \$ 0304

0311: 是, 取相同标志

0313: 存标志

0315: 结束

程序9.22几点说明:

(1) 用X寄存器作为计数器, 用Y寄存器放标志用。

(2) 只要字符串(又称字串)中有一个字符同另一字符串中一个字符不同, 即不再比较, 也不计数, 给出不同标志FF结束。

(3) 在两个字符串比较时, 是从前向后逐个字符比较, 相同时计数器加1, 只要没有比较完, 即再比较直至完毕, 若字符完全相同则给出00标志。

(4) 本程序结构较好, 只用一个Y寄存器作标志暂存, 最后结果看\$01单元内容, 相同为00, 不同为FF。

4. 检 索

检索的方法主要有三种:

- ① 直接取表法;
- ② 顺序查找法;
- ③ 对分搜索法。

(1) 直接取表法 直接取表法是一种查表技术, 它是程序设计中的一项重要内容。直接取表法适用于有序表格, 被查找的值与该值在表中的排列位置有严格的对应关系。直接取表法是最快速的查表方法, 因为不需要计算, 同时也比较简单, 因为不需要去设计数学方法以及测试这个方法。直接取表法常用来计算超越函数与三角函数、使输入线性化、转换数码以及执行其它数学上的事务等。

直接取表法, 又称索引定址法。其思路是:

① 必须首先建立一个表，表内只包含问题的所有答案即可。

② 表的组成是以问题的答案能从其内简单地得到为原则。

③ 为了从表中得到正确答案，必须知道答案在表内的位置（称为索引），以及表的基底位址（又称起始地址），这样，基底位址加索引即为答案所在的地方。

为了说明直接取表法的思想和用法，请看两个实例：

例 1：求 0—15 平方值

首先建立 0—15 平方值表，它们放在 \$6000 开始的单元中，即：

6000- 00 01 04 09 10 19 24 31

6008- 40 51 64 79 90 A9 C4 E1

其次安排一个存贮单元例如 \$01 单元，作为被求平方值的数字索引的地方。

最后编一段小程序，见 9.23。

程序 9.23：

0300-	A4	01	LDY	\$01
0302-	B9	00 60	LDA	\$6000,Y
0305-	85	02	STA	\$02
0307-	20	DA FD	JSR	\$FDDA
030A-	60		RTS	

程序说明：

0300：放求某数平方的数

0302：取答案

0305：放结果至 (02)

0307：显示 16 进制数答案

030A：结束

例如求 7 的平方，那么只要在 \$01 单元放数字 7，执行程序 9.23，即 300G↵，屏幕上显示 31，即 10 进制的 49。原理非常简单，当 Y 寄存器从 01 单元取数字 7 后，则 LDA \$6000, Y，即指向 \$6007 单元取数，即为答案。

在例 1 中，求平方值最大只能求 15 的平方（即 225 或 16 进制表示为 \$E1），若要求大于 15 的平方（如 $16^2 = 256$ ，16 进制表示为 \$100），则源程序无法处理，因为超过 15 的平方后，其值一个存储单元不能表示，为此，对于求 0 到 255 的平方值问题，只能另编程序。

例如，求 255^2 ，计算结果知 $255^2 = (65025)_{10} = (\$FE01)_{16}$ 。

我们可以取 \$06 单元作为存放被求平方值的数字索引地方，如求 255^2 可以在 \$06 单元放 \$FF = $(255)_{10}$ ，而把 \$6000 + \$FF = \$60FF 的地方放 \$FE，同时把 \$6100 + \$FF = \$61FF 的地方放 \$01，这样，两个地址的内容组合在一起，就是 255^2 的值。

$$\because 255^2 = 65025 = 65024 + 1 = \$FE00 + \$0001$$

故有程序 9.24。

程序 9.24:

0300-	A4	06		LDY	\$06
0302-	B9	00	60	LDA	\$6000,Y
0305-	20	DA	FD	JSR	\$FDDA
0308-	B9	00	61	LDA	\$6100,Y
030B-	20	DA	FD	JSR	\$FDDA
030E-	60			RTS	

其中 \$6000—\$60FF 存放的是 0—255 平方值的 16 进制数的高位值，而 \$6100—\$61FF 存放的是 0—255 平方值的 16 进制数的低位值。

\$6000—\$60FF以及\$6100—61FF 的值，可以方便地用一个BASIC程序计算出。即：

```
10  FOR I=0 TO 255
20  A=I*I
30  B=INT(A/256)*256
40  C=A-B
45  IF B>255 THEN B=B-INT
    (A/256)*255
50  POKE 24576+I,B: POKE 24832+
    I,C
60  NEXT
70  END
```

本题的运行方法：

① 首先运行BASIC程序，可以在监控下看到\$6000—\$60FF以及\$6100—\$61FF中自动存放了 0—255 的平方值（均为16进制数），高位在\$6000—\$60FF中，低位在\$6100—\$61FF中。

② 在监控下从\$06单元键入待求某数平方的数。

③ *300G↵，立即显示某数的平方（16进制数）。

实例：

*06:10↵	相当于求16的平方
*300G↵	
*100	即 $16^2 = 256 = \$100$
*06:FF↵	相当于求255的平方
*300G↵	
*FE01	即 $255^2 = 65025 = \$FE01$
*06:64↵	相当于求100的平方
*300G↵	
*2710	即 $100^2 = 10000 = \$2710$

例2：将16进制数转换成ASCII码，这个问题我们曾在机器语言程序设计一章中介绍过。现在用查表法就显得特别简单，请看程序9.25。

程序9.25：

```
0300-   A4   01           LDY   $ 01
0302-   B9   00   60     LDA   $ 6000,Y
0305-   85   02           STA   $ 02
0307-   20   DA   FD     JSR   $ FDDA
030A-   60           RTS
```

程序9.25和程序9.23完全一样，但\$01单元放的数是0—9和A—F中的任一个数字或字符，而\$6000开始的单元则顺序存放的是上述数字或字符对应的ASCII码值，即：

6000- B0 B1 B2 B3 B4 B5 B6 B7

6008- B8 B9 C1 C2 C3 C4 C5 C6

例如，在01单元放字母A，则300G↙，屏上显示C1，它就是A的ASCII码。

(2) 顺序查找法 顺序查找又称线性搜索，它是对无序表格进行检索的一种方法。因为无序表格中的每一纪录的排列没有任何规律，因此，这种查表方法是把信息从头到尾依次向下全部扫描一遍，从中找出所要查找的内容。这种方法的优点在于适用性强，无论记录按什么顺序排放，都可以查找出来。缺点是查找速度慢。不过，由于计算机本身的运算速度越来越快，加上我们用汇编语言编程，不是用高级语言编程，这个缺点也就不那么重要了。因此，对于大规模而不需要经常检索的线性表，采用顺序查找法是可行的，它可以省去对该表进行分类（又称排序）的操作。

顺序查找的实例，请见程序9.26。

例 1: 若内存中某个区域, 有 100 个数据或字符 (由对应的 ASCII 码表示), 要搜索其中是否有字符 \$ (它的 ASCII 码为 A 6), 若有则记下它的地址并显示出来。

设 01 单元存放待搜索的字符的 ASCII 码, 即 (01) = A 6

从 \$6000 单元开始到 \$6063 中存放 100 个数据或字符的 ASCII 码, 如:

* 6000.6063

6000-	A6	E9	EF	A6	E9	A6	EF	EF
6008-	ED	E9	EF	EF	E9	E9	EF	EF
6010-	F9	E9	EF	EF	E9	F9	EF	EF
6018-	ED	E9	EF	EF	E9	E9	EF	EF
6020-	F9	E9	EF	EF	E9	F9	EF	EF
6028-	ED	E9	EF	EF	E9	E9	EF	EF
6030-	F9	E9	EF	EF	E9	F9	FF	EF
6038-	ED	E9	EF	EF	E9	E9	EF	EF
6040-	10	00	06	06	00	10	16	16
6048-	04	10	06	16	00	00	16	16
6050-	10	10	06	06	00	10	16	06
6058-	04	10	06	16	00	00	06	16
6060-	A6	00	06	A6				

*

为了查找 \$ 字符, 最简便的方法就是从内存的起始地址 (要搜索的表格的) 开始, 把表格中的数据一个个依次取出来, 与要搜索的关键字相比较, 看其是否符合, 符合则显示其在内存的位置, 不符合再找继续比较, 直至所有的数据都比较完。故有程序 9.26。

程序 9.26:

0300-	A0	00	LDY	# \$ 00
-------	----	----	-----	---------

0302-	B9	00	60	LDA	\$ 6000, Y
0305-	45	01		EOR	\$ 01
0307-	D0	0C		BNE	\$ 0315
0309-	A9	60		LDA	# \$ 60
030B-	20	DA	FD	JSR	\$ FDDA
030E-	98			TYA	
030F-	20	DA	FD	JSR	\$ FDDA
0312-	20	48	F9	JSR	\$ F948
0315-	C8			INY	
0316-	C0	64		CPY	# \$ 64
0318-	D0	E8		BNE	\$ 0302
031A-	60			RTS	

程序说明:

0300: 寄存器Y作计数器

0320: 开始取第一个数

0305: 和关键字比较

0307: 不相等, 转

0309: 相等, 取地址高字节

03B: 显示

03E: 取地址低字节

03F: 显示

0312: 空三格

0315: $Y + 1 \rightarrow 1$

0316: 都比较完吗?

0318: 否, 转 \$ 0302

031A: 是, 结束

几点说明:

① 关键字存放在01单元, 本例是\$的ASCII码A 6, 若要查找其它字符, 只要在01单元放其它字符的ASCII码

值即可。

② 采用异或指令EOR, 如果表中的内容与01单元的内容完全一致, 则EOR的结果8位2进制均为0, 执行BNE指令时不执行分支动作, 而继续执行下一条指令; 反之跳转。

③ 值得指出的是, 本程序不能用LDX, TXA, INX, LDA \$6000, X这四条指令, 否则会出现死循环。请有兴趣的读者上机运行并分析原因。

④ 程序执行后会显示6000□□□6003□□□6005□□□6060□□□6063五个地址, 说明在这些地址单元中确实存放着\$字符的ASCII码值A 6。

(3) 对分搜索法 前面介绍的线性查找法其速度是比较慢的, 这是因为对于由N个记录组成的集合进行一次成功的查找, 在最好的情况下只需要一次关键字的比较; 而最坏的情况则需要进行N次的比较。其中平均查找次数为 $K_L = N/2$ 。所以当N足够大时, 线性查找的速度是很慢的。如 $N = 64K = 64 \times 1024 = 2^{16} = 65536$, 用线性查找其平均次数高达32768次。减少搜索次数, 提高查找速度, 找到一个更有效的检索方法, 无论在理论上还是实践中都是有意义的。

我们这里介绍的对分搜索法(又称二分查找法), 就是一个比较理想的查找方法。它的平均查找次数 $K_B = \log_2 N - 1$ 。如 $N = 64K = 2^{16}$, 则对与搜索的平均查找次数仅为 $K_B = \log_2 2^{16} - 1 = 15$ 次。

由此可知, 对分搜索要比顺序查找优越得多。一个明显的例子, 在一个藏书高达数万册的图书馆里, 要进行现代化管理, 对分搜索技术, 大有用武之地。

对分搜索的主要思想是: 先取数组中间值 $e_{N/2}$ ($N/2$)

处的值)与要搜索的值 X 相比较,看是否相等,若相等则搜索到;若不等则比较两数的大小,若 $X > e_{N/2}$,则下一次取 $N/2 - N$ 之间的中间值 $e_{3N/4}$ 与 X 相比较;若 $X < e_{N/2}$,则下一次取 $0 - N/2$ 之间的中间值 $e_{N/4}$ 与 X 相比较,这样,每搜索一次使区界缩小 $1/2$,如此一直进行下去,直至或者是被搜索的字找到;或者搜索的区间变为 0 ,则表示搜索不到所要找的数。

对分搜索的前提必须是数组中的值按序排到(从大到小,或从小到大)。若是字符,则应按其ASCII码值的大小顺序排列。对于无序数组,则应先设法加以排序(即分类)。

假定有一个有序数组,其记录放在\$6000—\$605D的94个单位(\$5E)中,记录的个数(\$5E)存放在零页地址\$06单元中,待检索的数放在\$07单元中,今要求从有序数组中快速检索出该元素,若有,则显示出该元素所在地址;若无,则给出NO字样。

源程序9.27安排在\$1000—\$1055单元中:

程序9.27:

1000-	A2	00	LDX	# \$00
1002-	A5	06	LDA	\$06
1004-	85	08	STA	\$08
1006-	C6	08	DEC	\$08
1008-	E4	08	CPX	\$08
100A-	10	31	BPL	\$103D
100C-	E6	08	INC	\$08
100E-	8A		TXA	
100F-	65	08	ADC	\$08
1011-	4A		LSR	

1012-	A8			TAY	
1013-	B9	00	60	LDA	\$ 6000,Y
1016-	C5	07		CMP	\$ 07
1018-	F0	0C		BEQ	\$ 1026
101A-	90	05		BCC	\$ 1021
101C-	84	08		STY	\$ 08
101E-	4C	06	10	JMP	\$ 1006
1021-	98			TYA	
1022-	AA			TAX	
1023-	4C	06	10	JMP	\$ 1006
1026-	84	08		STY	\$ 08
1028-	A9	60		LDA	# \$ 60
102A-	20	DA	FD	JSR	\$ FDDA
102D-	A5	08		LDA	\$ 08
102F-	A8			TAY	
1030-	20	DA	FD	JSR	\$ FDDA
1033-	20	48	F9	JSR	\$ F948
1036-	B9	00	60	LDA	\$ 6000,Y
1039-	20	DA	FD	JSR	\$ FDDA
103C-	60			RTS	
103D-	EA			NOP	
103E-	EA			NOP	
103F-	A4	08		LDY	\$ 08
1041-	B9	00	60	LDA	\$ 6000,Y
1044-	C5	07		CMP	\$ 07
1046-	D0	03		BNE	\$ 1048
1048-	4C	26	10	JMP	\$ 1026
104B-	A9	CE		LDA	# \$ CE
104D-	20	F0	FD	JSR	\$ FDF0

```

1050-    A9    CF            LDA    # $ CF
1052-    20    F0    FD      JSR     $ FDF0
1055-    60                        RTS

```

有序数组记录放在\$6000—605D中:

```

6000- 01  02  03  04  05  06  07  08
6008- 09  0A  0B  0C  0D  0E  0F  10
6010- 11  12  13  14  15  16  17  18
6018- 19  1A  1B  1C  1D  1E  1F  20
6020- 21  22  23  24  25  26  27  28
6028- 29  2A  2B  2C  2D  2E  2F  30
6030- 31  32  33  34  35  36  37  38
6038- 39  3A  3B  3C  3D  3E  3F  40
6040- 41  42  43  44  45  46  47  48
6048- 49  4A  4B  4C  4D  4E  4F  50
6050- 51  52  53  54  55  56  57  58
6058- 59  5A  5B  5C  5D  5E

```

运行实例:

```
]CALL  -151↵
```

```

*6:5E          (放数组元素的个数)
*7:00          (放待检索的元素)
*1000G         (运行源程序)
*NO            (未查找到)
*7:01          (放待检索的元素)
*1000G
  6000 01      (查找到了, 元素01在 $ 6000)
*7:4C          (放待检索的元素)
*1000G
  604B 4C      (查找到了, 元素 4 C在 $ 604B)
*7:80          (放待检索的元素)

```

* 1000G

NO (找不到)

源程序9.27的几点说明:

① 对分搜索到最后的被搜索记录还剩一个时,或者搜索不到,给出“NO”表示;或者搜索到,给出待搜索元素的地址及该元素。在程序中特别安排了一条DEC \$08指令以及CPX \$08和BPL \$103D两条指令,由于BPL指令是在N = 0 (结果为正)时转移,故当确实是最后一个记录时,应转向\$103D单元,并将最后一个记录地址中的内容与\$07单元待检索的元素比较,若两者不相等(执行BNE \$1049指令,结果不为0转),则转向显示找不到信息(见\$1049—\$1055);若两者相等(执行JMP \$1026指令),转向\$1026开始的一段子程序,显示找到的地址和内容(\$1026—\$103C)。

显然,如果被搜索的记录不是最后一个时,记录的个数应加1,见\$100C—E6 08 INC \$08,这是相对于DEC \$08指令安排而言的。

② 对分搜索是把待查找的记录一分为二,在程序中主要用了以下几条指令:

```
LDX  # $00; 取区间上限
LDA  $06;    $06单元为数组元素个数
STA  $08;    元素个数暂存$08单元初始化时为区间下限
TXA;        区间上限→A,初始化时,区间上限为0
ADC  $08;    区间上限加区间下限
LSR;        (区间上限+区界下限)/2
TAY;        区间一半的位置暂存Y
LDA  $6000,Y;取区间一半位置对应的元素之值
CMP  $07;    和待查搜索的關鍵字比较
```

BEQ \$ 1026; 搜索到关键字转 \$ 1026 的程序段, 并显示结果
BCC \$ 1021; 不是关键字, 再判一下有无进位, 无行位(借位)
转
STY \$ 08; 有进位暂存 \$ 08
JMP \$ 1006; 循环上去, 再对分搜索

5. 排 序

排序(又称分类)问题在数据(信息)处理中, 占有很重要的地位。其目的是根据某种预先规定的准则, 把数据按一定的顺序排列起来, 以便在使用时能尽快地找到它。

从数的符号来分, 有无符号数排序和有符号数排序两种。

从结构上来分, 有从前向后(从上到下)和从后向前(从下到上)排序两种。

从要求上来分, 有从小到大增序和从大到小逆序两种排序。

从方法上来分, 有冒泡排序、交换排序、脉动排序和快速排序等多种方法。

对于这些名目繁多的排序, 我们摘其主要的用实例分析的方法列举于后。

例 1: 8 位无符号数排序

目的: 将一组无符号数数组, 按由大到小的降序重新排列。数组的长度放在存贮单元 \$6000 中, 而数组本身从存贮单元 \$6001 开始存放。

示范题: $(6000) = 04$

$(6001) = 12$

$(6002) = 03$

(6003) = 15
 (6004) = 08
 结果: (6001) = 15
 (6002) = 12
 (6003) = 08
 (6004) = 03

程序编制按冒泡排序方法，从后向前（从下到上）逐个比较，最后按降序（由大到小）排列。见程序9.28。

程序9.28:

0300-	A0	00		LDY	# \$ 00
0302-	AE	00	60	LDX	\$ 6000
0305-	CA			DEX	
0306-	BD	00	60	LDA	\$ 6000,X
0309-	DD	01	60	CMP	\$ 6001,X
030C-	B0	0D		BCS	\$ 031B
030E-	A0	01		LDY	# \$ 01
0310-	48			PHA	
0311-	BD	01	60	LDA	\$ 6001,X
0314-	9D	00	60	STA	\$ 6000,X
0317-	68			PLA	
0318-	9D	01	60	STA	\$ 6001,X
031B-	CA			DEX	
031C-	D0	E8		BNE	\$ 0306
031E-	88			DEY	
031F-	F0	DF		BEQ	\$ 0300
0321-	60			RTS	

程序说明:

0300: →Y

0302: 取数据长度

0305: 长度减 1
 0306: 取前一个数
 0309: 与后一个数比较, 哪个大?
 030C: 前一个数大, 转 \$ 031B
 030E: 后一个数小, $Y + 1 \rightarrow Y$
 0310: 小数进栈
 0311: 取大数
 0314: 交换
 0317: 小数出栈
 0318: 放小数
 031B: $X - 1 \rightarrow X$
 031C: 一轮比较完? 否, 继续比
 031E: 是 $Y - 1 \rightarrow Y$, 准备下比下一轮
 031F: 最后一轮比完否? 否, 继续比
 0321: 是, 结束

运行9.28程序, 第一轮后数组的最后顺序为15、12、03、08; 第二轮后数组的最后顺序为15、12、08、03; 第三轮后元素都已按顺序排好, 不再交换。

本程序有一缺点, 即在第二轮结束后, 虽然已经排好序, 但还要重复一轮 (第三轮是重复的)。原因在于要求 $N - 1 = 4 - 1$ 轮比较。不仅如此, 如果原数组已经排好序, 执行本程序也还要进行 $N - 1$ 轮比较。所以本程序有很多需要改进的地方, 而且新的排序技巧也是目前研究的一个重要领域。

若要求由小到大增序排列, 从上到下 (从前到后) 逐个比较, 可用程序9.29, 数据长度为07。

程序9.29:

```
0300-   AD   00   60       LDA   $ 6000
```

0303-	85	07		STA	\$ 07
0305-	C6	07		DEC	\$ 07
0307-	A9	01		LDA	# \$ 01
0309-	85	06		STA	\$ 06
030B-	A2	00		LDX	# \$ 00
030D-	A4	06		LDY	\$ 06
030F-	BD	01	60	LDA	\$ 6001,X
0312-	D9	01	60	CMP	\$ 6001,Y
0315-	90	0B		BCC	\$ 0322
0317-	48			PHA	
0318-	B9	01	60	LDA	\$ 6001,Y
031B-	9D	01	60	STA	\$ 6001,X
031E-	68			PLA	
031F-	99	01	60	STA	\$ 6001,Y
0322-	C8			INY	
0323-	CC	00	60	CPY	\$ 6000
0326-	D0	E7		BNE	\$ 030F
0328-	E6	06		INC	\$ 06
032A-	E8			INX	
032B-	E4	07		CPX	\$ 07
032D-	D0	DE		BNE	\$ 030D
032F-	60			RTS	

程序说明:

0300: 取数据长度

0303: 长度 \rightarrow (07)

0305: (07) = 长度减 1

0307: 01 \rightarrow A

0309: (06) = 01

030B: X = 00

030D: $Y = 01$
030F: 前一数与后面各数比较
0312: 哪一个数大?
0315: 前一数小, 转 \$ 0322
0317: 前一数大, 进栈
0318: 交换位置, 小数放前, 大数放后
0322: $Y + 1 \rightarrow Y$
0323: 一轮比完否?
0326: 否, 继续比
0328: 是, $Y + 1, X + 1$
032A: 准备比下一轮
032B: 最后一轮比完吗?
032D: 否, 继续比
032F: 是, 结束

例 2: 一个实用排序实例

设有256个数, 它们存放在 \$ 6000—\$ 60FF 的256 单元中, 这些数未按值的大小存放。由于数据量较大, 我们可以利用随机函数的定义和POKE指令, 编制一个BASIC程序, 运行该程序后, 即可以在 \$ 6000—\$ 60FF 单元中, 看到这些随机数已存放好。

```
10  FOR I= 0  TO  255
20  A=INT(RND(1)*100)*2
30  POKE  24576+I,A
40  NEXT
50  END
]
```

为了实现无序数组的有序排列, 我们编制一个机器语言程序, 并设 \$ 06单元为存放数组中记录 (或元素) 的个数,

而\$08,\$09单元则为存放数组元素的起始地址,即在运行机器语言程序之前键入:

* 6: FF↵

* 8: 00↵

* 9: 60↵

然后*1000G↵,即可看到255个数已按大到小顺序排列好(*6000·60FF↵)

见程序9.30。

程序9.30:

1000-	A0	00	LDY	# \$00
1002-	A6	06	LDX	\$06
1004-	F0	1E	BEQ	\$1024
1006-	86	07	STX	\$07
1008-	B1	08	LDA	(\$08),Y
100A-	C8		INY	
100B-	D1	08	CMP	(\$08),Y
100D-	B0	0E	BCS	\$101D
100F-	48		PHA	
1010-	B1	08	LDA	(\$08),Y
1012-	88		DEY	
1013-	91	08	STA	(\$08),Y
1015-	68		PLA	
1016-	C8		INY	
1017-	91	08	STA	(\$08),Y
1019-	A9	00	LDA	# \$00
101B-	85	07	STA	\$07
101D-	CA		DEX	
101E-	D0	E8	BNE	\$1008

1020-	A5	07	LDA	\$ 07
1022-	F0	DC	BEQ	\$ 1000
1024-	60		RTS	

十、语句及自定义功能扩展

本章介绍中华学习机部分语句及自定义功能扩展的原理和方法，目的是扩大中华学习机使用范围，为操作和应用带来更多方便。把一种机器原来设计好的各种功能，进一步加以扩充和发展，是一个相当复杂和困难的技术，本章没有也不可能解决众多问题，而是对比较简单且有实用意义的问题作一些初步探讨，以求更进一步地学习和提高。

1. 单个命令或程序段功能键的自定义

中华学习机没有自定义功能键，这对操作者使用一些常用命令带来不便，例如列文件目录的命令 CATALOG，一般都是一个字符一个字符的从键盘敲入；又如删除文件命令 DELETE 既长，稍不在意又会敲错，如果我们用一个符号或一个按键代替命令，不仅节省字符敲入的时间，而且减少输入命令的错误。

用单个字符定义功能键最简单最常用的是用 & 键，它的入口地址为 \$03F5，通常情况下 \$03F5—4C 58 FF，其中 4C 为无条件转移指令 JMP 的指令操作码，其寻址方式是绝对寻址，因而 4C 后面的两个字节值，即为转向的地址。如果，我们把 \$3F6, \$3F7 中的内容改为某一命令的入口地址，则执行 & 命令后就转向了某一命令去执行，从而 & 键就被定义成另一个命令。

例如，3F5: 4C 6E A5 ↵ 并按 CTRL-C 返回 BASIC

后，则按下&键并回车，计算机就自动列出磁盘中所有文件的目录。这里的\$A56E是CATALOG命令的入口地址。

因此，在运行程序前键入某一命令的入口地址(低位放入\$3F6中，高位放入\$3F7中)，就可以用&键定义不同命令。

如3F5: 4C 49 D6↵，返回BASIC后，在BASIC程序的最后一句行号后面放入&，则当BASIC程序运行后就清除全部程序，从而达到程序的加密作用，因为\$D6 49是NEW命令的入口地址。当然，为了防止BASIC程序或机器语言程序在执行过程中被CTRL-RESET键而暂停，从而被列出程序，可在执行程序前将一数值(如\$80)写入\$3F4中。为了更好地保护程序，防止程序被CTRL-C中断，可在程序的最前面加入一句:ONERR GOTO XXXX，而XXXX则为程序最后一句的行号，这样，即使别人按CTRL-C，程序转向执行最后一句NEW，从而清除原程序。

由此可见，定义单个命令功能键的方法还是比较简单的。

当然，定义单个命令功能键，也不仅仅局限于用&，例如还可以用CTRL-Y来定义。由于CTRL-Y的入口地址为\$FECA，一般情况下\$FECA—4C F8 03，亦按CTRL-Y后相当于转向\$03F8，因此若在\$03F8单元开始，再安排一条跳转指令，如4CYY XX(相当于JMP \$XXYY)，则可跳转到以\$XXYY开头去执行。

如:]CALL-151↵

* 3F8: 4C 6E A5↵

* 0G↵

]CTRL-Y↵

则同样可以显示所有文件的目录，这里CTRL-Y和

CATALOG命令等效。

用&或CTRL-Y键也可以定义一段机器语言程序，例如用&调用一段机器语言程序。

```
]CALL-151↵
```

```
* 3F5: 4C D8 A0↵
```

```
* A0 D8: 20 7B DD 20 52 E7 20 1A D6 90 03  
4C 41 D9 A2 5A 4C 12 D4
```

```
* 0G↵
```

```
]
```

这样&键既可以从键盘送入，也可放在程序中执行，均可以运行\$A0D8到\$A0EA的一段机器语言子程序。需要说明的是机器语言子程序既可以放在\$3F5以后或\$300以后的区间，也可以存在内存中其它部分，前者适用于机器语言较短（字节数较少）；后者适用于机器语言较长（字节数很多），因为前者存贮空间留给使用者的位置不多，而后者也应放置在安全区，例如不会与其它有用程序干扰等等。

实际上，我们还可以用多个键（不一定是&，CTRL-Y）定义多个功能命令；也可以用一个&命令后面带不同数码定义多个功能命令等等，下面我们继续介绍。

2. 多个命令键的自定义

在进行机器语言编程和调试的过程中，常常在监控和汇编状态转换；完成调试机器语言后，又常常返回BASIC，为了操作方便，我们可以定义三个功能键：F4进入监控，F5进入汇编，TEST进入BASIC状态。程序如10.1所示：

程序10.1:

```
0300- A9 0B LDA # $0B
```

0302-	85	36	STA	\$ 36
0304-	A9	03	LDA	# \$ 03
0306-	85	37	STA	\$ 37
0308-	6C	F2 03	JMP	(\$ 03F2)
030B-	48		PHA	
030C-	AD	00 C0	LDA	\$ C000
030F-	C9	14	CMP	# \$ 14
0311-	D0	03	BNE	\$ 0316
0313-	4C	69 FF	JMP	\$ FF69
0316-	C9	06	CMP	# \$ 06
0318-	D0	03	BNE	\$ 031D
031A-	4C	50 D3	JMP	\$ D350
031D-	C9	16	CMP	# \$ 16
031F-	D0	03	BNE	\$ 0324
0321-	4C	A5 D6	JMP	\$ D6A5
0324-	A9	00	LDA	# \$ 00
0326-	BD	10 C0	LDA	\$ C010,X
0329-	68		PLA	
032A-	4C	F0 FD	JMP	\$ FDF0

程序说明:

0300:

⋮

0306: 指针初始化,指向 \$ 030B

030B: 进栈

030C: 读键盘

030F: 是F4?

0311: 否, 转 \$ 0316

0313: 是, 进监控

0316: 是F5?

0318: 否, 转\$031D

031A: 是, 进小汇编

031D: 是TEST?

031F: 否, 转\$0324

0321: 是, 进入BASIC

0326: 清键盘

0329: 出栈

032A: 输出字符至屏幕

本程序的设计思路是将输入指针指向本程序, 比较是否输入 F4、F5、TEST 键, 经判断后转相应程序, 若没有按键输入, 则调用输出字符至屏幕的子程序\$FDF0。本程序在不含DOS的西文状态下通过。

零页地址\$36和\$37两个单元, 系指出输出字符的外部设备地址, 通常\$36:F0, \$37:FD, 即指向显示器工作地址\$FD-F0, 将累加器中的字符送往屏幕; 若要转向打印机, 则将设备码\$C100置入\$36, \$37中, 即\$36:00, \$36:C1, 再调用\$FD-ED输出子程序, 把累加器 A 中的内容送往打印机; 本程序则用来初始化指针, 让其转至 \$030B (见\$0300—\$0307)。

程序中#\$14, #\$06, #\$16分别为CTRL-T, CTRL-F, CTRL-V的机内ASCII码, 等同F4、F5、TEST 键。

用&命令定义一个功能键, 我们已经熟悉了, 但不能满足定义多个功能键的要求, 下面的程序可以使&后面带上一些数字或字符, 从而可以定义许多功能键。例&8代表HGR 等等。

首先在监控下定义&键, 让其指向\$0300开始的机器语言程序:

```
]CALL-151↵
```


* 3F5: 4C 00 03 ✓

然后键入下述机器语言见程序10.2, 为方便阅读同时给出相应的助记符和简单注释。

程序10.2:

0300-	A0	00	LDY	# \$ 00
0302-	B1	B8	LDA	(\$ B8), Y
0304-	48		PHA	
0305-	C8		INY	
0306-	20	98 D9	JSR	\$ D998
0309-	68		PLA	
030A-	C9	30	CMP	# \$ 30
030C-	D0	03	BNE	\$ 0311
030E-	4C	58 FC	JMP	\$ FC58
0311-	C9	31	CMP	# \$ 31
0313-	D0	03	BNE	\$ 0318
0315-	4C	A5 D6	JMP	\$ D6A5
0318-	C9	32	CMP	# \$ 32
031A-	D0	03	BNE	\$ 031F
031C-	4C	6D F2	JMP	\$ F26D
031F-	C9	33	CMP	# \$ 33
0321-	D0	03	BNE	\$ 0326
0323-	4C	6F F2	JMP	\$ F26F
0326-	C9	34	CMP	# \$ 34
0328-	D0	03	BNE	\$ 032D
032A-	4C	84 FE	JMP	\$ FE84
032D-	C9	35	CMP	# \$ 35
032F-	D0	03	BNE	\$ 0334
0331-	4C	80 FE	JMP	\$ FE80
0334-	C9	36	CMP	# \$ 36

0336-	D0	03	BNE	\$ 033B
0338-	4C	80 F2	JMP	\$ F280
033B-	C9	37	CMP	# \$ 37
033D-	D0	03	BNE	\$ 0342
033F-	4C	40 FB	JMP	\$ FB40
0342-	C9	38	CMP	# \$ 38
0344-	D0	03	BNE	\$ 0349
0346-	4C	E2 F3	JMP	\$ F3E2
0349-	C9	39	CMP	# \$ 39
034B-	D0	03	BNE	\$ 0350
034D-	4C	D8 F3	JMP	\$ F3D8
0350-	C9	41	CMP	# \$ 41
0352-	D0	03	BNE	\$ 0357
0354-	4C	99 F3	JMP	\$ F399
0357-	C9	42	CMP	# \$ 42
0359-	D0	03	BNE	\$ 035E
035B-	4C	12 D9	JMP	\$ D912
035E-	C9	43	CMP	# \$ 43
0360-	D0	03	BNE	\$ 0365
0362-	4C	69 FF	JMP	\$ FF69
0365-	C9	44	CMP	# \$ 44
0367-	D0	03	BNE	\$ 036C
0369-	4C	6E A5	JMP	\$ A56E
036C-	4C	C9 DE	JMP	\$ DEC9
036F-	00		BRK	

程序说明:

030A: 是“0”吗?

030C: 否, 转

030E: 是, 转HOME

0311: 是“1”吗?
0313: 否, 转
0315: 是, 转LIST
0318: 是“2”吗?
031A: 否, 转
031C: 是, 转TRACE
031F: 是“3”吗?
0321: 否, 转
0323: 是, 转NOTRACE
0326: 是“4”吗?
0328: 否, 转
032A: 是, 转NORMAL
032D: 是“5”吗?
032F: 否, 转
0331: 是, 转INVERSE
0334: 是“6”吗?
0336: 否, 转
0338: 是, 转FLASH
033B: 是“7”吗?
033D: 否, 转
033F: 是, 转GR
0342: 是“8”吗?
0344: 否, 转
0346: 是, 转HGR
0349: 是“9”吗?
034B: 否, 转
034D: 是, 转HGR2
0350: 是“A”吗?
0352: 否, 转

0354: 是, 转TEXT
 0357: 是“B”吗?
 0359: 否, 转
 035B: 是, 转RUN
 035E: 是“C”吗?
 0360: 否, 转
 0362: 是, 转CALL-151
 0365: 是“D”吗?
 0367: 否, 转
 0369: 是, 转CATALOG

上述程序10.2检查无误后, 存盘备用。使用方法很简单, 例如键入&D↵, 则列目录, 键入&9↵, 则进入高分辨第二页并清屏幕。

各定义功能键说明如下:

&0 — HOME 将光标移到左上方, 并清除文本窗
 &1 — LIST 显示出全部程序清单
 &2 — TRACE 显示每个语句的行号, 它不能由RUN, CLEAR, NEW, DEL 或 RESET 等所关断, 只由 NOTTRACE 所中止
 &3 — NO TRACE 关掉TRACE方式
 &4 — NORMAL 置屏幕为黑底白字方式
 &5 — INVERSE 置屏幕为白底黑字方式
 &6 — FLASH 置“闪烁”方式, 屏幕上以黑底白字和白底黑字交替进行, 可用NORMAL返回正常方式
 &7 — GR 将屏幕置低分辨图形方式, 底部留有四行文本区, 屏幕清除为黑色, 光标移入文本窗, COLOR置0
 &8 — HGR 置高分辨率图形方式(280×160), 底部留四行文本窗, 屏幕清除为黑色, 显示第一页存贮区(\$2000—\$3FFF)

& 9——HGR2	置全屏幕高分辨图形方式(280×192), 屏幕清除黑底色, 显示第二页存贮区 (\$4000-\$5FFF), 文本存贮区不受影响。
& A——TEXT	置屏幕为非图形文本方式, 显示20行40列字符
& B——RUN	运行程序
& C——CALL-151	从BASIC状态进入监控
& D——CATALOG	列示所有文件目录

3. USR函数及其应用

中华学习机浮点BASIC(APPLESOFT)语言中, 有一个USR函数, 其格式是: USR (算术表达式)。该函数功能很强, 应用较广, 但使用方法特殊。在有关手册上只有很简单的几句交待, 许多BASIC教程不是避而不谈, 就是述而不明, 因而使许多用户望而却步, 无法应用, 这里作一介绍, 以补有关书籍之不足。

USR(X)函数中的X, 可以是一个数, 也可以是一个合法的BASIC表达式。该函数被调用时, 其执行过程为:

① 将X的值算出之后保存在APPLESOFT的主浮点累加器(FAC)中。这里的主浮点累加器FAC, 不是累加器A, 而是一个从\$9D到\$A3的共有7个字节的存贮空间的累加器。

② 然后跳到\$0A处执行指令。即将系统的控制权交到\$0A—\$0C单元, 用户可在\$0A—\$0C单元中自行安排一条无条件转移指令JMP \$XXYY, 通过该指令将程序转移到某一地址(如\$XXYY)去执行。换句话说, 用户可以改变这三个字节的内容, 使之进入我们定义USR所指定的

地址。

③ 返回时以\$9D—\$A3中的值作为函数值。返回是指子程序返回，用RTS指令处理。即当需要从子程序取回一个数值（出口参数）时，必须把该数转换成相应的浮点格式，放入主浮点累加器FAC中。因此，FAC的当前值即为USR的函数值。

值得指出的是，由于机器语言程序一般处理的是整数形式的操作数，因此，通过主浮点累加器传递数据的过程中，就存在如何将浮点数转换成整数，以及如何将整数转换成浮点数的问题。

BASIC系统子程序中，有两个重要的子程序可以完成上述两种数制之间的转换，其入口地址分别为\$E10C和\$E2F2。前者将主浮点累加器中的浮点数转换成两个字节的整数，并放置在\$A0（高字节）和\$A1（低字节）两个单元中；后者则将两个字节的整数转换成浮点数，亦放在主浮点累加器中，但在进入该子程序（\$E2F2）前，应将转换的整数放入累加器A（高字节）及变址寄存器Y（低字节）中。

综合所述，USR（X）函数的执行过程可以简单归结为三句话：在\$0A—\$0C中放入转某个机器语言子程序的指令；让子程序对主浮点累加器中的数进行处理；返回。这样，我们在程序设计中就多了一个自定义函数的功能。

现举几个实例，说明使用的具体方法。

例1：编制一个程序，根据输入的X值完成乘2的功能。

首先编制一个机器语言子程序，要求将函数USR送入的表达式（其值应小于16384的正整数）乘2（即左移一位），并存入主浮点累加器中，其子程序（取名X2.OBJ0）为10.3。

0300-	20	0C	E1	JSR	\$E10C
0303-	06	A0		ASL	\$A0
0305-	06	A1		ASL	\$A1
0307-	90	04		BCC	\$030D
0309-	E6	A0		INC	\$A0
030B-	A5	A0		LDA	\$A0
030D-	A4	A1		LDY	\$A1
030F-	20	F2	E2	JSR	\$E2F2
0312-	60			RTS	

程序说明:

300: 从浮点累加器中取值

303: 左移高位字节

305: 左移低位字节

307: 无进位, 转

309: 高位字节加1

30B: 高位字节存累加器A

30D: 低位字节存暂存器Y

30F: 浮点数存FAC

312: 返回BASIC

将子程序X2.OBJ0存盘备用。

其次安排跳转指令, 指向定义USR所指定的地址。由于机器语言子程序X2.OBJ0存放在\$300开始的单元中, 所以应在\$0A—\$0C处设置跳转指令JMP \$0300 (4C 00 03)即:

0A: 4C

0B: 00

0C: 03

对应的BASIC程序中, 则用POKE指令处理:

POKE 10, 76

POKE 11, 00

POKE 12, 03

最后编制一个BASIC程序10.4, 实现调用:

```
100 REM USR(X) = X * 2 ($0300)
110 POKE 10,76: POKE 11,0: POKE
    12,3
120 D$ = CHR$(4)
130 PRINT D$; "BLOAD X2.OBJ0"
140 INPUT "X=", X
150 IF X = -1 THEN 190
160 Y = USR(X)
170 PRINT "Y=", Y
180 GOTO 140
190 END
RUN
X = 5
Y = 10
X = -1
END
```

运行结果表明, 函数USR(X) 等价于X乘2。

若删去110句, 应在程序运行前设置:

]CALL-151

*0A: 4C 00 03 ✓

]

例2: 用USR(K)代替EXP(K)

本例和例1不同, 不需要编制一个计算 e^x 的机器语言子程序, 而直接调用BASIC运算子程序\$EF09 (EXP(K)函数的入口地址), 这样, 程序设计比较简捷。

为此,可在\$0A—\$0C (10—12)中设置 JMP \$EF09 指令,这样就直接转向APPLESOFT的EXP子程序。该子程序可将主浮点累加器FAC中的值进行指数运算后返回。即:

\$0A: 4C——POKE 10, 76

\$0B: 09——POKE 11, 09

\$0C: EF——POKE 12, 239

这样, BASIC解释程序一遇到 USR(K), 立即转向 \$EF09, 用EXP子程序处理\$9D—\$A3中的数据。

见程序10.5。

程序10.5:

```
200 REM USR(K) = EXP($EF09)
210 POKE 10,76, POKE 11,09, POKE
    12,239
220 FOR K = 1 TO 10
230 PRINT USR (K)
240 NEXT K
250 END
```

例3: 用USR(J)代替SGN(J)

本例思路同例2, 即只要在\$0A—\$0C中存入“JMP \$EB90”, 从而转入APPLESOFT符号函数子程序(入口地址是\$EB90), 这样, 本例中的USR(J)等价于SGN(J), 故容易写出以下程序10.6。

程序10.6:

```
300 REM USR(J)SGN ($EB90)
310 POKE 10, 76, POKE 11,114, POKE
    12,235
320 FOR J = 1 TO 9
330 PRINT USR (J)
```

340 NEXT J

350 END

从上面三个例子，使我们清楚地看到，巧用USR(X)函数，可以自定义新的功能。例1是原理性说明，例2、例3是实用性说明。读者不难根据以上方法，自行设计或定义新的函数。为便于使用，今列出各函数子程序入口地址：

函数	入口地址	函数	入口地址
SGN	\$EB90	RND	\$EFAE
ABS	\$EBAF	COS	\$EFEA
INT	\$EC23	SIN	\$EFF1
SQR	\$EE8D	TAN	\$F03A
LOG	\$E941	ATN	\$F09E
EXP	\$EF09	^(乘方)	\$EF97

4. GOTO带变量和表达式

中华学习机APPLESOFT BASIC语言中，规定GOTO语句后面的行号只允许带数字（如GOTO30），但不允许带变量（如GOTO A）和表达式（如GOTO K*10+20），为扩大GOTO语句的使用范围，可用机器语言来处理，以完成GOTO〈变量和表达式〉的功能。

首先定义&命令，让其转向机器语言入口地址\$0300，即：

]CALL-151↵

*3F5: 4C 00 03↵

*CTRL-C↵

]

或：

]行号 POKE 1013,76:POKE 1014,0:POKE 1015,3
然后建立GOTO〈变量〉的机器语言子程序10.7。

程序10.7

0300-	20	7B	DD	JSR	\$DD7B
0303-	20	52	E7	JSR	\$E752
0306-	20	1A	D6	JSR	\$D61A
0309-	90	03		BCC	\$030E
030B-	4C	41	D9	JMP	\$D941
030E-	A2	5A		LDX	# \$5A
0310-	4C	12	D4	JMP	\$D412

程序说明:

0300: 取TXTPTR所指的变量

0303: 换成行号并置入行号指示器

0306: 找出指定行号的地址

0309: 找不到, 转

030B: 找到, 使程序转移

030E: 5A→X暂存器

0310: 显示出错信号

其中\$DD7B、\$E752、\$D61A、\$D941、\$D412均为BASIC系统子程序的入口, 它们各自的功能已列在上述程序清单的注释中。需要说明的是, 中华学习机零页有一对重要的地址指示器TXTPTR = (\$B8, \$B9), 当执行BASIC程序时, 系统便利用TXTPTR来指出当前所要读取的字符码或语句代码的地址。通常\$B8, \$B9是读键盘字符。

将上述机器语言子程序存盘备用, 并取名GOTO OBJ0。

为了验证GOTO.OBJ0是否具有GOTO<变量>的功能, 可用以下BASIC程序10.8实验:

程序10.8:

```
100 PRINT CHR$ (4); "BLOAD GOTO.
    OBJ0"
```

```

110 POKE 1013, 76; POKE 1014, 0; POKE
    1015,3
120 INPUT "K = (150,160,170,180)?",K
130 & K
150 PRINT "HOME"; GOTO 120
160 PRINT "HGR2"; GOTO 120
170 PRINT "TEXT"; GOTO 120
180 PRINT "STOP"; END
RUN
K = (150,160,170,180)? 160↵
HGR2
K = (150,160,170,180)? 180↵
STOP

```

说明建立的GOTO.OBJ0程序确实起到了GOTO带变量的作用, 130句中& K和GOTO K等效。

至于GOTO语句后面带表达式的问题, 机器语言子程序GOTO.OBJ0同样具有这个功能, 可用另一个BASIC程序10.9验证:

程序10.9:

```

200 INPUT A
210 & A * 10 + 220
220 PRINT "THIS IS A BASIC PROGRAM"; GOTO 200
230 PRINT "CPU-CENTRY PROCESSING UNIT"; GOTO 200
240 PRINT "END!"; END
JRUN
?0
THIS IS A BASIC PROGRAM

```

71

CPU-CENTRY PROCESSING UNIT

72

END!

5. GOSUB 带表达式

按照基本BASIC规定，GOSUB 语句后面可以带行号，但不可以带表达式，仿照GOTO语句功能扩展的例子，也可以编写一个机器语言子程序 (GOSUB·OBJ1) 10.10。

程序10.10:

0300-	20	7B	DD	JSR	\$ DD7B
0303-	20	52	E7	JSR	\$ E752
0306-	20	1A	D6	JSR	\$ D61A
0309-	90	1D		BCC	\$ 0328
030B-	A9	03		LDA	# \$ 03
030D-	20	D6	D3	JSR	\$ D3D6
0310-	A5	B9		LDA	\$ B9
0312-	48			PHA	
0313-	A5	B8		LDA	\$ B8
0315-	48			PHA	
0316-	A5	76		LDA	\$ 76
0318-	48			PHA	
0319-	A5	75		LDA	\$ 75
031B-	48			PHA	
031C-	A9	B0		LDA	# \$ B0
031E-	48			PHA	
031F-	20	B7	00	JSR	\$ 00B7
0322-	20	41	D9	JSR	\$ D941
0325-	4C	D2	D7	JMP	\$ D7D2

```
0328-   A2 5A           LDX   # $5A
032A-   4C 12 D4       JMP    $D412
```

其中\$75, \$76是零页地址的二个单元, 存放正在执行的行号。

类似GOTO语句的功能扩展, 可用如下实例试验, 见程序10.11。

程序10.11:

```
300 PRINT CHR$(4); "BLOAD GOSUB
      .OBJ1"
310 POKE 1013,76; POKE 1014,0; POKE
      1015,3
320 INPUT A
330 & A * 10 + 340; GOTO 320
340 PRINT "####1", RETURN
350 PRINT "$$$ $2", RETURN
360 END
]RUN
?0
####1
?1
$$$ $2
?2
```

6. RESTORE 带行号

在PC-1500机或IBM-PC系列机中, 恢复数据区语句RESTORE后面可以带一个表达式:

<行号> RESTORE <表达式>

其中<表达式>可以是数值, 它代表行号; 也可以是字符

串，它代表标号，表示指针拨到的位置。如 PC—1500 机中下述程序10.12就可以执行。

程序10.12:

```
10 DATA 8,5
20 DATA 20,30, "BASIC LANGUAGE"
30 READ A,B
40 READ C,D,K$
50 RESTORE 2 * A + 4
60 READ X,Y,S$
```

结果有:

A = 8, B = 5, C = 20, D = 30, K\$ = BASIC
LANGUAGE, X = 20, Y = 30, S\$ = BASIC
LANGUAGE

60句中的READ语句，所以有数可读，是借助于50句RESTORE的作用，它把数据区的指针拨到 $2 * A + 4 = 20$ 行的开始位置。

但是，中华学习机及APPLE-Ⅱ及兼容机没有上述功能，为扩大RESTORE语句功能，可用RESTORE.OBJ2机器语言实现，见程序10.13。

程序10.13:

```
0300- A9 0B LDA #$0B
0302- 8D F6 03 STA $03F6
0305- A9 03 LDA #$03
0307- 8D F7 03 STA $03F7
030A- 60 RTS
030B- 20 7B DD JSR $DD7B
030E- 20 52 E7 JSR $E752
0311- 20 1A D6 JSR $D61A
```

0314-	90	0B	BCC	\$ 0321
0316-	C6	9B	DEC	\$ 9B
0318-	A4	9B	LDY	\$ 9B
031A-	A5	9C	LDA	\$ 9C
031C-	84	7D	STY	\$ 7D
031E-	85	7E	STA	\$ 7E
0320-	60		RTS	
0321-	A2	5A	LDX	# \$ 5A
0323-	4C	00 00	JMP	\$ 0000

程序中用到的几个子程序同 GOTO·OBJ0, \$0300—\$0309 实际上是定义 & 键, \$9B, \$9C 是 BASIC 系统设置的浮点数据临时存放区 2 (TEMP2) 中的二个单元, 用来存放计算过程的中间结果, \$7D, \$7E 也是零页地址中的二个单元, 用来存放 READ 语句正在执行 DATA 语句的行号。

为了验证 RESTORE·OBJ2 程序, 是否具有 RESTORE N 的功能, 可用 BASIC 程序 10.14 来检查。

程序 10.14:

```

50  PRINT  CHR$ (4), "BLOAD RESTOR
    E.OBJ2"
100  CALL 768, READ A,B, & 400
200  READ C, PRINT "A=";A, "B="; B,
    "C=";C
300  DATA 10,20,30,40,50,60,70
400  DATA 100

```

运行结果表明, & 400 确实起到 RESTORE 400 的作用。

上述扩展 RESTORE 语句功能的做法, 不影响标准 RESTORE 的正常使用, 而在需要指向任意数据区时, 灵活自如, 为我所用。

7. 控制反汇编行数

机器语言也有一个 LIST 命令, 简称“L”命令, 其作用是从指定的首地址开始反汇编 20 条指令。因此它不同于 BASIC 的 LIST 命令。

当我们编好一段机器语言程序后, 要想打印机器语言及汇编语言程序清单, 一般都要用“L”命令, 而在程序较长时要用几个“L”命令, 但具体要用几个又不清楚, 用多了后面带来一些乱七八糟的指令; 用少了又不够, 因而使用不太方便, 这给打印一张整齐美观且行数又正好的清单带来麻烦。

现在提供一种方法, 只要在 \$06—\$09 四个零页地址分别设置程序始址和终址, 再键入 CALL4096↵或在监控下用 1000G↵, 再配合 CTRL-S 键, 就可以控制反汇编的行数, 使列表方便。

程序清单如 10.14 所示。

程序 10.14

1000-	A5	06	LDA	\$06
1002-	A4	07	LDY	\$07
1004-	48		PHA	
1005-	68		PLA	
1006-	85	3A	STA	\$3A
1008-	84	3B	STY	\$3B
100A-	20	D0 F8	JSR	\$F8D0
100D-	20	53 F9	JSR	\$F953
1010-	48		PHA	
1011-	98		TYA	
1012-	C5	09	CMP	\$09
1014-	30	EF	BMI	\$1005

1016-	D0 07	BNE	\$ 101F
1018-	68	PLA	
1019-	C5 08	CMP	\$ 08
101B-	30 E9	BMI	\$ 1006
101D-	F0 E7	BEQ	\$ 1006
101F-	60	RTS	

8. DOS 命令的简化

中华学习机共有 28 个 DOS 命令，键入这些命令少则要击 2 个键，如 FP；多则要击七、八个键，如使用最频繁的 CATALOG 命令，就有 7 个字符，使用起来颇感不便。为了提高 DOS 命令的键入速度，方便使用，本节给出一种简化 DOS 命令的方法。例如若用 C. 代表 CATALOG 命令，那么只要键入 C.↵，即可以列目录清单，实为简便，大大地提高了计算机的使用效率。

如何才能用简化的命令(可以根据自己的习惯自行定义)来代替中华机 DOS 系统中的命令呢？简单地说就是修改 DOS 命令名表。

开机后，DOS 被引导调入内存，并建立命令名表和命令入口地址表。

命令名表		命令入口地址
序号	(\$ A884—\$ A908)	(\$ 9D1E—\$ 9D55)
\$ 00	INIT	A54E
\$ 02	LOAD	A412
\$ 04	SAVE	A397
\$ 06	RUN	A4D1
:	:	:
:	:	:

\$ 34	BRUN	A38E
\$ 36	VERIFY	A27D

命令名表采用ASCII码格式,而在每一命令的最后一字的最高位规定为1(指2进制数),以此作为该命令的结束标志。例如,INIT的ASCII代码为49, 4E, 49, 54,而在命令名表中对应地址\$A884—\$A887中则为49, 4E, 49, D4。其中最后一个字节T的ASCII码是54,但作为INIT命令的结束标志则在\$54上再加\$80,结果为\$D4。

DOS命令名表存放在\$A884至\$A907的内存区域,依次存放着从INIT、LOAD、SAVE、RUN、……、BRUN、VERIFY等28个DOS命令每一个字符的ASICC码。每一命令的最后一字是在对应字符的ASICC码上加上\$80,以示结束。这样的存贮方式,我们不妨称之为DOS命令样板表。

DOS命令样板表

\$ A880	49 4E 49 D4	INIT
\$ A888	4C 4F 41 C4 53 41 56 C5	LOADSAVE
\$ A890	52 55 CE 43 48 41 49 CE	RUNCHAIN
\$ A898	44 45 4C 45 54 C5 4C 4F	DELETELO
\$ A8A0	43 CB 55 4E 4C 4F 43 CB	CKUNLOCK
\$ A8A8	43 4C 4F 53 C5 52 45 41	CLOSEREAL
\$ A8B0	C4 45 58 45 C3 57 52 49	DEXECWRI
\$ A8B8	54 C5 50 4F 53 49 54 49	TEPOSITI
\$ A8C0	4F CE 4F 50 45 CE 41 50	ONOPENAP
\$ A8C8	50 45 4E C4 52 45 4E 41	PENDRENA
\$ A8D0	4D C5 43 41 54 41 4C 4F	MECATALO
\$ A8D8	C7 4D 4F CE 4E 4F 4D 4F	GMONNOMO
\$ A8E0	CE 50 52 A3 49 4E A3 4D	NPR#IN#M
\$ A8E8	41 58 46 49 4C 45 D3 46	AXFILESF

```

$A8F0 D0 49 4E D4 42 53 41 56    P I N T B S A V
$A8F8 C5 42 4C 4F 41 C4 42 52    E B L O A D B R
$A900 55 CE 56 45 52 49 46 D9    U N V E R I F Y

```

在DOS系统中，命令的判别是通过与DOS命令样板匹配进行的。例如，键入 CATALOG 命令，系统就将 CATALOG与命令样板逐一比较，直到有与CATALOG相匹配的样板，就转入样板处所指定的CATALOG命令的处理子程序。因此，如果改变系统中的DOS命令样板，也就改变了DOS命令的形式；换言之，要实现DOS命令的简化，只要修改样板的存贮形式。

如果我们想修改某一命令的名称，则只要把新命令名代替旧命令名，只是要注意新命令名的长度不要超过旧命令名的长度，然后调整命令名表，把代替过程中所留下的空位删除掉（若新、旧命令名长度一致则不必）。

例如，CATALOG命令名太长，使用起来实在不方便，我们可以用较短的名字代替，如改为CAT。由DOS命令样板表中查出CATALOG命令名是从\$A8D2开始，到\$A8D8结束，而现在只要保留CAT三个字符的ASCII码，即43,41,54,但由于“T”字符作为简化命令的结束字符，故应在\$54上加上\$80应为\$D4，同时还应删去后面的4个字符，操作如下，

* A8D4: D4 ✓

* A8D5<A8D9,A908 M ✓

如果对几个常用的DOS命令简化，可采用这样的方法，简化的DOS命令为原来DOS命令的第一或第二个字符后跟一个“.”，即：

原DOS命令	简化后命令
INIT	I.

LOAD	L.
DELETE	D.
CATALOG	C.
RUN	R.
SAVE	S.
RENAME	RN.
UNLOCK	UL.
LOCK	LC.
PR#	P.
BLOAD	BL.
BRUN	BR.
BSAVE	BS.

修改步骤如下:

- ① 引导DOS3.3系统盘
- ② 进入监控: CALL-151↵
- ③ 键入程序10.15的机器码
- ④ 返回BASIC: CTRL-RESET↵
- ⑤ 用I.HELLO↵初始化一张盘片

这样,经初始化的盘片,就是一张简化DOS命令的副盘,以后用副盘启动系统,即可用简化DOS命令操作机器,并可实现对磁盘文件的加密保护。

如程序10.15所示。

程序10.15:

```

A884- 49 AE 4C AE
A888- 53 AE 52 AE 43 48 41 49
A890- CE 44 AE 4C 43 AE 55 4C
A898- AE 43 4C 4F 53 C5 52 45
A8A0- 41 C4 45 58 45 C3 57 52

```

```

A8A8- 49 54 C5 50 4F 53 49 54
A8B0- 49 4F CE 4F 50 45 CE 41
A8B8- 50 50 45 4E C4 52 4E AE
A8C0- 43 AE 4D 4F CE 4E 4F 4D
A8C8- 4F CE 50 AE 49 4E A3 4D
A8D0- 41 58 46 49 4C 45 D3 46
A8D8- D0 49 4E D4 42 53 AE 42
A8E0- 4C AE 42 52 AE 56 45 52
A8E8- 49 46 D9 00

```

9. 一行列印16个字节

在中华学习机监控状态下，查看和打印内存单元中的内容时，一行列出8个内存单元的数据，在屏幕上显示既整齐美观又便于阅读，但在使用打印机打印清单时，打印纸浪费太多。

一行列印8个字节的内容，是原来机器功能设定的，为了节省纸张，可以稍加改变机器功能，编一个程序，用软件控制打印输出，使一行显示（或打印）16个内存单元字节的内容。见程序10.16。

在介绍程序之前，我们先说明一下程序中将要用到的8个零页地址单元：

\$06—\$09：这四个单元可以由用户自由选用，例如通常将它们设置成地址指针。

\$3C—\$3F：这四个单元作为读写磁带信息首地址指针；也可安排操作数地址缓冲区(A1)；还可以装置特址表地址(RWTS)。例如有两个地址A1和A2，常用\$3C存放A1低字节地址；\$3D存放A1高字节地址；\$3E存放A2低字节地址；

\$3F存放A2高字节地址。

本程序中把起始地址放入\$06, \$07单元, 把结束地址放入\$08, \$09, 单元, 都是低位在前, 高位在后。而把\$3C—\$3F作为缓冲区, 暂存一些信息, 以便随时调用。

程序设计思路有这样几点:

① 为了一行显示16个字节的内容, 不是在一行显示8个字节内容后换行, 而是在一行显示16个字节内容后换行。

② 程序应能显示打印出地址、破折号“——”、三个空格以及相应地址的16个内存信息, 并能自动换行, 顺序不能颠倒。

③ 为了屏幕显示10行, 每行显示16个字节, 共160个字节的需要, 应合理安排判断, 以保证显示一幕信息的内存内容不超过160个字节。

④ 为使程序简洁, 其中调用了监控中四个子程序: \$FDED——输出一个字符; \$FDDA——输出16进制字符; \$FCBA——对首、末地址进入写操作; \$FD92——显示地址、破折号、空出三个空格、显示存贮单元内容。

程序清单见10.16。

程序10.16:

8000-	A5	06	LDA	\$06
8002-	85	3C	STA	\$3C
8004-	A5	07	LDA	\$07
8006-	85	3D	STA	\$3D
8008-	A9	0F	LDA	# \$0F
800A-	85	3E	STA	\$3E
800C-	A5	3D	LDA	\$3D
800E-	85	3F	STA	\$3F

8010-	A5	3C	LDA	\$3C
8012-	29	0F	AND	#\$0F
8014-	D0	03	BNE	\$8019
8016-	20	92 FD	JSR	\$FD92
8019-	A9	A0	LDA	#\$A0
801B-	20	ED FD	JSR	\$FDED
801E-	B1	3C	LDA	(\$3C),Y
8020-	20	DA FD	JSR	\$FDDA
8023-	20	BA FC	JSR	\$FCBA
8026-	E6	06	INC	\$06
8028-	D0	02	BNE	\$802C
802A-	E6	07	INC	\$07
802C-	A5	09	LDA	\$09
802E-	C5	07	CMP	\$07
8030-	90	0B	BCC	\$803D
8032-	F0	03	BEQ	\$8037
8034-	4C	08 80	IMP	\$8008
8037-	A5	08	LDA	\$08
8039-	C5	06	CMP	\$06
803B-	E0	CB	BCS	\$8008
803D-	60		RTS	

程序说明:

0300: 取始址低位→A

0302: 存始址低位→\$3C

0304: 取始址高位→A

0306: 存始址高位→\$3D

0308: 取立即数15→A

030A: 暂存15个字节数→\$3E

030C: 调回始址高位

030E: 改存到 \$3F
 0310: 调回始址低位
 0312: 始址低位值与0F相“与”
 0314: 结果不为0, 转
 0316: 结果为0, 转 \$FD92
 0319: $A0 = (160)_{10} \rightarrow A$
 031B: 输出一个字符子程序
 031E: 始址 $\rightarrow A$
 0320: 调输出16进制字符子程序
 0323: 显示
 0326: 始址低位加1
 0328: 不是0转
 032A: 始址高位加1
 032C: 取末址高位 $\rightarrow A$
 032E: 末、首址高位比较
 0330: $C = 0$ 转, 不够减转
 0332: $Z = 1$ (结果为0) 转
 0334: 返至 \$0308循环
 0337: 末址低位 $\rightarrow A$
 0339: 末址高位与始址低位比较
 033B: $C = 1$ 转, 有借位
 033D: 结束

本程序操作方法:

- ① 将要显示或打印的机器语言程序调进内存。
- ② 在监控下将打印的程序起始地址和结束地址分存在 \$06—\$09单元。
- ③ 接通打印机。
- ④ JCALL 768 / 或 * 300G /。

实例: 例如有一段机器语言程序(10.17)存放在 \$8000-

\$80E9中，今欲将其按每行按16个字节打印出来，操作步骤：

- ①]BLOAD_文件名，A\$300↵ 调打印控制程序
进内存
- ②]BLOAD_文件名，A\$8000↵ 调被打印程序进
内存
- ③]CALL-151↵ 进监控
- ④ *6:00 80 E9 80↵ 键入被打印程序首末地址
- ⑤ *300G↵ 按每行16字节打印输出

程序10.17:

8000-	A9 EA	LDA	#\$EA
8002-	85 E8	STA	\$E8
8004-	A9 60	LDA	#\$60
8006-	85 E9	STA	\$E9
8008-	20 E2 F3	JSR	\$F3E2
800B-	A9 14	LDA	#\$14
800D-	85 E7	STA	\$E7
800F-	A9 8C	LDA	#\$8C
8011-	85 01	STA	\$01
8013-	A9 60	LDA	#\$60
8015-	85 02	STA	\$02
8017-	A9 01	LDA	#\$01
8019-	85 00	STA	\$00
801B-	AD 00 C0	LDA	\$C000
801E-	A0 00	LDY	#\$00
8020-	8C 10 C0	STY	\$C010
8023-	C9 9B	CMP	#\$9B
8025-	D0 01	BNE	\$8028
8027-	60	RTS	

8028-	C9	CA	CMP	## \$ CA
802A-	D0	02	BNE	\$ 802E
802C-	C6	01	DEC	\$ 01
802E-	C9	CB	CMP	## \$ CB
8030-	D0	02	BNE	\$ 8034
8032-	E6	01	INC	\$ 01
8034-	C9	C8	CMP	## \$ C8
8036-	D0	03	BNE	\$ 803B
8038-	4C	08 60	JMP	\$ 6008
803B-	C9	C9	CMP	## \$ C9
803D-	D0	0A	BNE	\$ 8049
803F-	C6	02	DEC	\$ 02
8041-	A5	02	LDA	\$ 02
8043-	B0	04	BCS	\$ 8049
8045-	A9	BE	LDA	## \$ BE
8047-	85	02	STA	\$ 02
8049-	C9	CD	CMP	## \$ CD
804B-	D0	0C	BNE	\$ 8059
804D-	E6	02	INC	\$ 02
804F-	A5	02	LDA	\$ 02
8051-	C9	BE	CMP	## \$ BE
8053-	90	04	BCC	\$ 8059
8055-	A9	00	LDA	## \$ 00
8057-	85	02	STA	\$ 02
8059-	C9	88	CMP	## \$ 88
805B-	D0	0C	BNE	\$ 8069
805D-	C6	E7	DEC	\$ E7
805F-	A5	E7	LDA	\$ E7
8061-	C9	01	CMP	## \$ 01

8063-	B0	04		BCS	\$ 8069
8065-	A9	01		LDA	# \$ 01
8067-	85	E7		STA	\$ E7
8069-	C9	95		CMP	# \$ 95
806B-	D0	02		BNE	\$ 806F
806D-	E6	E7		INC	\$ E7
806F-	C9	CC		CMP	# \$ CC
8071-	D0	04		BNE	\$ 8077
8073-	A9	00		LDA	# \$ 00
8075-	85	03		STA	\$ 03
8077-	C9	D0		CMP	# \$ D0
8079-	D0	04		BNE	\$ 807F
807B-	A9	01		LDA	# \$ 01
807D-	85	03		STA	\$ 03
807F-	C9	CF		CMP	# \$ CF
8081-	D0	04		BNE	\$ 8087
8083-	A9	02		LDA	# \$ 02
8085-	85	03		STA	\$ 03
8087-	C9	B1		CMP	# \$ B1
8089-	D0	02		BNE	\$ 808D
808B-	C6	04		DEC	\$ 04
808D-	C9	B2		CMP	# \$ B2
808F-	D0	02		BNE	\$ 8093
8091-	E6	04		INC	\$ 04
8093-	A2	03		LDX	# \$ 03
8095-	20	F0	F6	JSR	\$ F6F0
8098-	A5	03		LDA	\$ 03
809A-	D0	0F		BNE	\$ 80AB
809C-	20	D0	60	JSR	\$ 60D0
809F-	20	5D	F6	JSR	\$ F65D

80A2-	20	D0	60	JSR	\$ 60D0
80A5-	20	5D	F6	JSR	\$ F65D
80A8-	4C	C2	60	JMP	\$ 60C2
80AB-	20	D0	60	JSR	\$ 60D0
80AE-	20	01	F6	JSR	\$ F601
80B1-	A5	03		LDA	\$ 03
80B3-	C9	01		CMP	# \$ 01
80B5-	F0	0B		BEQ	\$ 80C2
80B7-	A2	04		LDX	# \$ 04
80B9-	20	F0	F6	JSR	\$ F6F0
80BC-	20	D0	60	JSR	\$ 60D0
80BF-	20	01	F6	JSR	\$ F601
80C2-	E6	00		INC	\$ 00
80C4-	A5	00		LDA	\$ 00
80C6-	C9	40		CMP	# \$ 40
80C8-	B0	03		BCS	\$ 80CD
80CA-	4C	1B	60	JMP	\$ 601B
80CD-	4C	17	60	JMP	\$ 6017
80D0-	A5	04		LDA	\$ 04
80D2-	20	A8	FC	JSR	\$ FCA8
80D5-	A2	01		LDX	# \$ 01
80D7-	20	30	F7	JSR	\$ F730
80DA-	A5	02		LDA	\$ 02
80DC-	A6	01		LDX	\$ 01
80DE-	A0	00		LDY	# \$ 00
80E0-	20	11	F4	JSR	\$ F411
80E3-	A5	00		LDA	\$ 00
80E5-	A6	1A		LDX	\$ 1A
80E7-	A4	1B		LDY	\$ 1B
80E9-	60			RTS	

8000-	A9	EA	85	E8	A9	60	85	E9	20	E2	F3	A9	14	85	E7	A9
8010-	8C	85	01	A9	60	85	02	A9	01	85	00	AD	00	C0	A0	00
8020-	8C	10	C0	C9	9B	D0	01	60	C9	CA	D0	02	C6	01	C9	CB
8030-	D0	02	E6	01	C9	C8	D0	03	4C	08	60	C9	C9	D0	0A	C6
8040-	02	A5	02	B0	04	A9	BE	85	02	C9	CD	D0	0C	E6	02	A5
8050-	02	C9	BE	90	04	A9	00	85	02	C9	88	D0	0C	C6	E7	A5
8060-	E7	C9	01	B0	04	A9	01	85	E7	C9	95	D0	02	E6	E7	C9
8070-	CC	D0	04	A9	00	85	03	C9	D0	D0	04	A9	01	85	03	C9
8080-	CF	D0	04	A9	02	85	03	C9	B1	D0	02	C6	04	C9	B2	D0
8090-	02	E6	04	A2	03	20	F0	F6	A5	03	D0	0F	20	D0	60	20
80A0-	5D	F6	20	D0	60	20	5D	F6	4C	C2	60	20	D0	60	20	01
80B0-	F6	A5	03	C9	01	F0	0B	A2	04	20	F0	F6	20	D0	60	20
80C0-	01	F6	E6	00	A5	00	C9	40	B0	03	4C	1B	60	4C	17	60
80D0-	A5	04	20	A8	FC	A2	01	20	30	F7	A5	02	A6	01	A0	00
80E0-	20	11	F4	A5	00	A6	1A	A4	1B	60						

细心的读者一定看得出来，从\$8000到\$803D的内容和\$300到\$33D的内容几乎相差甚微，实际上是将\$300—\$33D的内容搬至\$8000—\$803D中，而改变了几个转移地址，故\$8000—\$803D的程序，同样能控制一行16个字节的输出，方法是在监控下直接执行8000G即可。

10. 一行显示任意个字节

前节曾经介绍一行列印16个字节的程序，它可以将机器原来每行显示8个单元内容，扩展成每行显示16个字节。事实上，仿照上节程序，可以在一行内显示任意多个字节（以不超过屏幕极限，并应清晰整齐），下面提供的程序，其原理和一些子程序作用和一行列印16个字节程序十分相似，不再重复。

这里主要说明的还是零页地址的使用。在运行程序前，应先将显示的开始和结束地址分别送入\$06至\$09单元，而把\$3C—\$3F作为缓冲区，实际上也是处理\$06—\$09单元的信息。

把每一行要显示的单元个数，用16进制数表示，送入\$031C单元中，这是本程序的关键所在，若\$031C单元中预置的数是\$08，则一行显示8个单元字节；若\$031C单元中预置的数是\$10，则一行显示16个单元字节。接通打印机，键入*300G则可将显示内容列印在打印纸上。

程序清单如10.18所示：

程序10.18：

0300-	A9 00	LDA	# \$ 00
0302-	85 FD	STA	\$ FD
0304-	A5 09	LDA	\$ 09

0306-	85	3F	STA	\$ 3F
0308-	A5	08	LDA	\$ 08
030A-	85	3E	STA	\$ 3E
030C-	A5	07	LDA	\$ 07
030E-	85	3D	STA	\$ 3D
0310-	A5	06	LDA	\$ 06
0312-	85	3C	STA	\$ 3C
0314-	4C	25 03	JMP	\$ 0325
0317-	E6	FD	INC	\$ FD
0319-	A5	FD	LDA	\$ FD
031B-	C9	10	CMP	# \$ 10
031D-	D0	09	BNE	\$ 0328
031F-	A9	00	LDA	# \$ 00
0321-	85	FD	STA	\$ FD
0323-	A5	3C	LDA	\$ 3C
0325-	20	92 FD	JSR	\$ FD92
0328-	A9	A0	LDA	# \$ A0
032A-	20	ED FD	JSR	\$ FDED
032D-	B1	3C	LDA	(\$ 3C),Y
032F-	20	DA FD	JSR	\$ FDDA
0332-	20	BA FC	JSR	\$ FCBA
0335-	90	E0	BCC	\$ 0317
0337-	60		RTS	

十一、绘图原理和方法

众所周知，中华学习机在APPLESOFT支持下，可以用GR, PLOT, HLIN, VLIN, COLOR 等命令绘制低分辨率图形；也可以用HGR, HGR2, HCOLOR 和 HPLOT 等命令进行高分辨作图。但是在机器语言状态下如何实现图形绘制，一般资料和书籍均未介绍，本章就绘图原理和方法作些阐明，目的是通过这个领域的应用，进一步熟悉机器语言的使用。

实际上，中华学习机是一种比较好的绘图系统，而6502机器语言又是一种比较好的绘图语言，无论在图形效果，占用存贮空间还是在绘图速度上都比高级语言强得多。同时，在任何一个BASIC程序中，巧用一些机器语言相配合，可以实现BASIC语言无法实现的功能。可以这样说，不了解机器语言绘图技术，要想设计出更加完美的游戏程序，动画程序，或计算机辅助教学程序，将十分困难。

任何一幅图形，都可以说是由点和线（水平线，垂直线）构成的，而6502机器语言的监控程序中为我们提供了许多绘点画线的子程序，因此我们可以设想，一般的绘图程序无非是调用这些子程序，画出水平线、垂直线及点。各种不同的图形无非是调用前的坐标不同，调用的次数不同。若在调用这些子程序之前，将坐标写入指定的寄存器，则我们可以设计出形式多样、变化万千的图形来。

本章仅就图形程序的编写进行一般的讨论，更详细的研

究请参阅《中华学习机图形管理》、《中华学习机图形技巧》两书介绍。

1. 低分辨率绘图形程序

中华学习机监控程序中可调用的低分辨率绘图形程序见表11.1。

表11.1

入口地址	功能说明	调用前需预置的寄存器
\$FB40	设置低分辨率图形和文本(四行)混合方式, 清屏为黑色	没有
\$F832	使整个图形屏幕为黑色	没有
\$F836	使彩色图形屏幕上部为黑色, 保留四行文本	没有
\$F864	置低分辨率图形彩色	彩色数字送累加器A
\$FC58	清屏幕(文本状态), 光标在左上角	没有
\$F800	在低分辨率图形第1页上画一个图形点	横坐标送Y寄存器, 纵坐标送A累加器, 颜色代码放\$30
\$F819	画一条低分辨率水平线	纵坐标送累加器A, 横坐标(起始值)送Y寄存器, 横坐标(终了值)送\$2C存贮单元
\$F828	画一条低分辨率垂直线	横坐标送寄存器Y, 纵坐标(起始值)送累加器A, 纵坐标(结束值)送\$2D存贮单元

低分辨率图形共有16种颜色, 相应的代码如表11.2所

示:

表11.2

0	黑色	8	棕色
1	洋红	9	桔黄
2	深蓝	10	灰色
3	紫色	11	粉红
4	深绿	12	浅绿
5	深灰	13	黄色
6	蓝色	14	海蓝
7	浅蓝	15	白色

根据上述表,我们可以设计出简单的机器语言绘制低分辨率图形。

例1:在低分辨率屏幕上画一彩色对角线,见程序11.1。

程序11.1:

```
0300- 20 58 FC      JSR    $FC58
0303- 20 40 FB      JSR    $FB40
0306- A9 09          LDA    # $09
0308- 20 64 F8      JSR    $F864
030B- A0 00          LDY    # $00
030D- 98            TYA
030E- 20 00 F8      JSR    $F800
0311- C8            INY
0312- C0 28          CPY    # $28
0314- D0 F7          BNE    $030D
0316- 60            RTS
```

程序说明:

0300: 清全屏文本

0303: 置混合方式

0306: 取颜色代码

0308: 置颜色为橙色

030B: 设置横坐标

030D: 设置纵坐标

030E: 画一个点

0311: 横坐标加1

0312: 画到右下角吗?

0314: 未完, 再画

例2: 在低分辨率屏幕上作一水平线, 见程序11.2。

程序11.2:

1000-	20	40	FB	JSR	\$FB40
1003-	A9	10		LDA	# \$10
1005-	48			PHA	
1006-	A9	02		LDA	# \$02
1008-	98			TYA	
1009-	68			PLA	
100A-	A2	20		LDX	# \$20
100C-	86	2C		STX	\$2C
100E-	A2	05		LDX	# \$05
1010-	86	30		STX	\$30
1012-	20	19	F8	JSR	\$F819
1015-	60			RTS	

程序说明:

1000: 置混合方式

1003: 置纵坐标

1005: 进栈

1006: 置横坐标起始值, $A \rightarrow y$

1009: 出栈 $\rightarrow A$

100A: 置横坐标结束值, $X \rightarrow \$2C$

100E: 取颜色代码 $\rightarrow \$30$

2. 高分辨率绘图子程序

中华学习机监控程序中可调用的高分辨率绘图子程序见表11.3。

表11.3

入口地址	功能说明	调用前需预置的寄存器
\$F3E2	启动高分辨率第1作图页(设置HGR方式), 清除屏幕	无
\$F3D8	启动高分辨率第2作图页(设置HGR2方式), 清除屏幕	无
\$F6F0	设置高分辨率作图的颜色(设置HCOLOR方式)	颜色序号(0—7)送入X寄存器, 颜色代码放入\$E4存贮单元
\$F457	在坐标(X0,Y0)处画一个点(同HPILOT X0,Y0)	横坐标X0高位在Y寄存器, 低位在X寄存器, 纵坐标Y0在累加器A中
\$F715	从当前坐标画一条到坐标(X1,X1)的直线(同HPILOT TO X1,Y1)	横坐标、纵坐标的设置同\$F457子程序
\$F53A	画线子程序(同HPILOT X0,Y0 TO X1,Y1)	横坐标低位在累加器A中, 高位在X寄存器中, 纵坐标在Y寄存器中

注: X0的范围: 0—279 (\$00—\$117), Y0的范围为: 0—191 (\$00—\$BF)。高分辨画点颜色设置必须通过调用\$F6F0开始的子程序来实现, 调用前将颜色序号(0—7)送入X寄存器。颜色序号和代码的对应关系为: 0—00, 1—2A, 2—52, 3—7F, 4—80, 5—AA, 6—D5, 7—FF, 颜色代码放入\$E4单元。颜色代码首地址为\$F6F6。

高分辨图形共有 8 种颜色，相应的序号如表 11.4 所示：

表11.4

0	黑 1	4	黑 2
1	绿 (决定于TV)	5	红 (决定于TV)
2	蓝 (决定于TV)	6	紫 (决定于TV)
3	白 1	7	白 2

和绘制低分辨图形一样，只要调用表11.3和表11.4的信息，可以方便地进行高分辨率图形的机器语言绘制。

例 1：在 (X0, Y0) — (X1, Y1) 画一直线，见程序11.3。

程序11.3:

0300-	A6	4A	LDX	\$4A
0302-	20	F0 F6	JSR	\$E6F0
0305-	A6	4B	LDX	\$4B
0307-	A4	4C	LDY	\$4C
0309-	A5	4D	LDA	\$4D
030B-	20	57 F4	JSR	\$F457
030E-	A6	CD	LDX	\$CD
0310-	A4	CE	LDY	\$CE
0312-	A5	CF	LDA	\$CF
0314-	20	15 F7	JSR	\$F715
0317-	60		RTS	

程序说明：

300：取颜色代码→X

302：置颜色

305：取X0的低位

307：取X0的高位

309：取Y0

30B: 调画点子程序

30E: 取X1的低位

310: 取X1的高位

312: 取Y1

314: 调画线子程序

317: 返回

注意本程序运行前必须将 (X0,Y0) , (X1,Y1) 坐标分别置于\$4B—\$4D和\$CD—\$CF中, 也可放入用户自己选用的零页地址\$06—\$08和\$18—\$1A中, 颜色代码置入\$09中。

例2: 从屏幕左上角到右下角画一线段, 见程序11.4。

程序11.4:

1000-	20	DB	F3	JSR	\$F3D8
1003-	A2	04		LDX	# \$ 04
1005-	20	F0	F6	JSR	\$F6F0
1008-	A0	00		LDY	# \$ 00
100A-	A2	00		LDX	# \$ 00
100C-	A9	00		LDA	# \$ 00
100E-	20	57	F4	JSR	\$F457
1011-	A0	BF		LDY	# \$ BF
1013-	A2	01		LDX	# \$ 01
1015-	A9	17		LDA	# \$ 17
1017-	20	3A	F5	JSR	\$F53A
101A-	60			RTS	

程序说明:

1000: 启动第2 高分辨率作图页

1003: 取得颜色代码

1005: 设置颜色

1008: 设置X0高位初值
 100A: 设置X0低位初值
 100C: 设置Y0初值
 100E: 调画点 (X0, Y0) 子程序
 1011: 设置Y1终值
 1013: 设置X1高位终值
 1015: 设置X2低位终值
 1017: 调用画线子程序
 101A: 返回

3. 图形清屏

例如, 将高分辨第 1 页图形页面清除为黑色。这个问题虽可用图形语句HGR即可方便实现, 但我们希望编制一个机器语言程序来处理。

表面上看, 这个问题难于下手, 但实际上它是一个典型的数据块传送实例, 因为, 将高分辨图形页面第 1 页清洗干净, 就是清屏, 因此, 只要将内存\$2000—\$3FFF的 8191个存储单元, 全部置为 0 即可 (当屏幕上点的相应位为 0 时, 屏上是一个黑点)。所以, 问题最终归结为编制一个程序, 将 0 送至 \$2000—\$3FFF 的每一个内存单元, 故有程序 11.5。

程序11.5:

0300-	A2	00	LDX	# \$ 00
0302-	A9	00	LDA	# \$ 00
0304-	9D	00 20	STA	\$ 2000,X
0307-	E8		INX	
0308-	D0	FA	BNE	\$ 0304
030A-	EE	06 03	INC	\$ 0306

030D-	AD 06 03	LDA	\$0306
0310-	C9 40	CMP	# \$40
0312-	D0 EE	BNE	\$0302
0314-	60	RTS	

程序11.5简述如下：初始化时，变址寄存器X先置0，累加器A也置0，执行STA \$2000，X指令，第一次将0送入\$2000单元中，执行INX指令，使 $X+1 \rightarrow X$ ，执行BNE \$0304指令，只要 $Z=0$ （即\$2000以后的单元中不为0），都循环上去（转至\$0304）不断送0，X变址不断增1，一直到X中的值变到\$FF之前，都送0，而当X再增1至\$2100时， $Z=1$ （指令执行完毕其值全为0）时，执行INC指令，使内存单元加1，因此时转向\$0306，故\$0306单元中的值由20变为21（即\$2100），执行LDA \$0306指令，就从\$2100开始再送0，用CMP # \$40判断，只要不到\$4000（即\$3FFF之前，包括\$3FFF）都转向\$0302，不断送0，而超过\$3FFF后，结束。

从\$2000到\$3FFF共8191个存贮单元，也就是说本题要送8191个零，数据量较大，但由于机器语言执行速度快，这种传送8191个零的过程瞬间即告完成。

为了检验上述机器语言程序清屏的功能，可用下述步骤：

- ①]HGR↵
- ②]BLOAD PIC A\$2000↵
- ③]CALL 768↵

其中PIC为调入内存的高分辨图形，当图形显示完毕后，执行CALL 768↵，屏幕即为黑色。

4. 图形反相显示

在图形处理中,为使图形画面显示更为生动,可以设法使图形反相显示。例如,将高分辨第1页图形进行反相显示处理。

处理此类问题首先应弄清两个基本概念,什么是图形反相显示,怎样才能对图形信息求反。

图形信息中包含了大量的“0”和“1”,是“1”的显示一个亮点,是“0”的显示一个黑点,因此,要使图形能够反相显示(亮点变黑点,黑点变亮点),应设法使“1”变成“0”,使“0”变成“1”。

对任意一个8位2进制数来说,要使它的每一位取反(1变成0,0变成1),通常有两种方法。其一是用\$FF(11111111)去减一个8位2进制数,所得差值即为该8位2进制的反;其二是用\$FF与一个8位2进制数异或(相同出0,不同出1),所得异或结果即为该8位2进制数的反。

因此,这两种不同的算法,就是编写程序的两个不同设计思想,而最终又归结为根据上述思想编写的数据块传送的程序。

(1) 选用异或处理方法的程序见11.6。

程序11.6:

```
0300-   A2  00           LDX    # $00
0303-   BD  00   20     LDA    $2000,X
0305-   49  FF           EOR    # $FF
0307-   9D  00   20     STA    $2000,X
030A-   E8              INX
```

030B-	D0	F5	BNE	\$ 0302
030D-	EE	04 03	INC	\$ 0304
0310-	EE	09 03	INC	\$ 0309
0313-	AD	09 03	LDA	\$ 0309
0316-	C9	40	CMP	# \$ 40
0318-	D0	E8	BNE	\$ 0302
031A-	60		RTS	

程序11.6结构简单清楚, 首先置X变址为 0, 接着取\$2000中的信息(第一次为\$2000), 与\$FF异或(EOR #\$FF), 并把结果再放入\$2000中(仍对第一次而言), 然后X+1→X, 执行BNE \$0302指令, Z=0转移, Z=1继续执行下一条指令, 这样某一个循环执行完后, 就把\$2000—\$20FF单元内的值求反了, 然后\$20FF地址的高位加1, (20→21, 注意执行两条INC指令), 并取到累加器, 通过CPM # \$40比较, 只要不是\$40, (因为第一页的末地址高位是\$3F), 再循环上去, 如此往复, 最后将\$2000—\$3FFF的全部单元信息取反。

检验图形求反的方法是:

] HGR↙

] BLOAD 图形文件名, A\$2000↙ (看到的图形为正常图形)

] CALL 768↙ (看到源图形已经反相了)

(2) 采用差值处理方法的程序见11.7。

程序11.7:

0300-	A9	00	LDA	# \$ 00
0302-	85	00	STA	\$ 00
0304-	A9	40	LDA	# \$ 40
0306-	85	01	STA	\$ 01

0308-	A0	00	LDY	# \$ 00
030A-	A9	FF	LDA	# \$ FF
030C-	F1	00	SBC	(\$ 00), Y
030E-	91	00	STA	(\$ 00), Y
0310-	C8		INY	
0311-	D0	F7	BNE	\$ 030A
0313-	E6	01	INC	\$ 01
0315-	A5	01	LDA	\$ 01
0317-	C9	60	CMP	# \$ 60
0319-	90	EF	BCC	\$ 030A
031B-	60		RTS	

程序说明:

0300 }
 : } (01) (00) = 4000
 0306 }

0308, 00 → Y

030A, FF → A

030C }
 030E } 求反

0310: Y + 1 → Y

0311: 只要不是全 0 (Z = 0) 循环

0313: 是 0, 地址高位加 1

0315: 01 → A

0317: 是 \$ 60 吗? 第二页完地址高位是 5F

0319: 不是 60, 再循环上去 (C = 0)

031B: C = 1, 结束。

程序 11.7 是将高分辨图形第 2 页求反。若对第一页高分辨率图形求反。则将 \$0305 改 20, \$0318 改 40。

5. 图形叠加

图形叠加是指高分辨第1页的图形合并到高分辨第2页图形上去；或者相反，将第2页的图形合并到第1页上去。图形叠加的方法很多，这里主要介绍用逻辑运算指令处理图形叠加的方法和技巧。

例如，将高分辨第2页图形，通过“与”、“异或”、“或”处理搬入第1页，从而实现两页图形的叠加。

“与”指令AND，其功能是将累加器A中的内容与指定存储器M单元中的内容按位相“与”，结果送累加器，存储器中内容不变，即 $A \wedge M \rightarrow A$ 。AND指令共有八种寻址方式，若采用后变址(Y)间接寻址时，其指令格式为AND(\$NN),Y,代码为31 NN。按位相“与”即全1出1，有0出0。

“或”指令ORA，其功能是将累加器A中的内容与指定存储器单元中的内容按位“或”，结果送累加器，不改变存储单元内容，即 $A \vee N \rightarrow A$ 。ORA指令共有八种寻址方式，如后变址(Y)间接寻址的指令是ORA(\$NN),Y,代码为11NN。按位“或”即有1出1，全0出0。

“异或”指令EOR，功能是将累加器中的内容与指定存储单元中的内容按位“异或”，结果送累加器，存储器中内容不变，即 $A \oplus M \rightarrow A$ 。EOR指定也有8种寻址方式，采用后变址(Y)间接寻址的指令是EOR(\$NN),Y,代码为51 NN。按位“异或”即相同出0，不同出1。

若在下面程序11.8中，\$031C单元的操作码为\$11时，则进行“或”显示；置\$31时为“与”显示；置\$51时，为“异或”显示。

程序11.8:

0300-	A9	00	LDA	#\$00
0302-	85	42	STA	\$42
0304-	A9	20	LDA	#\$20
0306-	85	43	STA	\$43
0308-	A9	00	LDA	#\$00
030A-	85	3C	STA	\$3C
030C-	A9	40	LDA	#\$40
030E-	85	3D	STA	\$3D
0310-	A9	FF	LDA	#\$FF
0312-	85	3E	STA	\$3E
0314-	A9	5F	LDA	#\$5F
0316-	85	3F	STA	\$3F
0318-	A0	00	LDY	#\$00
031A-	B1	3C	LDA	(\$3C),Y
031C-	11	42	ORA	(\$42),Y
031E-	91	42	STA	(\$42),Y
0320-	20	B4 FC	JSR	\$FCB4
0323-	90	F5	BCC	\$031A
0325-	60		RTS	

程序说明:

0300: 00→A
 0302: (42) = 00
 0304: 20→A
 0306: (43) = 20
 0308: 00→A
 030A: (3C) = 00
 030C: (40) →A
 030E: (3D) = 40

0310: FF→A
 0312: (3E) = FF
 0314: 5F→A
 0316: (3E) = 5F
 0318: 00→Y暂存器
 031A: 调4000+Y的值→A
 031C: 或
 031E: 或后的结果→A
 0320: 调地址增1 指针子程序
 0323: 未增完转
 0325: 结束

上述程序中巧用了搬家程序和地址指针加1 程序。注意程序中一定要设置Y = 0。

6. 图形动画显示

在用计算机进行图形处理和显示时，我们经常希望显示器上出现动画图形。利用中华学习机高分辨率图形具有两个存贮器映象区的特点，我们可以利用汇编语言编写一段动画图形显示程序，通过该程序的执行，我们可将三幅存贮于微机内存中的图形按所希望的速度进行连续动画显示，形成一个很有趣的CARTOON。

下面这段名为CATON的程序可将三幅事先存于内存贮器中的高分辨图形进行间隔约1 秒钟的连续动画显示。程序中的LOOP循环为主程序，通过调用显示开关、数据块移动、延时等子程序，完成整个动画显示控制任务。程序中P1为高分辨第一页显示开关子程序；P2为高分辨第二页显示开关子程序；MOVE1、MOVE2为数据块移动子程序；WAST1、WAST2、WAST3为延时子程序，通过改变延时子程序中 X

寄存器的值，可达到不同的画面延时效果。

使用前，先将CATON程序进行汇编，形成机器码文件，并将预动画显示的三幅图顺序存放于\$4000—\$5FFF，\$2000—\$3FFF，\$6000—\$7FFF区域内，然后在系统状态下运行汇编后的CATON程序就可以出现连续的动画图形了。

1000-	20	82	10	JSR	\$ 1082	
1003-	20	02	11	JSR	\$ 1102	
1006-	20	75	10	JSR	\$ 1075	(LOOP)
1009-	20	02	11	JSR	\$ 1102	
100C-	20	8F	10	JSR	\$ 108F	
100F-	20	82	10	JSR	\$ 1082	
1012-	20	0D	11	JSR	\$ 110D	
1015-	20	C3	10	JSR	\$ 10C3	
1018-	20	75	10	JSR	\$ 1075	
101B-	20	02	11	JSR	\$ 1102	
101E-	20	8F	10	JSR	\$ 108F	
1021-	20	82	10	JSR	\$ 1082	
1024-	20	02	11	JSR	\$ 1102	
1027-	20	C3	10	JSR	\$ 10C3	
102A-	20	75	10	JSR	\$ 1075	
102D-	20	0D	11	JSR	\$ 110D	
1030-	20	8F	10	JSR	\$ 108F	
1033-	20	82	10	JSR	\$ 1082	
1036-	20	F7	10	JSR	\$ 10F7	
1039-	20	C3	10	JSR	\$ 10C3	
103C-	20	75	10	JSR	\$ 1075	
103F-	20	F7	10	JSR	\$ 10F7	
1042-	20	8F	10	JSR	\$ 108F	
1045-	20	82	10	JSR	\$ 1082	

1048-	20	0D	11	JSR	\$ 110D	
104B-	20	C3	10	JSR	\$ 10C3	
104E-	20	75	10	JSR	\$ 1075	
1051-	20	F7	10	JSR	\$ 10F7	
1054-	20	8F	10	JSR	\$ 108F	
1057-	20	82	10	JSR	\$ 1082	
105A-	20	F7	10	JSR	\$ 10F7	
105D-	20	C3	10	JSR	\$ 10C3	
1060-	20	75	10	JSR	\$ 1075	
1063-	20	0D	11	JSR	\$ 110D	
1066-	20	8F	10	JSR	\$ 108F	
1069-	20	82	10	JSR	\$ 1082	
106C-	20	02	11	JSR	\$ 1102	
106F-	20	C3	10	JSR	\$ 10C3	
1072-	4C	06	10	JMP	\$ 1006	
1075-	AD	54	C0	LDA	\$ C054	(P ₁)
1078-	AD	57	C0	LDA	\$ C057	
107B-	AD	52	C0	LDA	\$ C052	
107E-	AD	50	C0	LDA	\$ C050	
1081-	60			RTS		
1082-	AD	57	C0	LDA	\$ C057	(P ₂)
1085-	AD	55	C0	LDA	\$ C055	
1088-	AD	52	C0	LDA	\$ C052	
108B-	AD	50	C0	LDA	\$ C050	
108E-	60			RTS		
108F-	A9	00		LDA	# \$ 00	(MOVE ₁)
1091-	85	50		STA	\$ 50	
1093-	85	60		STA	\$ 60	
1095-	85	70		STA	\$ 70	

1097-	85	80	STA	\$80	
1099-	A9	60	LDA	#\$60	
109B-	85	61	STA	\$61	
109D-	A9	20	LDA	#\$20	
109F-	85	51	STA	\$51	
10A1-	A9	40	LDA	#\$40	
10A3-	85	71	STA	\$71	
10A5-	A9	80	LDA	#\$80	
10A7-	85	81	STA	\$81	
10A9-	A0	00	LDY	#\$00	
10AB-	B1	70	LDA	(\$70),Y	
10AD-	91	80	STA	(\$80),Y	
10AF-	B1	60	LDA	(\$60),Y	
10B1-	91	70	STA	(\$70),Y	
10B3-	C8		INY		
10B4-	D0	F5	BNE	\$10AB	
10B6-	E6	61	INC	\$61	
10B8-	E6	71	INC	\$71	
10BA-	E6	81	INC	\$81	
10BC-	A5	71	LDA	\$71	
10BE-	C9	60	CMP	#\$60	
10C0-	90	E7	BCC	\$10A9	
10C2-	60		RTS		
10C3-	A9	00	LDA	#\$00	(MOVE ₂)
10C5-	85	50	STA	\$50	
10C7-	85	60	STA	\$60	
10C9-	85	70	STA	\$70	
10CB-	85	80	STA	\$80	
10CD-	A9	60	LDA	#\$60	

10CF-	85	61		STA	\$ 61
10D1-	A9	20		LDA	# \$ 20
10D3-	85	51		STA	\$ 51
10D5-	A9	40		LDA	# \$ 40
10D7-	85	71		STA	\$ 71
10D9-	A9	80		LDA	# \$ 80
10DB-	85	81		STA	\$ 81
10DD-	A4	81		LDY	\$ 81
10DF-	B1	50		LDA	(\$ 50),Y
10E1-	91	60		STA	(\$ 60),Y
10E3-	B1	80		LDA	(\$ 80),Y
10E5-	91	50		STA	(\$ 50),Y
10E7-	C8			INY	
10E8-	D0	F5		BNE	\$ 10DF
10EA-	E6	51		INC	\$ 51
10EC-	E6	61		INC	\$ 61
10EE-	E6	81		INC	\$ 81
10F0-	A5	51		LDA	\$ 51
10F2-	C9	40		CMP	# \$ 40
10F4-	90	E7		BCC	\$ 10DD
10F6-	60			RTS	
10F7-	A2	10		LDX	# \$ 10 (WAST ₁)
10F9-	A9	FF		LDA	# \$ FF
10FB-	20	A8	FC	JSR	\$ FCA8
10FE-	CA			DEX	
10FF-	D0	F8		BNE	\$ 10F9
1101-	60			RTS	
1102-	A2	02		LDX	# \$ 02 (WAST ₂)
1104-	A9	FF		LDA	# \$ FF

1106-	20	A8	FC	JSR	\$FCAB	
1109-	CA			DEX		
110A-	D0	F8		BNE	\$1104	
110C-	60			RTS		
110D-	A2	50		LDX	#\$50	(WAST ₃)
110F-	A9	FF		LDA	#\$FF	
1111-	20	A8	FC	JSR	\$FCA8	
1114-	CA			DEX		
1115-	D0	F8		BNE	\$110F	
1117-	60			RTS		

7. 汇编语言调用造型表

在辅助教学和游戏软件设计中，经常采用造型表方法，绘制一些图形，然后用BASIC程序调用。实际上，编制一段汇编语言来调用造型表，不仅在速度上要比BASIC语言调用造型表快好几倍，而且用在动画设计上，其动态效果突出且无闪烁感。

编制汇编语言调用造型表，并不十分麻烦，只需了解一些造型表的存贮结构，调用和造型有关的机器语言入口地址及相应子程序，同时正确存贮造型表。

我们先给出调用造型表的机器语言子程序(\$300-\$32B，见程序11.9)，并给出对应的汇编语言程序(见程序11.10)，然后逐段加以解释，从中可以了解到程序编制的方法。

程序11.9:

0300-	20	E2	F3	A9	00	85	E8	A9
0308-	60	85	E9	A2	03	20	F0	F6
0310-	A9	03	85	E7	A2	01	20	30
0318-	F7	A2	80	A0	00	A9	32	20

```

0320- 11 F4 A6 1A A4 1B A9 00
0238- 20 5D F6 60
031D- A9 32          LDA    # $32

```

程序11.10

```

0300- 20 E2 F3      JSR     $F3E2
0303- A9 00          LDA    # $00
0305- 85 E8          STA    $E8
0307- A9 60          LDA    # $60
0309- 85 E9          STA    $E9
030B- A2 03          LDX    # $03
030D- 20 F0 F6      JSR     $F6F0
0310- A9 03          LDA    # $03
0312- 85 E7          STA    $E7
0314- A2 01          LDX    # $01
0316- 20 30 F7      JSR     $F730
0319- A2 80          LDX    # $80
031B- A0 00          LDY    # $00
031D- A9 32          LDA    # $32
031F- 20 11 F4      JSR     $F411
0322- A6 1A          LDX    $1A
0324- A4 1B          LDY    $1B
0326- A9 00          LDA    # $00
0328- 20 5D F6      JSR     $F65D
032B- 60             RTS

```

(1) 高分辨图形显示页面的设置 同APPLESOFT一样,为了调用并显示造型,首先应该设置高分辨图形显示页面。例如,高分辨第一页作图页(相当于HGR)的入口地址为\$F3E2,高分辨第二页作图页(相当于HGR2)的入口地址为\$F3D8。用汇编语言调用,其指令分别是;

20 E2 F3 JSR \$F3E2; 启动第一页

20 D8 F3 JSR \$F3D8; 启动第二页

本例是启动高分辨第一页作图页面, 见\$300—\$302。

同BASIC一样, 若要清除当前屏幕画面 (用HOME指令), 机器语言则应这样安排:

20 58 FC JSR \$FC58

其中\$FC58为HOME指令的入口地址, 本例没有安排。

(2) 造型表的地址 在BASIC程序中, \$E8(232)和\$E9(233)两个单元是专门用来存放造型表首地址的, 并规定造型表首地址的低字节 (本例是\$00) 放在\$E8单元中; 造型表首地址的高字节 (本例是\$60) 放在\$E9单元中。因此, 在BASIC程序中常用POKE指令设置, 即:

POKE 232, 0; POKE 233, 96

对应的汇编语言则为:

A9 00 LDA #\$00

85 E8 STA \$E8

A9 60 LDA #\$60

85 E9 STA \$E9

本例造型表地址为\$6000, 机器语言存贮在\$0303—\$030A单元中。只有这样安排, 造型表才能送入内存并付诸使用。

(3) 高分辨图形的颜色 在BASIC程序中, 通过HCOLOR指令设定高分辨图形的颜色, 例如HCOLOR = N (N取0—7)。但在汇编语言中颜色代码 (可取\$00—\$07中任一个) 应放在X暂存器中, 但必须通过程序名为HCOLOR的机器语言子程序来实现, 其入口地址为\$F6F0。所以, 在机器语言中高分辨图形的颜色应由下面两条指令来实现:

```
A2 03      LDX # $03
20 F0 F6 JSR $F6F0
```

本例颜色代码为03，两条指令的机器码放在\$030B—\$030F单元中。

(4) 绘图矢量的大小 在BASIC语言中，关于绘图矢量的大小是用指令SCALE设定的，例如SCALE=1，表示造型的放大倍数为1（即不放大）。用汇编语言处理该指令时，应将造型放大的值放在\$E7单元中，即

```
A9 03 LDA # $03
85 E7 STA $E7
```

本例造型放大值取03，见\$0310—\$0313。

(5) 造型显示位置的确定 造型显示位置是指造型显示的起点位置。在BASLC程序中，如执行DRAW 1 AT 140, 94，则表示将第一个造型绘于141列95行开始的地方（相当于HGR2状态时屏幕中央的位置）。在机器语言中，则要调用\$F411开始的一段机器语言子程序，而在调用该子程序之前，应将显示起始位置水平坐标的低位放入X暂存器中，水平坐标的高位放入Y暂存器中，至于造型显示起始位置的垂直坐标，则放在累加器A中。即有下面四条指令：

```
A2 80      LDX # $80
A0 00      LDY # $00
A9 32      LDA # $32
20 11 F4 JSR $F411
```

本例坐标选在X=128，Y=00这一点显示造型，4条指令存贮在\$0319—\$0321单元中。

(6) 调用造型指针的设定 调用造型的指针设定，是指调用第几个造型，这也可以用机器语言子程序来实现，其

入口地址在\$F730中,调用前应将造型序号(第几个造型)之值放入X暂存器中,即:

```
A2 01      LDX  # $01
20 30 F7    JSR  $F730
```

本例是调第1个造型,指针号为01。此值相当于BASIC语言中造型绘图命令DRAW(或XDRAW)后面紧跟的参数。上述两条指令存放在\$0314—\$0318单元中。

(7) 关于造型显示或清除 在BASIC语言中有两个指令DRAW和XDRAW。前者绘出造型,后者消除一个造型,实际上是用互补颜色再绘一次造型而达到抵消的目的。在机器语言中这两条指令的入口地址不同,前者是\$F601,后者是\$F65D。在调用这两个子程序前,应将旋转值(对应BASIC命令ROT=N)放在累加器A中,至于造型向量绝对地址的低位、高位值分别放在X、Y暂存器中(其值为\$1A, \$1B),即:

```
A6 1A      LDX  $1A
A4 1B      LDY  $1B
A9 00      LDA  # $00
20 5D F6    JSR  $F65D
```

本例旋转值为\$00,即不旋转,以上四条指令的机器码见\$0322—\$032A。

(8) 返回 返回指令RTS的机器码为60,它放在\$032B单元中。

至此,整个机器语言调用造型表的设计原理和方法叙述完成。

最后,我们给出一个造型数据(见程序11.11),它放在\$6000—\$6072单元中。在执行程序11.9的机器语言之前,

应将程序 11.11 先调入内存，然后在 BASIC 状态下执行 CALL 768↵，或在监控状态下执行 300G↵即可看到一辆汽车在屏幕上闪动。

程序 11.11:

```

6000- 01 00 05 00 00 41 38 20
6008- 25 2C 2D 65 0C 0C 0C 2D
6010- 2D 2D 2D 2D 15 15 AD 2D
6018- 2D 15 2D 2D 2E 2E 3E 3E
6020- BF 37 3F 3C 1C 24 1C 3F
6028- 17 17 36 3F 3F 3F F7 27
6030- 3F E4 1C 3F 17 3E 3F 2D
6038- 25 6D 15 F6 3F 07 20 64
6040- 2D 04 20 05 28 28 2D 2D
6048- 2D 35 35 35 3F 3F 3F 24
6050- 34 36 3F 3F 3F 4A 49 92
6058- 3A 2D 4E 49 49 20 64 2D
6060- B6 1E 7F 49 09 2C 25 3C
6068- 2C 2D 2D 3C 3F 3F EF 3F
6070- 3F 27 2D

```

8. 图形绘制工具软件

本节介绍一个用机器语言编制的图形绘制工具软件。本软件采用一条长度可以调节的线段作为绘图工具，它象一支神奇的笔，在屏幕上画出千姿百态的图案来。这支笔既可以画点，也可以画线，还可以作图；既可以上、下、左、右移动，也可以旋转；既可以绘图着色，也可以换色清除；既可以加快或减慢绘图速度，也可以放大或缩小图形。因此别具一格，令人耳目一新。

这支笔实际上是一个点的造型，为了便于操作者使用方便，定义了以下13个功能键，其作用分述如下：

- ① P：旋转并绘色
- ② L：旋转
- ③ O：旋转并清除
- ④ ESC：退出运行
- ⑤ M：下移
- ⑥ I：上移
- ⑦ J：左移
- ⑧ K：右移
- ⑨ H：清屏
- ⑩ →：缩小
- ⑪ ←：放大
- ⑫ 1：加快
- ⑬ 2：减慢

软件全部用汇编语言编写，为阅读方便，我们对软件的各项功能，语句安排，指令作用，指针设置作了详细的注释和说明。

整个程序设计分三大部分：

- ① \$6000—\$60E9：是软件的主体
- ② \$60EA—\$60EF：造型表
- ③ \$60F0—\$610C：控制部分

现在，我们按标号顺序逐一叙述。

(1) 造型表起始地址放\$E8，\$E9单元。\$E8—\$E9是零页地址的两个专用单元，用来存放造型表开始指针。由于本软件造型表放在\$60EA开始的单元，故有：

6000-	A9	EA	LDA	# \$EA
6002-	85	E8	STA	\$E8

```

6004-   A9  60      LDA   # $60
6006-   85  E9      STA   $E9

```

(2) 启动高分辨作图页。中华学习机高分辨率第1作图页地址为\$F3E2, 高分辨率第2页作图页地址为\$F3D8, 因此启动高分辨率作图页可以这样安排:

```

20 E2 F3   JSR $F3E2: 启动第1 高分辨作图页
20 D8 F3   JSR $F3D8: 启动第2 高分辨作图页

```

若用软件开关控制则有:

```

60F3-   8D  50 C0      STA   $C050
60F6-   8D  52 C0      STA   $C052
60F9-   8D  55 C0      STA   $C055
60FC-   8D  57 C0      STA   $C057

```

程序说明:

60F3: 图形状态

60F6: 全屏幕方式

60F9: 第2 页面

60FC: 高分辨率

以上四条指令相当于执行BASIC中的HGR2命令。

```

8D  50 C0   STA $C050: 图形状态
8D  53 C0   STA $C053: 混合方式
8D  54 C0   STA $C054: 第1 页面
8D  57 C0   STA $C057: 高分辨率

```

以上四条指令相当于执行BASIC中的HGR命令。

本软件采用启动高分辨第1页作为作图区, 即6008—20E2 F3 JSR \$F3E2。为了显示中文菜单(即前面定义的13个功能键), 安排\$60F3—\$60FE语句。这是因为中华学习机汉字显示方式在高分辨第2页。

(3) 造型参数指针设置。

① 造型放大倍数值：造型放大倍数(SCALE)值放在零页地址专用单元\$E7中，如造型放大倍数值为\$14，则：

```
600B-   A9   14   LDA   # $14
600D-   85   E7   STA   $E7
```

② 造型显示起始值：为了显示造型，必须调用\$F411开始的机器语言子程序，而在调用该子程序之前，应将显示起始位置水平坐标低位放入X暂存器中，水平坐标高位放入Y暂存器中，垂直坐标放入累加器A中，一般有：

```
A2 80      LDX # $80: 水平坐标低址
A0 00      LDY # $00: 水平坐标高址
A9 32      LDA # $32: 垂直坐标
20 11 F4 JSR $F411: 扫描子程序
```

上述四条指令即在 $X=128$ ， $Y=50$ 处显示造型。

本软件则在二处分头存贮和调用。

一处安排在\$600F—\$6016中，即：

```
600F-   A9   8C   LDA   # $8C
6011-   85   01   STA   $01
6013-   A9   60   LDA   # $60
6015-   85   02   STA   $02
```

程序说明：

600F：水平坐标140→A

6011：暂存\$01单元

6013：垂直坐标96→A

6015：暂存\$02单元

这里水平及垂直坐标分别用\$01和\$02单元暂存，目的是为了以后控制上、下、左、右移动时直接从\$01和\$02单元调用。

另一处安排在\$60DA—\$60E2中，即：

60DA-	A5	02	LDA	\$ 02
60DC-	A6	01	LDX	\$ 01
60DE-	A0	00	LDY	# \$ 00
60E0-	20	11 F4	JSR	\$ F411

程序说明:

60DA: 调垂直坐标→A

60DC: 调水平坐标低址→x

60DE: 调水平坐标高址→y

60E0: 调扫描子程序

这样, 可以在指定位置显示造型。

③ 初始旋转角度。零页地址\$F9中, 用来存放ROT值, 即旋转角度值。本软件中没有设置\$F9单元, 而是选用了\$00单元, 这也是灵活的, 在软件中凡是调用旋转值时, 即可从\$00单元去取。

设初始旋转角度值为\$01, 则有:

6017-	A9	01	LDA	# \$ 01
6019-	85	00	STA	\$ 00

(4) 按键功能控制。本软件共有13个功能(如前述), 均用按键控制, 现分述如下:

① 读取按键并清除:

601B-	AD	00 C0	LDA	\$ C000
601E-	A0	00	LDY	# \$ 00
6020-	8C	10 C0	STY	\$ C010

程序说明:

601B: 读键盘

6020: 清键盘

② 若是ESC键则退出:

6023-	C9	9B	CMP	# \$ 9B
-------	----	----	-----	---------

```

6025-    D0    01    BNE    $ 6028
6027-    60          RTS

```

程序说明:

6023: 是ESC键吗?

6025: 否, 转, 再判其它键

6027: 退出

③ 造型上下左右移动和清屏: 造型上、下、左、右移动及清除屏幕, 可分别用M、I、J、K及H键控制, 其对应的ASCII的值分别为\$CD、\$C9、\$CA、\$CB及\$C8。由于造型显示的起始值已放在\$01(水平坐标值)和\$02(垂直坐标值)单元中, 因此, 只要改变\$01和\$02单元的数值, 即可控制造型在四个方向移动。而清屏即可以在判断是H键后转向HOME(\$FC58)子程序, 也可直接转向\$6008(20 E2 F3 JSR \$F3E2)。

```

6028-    C9    CA          CMP    # $CA
602A-    D0    02          BNE    $ 602E
602C-    C6    01          DEC    $ 01
602E-    C9    CB          CMP    # $CB
6030-    D0    02          BNE    $ 6034
6032-    E6    01          INC    $ 01
6034-    C9    C8          CMP    # $C8
6036-    D0    03          BNE    $ 603B
6038-    4C    08    60    JMP    $ 6008

```

程序说明:

6028: 是J键?

602A: 否, 转

602C: 是, (01) -1, 左移

602E: 是K键?

6030: 否, 转

6032: 是 (01) + 1, 右移

6034: 是H键?

6036: 否, 转

6038: 是, 清屏

从程序中看出是某个键, 则执行该键对应的功能操作, 不再判其它键。所用指令也比较简单, 但必须注意因为是左、右移位, 要改变\$01中的值。

至于用按键控制上、下移动, 我们加了防止按键过头超出屏幕的处理, 因而程序稍许复杂一些:

603B-	C9	C9	CMP	#\$C9
603D-	D0	0A	BNE	\$6049
603F-	C6	02	DEC	\$02
6041-	A5	02	LDA	\$02
6043-	B0	04	BCS	\$6049
6045-	A9	BE	LDA	#\$BE
6047-	85	02	STA	\$02
6049-	C9	CD	CMP	#\$CD
604B-	D0	0C	BNE	\$6059
604D-	E6	02	INC	\$02
604F-	A5	02	LDA	\$02
6051-	C9	BE	CMP	#\$BE
6053-	90	04	BCC	\$6059
6055-	A9	00	LDA	#\$00
6057-	85	02	STA	\$02

程序说明:

603B, 是I键?

603D, 否, 转

603F, 是, (02) - 1, 上移

6041, 调上移后的坐标值→A

6043: C = 1时转移
 6045: 调垂直坐标最大值→A
 6047: 回屏幕底部
 6049: 是M?
 604B: 否, 转
 604D: 是, (02) + 1, 下移
 604F: 调下移后的坐标→A
 6051: 是BE = (191)₁₀吗?
 6053: C = 0转移
 6055: 否, 调垂直坐标最小值→A
 6057: 回屏幕顶部

④ 造型放大和缩小: 由于造型放大倍数 值放在\$E7单元中, 因此改变\$E7单元的值就可以放大(\$E7 + 1)和缩小(\$E7 - 1), 我们用左剪头←和右剪头→分别控制, 这两个按键的ASCII码分别为\$95和\$88。故有程序:

6059-	C9	88	CMP	# \$88
605B-	D0	0C	BNE	\$6069
605D-	C6	E7	DEC	\$E7
605F-	A5	E7	LDA	\$E7
6061-	C9	01	CMP	# \$01
6063-	B0	04	BCS	\$6069
6065-	A9	01	LDA	# \$01
6067-	85	E7	STA	\$E7
6069-	C9	95	CMP	# \$95
606B-	D0	02	BNE	\$606F
606D-	E6	E7	INC	\$E7

程序说明:

6059: 是→吗?
 605B: 否, 转

605D: 是, (E7) - 1, 造型缩小

605F: 调缩小后的值→A

6061: 是最小值?

6063: C = 1转移

6065: 调最小值→A

6067: 保持最小值

6069: 是←吗?

606B: 否, 转

606D: 是, (E7) + 1, 造型放大

本段程序中对造型缩小可能达到极值进行了控制, 而对最大值表面上没有限制, 实际上可以按右剪头按钮→让其缩小。

⑤ 旋转、绘画和擦图:

通过在\$03单元中放置不同的值达到旋转、绘图和擦图的控制:

\$03:00, 则旋转, 用L键控制 (代码为\$CC)

\$03:01, 则旋转并绘图, 用P键 (代码\$D0)

\$03:02, 则旋转并清除, 用O键 (代码\$CF)

程序段为:

606F-	C9	CC	CMP	#\$CC
6071-	D0	04	BNE	\$6077
6073-	A9	00	LDA	#\$00
6075-	85	03	STA	\$03
6077-	C9	D0	CMP	#\$D0
6079-	D0	04	BNE	\$607F
607B-	A9	01	LDA	#\$01
607D-	85	03	STA	\$03
607F-	C9	CF	CMP	#\$CF
6081-	D0	04	BNE	\$6087

6083- A9 02 LDA # \$ 02

6085- 85 03 STA \$ 03

程序说明:

606F: 是L键?

6071: 否, 转

6073: 是, 取00

6075: (03) = 00

6077: 是P键?

6079: 否, 转

607B: 是, 取01

607D: (03) = 01

607F: 是O键?

6081: 否, 转

6083: 是, 取02

6085: (03) = 02

⑥ 速度控制: 通过数字键 1 和 2, 控制画笔的移动或旋转的速度:

按 1 键 (ASCII码为 \$ B1) , \$ 04单元减 1, 加速,
按 2 键 (ASCII码为 \$ B2) , \$ 04单元加 1, 减速:

6087- C9 B1 CMP # \$ B1

6089- D0 02 BNE \$ 608D

608B- C6 04 DEC \$ 04

608D- C9 B2 CMP # \$ B2

608F- D0 02 BNE \$ 6093

6091- E6 04 INC \$ 04

程序说明:

6087: 是 1 键?

6089: 否, 转

608B: 是, (04) - 1

608D: 是2键?

608F: 否, 转

6091: 是, (04) + 1

(5) 颜色设置。颜色设置必须调用\$F6F0单元开始的子程序, 并在调用前将颜色代码放入X暂存器中:

6093—A2 03 LDX # \$03; 取颜色代码

6095—20 F0 F6 JSR \$F6F0; 颜色设置

顺便提及的是, 零页地址\$30单元为低分辨率颜色暂存器, 高分辨率方式颜色代码也可放在\$30中。

(6) 造型绘制和清除。在BASIC语言中, 绘出造型用DRAW命令, 清除造型用XDRAW命令 (实际上是用互补颜色绘出造型而达到抵消的目的), 前者对应的机器语言入口地址是\$F601; 后者对应的机器语言入口地址是\$F65D。在调用这二个子程序之前应将旋转值放在累加器A中: 即

6098-	A5	03	LDA	\$03	
609A-	D0	0F	BNE	\$60AB	
609C-	20	D0	60	JSR	\$60D0
609F-	20	5D	F6	JRS	\$F65D
60A2-	20	D0	60	JSR	\$60D0
60A5-	20	5D	F6	JSR	\$F65D
60A8-	4C	C2	60	JMP	\$60C2
60AB-	20	D0	60	JSR	\$60D0
60AE-	20	01	F6	JSR	\$F601
60B1-	A5	03	LDA	\$03	
60B3-	C9	01	CMP	# \$01	
60B5-	F0	0B	BEQ	\$60C2	
60B7-	A2	04	LDX	# \$04	
60B9-	20	F0	F6	JSR	\$F6F0

60BC-	20	D0	60	ISR	\$ 60D0
60BF-	20	01	F6	ISR	\$ F601

程序说明:

6098: (03) → A

609A: (A) 不为零, 转

609C: (A) = 0, 转延迟

609F: XDRAW

60A2: 调延迟

60A5: XDRAW

60A8: 转加大旋转子程序

60AB: 调延迟

60AE: DRAW

60B1: (03) → A

60B3: 是01吗?

60B5: 是, 转

60B7: 否, 换颜色代码

60B9: 颜色放置

60BC: 调延时

60BF: 清除

(7) 加大旋转角

60C2-	E6	00	INC	\$ 00
60C4-	A5	00	LDA	\$ 00
60C6-	C9	40	CMP	井 \$ 40
60C8-	B0	03	BCS	\$ 60CD
60CA-	4C	1B	60	JMP \$ 601B
60CD-	4C	17	60	JMP \$ 6017

程序说明:

60C2: (00) + 1, 旋转值 + 1

60C4: (00) → A

60C6: 是\$40吗? (最大值)

60C8: 够减, 无借位, 转

60CA: 转读键盘子程序

60CD: 重置旋转角

(8) 延时

60D0—A5 04 LDA \$04: 置延时值

60D2—20 A8 FC JSR \$FCA8: 调延迟子程序

在调用延迟(或延时)子程序前, 应在累加器A中置延时值, 这里取\$04单元的值。

(9) 造型显示子程序

60D5- A2 01 LDX # \$01

60D7- 20 30 F7 JSR \$F730

60DA- A5 02 LDA \$02

60DC- A6 01 LDX \$01

60DE- A0 00 LDY # \$00

60E0- 20 11 F4 JSR \$F411

60E3- A5 00 LDA \$00

60E5- A6 1A LDX \$1A

60E7- A4 1B LDY \$1B

60E9- 60 RTS

程序说明:

60D5: 取一个造型→X

60D7: 调用造型子程序

60DA: 调垂直坐标→A

60DC: 调水平坐标低位→X

60DE: 调水平坐标高位→Y

60E0: 调扫描子程序

60E3: 调旋转值→A

60E5: 造型向量绝对地址低值

60E7: 造型向量绝对地址高值

60E9: 结束

上述造型显示子程序比较重要,造型个数必须放在X寄存器中,并配合使用\$F730开始的子程序,即这两条指令完成调用造型指针的安排。\$1A—\$1B是零页地址的两个存贮单元,在高分辨率清屏时,放图形首地址,清完下一行后,在\$1B中的数加1,直至完全清屏。用图形表作图时(本例属此种情况), \$1A—\$1B中被置入图形表首址。

(10) 造型表。从\$60FA到\$60EF放置了造型表:

\$60EA:01: 存放造型的数目,本例为1个造型

\$60EB:00: 按规定不用,用00填空

\$60EC:04 00: 第一个造型相对于造型表首的地址,低址在前,
高址在后

\$60EE:04: 造型,实际上是一个点

\$60EF:00: 造型结束标志

(11) 控制部分。最后介绍一下控制部分。

首先编制一个BASIC程序,利用中华学习机的汉字功能,把本软件全部说明性内容、操作方法,各定义键功能键用汉字在屏幕上显示出来,然后运行这个BASIC程序,再将画面的汉字作为图形存贮在盘序中,取名HZJH,即:

]BSAVE HZJH, A \$ 4000, L \$ 2000↵

实际使用时,再将汉字说明调进内存:

]BLOAD HZJH, A \$ 4000↵

控制用CTRL-Y进行,为此先定义:

]CALL-151↵

* 3F8:4C F0 60↵

最后在\$60F0—\$610D存入以下内容(为清楚起见,

指令后面均加了助记符和说明)：

60F0-	20	58	FC	ISR	\$FC58
60F3-	8D	50	C0	STA	\$C050
60F6-	8D	52	C0	STA	\$C052
60F9-	8D	55	C0	STA	\$C055
60FC-	8D	57	C0	STA	\$C057
60FF-	20	0C	FD	JSR	\$FD0C
6102-	C9	A0		CMP	#\$A0
6104-	D0	03		BNE	\$6109
6106-	4C	00	60	JMP	\$6000
6109-	C9	8D		CMP	#\$8D
610B-	D0	E3		BNE	\$60F0
610D-	60			RTS	

程序说明：

60F0：清屏

60F3：图形显示方式

60F6：全屏幕方式

60F9：第二页

60FC：高分辨率图形

60FF：读键盘

6102：是SPACE键？

6104：否，转

6106：是，进入绘图程序

6109：是RETURN键？

610B：否，转

610D：返回

当调进汉字说明菜单以后（此菜单比较简单，这里省去），按CTRL-Y/即显示中文菜单，按SPACE键，进入绘图状态，按RETURN键结束控制。

至此，用机器语言绘制图形工具软件的内容全部介绍完毕。

源程序清单见11.12:

6000-	A9	EA		LDA	#\$EA
6002-	85	E8		STA	\$E8
6004-	A9	60		LDA	#\$60
6006-	85	E9		STA	\$E9
6008-	20	E2	F3	ISR	\$F3E2
600B-	A9	14		LDA	#\$14
600D-	85	E7		STA	\$E7
600F-	A9	8C		LDA	#\$8C
6011-	85	01		STA	\$01
6013-	A9	60		LDA	#\$60
6015-	85	02		STA	\$02
6017-	A9	01		LDA	#\$01
6019-	85	00		STA	\$00
601B-	AD	00	C0	LDA	\$C000
601E-	A0	00		LDY	#\$00
6020-	8C	10	C0	STY	\$C010
6023-	C9	9B		CMP	#\$9B
6025-	D0	01		BNE	\$6028
6027-	60			RTS	
6028-	C9	CA		CMP	#\$CA
602A-	D0	02		BNE	\$602E
602C-	C6	01		DEC	\$01
602E-	C9	CB		CMP	#\$CB
6030-	D0	02		BNE	\$6034
6032-	E6	01		INC	\$01
6034-	C9	C8		CMP	#\$C8

6036-	D0	03		BNE	\$ 603B
6038-	4C	08	60	JMP	\$ 6008
603B-	C9	C9		CMP	\$ C9
603D-	D0	0A		BNE	\$ 6049
603F-	C6	02		DEC	\$ 02
6041-	A5	02		LDA	\$ 02
6043-	B0	04		BCS	\$ 6049
6045-	A9	BE		LDA	# \$ BE
6047-	85	02		STA	\$ 02
6049-	C9	CD		CMP	# \$ CD
604B-	D0	0C		BNE	\$ 6059
604D-	E6	02		INC	\$ 02
604F-	A5	02		LDA	\$ 02
6051-	C9	BE		CMP	# \$ BE
6053-	90	04		BCC	\$ 6059
6055-	A9	00		LDA	# \$ 00
6057-	85	02		STA	\$ 02
6059-	C9	88		CMP	# \$ 88
605B-	D0	0C		BNE	\$ 6069
605D-	C6	E7		DEC	\$ E7
605F-	A5	E7		LDA	\$ E7
6061-	C9	01		CMP	# \$ 01
6063-	B0	04		BCS	\$ 6069
6065-	A9	01		LDA	# \$ 01
6067-	85	E7		STA	\$ E7
6069-	C9	95		CMP	# \$ 95
606B-	D0	02		BNE	\$ 606F
606D-	E6	E7		INC	\$ E7
606F-	C9	CC		CMP	# \$ CC

6071-	D0	04		BNE	\$ 6077
6073-	A9	00		LDA	# \$ 00
6075-	85	03		STA	\$ 03
6077-	C9	D0		CMP	# \$ D0
6079-	D0	04		BNE	\$ 607F
607 B-	A9	01		LDA	# \$ 01
607 D-	85	03		STA	\$ 03
607 F-	C9	CF		CMP	# \$ CF
6081-	D0	04		BNE	\$ 6087
6083-	A9	02		LDA	# \$ 02
6085-	85	03		STA	\$ 03
6087-	C9	B1		CMP	# \$ B1
6089-	D0	02		BNE	\$ 608D
608 B-	C6	04		DEC	\$ 04
608 D-	C9	B2		CMP	# \$ B2
608 F-	D0	02		BNE	\$ 6093
6091-	E6	04		INC	\$ 04
6093-	A2	03		LDX	# \$ 03
6095-	20	F0	F6	JSR	\$ F6F0
6098-	A5	03		LDA	\$ 03
609 A-	D0	0F		BNE	\$ 60AB
609 C-	20	D0	60	JSR	\$ 60D0
609 F-	20	5D	F6	JSR	\$ F65D
60 A2-	20	D0	60	JSR	\$ 60D0
60 A5-	20	5D	F6	JSR	\$ F65D
60 A8-	4C	C2	60	JMP	\$ 60C2
60AB-	20	D0	60	JSR	\$ 60D0
60AE-	20	01	F6	JSR	\$ F601
60 B1-	A5	03		LDA	\$ 03

60B3-	C9	01		CMP	#\$01
60B5-	F0	0B		BEQ	\$60C2
60B7-	A2	04		LDX	#\$04
60B9-	20	F0	F6	JSR	\$F6F0
60BC-	20	D0	60	JSR	\$60D0
60BF-	20	01	F6	JSR	\$F601
60C2-	E6	00		INC	\$00
60C4-	A5	00		LDA	\$00
60C6-	C9	40		CMP	#\$40
60C8-	B0	03		BCS	\$60CD
60CA-	4C	1B	60	JMP	\$601B
60CD-	4C	17	60	JMP	\$6017
60D0-	A5	04		LDA	\$04
60D2-	20	A8	FC	JSR	\$FCA8
60D5-	A2	01		LDX	#\$01
60D7-	20	30	F7	JSR	\$F730
60DA-	A5	02		LDA	\$02
60DC-	A6	01		LDX	\$01
60DE-	A0	00		LDY	#\$00
60E0-	20	11	F4	JSR	\$F411
60E3-	A5	00		LDA	\$00
60E5-	A6	1A		LDX	\$1A
60E7-	A4	1B		LDY	\$1B
60E9-	60			RTS	
60EA-	01	00	04 00 04 00		
60F0-	20	58	FC	JSR	\$FC58
60F3-	8D	50	C0	STA	\$C050
60F6-	8D	52	C0	STA	\$C052
60F9-	8D	55	C0	STA	\$C055

60FC-	8D	57	C0	STA	\$C057
60FF-	20	0C	FD	JSR	\$FD0C
6102-	C9	A0		CMP	#\$A0
6104-	D0	03		BNE	\$6109
6106-	4C	00	60	JMP	\$6000
6109-	C9	8D		CMP	#\$8D
610B-	D0	E3		BNE	\$60F0
610D-	60			RTS	

十二、数学运算子程序

本章从最简单的加减法开始，较详细地介绍了加、减、乘、除四则运算子程序，并结合上机实习，用中华学习机单步指令观察运算结果，使初学者进一步理解机器语言的各种指令的使用方法及编程技巧。通过上机实习，逐步培养调试、找错和修改程序的技能。

本章最后一节还介绍了调用中华学习机CEC-BASIC运算子程序进行数学运算。通过调用举例，使读者熟悉这一方法，并能应用到自己编制的程序中去。

在本章所介绍的程序中，我们采用了部分0页地址单元来存放要运算的数据和运算结果，而将程序安排在第3页的单元中，这样做的目的是由于0页寻址指令的字节短，执行速度快。但需要注意的是0页单元绝大部分被系统占用（参见《中华学习机CEC-I参考手册》），用户可以使用的只有\$06—\$09，\$18—\$1F，\$EB—\$EF等几个单元。而读者则可以根据自己的需要来安排这些地址单元。

1. 加法程序

（1）8位2进制加法 我们将被加数存放在\$06单元中，加数存放在\$08单元中，运算结果存放在\$1A单元中，则程序P1如下：

0300-	D8	CLD
0301-	18	CLC

0302-	A5	06	LDA	\$ 06
0304-	65	08	ADC	\$ 08
0306-	85	1A	STA	\$ 1A
0308-	00		BRK	

程序说明:

0300: 2 进制运算, 清10进制运算标志位 D

0301: 清进位标志位 C

0302: 取被加数

0304: 与加数相加

0306: 结果送 \$ 1 A

0308: 中断, 暂停

由于ADC指令是带进位加指令, 因此在使用之前必须先清进位标志位, 即让 $C = 0$, 以免影响运算结果。另外, 6502指令中加法指令只有ADC一条, 无论是2 进制 运算还是10进制运算都用它。区别是2 进制运算还是10进制运算就是看10进制标志位D是0 还是1。CLD指令置D为0, 表示进行2 进制运算。

上机实习:

在监控状态下键入上面的程序, 并设 $(06) = 3B$, $(08) = 26$, 则结果 $(1A)$ 应为61。

* 300: D8┐18┐A5┐06┐65┐08┐85┐1A┐00 ↵

* 06: 3B↵

* 08: 26↵

单步运行:

* 300S↵

显示: 0300-D8 CLD

A = 00 X = 60 Y = 0D P = 30 S = DC

这里A累加器、X、Y寄存器、堆栈S的值都为随机数，

因为并未设定，而P状态寄存器为30，将其展开

N	V	B	D	I	Z	C	
0	0	1	1	0	0	0	可知D被置为0了。

继续运行：*S↵

显示：0301-18 C<C

A=00 X=60 Y=0D P=30 S=DC，将P展开

D				C				
0	0	1	1	0	0	0	0	可知C也被置为0了。

再继续运行：*S↵

显示：0302-A5 06 LDA \$06

A=3B X=60 Y=0D P=30 S=DC

被加数已被取到A累加器了。

继续运行：*S↵

显示：0304-65 08 ADC \$08

A=61 X=60 Y=0D P=30 S=DC

被加数与加数相加，结果留在A累加器中。

继续运行：*S↵

显示：0306-86 1A STA \$1A

A=61 X=60 Y=0D P=30 S=DC

键入*1A↵ 显示：001A-61

可见结果已放入\$1A单元中。

(2) 两位10进制数加法 由于一个单字节只能放置两位10进制数(BCD码)，因此单字节加法只能进行两位10进制数相加。由上面分析我们已经知道，要进行10进制运算，

只要将10进制标志位 D 置 1, 其它程序 完全一样即可。见 P2。

```

0300-   F8           SED
0301-   18           CLC
0302-   A5  06       LDA    $ 06
0304-   65  08       ADC    $ 08
0306-   85  1A       STA    $ 1A
0308-   00           BRK

```

程序说明:

0300: 10进制运算, 是 D = 1

0301: 清进位标志位 C

0302: 取被加数

0304: 与加数相加

0306: 结果送 \$ 1 A

0308: 中断、暂停

上机实习:

设 (06) = 25, (08) = 19, 则 (1A) 应为44。

键入 P₂, 并单步执行: * 300 S ↵

显示: 0300 - F8 SED

A = 98 X = 60 Y = 0D P = 38 S = DC 展开

D							
0	0	1	1	1	0	0	0

可知 D 被置 1。

继续运行: * S ↵

显示: 0301 18 CLC

A = 98 X = 60 Y = 0D P = 38 S = DC 展开 P

C							
0	0	1	1	1	0	0	0

可知 C 被置 0。

继续运行：* S ↙

显示：0302-A5 06 LDA \$06

A = 25 X = 60 Y = 0D P = 38 S = DC被加数被取入A中。

继续运行：* S ↙

显示：0304-65 08 ADC \$08

A = 44 X = 60 Y = 0D P = 38 S = DC被加数与加数相加的结果留在A中。

继续运行：* S ↙

显示：0306-85 1A STA \$1A

A = 44 X = 60 Y = 0D P = 38 S = DC键入 * 1A ↙
可见001A-44。结果已送入\$1A中。

(3) 16位2进制加法 16位2进制加法与8位2进制加法类似，需要注意的是16位2进制运算时应先进行低8位运算，然后再进行高8位运算，这样低8位如有进位产生，则会被加到高8位中去，保证了结果的准确。程序见P3。

设被加数低8位放在\$06单元，高8位放在\$07单元，加数低8位放在\$08单元，高8位放在\$09单元，结果低8位放在\$1A单元，高8位放在\$1B单元。

0300-	DB	CLD	
0301-	18	CLC	
0302-	A5 06	LDA	\$ 06
0304-	65 08	ADC	\$ 08
0306-	85 1A	STA	\$ 1A
0308-	A5 07	LDA	\$ 07
030A-	65 09	ADC	\$ 09
030C-	85 1B	STA	\$ 1B

程序说明:

300: 2 进制运算, 清10进制运算标志位 D

301: 清进位标志位 C

302: 取被加数低 8 位

304: 与加数低 8 位相加

306: 结果低 8 位送 \$ 1 A

308: 取被加数高 8 位

30 A: 与加数高 8 位相加

30 C: 结果高 8 位送 \$ 1 B

30 E: 中断, 暂停

上机实习:

设 (07) = 65, (06) = 2 C, (09) = 13, (08) = F 6, 则 (1B) 应为 79, (1A) 应为 22。

键入 P3 程序, 并单步运行: * 300SS ↵

显示: 0300-D8 CLD

A = 00 X = 60 Y = 0D P = 30 S = F8

0301-18 CLC

A = 00 X = 60 Y = 0D P = 30 S = F8 这两步同

上介绍的一样, D 是 0, C 是 0。

继续运行 * S S S ↵

显示: 0302-A5 06 LDA \$06

A = 2C X = 60 Y = 0D P = 30 S = F8

0304-65 08 ADC \$08

A = 22 X = 60 Y = 0D P = 31 S = F8

0306-85 1A STA \$1A

A = 22 X = 60 Y = 0D P = 31 S = F8

这里要注意的是低八位相加产生了进位，将P展开

C							
0	0	1	1	0	0	0	1

可见C为1。

继续进行：*SSS↙

显示：0308-A5 07 LDA \$07

A = 65 X = 60 Y = 0D P = 31 S = F8

030A-65 09 ADC \$09

A = 79 X = 60 Y = 0D P = 30 S = F8

030C-85 1B STA \$1B

A = 79 X = 60 Y = 0D P = 30 S = F8

键入*1A、1B↙ 显示：001A-22↙79可见结果正确。

这次实习，我们加快了运行速度，一次运行几步，这是因为我们已知道中间运算结果，在运行时我们注意核对一下中间结果是否正确就可以了。对较大的程序在调试时我们可人为地设置一些断点，以检验中间的结果是否正确，这样对整个程序的调试纠错很有帮助。

(4) 多字节2进制加法 P4是一个四字节2进制加法程序。设被加数和加数按字节的高→低顺序分别存放在\$06—\$09和\$1A—\$1D单元中，而结果按同样顺序放在\$EA—\$ED单元中。

0300-	A2	04	LDX	# \$04
0302-	D8		CLD	
0303-	18		CLC	
0304-	B5	05	LDA	\$05,X
0306-	75	19	ADC	\$19,X
0308-	95	E9	STA	\$E9,X

030A-	CA	DEX	
030B-	D0 F7	BNE	\$ 0304
030D-	00	BRK	

程序说明:

300: 字节数存X寄存器
 302: 清10进制运算标志D
 303: 清进位位C
 304: 取被加数
 306: 加加数
 308: 存运算结果
 30A: 计数一次
 30B: 判计算完否, 未完继续, 循环4次
 30D: 计算完, 暂停

上机实习:

设 (06) = 10, (07) = 85, (08) = 59, (09) = 90,
 (1A) = 26, (1B) = 87, (1C) = 11, (1D) = 18,
 则结果 (EA) 应为37, (EB) 应为0C, (EC) 应为6A,
 (ED) 应为A8。

键入P4, 键入被加数与加数并单步运行: * 300 S ↵

显示: 0300- A2 04 LDX # \$04

A = 00 X = 04 Y = 0D P = 30 S = 8 A

字节数被存入X, 其余均为随机数。

继续运行: * S ↵

显示: 0302- D8 CLD

A = 00 X = 04 Y = 0D P = 30 S = 8 A

D

展开 P

0	0	1	1	0	0	0	0
---	---	---	---	---	---	---	---

 可见D置0。

继续运行: * S ↙

显示: 0303- 18 CLC

A = 00 X = 04 Y = 0D P = 30 S = 8 A

展开 P

C							
0	0	1	1	0	0	0	0

 可见C置0。

继续运行: * S ↙

显示: 0304-B5 05 LDA \$05, X

A = 90 X = 04 Y = 0D P = B0 S = 8 A

取被加数最低字节至A。

继续运行: * S ↙

显示: 0306-75 19 ADC\$19, X

A = A8 X = 04 Y = 0D P = B0 S = 8 A

与加数最低字节相加留在A中。

继续运行: * S ↙

显示: 0308-95 E9 STA\$E9, X

A = A8 X = 04 Y = 0D P = B0 S = 8 A

键入 * ED ↙ 显示: 00ED-A8 为结果的最低字节数。

继续运行: * S ↙

显示: 030A-CA DEX

A = A8 X = 03 Y = 0D P = 30 S = 8 A 计数器减1。

表示已计算了一次。

继续运行: * S ↙

显示: 030B-D0 F7 BNE \$0304

A = A8 X = 03 Y = 0D P = 30 S = 8 A 断判X ≠

0, 展开可见

Z							
0	0	1	1	0	0	0	0

 Z = 0。

转\$304继续

继续运行: * S ✓

显示: 0304-B5 05 LDA \$05, X

A = 59 X = 03 Y = 0D P = 30 S = 8A

因为X为03, 取\$05 + \$03 = \$08单元内容至A。

继续运行: * S ✓

显示: 0306-75 19 ADC \$19, X

A = 6A X = 03 Y = 0D P = 30 S = 8A与\$19 + \$03
= \$1C单元内容相加留在A中。

继续运行: * S ✓

显示: 0308-95 E9 STA \$E9, X

A = 6A X = 03 Y = 0D P = 30 S = 8A

键入 * EC ✓ 显示00EC-6A

继续运行: * S ✓

显示: 030A-CA DEX

A = 6A X = 02 Y = 0D P = 30 S = 8A

计数器再减1, 表示计算了两次。

继续运行: * S ✓

显示: 030B-D0 F7 BNE\$0304

A = 6A X = 02 Y = 0D P = 30 S = 8A

断判 $X \neq 0$, 即 $Z = 0$, 转\$0304继续

继续运行: * S ✓

显示: 0304-B5 05 LDA \$05, X

A = 85 X = 02 Y = 0D P = 30 S = 8A

取\$05 + \$02 = \$07单元内容至A。

继续运行: * S ✓

显示: 0306-75 19 ADC \$19, X

A = 0C X = 02 Y = 0D P = 71 S = 8A 与 \$19 + \$02
= \$1B 单元内容相加留在A中，展开P，

							C
0	1	1	1	0	0	0	1

可看到有进位C = 1。

继续运行：* S ↙

显示：0308-95 E9 STA \$E9, X

A = 0C X = 02 Y = 0D P = 71 S = 8A

键入 * EB ↙ 显示 00EB-0C

继续运行：* S ↙

显示：030A-CA DEX

A = 0C X = 01 Y = 0D P = 71 S = 8A

计数器再减 1，表示已计算了三次。

继续运行：* S ↙

显示：030B-D0 F7 BNE \$0304

A = 0C X = 01 Y = 0D P = 71 S = 8A 判断 X ≠

0，即 Z = 0，转 \$0304 继续

继续运行：* S ↙

显示：0304-B5 05 LDA \$05, X

A = 10 X = 01 Y = 0D P = 71 S = 8A

取被加数最高位字节至A中。

继续运行：* S ↙

显示：0306-75 19 ADC \$19, X

A = 37 X = 01 Y = 0D P = 30 S = 8A

与加数最高字节相加，并加进位位留在A中

继续运行：* S ↙

显示：0308-95 E9 STA \$E9, X

A = 37 X = 01 Y = 0D P = 30 S = 8A

键入 * E A ↵ 显示 00 E A - 37

继续运行: * S ↵

显示: 030 A - C A D E X

A = 37 X = 00 Y = 0D P = 32 S = 8A

计数器减至 0, 表示 4 次已计算完。展开 P

Z							
0	0	1	1	0	0	1	0

 可见 Z = 1。

继续运行: * S ↵

显示: 030 B - D 0 F 7 B N E \$ 0304

A = 37 X = 00 Y = 0D P = 32 S = 8A

判断 X = 0, 即 Z = 1, 不转, 往下执行, 中断结束。

(5) 多字节10进制加法 多字节10进制加法 程序见 P5, 它同 P4 基本一样, 只是改了一条指令, 将 CLD 改为 SED。

0300-	A2	04	LDX	# \$ 04
0302-	F8		SED	
0303-	18		CLC	
0304-	B5	05	LDA	\$ 05, X
0306-	75	19	ADC	\$ 19, X
0308-	95	E9	STA	\$ E9, X
030A-	CA		DEX	
030B-	D0	F7	BNE	\$ 0304
030D-	00		BRK	

程序说明:

300: 字节数存 X

302: 置 10 进制运算标志

- 303: 清进位位
- 304: 取被加数
- 306: 加加数
- 308: 存运算结果
- 30 A: 计数一次
- 30 B: 判计算完否, 未完继续, 循环四次。
- 30 D 计算完, 中断。

为什么一定要改 CLD 指令为 SED 指令呢? 它们究竟有什么区别呢? 原来计算机中的 10 进制数多数是用 BCD 码来表示的, 它与 10 进制数及 2 进制数的关系如表 12.1。从表 12.1 可

表 12.1

10 进制数	2 进制数	BCD 码	
0	0000		0000
1	0001		0001
2	0010		0010
3	0011		0011
4	0100		0100
5	0101		0101
6	0110		0110
7	0111		0111
8	1000		1000
9	1001		1001
10	1010	0001	0000
11	1011	0001	0001
12	1100	0001	0010
13	1101	0001	0011
14	1110	0001	0100
15	1111	0001	0101

看出, BCD 码是用 4 位 2 进制数来表示 10 进制数的, 所以又称 BCD 码为 2—10 进制数。

当两个 10 进制数相加时, 例如:

$$\begin{array}{r} 0101\ 0011\ (53) \\ + 0011\ 0111\ (37) \\ \hline 1000\ 1010\ (8A) \end{array}$$

计算机按 2 进制加法相加，所得结果为 8 A，但这不是BCD码，所以结果不正确，这时的结果应再加 6 才对。即：

$$\begin{array}{r} 1000\ 1010 \\ + \quad\quad 0110 \\ \hline 1001\ 0000\ (90) \end{array}$$

$$\begin{array}{r} \text{又如 } 1001\ 1001\ (99) \\ + 0110\ 0111\ (67) \\ \hline \end{array}$$

10000 0000 (1,00)产生进位，结果也不正确，应再加66，即

$$\begin{array}{r} 1\ 0000\ 0000 \\ +\ 0110\ 0110 \\ \hline 1\ 0110\ 0110\ (1,66) \end{array}$$

由上面两个例子我们可看出，两个10进制数相加，当有进位产生时，必须对结果进行调整，如果后 4 位向前 4 位有进位时，结果必须加0110 (6)，当前 4 位有进位时，结果必须加0110 0000 (60)，当两个 4 位都有进位时，结果必须加0110 0110(66)。这就叫10进制调整。执行ADC指令进行10进制数相加时，CPU会自动调整结果，而进行 2 进制数相加，结果则不必调整。因此，在执行 ADC 指令进行10进制数相加时，必须用SED指令置标志D为 1。

如果对存入X寄存器的值修改一下，并有意修改被加数、加数和结果的存放地址，则可将P5扩大为处理 1—256 中任意个字节的相加运算。P5的上机实习我们就留给读者自己去做了。

2. 减 法 程 序

(1) 8 位 2 进制减法 同加法程序一样, 我们将被减数存放在\$06单元中, 减数存放在\$08单元中, 运算结果存放在\$1A单元中, 则 8 位 2 进制减法程序如P6。

0300-	D8	CLD
0301-	38	SEC
0302-	A5 06	LDA \$ 06
0304-	E5 08	SBC \$ 08
0306-	85 1A	STA \$ 1A
0308-	00	BRK

程序说明:

0300: 2 进制运算, 清10进制运算标志D。

0301: 进位标志C置1

0302: 取被减数

0304: 与减数相减

0306: 结果送\$1A

0308: 中断, 暂停

由于SBC指令是带借位减, 而6502CPU规定C表示借位, 当 $\overline{C} = 0$ (即 $C = 1$) 表示无借位。因此在减法运算开始前都要用一条指令SEC将标志位C置1。同加法程序一样, 减法指令也只有一条, 必须区别是2 进制运算还是10进制运算。

上机实习:

在监控状态下键入P6程序, 并设(06) = C4, (08) = 35, 则结果(1A) 应为8 F。

* 300: D8□38□A5□06□E5□08□85□1A□00✓

* 06: C4

* 08: 35

单步运行: * 300 S ✓

显示: 0300-D8 CLD

A = 98 X = 60 Y = 0D P = 30 S = 8A

将P状态寄存器展开

D							
0	0	1	1	0	0	0	0

可知D已被置0, 表示正在进行的是2进制运算。而其余各位及各寄存器的内容均为随机数。

继续运行: * S ✓

显示: 0301-38 SEC

A = 98 X = 60 Y = 0D P = 31 S = 8A

将P展开

C							
0	0	1	1	0	0	0	1

可知C被置

1表示无借位。

继续运行: * S ✓

显示: 0302-A5 06 LDA \$06

A = C4 X = 60 Y = 0D P = B1 S = 8A

被减数已被取到A累加器。

继续运行: * S ✓

显示: 0304-E5 08 SBC \$08

A = 8F X = 60 Y = 0D P = B1 S = 8A

被减数与减数相减, 结果留在A中。

继续运行: * S ✓

显示: 0306-85 1A STA \$1A

A = 8F X = 60 Y = 0D P = B1 S = 8A

键入: * 1A ✓ 显示001A-8 F

可见结果已放入\$1A中。

对比加法程序，可知上机实习减法程序的步骤与加法程序完全一样，因此对下面的几个减法程序，我们只给出程序，不再详列上机实习步骤了，请读者参照加法程序的上机步骤自行上机实习。

(2) 2位10进制数减法 程序见P7。

0300-	F8	SED	
0301-	38	SEC	
0302-	A5 06	LDA	\$ 06
0304-	E5 08	SBC	\$ 08
0306-	85 1A	STA	\$ 1A
0308-	00	BRK	

程序说明：

0300: 10进制运算，置D = 1
0301: 清借位 (C = 1无借位)
0302: 取被减数
0304: 与减数相减
0306: 结果送 \$ 1A
0308: 中断，暂停

(3) 16位2进制减法 程序见P8。被减数、减数及结果的存贮形式同加法。

0300-	D8	CLD	
0301-	38	SEC	
0302-	A5 06	LDA	\$ 06
0304-	E5 08	SBC	\$ 08
0306-	85 1A	STA	\$ 1A
0308-	A5 07	LDA	\$ 07
030A-	E5 09	SBC	\$ 09
030C-	85 1B	STA	\$ 1B

030E- 00

BRK

程序说明:

300: 2 进制运算, D = 0

301: 清借位 (C = 1 无借位)

302: 取被减数低 8 位

304: 与减数低 8 位相减

306: 结果低 8 位送 \$ 1A

308: 取被减数高 8 位

30A: 与减数高 8 位相减

30C: 结果高 8 位送 \$ 1B

30E: 中断, 暂停

(4) 多字节 2 进制减法 程序见 P9。被减数、减数及结果的存贮形式同加法。

0300- A2 04 LDX # \$ 04

0302- D8 CLD

0303- 38 SEC

0304- B5 05 LDA \$ 05,X

0306- F5 19 SBC \$ 19,X

0308- 95 E9 STA \$ E9,X

030A- CA DEX

030B- D0 F7 BNE \$ 0304

030D- 00 BRK

程序说明:

300: 字节数存X

302: 清10进制运算标志

303: 置进位标志

304: 取被减数 (由低向高)

306: 减减数

308: 存运算结果

30 A: 计数一次

30 B: 判计算完否, 未完继续, 循环 4 次

30 D: 计算完, 暂停。

(5) 多字节10进制减法 程序见P10。被减数、减数及结果的存贮形式同加法。

0300-	A2 04	LDX	# \$ 04
0302-	F8	SED	
0303-	38	SEC	
0304-	B5 05	LDA	\$ 05,X
0306-	F5 19	SBC	\$ 19,X
0308-	95 E9	STA	\$ E9,X
030A-	CA	DEX	
030B-	D0 F7	BNE	\$ 0304
030D-	00	BRK	

程序说明:

300: 字节数存X

302: 置10进制运算标志

303: 置进位位

304: 取被减数

306: 减减数

308: 存运算结果

30 A: 计数一次

30 B: 判计算完否, 未完继续, 循环 4 次

30 D: 计算完, 暂停

3. 乘法程序

在介绍乘法程序之前, 我们先介绍一下 2 进制乘法的运算方法:

2 进制的乘法与10进制乘法类似:

例如：被乘数 10111101

乘数 $\times 10011101$

```
      10111101
      00000000
      10111101
      10111101
      10111101
      00000000
      00000000
      10111101
      -----
```

乘积 111001111101001

相乘时可用乘数的每一位去乘被乘数，乘得的中间结果的最低有效位与相应的乘数位对齐，最后把这些中间结果同时相加。由于加法太复杂，计算机计算时就不这样做了，而是采用边乘边相加的方法，如下面例子所示。

```
      被乘数      10111101
      乘数       $\times 10011101$ 
      -----
                        10111101
                        + 00000000
第一次部分积      10111101
                        + 10111101
第二次部分积      1110110001
                        + 10111101
第三次部分积      100110011001
                        + 10111101
第四次部分积      1010101101001
                        + 10111101
乘积      111001111101001
```

每一次的中间结果，取决于乘数，若乘数的某一位为1，则

中间结果为被乘数，若这一位为 0，则中间结果为 0。上述的运算方法，在计算机中是用移位和相加的操作来实现的。我们可以用被乘数左移来实现，也可以用部分积右移来实现，还可以用部分积左移来实现。

这里，我们举例说明如下：

(1) 被乘数左移法

乘数	被乘数	部分积
10011101	10111101	00000000
① 乘数最低位为 1， 把被乘数加至部分 积上，然后把被乘数 左移 101111010		<u>+ 10111101</u> 10111101
② 乘数为 0，不加被 乘数，被乘数左移 1011110100	1011110100	
③ 乘数为 1，加已左 移后的被乘数 被 乘数左移 10111101000		<u>10111101</u> + 1011110100
④ 乘数为 1，加已左 移后的被乘数 被 乘数左移 101111010000		<u>1110110001</u> + 10111101000
⑤ 乘数为 1，加已左 移后的被乘数 被 乘数左移 1011110100000		<u>100110011001</u> + 101111010000
⑥ 乘数为 0，不加被 乘数 被乘数左移 10111101000000	10111101000000	<u>1010101101001</u>
⑦ 乘数为 0，不加被 乘数 被乘数左移 101111010000000	101111010000000	

1010101101001

⑧ 乘数为 1，加已左

+ 101111010000000

移后的被乘数得乘积

111001111101001

由上例可看出，两个 8 位 2 进制数相乘，乘积为 16 位，在运算过程中，这 16 位都有可能相加，故要用 16 位加法器。

(2) 部分积右移法

乘数	被乘数	部分积
10011101	10111101	00000000
① 乘数最低位为 1，加 数乘数部分积右移一位		+ 10111101 10111101 010111101
② 乘数为 0，不加被乘 数部分积右移一位		0010111101 + 10111101
③ 乘数为 1，加被乘数 部分积右移一位		1110110001 01110110001 + 10111101
④ 乘数为 1，加被乘数 部分积右移一位		100110011001 0100110011001 + 010111101
⑤ 乘数为 1，加被乘数 部分积右移一位		1010101101001 01010101101001
⑥ 乘数为 0，不加被乘 数部分积右移一位		01010101101001
⑦ 乘数为 0，不加被乘 数部分积右移一位		001010101101001
⑧ 乘数为 1，加被乘数 部分积右移一位		+ 10111101 111001111101001 0111001111101001

最后的结果相同。由于部分积每次移出去的数不参加运算，所

以参加相加操作的只有8位，用8位加法器就可以运算了。

(3) 部分积左移法

被乘数	乘数	部分积
10111101	10011101	00000000
① 乘数最高位为1,加被乘数	乘数	+ 10111101
数左移一位, 部分积左移一位		10111101
	100111010	101111010
② 乘数为0,不加被乘数	乘数	
左移一位, 部分积左移一位		
	1001110100	1011110100
③ 乘数为0,不加被乘数	乘数	
左移一位, 部分积左移一位		
	10011101000	10111101000
④ 乘数为1, 加被乘数	乘数	+ 10111101
左移一位, 部分积左移一位		11010100101
	100111010000	110101001010
⑤ 乘数为1,加被乘数	乘数	+ 10111101
左移一位, 部分积左移一位		111000000111
	1001110100000	1110000001110
⑥ 乘数为1,加被乘数	乘数左	+ 10111101
移一位, 部分积左移一位		1110011001011
	10011101000000	11100110010110
⑦ 乘数为0,不加被乘数	乘数	
左移一位, 部分积左移一位		
	100111010000000	111001100101100
⑧ 乘数为1, 加被乘数得	乘数	+ 10111101
乘积		111001111101001

按照这些方法，就可以编制程序了。

(1) 8位2进制整数乘法程序：

① 被乘数左移法：将被乘数放入\$06单元，并扩充至16位，高8位为\$07单元，乘数放入\$08单元，运算结果高8位放\$1A单元，低8位放\$1B单元。程序P11如下：

0300-	D8		CLD	
0301-	A9	00	LDA	# \$ 00
0303-	85	1A	STA	\$ 1A
0305-	85	1B	STA	\$ 1B
0307-	85	07	STA	\$ 07
0309-	A2	08	LDX	# \$ 08
030B-	46	08	LSR	\$ 08
030D-	90	0D	BCC	\$ 031 C
030F-	18		CLC	
0310-	A5	1B	LDA	\$ 1 B
0312-	65	06	ADC	\$ 06
0314-	85	1A	STA	\$ 1 B
0316-	A5	1B	LDA	\$ 1A
0318-	65	07	ADC	\$ 07
031A-	85	1A	STA	\$ 1 A
031C-	06	06	ASL	\$ 06
031E-	26	07	ROL	\$ 07
0320-	CA		DEX	
0321-	D0	E8	BNE	\$ 030 B
0323-	00		BRK	

程序说明：

300：清10进制运算标志

301：累加器清零

303：存放结果单元清零

- 305: 存放结果单元清零
- 307: 被乘数扩至16位
- 309: 预置移位次数
- 30B: 乘数带进位位右移
- 30D: 无进位, 不加被乘数
- 30F: 有进位, 清掉
- 310: 取部分积低位
- 312: 加被乘数低位
- 314: 存入部分积低位
- 316: 取部分积高位
- 318: 加被乘数高位
- 31A: 存部分积高位
- 31C: 被乘数低位左移一位
- 31E: 被乘数扩充位左移一位
- 320: 计数一次
- 321: 判计算完否, 未完循环
- 323: 计算完中断, 暂停

上机实习:

设(06)=34, (08)=5A, 结果(1A)应为12, (1B)应为48。在监控状态下键入 P11程序和乘数、被乘数, 单步运行:

* 06:34 ↵

* 08:5A ↵

* 300S ↵

显示: 0300-D8 CLD

A = 98 X = 14 Y = D8 P = 30 S = F4

展开状态寄存器 P

D							
0	0	1	1	0	0	0	0

,

可知D被置0。

继续运行: * S ↙

显示: 0301-A900 LDA # \$00

A = 00 X = 14 Y = D8 P = 32 S = F4

累加器置 0, 影响标志位 Z。

继续运行: * SS ↙

显示: 0303-85 1A STA \$1A

A = 00 X = 14 Y = D8 P = 32 S = F4

0305-85 1B STA \$1B

A = 00 X = 14 Y = D8 P = 32 S = F4

键入 * 1A, 1B ↙ 显示: 001A-00 00 可知结果存放单元已被置 0。

继续运行: * S ↙

显示: 0307-8507 STA \$07

A = 00 X = 14 Y = D8 P = 32 S = F4

被乘数扩展位也被置 0。

继续运行: * S ↙

显示: 0309-A2 08 LDX # \$08

A = 00 X = 08 Y = D8 P = 30 S = F4

X 寄存器已置入移位次数。

继续运行: * S ↙

显示: 30B-46 08 LSR \$08

A = 00 X = 08 Y = D8 P = 30 S = F4

乘数右移一位, 由 C 位为 0 可知乘数最低位为 0。

继续运行: * S ↙

显示: 030D-90 0D BCC \$31C

A = 00 X = 08 Y = D8 P = 30 S = F4

判别 C 为 0, 不加被乘数, 转至 \$031C 继续运算。

继续运行: * S↙

显示: 031C- 06 06 ASL \$06

A = 00 X = 08 Y = D8 P = 30 S = F4

被乘数低位左移一位。

继续运行: * S↙

显示: 031E- 26 07 ROL \$07

A = 00 X = 08 Y = D8 P = 32 S = F4

被乘数扩展位连同进位位左移。

继续运行: * S↙

显示: 0320- CA DEX

A = 00 X = 07 Y = D8 P = 30 S = F4

X寄存器减1, 表示运算了1次。

继续运行: * S↙

显示: 0321- D0 E8 BNE \$30B

A = 00 X = 07 Y = D8 P = 30 S = F4

判计算完否, 结果不为0 ($Z = 0$) 转\$030B继续。

继续运行: * S↙

显示: 030B- 46 08 LSR \$08

A = 00 X = 07 Y = D8 P = 31 S = F4

乘数再右移一位, 由C位为1可知移出位为1。

继续运行: * S↙

显示: 030D- 90 0D BCC \$031C

A = 00 X = 07 Y = D8 P = 31 S = F4

判C位为1, 加被乘数, 不转。

继续运行: * S↙

显示: 030F- 18 CLC

A = 00 X = 07 Y = D8 P = 30 S = F4

清除 C，为加被乘数作准备。

继续运行：* S ↙

显示：0310- A5 1B LDA \$1B

A = 00 X = 07 Y = D8 P = 32 S = F4

取部分积低 8 位至累加器。

继续运行：* S ↙

显示：0312- 65 06 ADC \$06

A = 68 X = 07 Y = D8 P = 30 S = F4

与被乘数低 8 位相加，结果在累加器中。

继续运行：* S ↙

显示：0314- 85 1B STA \$1B

A = 68 X = 07 Y = D8 P = 30 S = F4

将结果送回部分积中。

继续运行：* S ↙

显示：0316- A5 1A LDA \$1A

A = 00 X = 07 Y = D8 P = 32 S = F4

再取部分积高 8 位至累加器。

继续运行：* S ↙

显示：0318-65 07 ADC \$07

A = 00 X = 07 Y = D8 P = 32 S = F4

与被乘数扩展位相加，结果在累加器中。

继续运行：* S ↙

显示：031A- 85 1A STA \$1A

A = 00 X = 07 Y = D8 P = 32 S = F4

将结果送回部分积中。

继续运行：* S ↙

显示：031C- 06 06 ASL \$06

A = 00 X = 07 Y = D8 P = B0 S = F4

被乘数低位左移一位。

继续运行: * S ↙

显示: 031E- 26 07 ROL \$07

A = 00 X = 07 Y = D8 P = 32 S = F4

被乘数扩展位连同进位位左移。

继续运行: * S ↙

显示: 0320- CA DEX

A = 00 X = 06 Y = D8 P = 30 S = F4

X寄存器再减1, 表示已计算了2次。

继续运行: * S ↙

显示: 0321- D0 EB BNE \$030B

A = 00 X = 06 Y = DB P = 30 S = F4

判计算完否, 未完转\$030B继续运算。重复上面的运算, 直至X减为0, 计算结束, 程序中断。键入: *1A.1B ↙ 显示: 001A- 12 48 可见计算结果正确。

后面两个8位2进制乘法程序, 上机实习步骤与上述相同, 这里就不再说明了, 留待读者自己去实习。

②. 部分积右移法: 将被乘数放入\$06单元, 乘数放入\$08单元, 运算结果高8位放\$1A单元, 低8位放\$1B单元。程序P12如下:

0300-	D8	CLD	
0301-	A9 00	LDA	# \$ 00
0303-	85 1A	STA	\$ 1A
0305-	85 1B	STA	\$ 1B
0307-	A2 08	LDX	# \$ 08
0309-	46 08	LSR	\$ 08

030B-	90	03	BCC	\$ 0310
030D-	18		CLC	
030E-	65	06	ADC	\$ 06
0310-	6A		ROR	
0311-	66	1B	ROR	\$ 1 B
0313-	CA		DEX	
0314-	D0	F3	BNE	\$ 0309
0316-	85	1A	STA	\$ 1 A
0318-	00		BRK	

程序说明:

300: 清10进制标志位

301: 累加器清零

303: 存放结果单元清零

305: 存放结果单元清零

307: 预置移位次数

309: 乘数带进位位右移

30B: 无进位, 不加被乘数, 转

30D: 有进位, 清掉

30E: 部分积和被乘数相加

310: 部分积右移 1 位, 相加产生的进位C移进最高位

311: 移出的低位送 \$ 1B高位

313: 计数一次

314: 判计算完否, 未完循环

316: 计算完, 把乘积高 8 位送入 \$ 1A

318: 中断, 暂停

③ 部分积左移法: 将被乘数放入 \$ 06单元, 乘数放入 \$ 08单元, 运算结果高 8 位放入 \$ 1A 单元, 低 8 位放入 \$ 1B单元。程序P13如下:

0300-	D8	CLD
-------	----	-----

0301-	A9	00	LDA	# \$ 00
0303-	85	1A	STA	\$ 1 A
0305-	85	1B	STA	\$ 1 B
0307-	A2	08	LDX	# \$ 08
0309-	0A		ASL	
030A-	26	1A	ROL	\$ 1 A
030C-	06	08	ASL	\$ 08
030E-	90	07	BCC	\$ 0317
0310-	18		CLC	
0311-	65	06	ADC	\$ 06
0313-	90	02	BCC	\$ 0317
0315-	E6	1A	INC	\$ 1 A
0317-	CA		DEX	
0318-	DO	EF	BNE	\$ 0309
031A-	85	1B	STA	\$ 1 B
031C-	00		BRK	

程序说明:

300: 清10进制运算标志

301: 累加器清零

303: 存放结果单元清零

305: 存放结果单元清零

307: 预置移位次数

309: 部分积低 8 位左移

30A: 部分积高低位串起来左移

30C: 乘数左移

30E: 判别乘数移出位为 0 , 不加被乘数

310: 乘数移出位为 1 , 清掉

311: 加被乘数

313: 判有无进位位, 无转

315: 有进位, 部分积高八位加进位

317: 计数一次

318: 判计算完否, 未完循环

31A: 乘积低 8 位送 \$ 1B

31C: 计算完中断

(2) 16 位 2 进制整数乘法程序:

① 被乘数左移法: 两个 16 位 2 进制数相乘, 其结果为 32 位, 用被乘数左移法时, 要将被乘数扩展至 32 位, 乘数移位时要安排成 16 位右移, 程序见 P14。

将被乘数高 8 位存入 \$ 08 单元, 低 8 位存入 \$ 09 单元, 并扩展至 \$ 06、\$ 07 单元, 乘数高 8 位存入 \$ 1A 单元, 低 8 位存入 \$ 1B 单元, 结果高 16 位存入 \$ EA、\$ EB 单元, 低 16 位存入 \$ EC、\$ ED 单元。

0300-	D8		CLD	
0301-	A9	00	LDA	# \$ 00
0303-	85	EA	STA	\$ EA
0305-	85	EB	STA	\$ EB
0307-	85	EC	STA	\$ EC
0309-	85	ED	STA	\$ ED
030B-	85	06	STA	\$ 06
030D-	85	07	STA	\$ 07
030F-	A2	10	LDX	# \$ 10
0311-	46	1A	LSR	\$ 1A
0313-	66	1B	ROR	\$ 1B
0315-	90	19	BCC	\$ 0330
0317-	18		CLC	
0318-	A5	ED	LDA	\$ ED
031A-	65	09	ADC	\$ 09
031C-	85	ED	STA	\$ ED

031E-	A5	EC	LDA	\$ EC
0320-	65	08	ADC	\$ 08
0322-	85	EC	STA	\$ EC
0324-	A5	EB	LDA	\$ EB
0326-	65	07	ADC	\$ 07
0328-	85	EB	STA	\$ EB
032A-	A5	EA	LDA	\$ EA
032C-	65	06	ADC	\$ 06
032E-	85	EA	STA	\$ EA
0330-	06	09	ASL	\$ 09
0332-	26	08	ROL	\$ 08
0334-	26	07	ROL	\$ 07
0336-	26	06	ROL	\$ 06
0338-	CA		DEX	
0339-	D0	D6	ENE	\$ 0311
033B-	00		BRK	

程序说明:

300: 清10进制运算标志

301: 累加器清零

303: 存放结果单元清零

305: 存放结果单元清零

307: 存放结果单元清零

309: 存放结果单元清零

30B: 被乘数扩展单元清零

30D: 被乘数扩展单元清零

30F: 预置移位次数

311: 乘数16位右移

313: 判乘数最低位代码

315: 为0, 不加被乘数, 转被乘数左移一位

317: 为 1, 加被乘数, 清C
 318: 取部分积低16位
 31A: 与被乘数低 8 位相加
 31C: 送回低16位
 31E: 取部分积低16位
 320: 与被乘数高 8 位相加
 322: 送回低16位
 324: 取部分积高16位
 326: 与被乘数扩展位相加
 328: 送回高16位
 32A: 取部分积高16位
 32C: 与被乘数扩展值相加
 32E: 送回高16位
 330: 被乘数低 8 位左移一位
 332: 被乘数高 8 位左移一位
 334: 被乘数扩展16位左移
 336: 被乘数扩展16位左移
 338: 计数一次
 339: 判计算完否, 未完转 \$ 311继续循环
 33B: 计算完中断

上机实习:

设 (08) = 61, (09) = 04, (1A) = 40, (1B) = 24, 结果
 (EA) 应为18, (EB) 应为4E, (EC)应为 A4, (ED)
 应为90。在监控状态下键入P14程序和16 位乘数、被乘数,
 并单步运行:

* 08 : 61 □ 04 ↵
 * 1A : 40 □ 24 ↵
 * 300S ↵

显示: 0300- D8 CLD

A = 98 X = 04 Y = D8 P = 30 S = F4

继续运行: * SSSSSSS ✓

显示: 0301- A 900 LDA # \$ 00

A = 00 X = 04 Y = D8 P = 32 S = F4

0303- 85 EA STA \$EA

A = 00 X = 04 Y = D8 P = 32 S = F4

0305- 85 EB STA \$EB

A = 00 X = 04 Y = D8 P = 32 S = F4

0307- 85 EC STA \$EC

A = 00 X = 04 Y = D8 P = 32 S = F4

0309- 85 ED STA \$ED

A = 00 X = 04 Y = D8 P = 32 S = F4

030B- 85 06 STA \$06

A = 00 X = 04 Y = D8 P = 32 S = F4

030D- 85 07 STA \$07

A = 00 X = 04 Y = D8 P = 32 S = F4

键入: * EA,ED ✓

显示: 00EA- 00 00 00 00

键入: * 06,07 ✓

显示: 0006—00 00

可见上述单元均已被清零。

继续运行: * S ✓

显示: 030F- A2 10 LDX # \$10

A = 00 X = 10 Y = D8 P = 30 S = F4

移位次数已被置入X寄存器。

继续运行: * SS ✓

显示: 0311- 46 1A LSR \$1A

A = 00 X = 10 Y = D8 P = 30 S = F4

0313-66 1B ROR \$1B

A = 00 X = 10 Y = D8 P = 30 S = F4

乘数 16 位高低位串起来右移一次, 由 P 可知, 移出位为 0 (C = 0)。

继续运行: * S ↙

显示: 0315- 90 19 BCC \$330

A = 00 X = 10 Y = D8 P = B0 S = F4

判 C = 0, 不加被乘数, 转 \$330 继续。

继续运行: * SSSS ↙

显示: 0330- 06 09 ASL \$09

A = 00 X = 10 Y = D8 P = 30 S = F4

0332- 26 08 ROL \$08

A = 00 X = 10 Y = D8 P = B0 S = F4

0334- 26 07 ROL \$07

A = 00 X = 10 Y = D8 P = 32 S = F4

0336- 26 06 ROL \$06

A = 00 X = 10 Y = D8 P = 32 S = F4

被乘数包括扩展 16 位一起左移一位。

继续运行: * S ↙

显示: 0338- CA DEX

A = 00 X = 0F Y = D8 P = 30 S = F4

X - 1, 计数一次。

继续运行: * S ↙

显示: 0339- D0 D6 BNE \$0311

A = 00 X = 0F Y = D8 P = 30 S = F4

X不为0，转\$311继续运算。

继续运行：*SS↙

显示：0311- 46 1A LSR \$1A

A = 00 X = 0F Y = D8 P = 3D S = F4

0313- 66 1B ROR \$1B

A = 00 X = 0F Y = D8 P = 30 S = F4

乘数16位再次右移，由P可知，移出位仍为0。

继续运行：*S↙

显示：0315-90 19 BCC \$330

A = 00 X = 0F Y = D8 P = 30 S = F4

判C = 0，不加被乘数，转\$330继续。

继续运行：*SSSS↙

显示：0330- 06 09 ASL \$09

A = 00 X = 0F Y = D8 P = 3D S = F4

0332- 26 08 ROL \$08

A = 00 X = 0F Y = D8 P = B1 S = F4

0334- 26 07 ROL \$07

A = 00 X = 0F Y = D8 P = 30 S = F4

0336- 26 06 ROL \$06

A = 00 X = 0F Y = D8 P = 32 S = F4

被乘数包括扩展16位一起左移一位。

继续运行：*SS↙

显示：0338- CA DEX

A = 00 X = 0E Y = D8 P = 30 S = F4

0339- D0 D6 BNE \$311

计数，并判X不为0，转\$311继续运算。

继续运行：*SS↙

显示: 0311- 46 1A LSR \$1A
 A = 00 X = 0E Y = D8 P = 30 S = F4
 0313- 66 1B ROR \$1B
 A = 00 X = 0E Y = D8 P = 31 S = F4

乘数16位再次右移, 由P可知, C = 1。

继续运行: * S ↙

显示: 0315- 90 19 BCC \$0330
 A = 00 X = 0E Y = D8 P = 31 S = F4

C = 1, 不转, 继续往下运行, 加被乘数。

继续运行: * S ↙

显示: 0317- 18 CLC
 A = 00 X = 0E Y = D8 P = 30 S = F4

清C标志, 为加被乘数作准备。

继续运行: * SSS ↙

显示: 0318- A5 ED LDA \$ED
 A = 00 X = 0E Y = D8 P = 32 S = F4
 031A- 65 09 ADC \$09
 A = 10 X = 0E Y = D8 P = 30 S = F4
 031C- 85 ED STA \$ED
 A = 10 X = 0E Y = D8 P = 30 S = F4

取部分积单元结果与被乘数低8位相加后送回。键入: * ED ↙

显示: 00ED-10

继续运行: * SSS ↙

 031E- A5 EC LDA \$EC
 A = 00 X = 0E Y = D8 P = 32 S = F4
 0320- 65 08 ADC \$08
 A = 84 X = 0E Y = D8 P = B0 S = F4

0322- 85 EC STA \$EC

A = 84 X = 0E Y = D8 P = B0 S = F4

取部分积单元结果与被乘数高 8 位相加后送回。

键入: *EC↵ 显示: 00EC-84

继续运行: *SSS↵

显示: 0324- A5 EB LDA \$EB

A = 00 X = 0E Y = D8 P = 32 S = F4

0326- 65 07 ADC \$07

A = 01 X = 0E Y = D8 P = 30 S = F4

0328- 85 EB STA \$EB

A = 01 X = 0E Y = D8 P = 30 S = F4

取部分积高16位与被乘数扩展16位相加后送回。\$07单元中的01即是被乘数左移的结果。

继续运行: *SSS↵

显示: 032A- A5 EA LDA \$EA

A = 00 X = 0E Y = D8 P = 32 S = F4

032C- 65 06 ADC \$06

A = 00 X = 0E Y = D8 P = 32 S = F4

032E- 85 EA STA \$EA

A = 00 X = 0E Y = D8 P = 32 S = F4

取部分积高16位与被乘数扩展16位相加后送回。

继续运行: *SSSS↵

显示: 0330- 06 09 ASL \$09

A = 00 X = 0E Y = D8 P = 30 S = F4

0332- 26 08 ROL \$08

A = 00 X = 0E Y = D8 P = 31 S = F4

0334- 26 07 ROL \$07

A = 00 X = 0E Y = D8 P = 30 S = F4

0336- 26 06 ROL \$ 06

A = 00 X = 0E Y = D8 P = 32 S = F4

被乘数包括扩展16位一起左移一位。

继续运行: *SS↙

显示: 0338- CA DEX

A = 00 X = 0D Y = D8 P = 30 S = F4

0339- D0 D6 BNE \$ 311

计数, 判X不为0, 转\$ 311继续运算。

……直至运算结束。

键入: *EA,ED↙ 显示: 00EA—18□ 4E□ A4□ 90
可见结果正确。

② 部分积右移法: 两个16位二进制数相乘, 其结果为32位, 因此, 用部分积右移法, 乘数移位时要安排成16位右移, 部分积移位时要安排成32位右移。程序见P15。

将被乘数高8位存入\$ 06单元, 低8位存入\$ 07单元, 乘数高8位存入\$ 08单元, 低8位存入\$ 09单元, 结果高16位存放\$ 1A、\$ 1B单元, 低16位存放\$ 1C、\$ 1D单元。

0300-	D8	CLD	
0301-	A9 00	LDA	# \$ 00
0303-	85 1A	STA	\$ 1A
0305-	85 1B	STA	\$ 1B
0307-	85 1C	STA	\$ 1C
0309-	85 1D	STA	\$ 1D
030B-	A2 10	LDX	# \$ 10
030D-	46 08	LSR	\$ 08
030F-	66 09	ROR	\$ 09

0311-	90	0D	BCC	\$ 0320
0313-	18		CLC	
0314-	A5	1B	LDA	\$ 1B
0316-	65	07	ADC	\$ 07
0318-	85	1B	STA	\$ 1B
031A-	A5	1A	LDA	\$ 1A
031C-	65	06	ADC	\$ 06
031E-	85	1A	STA	\$ 1A
0320-	66	1A	ROR	\$ 1A
0322-	66	1B	ROR	\$ 1B
0324-	66	1C	ROR	\$ 1C
0326-	66	1D	ROR	\$ 1D
0328-	CA		DEX	
0329-	D0	E 2	BNE	\$ 030D
032B-	00		BRK	

程序说明:

300: 清10进制标志位

301: 累加器清零

303: 结果单元清零

305: 结果单元清零

307: 结果单元清零

309: 结果单元清零

30B: 预置移位次数

30D: 乘数16位右移

30F: 判乘数最低位代码

311: 为 0, 不加被乘数, 转部分积移位

313: 为 1, 加被乘数, 清C

314: 取部分积

316: 与被乘数低 8 位相加

318: 送回部分积单元
 31A: 取部分积
 31C: 与被乘数高 8 位相加
 31E: 送回部分积单元
 320: 部分积右移一位
 328: 计数一次
 329: 判计算完否, 未完循环
 32B: 计算完中断

本程序和下一个程序的上机实习就留给读者自己去进行了。

③ 部分积左移法: 同P14一样, 将被乘数高 8 位存入 \$06 单元, 低 8 位存入 \$07 单元, 乘数高 8 位存入 \$08 单元, 低 8 位存入 \$09 单元, 结果高 16 位存入 \$1A、\$1B 单元, 低 16 位存入 \$1C、\$1D 单元。程序见 P16。

0300-	D8	CLD	
0301-	A9 00	LDA	# \$ 00
0303-	85 1A	STA	\$ 1 A
0305-	85 1B	STA	\$ 1 B
0307-	85 1C	STA	\$ 1 C
0309-	85 1D	STA	\$ 1 D
030B-	A2 10	LDX	# \$ 10
030D-	26 1D	ROL	\$ 1 D
030F-	26 1C	ROL	\$ 1 C
0311-	26 1B	ROL	\$ 1 B
0313-	26 1A	ROL	\$ 1 A
0315-	06 09	ASL	\$ 09
0317-	26 08	ROL	\$ 08
0319-	90 11	BCC	\$ 032C
031B-	18	CLC	

031C-	A5	1D	LDA	\$ 1D
031E-	65	07	ADC	\$ 07
0320-	85	1D	STA	\$ 1D
0322-	A5	1C	LDA	\$ 1C
0324-	65	06	ADC	\$ 06
0326-	85	1C	STA	\$ 1C
0328-	90	02	BCC	\$ 032 C
032A-	E6	1B	INC	\$ 1B
032C-	CA		DEX	
032D-	D0	DE	BNE	\$ 030 D
032F-	00		BRK	

程序说明:

300: 清10进制标志位
 301: 累加器清零
 303: 结果单元清零
 305: 结果单元清零
 307: 结果单元清零
 309: 结果单元清零
 30B: 预置移位次数
 30D: 部分积左移一位
 30F: 部分积左移一位
 311: 部分积左移一位
 313: 部分积左移一位
 315: 乘数16位左移
 317: 判乘数最高位代码
 319: 为0, 不加被乘数, 转 \$ 32C
 31B: 为1, 加被乘数, 清C
 31C: 取部分积低位
 31E: 与被乘数低8位相加

320: 送回部分积单元
 322: 取部分积低位
 324: 与被乘数高 8 位相加
 326: 送回部分积单元
 328: 判有无进位位, 无转 \$ 32C
 32A: 有进位, 加入部分积高位
 32C: 计数一次
 32D: 判计算完否, 未完循环
 32F: 计算完中断。

(3) 10进制乘法: 10进制乘法运算的原理与 2 进制基本相同, 但在 10 进制中, 乘数和积的左移一位相当于 2 进制的 4 位, 并且 10 进制数是从 0 到 9 变化, 因而积加被乘数运算的重复次数等于正在运算的那位 10 进制数值。这里我们再介绍两个 10 进制乘法程序。

① 2 位乘 2 位: 将被乘数存入 \$06 单元, 乘数存入 \$08 单元, 注意要按 2—10 进制 BCD 码存入, 结果在 \$1A、\$1B 中。程序见 P17。

0300-	F8	SED	
0301-	A9 00	LDA	# \$ 00
0303-	85 1A	STA	\$ 1A
0305-	85 1B	STA	\$ 1B
0307-	A2 02	LDX	# \$ 02
0309-	A5 08	LDA	\$ 08
030B-	29 F0	AND	# \$ F0
030D-	F0 15	BEQ	\$ 0324
030F-	4A	LSR	
0310-	4A	LSR	
0311-	4A	LSR	
0312-	4A	LSR	

0313-	A8	TAY	
0314-	18	CLC	
0315-	A5 1B	LDA	\$ 1B
0317-	65 06	ADC	\$ 06
0319-	85 1B	STA	\$ 1B
031B-	A9 00	LDA	# \$ 00
031D-	65 1A	ADC	\$ 1A
031F-	85 1A	STA	\$ 1A
0321-	88	DEY	
0322-	D0 F1	BNE	\$ 0315
0324-	CA	DEX	
0325-	F0 12	BEQ	\$ 0339
0327-	A0 04	LDY	# \$ 04
0329-	06 1B	ASL	\$ 1B
032B-	26 1A	ROL	\$ 1A
032D-	88	DEY	
032E-	D0 F9	BNE	\$ 0329
0330-	A5 08	LDA	\$ 08
0332-	29 0F	AND	# \$ 0F
0334-	F0 03	BEQ	\$ 0339
0336-	4C 13 03	JMP	\$ 0313
0339-	00	BRK	

程序说明:

- 300: 置10进制运算
- 301: 累加器清零
- 303: 结果单元清零
- 305: 结果单元清零
- 307: 运算位数置入X
- 309: 取乘数

30B: 屏蔽低位 (10进制个位)
 30D: 判结果为 0 转 \$ 324进行下一位运算
 30F- 将高位移往低位,
 (即10进制10位移到个位)
 312: 结果缩小10倍
 313: 送入Y计数
 314: 清C
 315: 结果取低位
 317: 加被乘数
 319: 送回
 31B- 如有进位, 则加至
 320: 结果高位
 321: Y - 1
 322: 判加完否, 未完继续
 324: 加完, X - 1, 计算下一位
 325: 判是否全计算完
 327- 未完将运算结果乘
 10 (10进制), 即恢复
 32F: 原结果
 330: 取乘数
 332: 屏蔽高位 (10进制10位数)
 334: 判结果为零结束
 336: 转 \$ 313继续计算
 339: 计算完结束

上机实习:

设(06) = 95, (08) = 74, 结果(1A) 应为 70, (1B)应
 为30。在监控状态下键入P17程序, 并输入被乘数与 乘数,
 然后单步运行:

* 06: 95✓

* 08: 74↵

* 300S↵

显示: 0300-F8 SED

A = 00 X = 03 Y = 09 P = 38 S = D2

由P可知, D置1, 表示进行10进制运算。

继续运行: SSS↵

显示: 0301- A9 00 LDA # \$00

A = 00 X = 03 Y = 09 P = 3A S = D2

0303- 85 1A STA \$1A

A = 00 X = 03 Y = 09 P = 3A S = D2

0305- 85 1B STA \$1B

A = 00 X = 03 Y = 09 P = 3A S = D2

将结果单元清零。键入: 1A、1B↵

显示: 001A- 00 00

可见结果单元已被清零。

继续运行: S↵

显示: 0307- A2 02 LDX # \$02

A = 00 X = 02 Y = 09 P = 38 S = D2

乘数的位数置入X寄存器。

继续运行: S↵

显示: 0309- A5 08 LDA \$08

A = 74 X = 02 Y = 09 P = 38 S = D2

乘数已取入累加器。

继续运行: S↵

显示: 030B-29 F0 AND # \$F0

A = 70 X = 02 Y = 09 P = 38 S = D2

乘数个位数被屏蔽掉。

继续运行: SSS✓

显示: 030D- F0 15 BEQ \$0324

A = 70 X = 02 Y = 09 P = 38 S = D2

030F- 4A LSR

A = 38 X = 02 Y = 09 P = 38 S = D2

0310- 4A LSR

A = 1C X = 02 Y = 09 P = 38 S = D2

0311- 4A LSR

A = 0E X = 02 Y = 09 P = 38 S = D2

0312- 4A LSR

A = 07 X = 02 Y = 09 P = 38 S = D2

2 进制移动 4 位才是 10 进制移动一位, 乘数 10 位数被移至个位。

继续运行: S ✓

显示: 0313- A8 TAY

A = 07 X = 02 Y = 07 P = 38 S = D2

乘数 10 位数移入 Y 寄存器作计数用。

继续运行: S ✓

显示: 0314- 18 CLC

A = 07 X = 02 Y = 07 P = 38 S = D2

清 C, 为 10 进制加法运算作准备。

继续运行: SSS✓

显示: 0315- A5 1B LDA \$1B

A = 0 X = 02 Y = 07 P = 3A S = D2

0317- 65 06 ADC \$06

A = 9 X = 02 Y = 07 P = B8 S = D2

0319- 85 1B STA \$1B

A = 95 X = 02 Y = 07 P = B8 S = D2

加被乘数至结果单元。

继续运行: SSS↙

显示: 031B- A9 00 LDA # \$00

A = 00 X = 02 Y = 07 P = 3A S = D2

031D- 65 1A ADC \$1A

A = 00 X = 02 Y = 07 P = 3A S = D2

031F- 85 1A STA \$1A

A = 00 X = 02 Y = 07 P = 3A S = D2

将10进制加法产生的进位加至\$1A单元。

继续运行: S↙

显示: 0321- 88 DEY

A = 00 X = 02 Y = 06 P = 38 S = D2

计数一次(减少)。

继续运行: S↙

显示: 0322- D0 F1 BNE \$0315

A = 00 X = 02 Y = 06 P = 38 S = D2

.....

由P可知结果不为0 ($Z = 0$), 转\$315继续加被乘数, 直至Y减为0。这与2进制乘法是有明显区别的。

继续运行: S↙

显示: 0324- CA DEX

A = 06 X = 01 Y = 0 P = 38 S = D2

计算完一位, $X - 1$ 。

继续运行: S↙

显示: 0325- F0 12 BEQ \$0339

A = 06 X = 01 Y = 0 P = 38 S = D2

判 $X = 0$ 否, 为 0 表示计算完, 转\$339结束。不为 0, 不转继续往下运行。

继续运行: SSSSS✓

```
显示: 0327- A0 04      LDY  #$04
A = 06  X = 01  Y = 04  P = 38  S = D2
      0329- 06 1B      ASL  $1B
A = 06  X = 01  Y = 04  P = B8  S = D2
      032B- 26 1A      ROL  $1A
A = 06  X = 01  Y = 04  P = 38  S = D2
      032D- 88        DEY
A = 06  X = 01  Y = 03  P = 38  S = D2
      032E- D0 F9      BNE  $0329
A = 06  X = 01  Y = 03  P = 38  S = D2
```

.....

由 P 可知结果不为 0, 转\$329继续移, 直至 Y 减为 0。这实际上是10进制移一位 (乘10)。

继续运行: SSS✓

```
显示: 0330- A5 08      LDA  $08
A = 74  X = 01  Y = 00  P = 38  S = D2
      0332- 29 0F      AND  #$0F
A = 04  X = 01  Y = 00  P = 38  S = D2
      0334- F0 03      BEQ  $0339
A = 04  Y = 01  Y = 00  P = 38  S = D2
```

取乘数个位继续运算, 如果为 0 则结束, 不为 0 则往下运行。

继续运行: S✓

```
显示: 0336- 4C 13 03    JMP  $0313
```

A = 04 X = 01 Y = 00 P = 38 S = D2

转\$313进行个位乘法运算，即重复上述步骤，直至运算到X - 1 = 0，则跳转\$339结束。

键入：1A、1B↵显示：001A-70┐30

可见结果正确。

② 4位乘2位：将被乘数高位存入\$06单元，低位存入\$07单元，乘数存入\$08单元，结果存在\$1A、\$1B、\$1C单元中。程序见P18。

0300-	F8	SED	
0301-	A9 00	LDA	# \$ 00
0303-	85 1A	STA	\$ 1A
0305-	85 1B	STA	\$ 1B
0307-	85 1C	STA	\$ 1C
0309-	A2 02	LDX	# \$ 02
030B-	A5 08	LDA	\$ 08
030D-	29 F0	AND	# \$ F0
030F-	F0 1B	BEQ	\$ 032C
0311-	4A	LSR	
0312-	4A	LSR	
0313-	4A	LSR	
0314-	4A	LSR	
0315-	A8	TAY	
0316-	18	CLC	
0317-	A5 1C	LDA	\$ 1C
0319-	65 07	ADC	\$ 07
031B-	85 1C	STA	\$ 1C
031D-	A5 1B	LDA	\$ 1B
031F-	65 06	ADC	\$ 06

0321-	85	1B	STA	\$ 1B
0323-	A9	00	LDA	# \$ 00
0325-	65	1A	ADC	\$ 1A
0327-	85	1A	STA	\$ 1A
0329-	88		DEY	
032A-	D0	EB	BNE	\$ 0317
032C-	CA		DEX	
032D-	F0	14	BEQ	\$ 0343
032F-	A0	04	LDY	# \$ 04
0331-	06	1C	ASL	\$ 1C
0333-	26	1B	ROL	\$ 1B
0335-	26	1A	ROL	\$ 1A
0337-	88		DEY	
0338-	D0	F7	BNE	\$ 0331
033A-	A5	08	LDA	\$ 08
033C-	29	0F	AND	# \$ 0F
033E-	F0	03	BEQ	\$ 0343
0340-	4C	15 03	JMP	\$ 0315
0343-	00		BRK	

程序说明:

300: 置10进制运算标志

301: 累加器清零

303: 结果单元清零

305: 结果单元清零

307: 结果单元清零

309: 运算位数置入X寄存器

30B: 取乘数

30D: 屏蔽低位 (10进制个位)

30F: 判结果为 0 转 \$ 32C 计算下一位

311-将高位移动4次,即
把10进制10位数移到
314: 个位。
315: 送入Y计数
316: 清C
317: 取结果低位
319: 加被乘数低位
31B: 送回
31D: 取结果中间位
31F: 加被乘数高位
321: 送回
323: 上述计算如有进位,
325: 加入结果高位
327: 送回
329: Y - 1
32A: 判加完否, 未完继续。
32C: 加完, X - 1, 计算下一位。
32D: 判是否全计算完, 完转 \$ 33F 结束。
32F: 未完, 将运算结果乘10 (10进制), 即恢复原
339: 结果
33A: 取乘数
33C: 屏蔽高位
33E: 判结果为0 转 \$ 343
340: 转 \$ 315继续运算
343: 中断, 结束
请读者自行上机验证。

4. 除法程序

除法是乘法的逆运算,其运算过程与10进制类似。我们举个例子予以说明。设被除数为一个16位数7460,除数为一个8位数78,所得结果也为一个8位数F8,余数为20,运算过程如下:

$$\begin{array}{r}
 0000000011111000 \\
 01111000 \overline{) 0111010001100000} \\
 \underline{01111000} \\
 11100001 \\
 \underline{01111000} \\
 11010011 \\
 \underline{01111000} \\
 10110110 \\
 \underline{01111000} \\
 01111100 \\
 \underline{01111000} \\
 00100000
 \end{array}$$

运算时,我们是从被除数的最高位开始检查,不够除时,商上0,够除时商上1,并减去除数,然后把被除数的下一位下移到余数上,再看够不够除(即余数够不够减去除数),够则商上1,余数减去除数,把被除数的再下一位移到余数上,不够则商上0,不减除数,把被除数的再下一位移到余数上,一直除下去,直到全部被除数的位都下移完为止。

计算机在运算时,一般是采用移位相减的方法。我们仍用上例来说明:

被除数	0111010001100000	商	00000000
左移	1110100011000000	左移	00000000
- 除数	01111000		

够减 C = 1	0111000011000000	加 1	00000001
左移	1110000110000000	左移	00000010
- 除数	01111000		
够减 C = 1	0110100110000000	加 1	00000011
左移	1101001100000000	左移	00000110
- 除数	01111000		
够减 C = 1	0101101100000000	加 1	00000111
左移	1011011000000000	左移	00001110
- 除数	01111000		
够减 C = 1	0011111000000000	加 1	00001111
左移	0111110000000000	左移	00011110
- 除数	01111000		
够减 C = 1	0000010000000000	加 1	00011111
左移	0000100000000000	左移	00111110
- 除数	01111000		
不够减, 不减	0000100000000000	不加	00111110
(C = 0)			
左移	0001000000000000	左移	01111100
- 除数	01111000		
不够减, 不减	0001000000000000	不加	01111100
(C = 0)			
左移	0010000000000000	左移	11111000
- 除数	01111000		
不够减, 不减	00100000	不加	11111000
(C = 0)			

移满 8 次, 结束, 结果商为 F8, 余数为 20。下面我们介绍具体程序。

(1) 8 位 2 进制除法程序 用机器语言编制除法程序，要防止由于溢出而造成运算错误，其主要方法是对参与运算的数进行限制。对于不带符号的 8 位 2 进制除法，被除数为 16 位，除数为 8 位，并规定被除数和除数的最高位是零，被除数的高 8 位小于除数。符合这一规定的数称为数据规格化的数。上面的例子就是数据规格化的数。程序 P19 也是按数据规格化的数设计的。

程序 P19 如下：

将被除数高 8 位存入 \$06 单元，低 8 位存入 \$07 单元，除数存入 \$08 单元，商放在 \$1A 单元，余数放在 \$1B 单元。

0300-	D8	CLD	
0301-	A2 08	LDX	# \$ 08
0303-	A5 07	LDA	\$ 07
0305-	85 1A	STA	\$ 1A
0307-	A5 06	LDA	\$ 06
0309-	06 1A	ASL	\$ 1A
030B-	2A	ROL	
030C-	C5 08	CMP	\$ 08
030E-	90 04	BCC	\$ 0314
0310-	E5 08	SBC	\$ 08
0312-	E6 1A	INC	\$ 1A
0314-	CA	DEX	
0315-	D0 F2	BNE	\$ 0309
0317-	85 1B	STA	\$ 1B
0319-	00	BRK	

程序说明：

0300：清 10 进制运算标志

0301：预置移位次数

0303: 取被除数低 8 位
 0305: 放入存放商的单元中
 0307: 取被除数高 8 位
 0309-30B: 将商与被除数同时左移
 030C: 与除数比较, 看是否够减
 030E: C 为 0, 不够减转
 0310: C 为 1, 够减, 减去除数
 0312: 商加 1
 0314: 计数一次
 0315: 判计算完否, 未完继续循环
 0317: 计算完, 将余数放入 \$1B 单元
 0319: 中断, 结束

本程序在编制上采用了一些技巧, 将被除数的低 8 位放入商中与高 8 位串起来左移, 同时实现了 16 位被除数的左移和商的左移, 由于被除数每移一次要丢掉一位, 所以商与被除数共用 16 位存贮器单元不会影响结果。如不采取这种方法, 而是商与被除数分开, 则又要增加好几条指令才行。

上机实习:

设 (06) = 4D, (07) = 5E, (08) = 69, 则商 (1A) 应为 BC, 余数 (1B) 为 42。

键入 P19 程序, 并输入被除数与除数, 单步运行:

* 06: 4D □ 5E □ 69 ↵

* 300 S ↵

显示: 0300- D8 CLD

A = 98 X = 00 Y = 0D P = 30 S = E0

D 置 0, 表示进行 2 进制运算。

继续运行: * S ↵

显示: 0301- A2 08 LDX # \$08

A = 98 X = 08 Y = 0D P = 30 S = E0

移位次数08置入X寄存器。

继续运行: * SS↙

显示: 0303- A5 07 LDA \$07

A = 5E X = 08 Y = 0D P = 30 S = E0

0305- 85 1A STA \$1A

A = 5E X = 08 Y = 0D P = 30 S = E0

被除数低8位存入商中。

继续运行: * SSS↙

显示: 0307- A5 06 LDA \$06

A = 4D X = 08 Y = 0D P = 30 S = E0

0309- 06 1A ASL \$1A

A = 40 X = 08 Y = 0D P = B0 S = E0

030B- 2A ROL

A = 9A X = 08 Y = 0D P = B0 S = E0

商与被除数同时左移。

继续运行: * S↙

显示: 030C- C5 08 CMP \$08

A = 9A X = 08 Y = 0D P = 31 S = E0

与除数比较。

继续运行: * S↙

显示: 030E- 90 04 BCC \$0314

A = 9A X = 08 Y = 0D P = 31 S = E0

由C = 1可知, 被除数够除, 不转\$314。

继续运行: * S↙

显示: 0310- E5 08 SBC \$08

A = 31 X = 08 Y = 0D P = 71 S = E0

减去除数。

继续运行: * S ↙

显示: 0312- E6 1A INC \$1A

A = 31 X = 08 Y = 0D P = F1 S = E0

商加 1。

继续运行: * S ↙

显示: 0314- CA DEX

A = 31 X = 07 Y = 0D P = 71 S = E0

X - 1, 计数一次。

继续运行: * S ↙

显示: 0315- D0 F2 BNE \$0309

A = 31 X = 07 Y = 0D P = 71 S = E0

判X不为 0, 转\$309继续运算……,直至X减为 0,再继续运行: * S ↙

显示: 0315- D0 F2 BNE \$0309

A = 42 X = 00 Y = 0D1 P = 32 S = E0

X减为 0, 不转\$03 09

继续运行: * S ↙

显示: 0317- 85 1B STA \$1B

A = 42 X = 00 Y = 0D P = 32 S = E0

留在A中的余数送至\$1B保存。

继续运行: * S ↙

显示: 0319- 00 BRK

A = 42 X = 00 Y = 0D P = 32 S = E0

结束。键入 * 1A、1B ↙

显示: 001A- BC 42

说明运算结果正确。

(2) 多字节 2 进制除法程序 对多字节 2 进制除法, 同样要注意参加运算的数必须是规格化的数据。为什么要限制被除数和除数的最高位必须为零呢? 这是由计算机采用移位相减的运算方法决定的。被除数最高位为零, 是为防止左移时溢出而造成错误, 而除数最高位为零, 是为避免被除数无论左移几次, 其高 8 位总比除数小, 以致使商错误地为零。同时, 为了得到满足精度要求的商, 除数要大于被除数的高 8 位。

下面我们介绍一个被除数为 24 位, 除数为 16 位, 商为 16 位 (其中后 8 位为小数) 的多字节 2 进制除法程序。

将被除数 24 位按高低顺序依次存入 \$6、\$7、\$8 单元, 除数存入 \$1A、\$1B 单元, 结果放在 \$EA、\$EB 单元, 其中 \$EB 单元中的内容为小数, 而余数则舍去。程序见 P20。

0300-	D8	CLD	
0301-	A9 00	LDA	# \$ 00
0303-	85 EA	STA	\$ EA
0305-	85 EB	STA	\$ EB
0307-	A2 10	LDX	# \$ 10
0309-	06 08	ASL	\$ 08
030B-	26 07	ROL	\$ 07
030D-	26 06	ROL	\$ 06
030F-	06 EB	ASL	\$ EB
0311-	26 EA	ROL	\$ EA
0313-	38	SEC	
0314-	A5 07	LDA	\$ 07
0316-	E5 1B	SBC	\$ 1B
0318-	85 07	STA	\$ 07
031A-	A5 06	LDA	\$ 06

031C-	E5	1A	SBC	\$ 1A
031E-	85	06	STA	\$ 06
0320-	90	0F	BCC	\$ 0331
0322-	18		CLC	
0323-	A5	EB	LDA	\$ EB
0325-	69	01	ADC	井 \$ 01
0327-	85	EB	STA	\$ EB
0329-	90	02	BCC	\$ 032D
032B-	E6	1A	INC	\$ 1A
032D-	CA		DEX	
032E-	D0	D9	BNE	\$ 0309
0330-	00		BRK	
0331-	18		CLC	
0332-	A5	07	LDA	\$ 07
0334-	65	1B	ADC	\$ 1B
0336-	85	07	STA	\$ 07
0338-	A5	06	LDA	\$ 06
033A-	65	1A	ADC	\$ 1A
033C-	85	06	STA	\$ 06
033E-	4C	2D 03	JMP	\$ 032D

程序说明:

0300: 清10进制标志

0301-0306: 结果单元清零

0303

0307: 预置移位次数

0309-030E: 被除数左移一位

030F-0312: 商左移一位

0313: C标志置 1

0314-031F: 被除数减除数

0320: 判进位位为 0 转

0322: 为 1, 清C

0323-0328: 商加 1

0329: 判低位满否, 未满转

032B: 低位满高位加 1

032D: 计数一次

032E: 判计算完否, 未完循环, 继续运算

0330: 完, 结束

0331-033C: 进位位为 0 表示被

除数不够减除数,

因前面被除数已减

除数, 这里就要加

除数, 即恢复原来

的被除数, 然后转

\$ 32D 计数, 继续运算。

033E

上机实习:

设(06) = 74, (07) = 6B, (08) = 68, 除数(1A) = 78,
(1B) = 32, 则商(EA) = F7, (EB) = F5, F5为小数。

键入P20程序, 并输入被除数与除数, 单步运行:

* 06: 74 □ 6B □ 68 ✓

* 1A: 78 □ 32 ✓

* 300 S ✓

显示: 0300- D8 CLD

A = 00 X = 00 Y = 0D P = 30 S = DC

D置 0, 进行 2 进制运算。

继续运行: * SSS ✓

显示: 0301- A9 00 LDA # \$00

A = 00 X = 00 Y = 0D P = 32 S = DC

0303- 85 EA STA \$EA

A = 00 X = 00 Y = 0D P = 32 S = DC

0305- 85 EB STA \$EB

A = 00 X = 00 Y = 0D P = 32 S = DC

键入 *EA、EB↵

显示: 00EA- 00□00

可知商结果单元被清零。

继续运行: *S↵

显示: 0307- A2 10 LDX #10

A = 00 X = 10 Y = 0D P = 30 S = DC

移位次数放入X寄存器。

继续运行: *SSS↵

显示: 0309- 06 08 ASL \$08

A = 00 X = 10 Y = 0D P = B0 S = DC

030B- 26 07 ROL \$07

A = 00 X = 10 Y = 0D P = B0 S = DC

030D- 26 06 ROL \$06

A = 00 X = 10 Y = 0D P = B0 S = DC

键入 *6、8↵

显示: 0006- E8□D6□D0

可知被除数左移了一位。

继续运行: *SS↵

显示: 030F- 06 EB ASL \$EB

A = 00 X = 10 Y = 0D P = 32 S = DC

0311- 26 EA ROL \$EA

A = 00 X = 10 Y = 0D P = 32 S = DC

商左移一位。

继续运行: * SSSSSSS ✓

显示: 0313- 38 SEC

A = 00 X = 10 Y = 0D P = 33 S = DC

0314- A5 07 LDA \$07

A = D6 X = 10 Y = 0D P = B1 S = DC

0316- E5 1B SBC \$1B

A = A4 X = 10 Y = 0D P = B1 S = DC

0318- 85 07 STA \$07

A = A4 X = 10 Y = 0D P = B1 S = DC

031A- A5 06 LDA \$06

A = E8 X = 10 Y = 0D P = B1 S = DC

031C- E5 1A SBC \$1A

A = 70 X = 10 Y = 0D P = 71 S = DC

031E- 85 06 STA \$06

A = 70 X = 10 Y = 0D P = 71 S = DC

被除数已减去除数。

继续运行: * S ✓

显示: 0320- 90 0F BCC \$0331

A = 70 X = 10 Y = 0D P = 71 S = DC

由 P = 71 可知 C 为 1, 表示够减, 不转

继续运行: * SSSSS ✓

显示: 0322- 18 CLC

A = 70 X = 10 Y = 0D P = 70 S = DC

0323- A5 EB LDA \$EB

A = 00 X = 10 Y = 0D P = 72 S = DC

0325- 69 01 ADC #\$01

A = 01 X = 10 Y = 0D P = 30 S = DC

0327- 85 EB STA \$EB

A = 01 Y = 10 Y = 0D P = 30 S = DC

0329- 90 02 BCC \$032D

A = 01 X = 10 Y = 0D P = 30 S = DC

商加 1 并判有无进位, 无进位 (C = 0) 转 \$32D。

继续运行: *SS↙

显示: 032D- CA DEX

A = 01 X = 0F Y = 0D P = 30 S = DC

032E- D0 D9 BNE \$309

A = 01 X = 0F Y = 0D P = 30 S = DC

X - 1, 计数一次, 并判结果不为 0 (Z = 0), 转 \$309 继续计算。

重复计算过程略, 直至 X 减为 0 结束。

键入 *EA、EB↙显示: 00EA-F7 F5 可见结果正确。

(3) 10 进制除法程序 10 进制除法运算的原理与 2 进制基本相同, 和 10 进制乘法一样, 被除数和商的移位均要移 4 次才是一位。被除数减除数的次数也要重复数次直至不够减才行。我们举一个被除数为 4 位, 除数为 2 位, 结果为 4 位的例子来说明。

将被除数按高低位顺序存入 \$08、\$09 单元, 并置 \$06、\$07 单元为运算单元, 除数存入 \$1A 单元, 置 \$1B 单元为运算单元, 结果放入 \$EA、\$EB 单元, 余数舍去。程序见 P21。

0300-	F8	SED
0301-	A9 00	LDA # \$ 00
0303-	85 EA	STA \$ EA
0305-	85 EB	STA \$ EB
0307-	85 06	STA \$ 06

0309-	85	07	STA	\$ 07
030B-	85	1B	STA	\$ 1B
030D-	A2	04	LDX	# \$ 04
030F-	A0	04	LDY	# \$ 04
0311-	06	09	ASL	\$ 09
0313-	26	08	ROL	\$ 08
0315-	26	07	ROL	\$ 07
0317-	26	06	ROL	\$ 06
0319-	88		DEY	
031A-	D0	F5	BNE	\$ 0311
031C-	38		SEC	
031D-	A5	07	LDA	\$ 07
031F-	E5	1A	SBC	\$ 1A
0321-	85	07	STA	\$ 07
0323-	A5	06	LDA	\$ 06
0325-	E5	1B	SBC	\$ 1B
0327-	85	06	STA	\$ 06
0329-	90	11	BCC	\$ 033C
032B-	A5	EB	LDA	\$ EB
032D-	69	00	ADC	# \$ 00
032F-	85	EB	STA	\$ EB
0331-	90	06	BCC	\$ 0339
0333-	A5	EA	LDA	\$ EA
0335-	69	00	ADC	# \$ 00
0337-	85	EA	STA	\$ EA
0339-	4C	1C 03	JMP	\$ 031C
033C-	A5	07	LDA	\$ 07
033E-	65	1A	ADC	\$ 1A
0340-	85	07	STA	\$ 07

0342-	A5 06	LDA	\$ 06
0344-	65 1B	ADC	\$ 1B
0346-	85 06	STA	\$ 06
0348-	CA	DEX	
0349-	F0 0B	BEQ	\$ 0356
034B-	A0 04	LDY	# \$ 04
034D-	06 EB	ASL	\$ EB
034F-	26 EA	ROL	\$ EA
0351-	88	DEY	
0352-	D0 F9	BNE	\$ 034D
0354-	F0 B9	BEQ	\$ 030F
0356-	00	BRK	

程序说明:

0300: 置10进位运算标志

0301: 累加器清零

0303-0306: 结果单元清零

0307-30C: 运算单元清零

030D: 置移位次数

030F-031B: 被除数左移一位

031A: 判左移完否, 未完继续

031C-0328: 被除数减除数

0329: 判C = 0 不够减转

032B-0330: 够减商加 1

0331-0338: 判低位商加满否加满高位商加 1

0339: 转 \$ 31C重复减

033C-0347: 被除数加除数

0348: 计数一次

0349: 判计算完否, 完转

034B-0351: 未完, 商左移一位

0352: 判左移完否, 未完继续

0354: 移完转 \$ 30F 继续计算

0356: 结束。

上机实习:

设被除数(08)=96, (09)=25, 除数(1A)=55, 则结果(EA)应为01, (EB)应为75。

键入 P21 程序, 并输入被除数与除数, 并单步执行:

* 08: 96 □ 25 ↵

* 1A: 55 ↵

* 300: S ↵

显示: 0300-F8 SED

A = 00 X = 60 Y = 0D P = 38 S = B6 D置 1,

表示进行10进制运算。

继续运行, * S S S S S S ↵

显示: 0301- A9 00 LDA # \$ 00

A = 00 X = 60 Y = 0D P = 3A S = B6

0303- 85 EA STA \$ EA

A = 00 X = 60 Y = 0D P = 3A S = B6

0305- 85 EB STA \$ EB

A = 00 X = 60 Y = 0D P = 3A S = B6

0307- 85 06 STA \$ 06

A = 00 X = 60 Y = 0D P = 3A S = B6

0309- 85 07 STA \$ 07

A = 00 X = 60 Y = 0D P = 3A S = B6

030B- 85 1B STA \$ 1B

A = 00 X = 60 Y = 0D P = 3A S = B6

键入 * EA、EB ↵ 显示: 00EA- 00 □ 00, 键入 * 06、07 ↵

显示: 0006—00—00, 键入 * 1B↵ 显示: 001B—00 可见结果单元与运算单元均被清零。

继续运行: * S↵

显示: 030D— A2 04 LDX # \$ 04

A = 00 X = 04 Y = 0D P = 38 S = B6

移位次数置入 X 寄存器。

继续运行: * S S S S S S S S↵

显示: 030F— A0 04 LDY # \$ 04

A = 00 X = 04 Y = 04 P = 38 S = B6

0311— 06 09 ASL \$ 09

A = 00 X = 04 Y = 04 P = 38 S = B6

0313— 26 08 ROL \$ 08

A = 00 X = 04 Y = 04 P = 39 S = B6

0315— 26 07 ROL \$ 07

A = 00 X = 04 Y = 04 P = 38 S = B6

0317— 26 06 ROL \$ 06

A = 00 X = 04 Y = 04 P = 3A S = B6

0319— 88 DEY

A = 00 X = 04 Y = 03 P = 38 S = B6

031A— D0 F5 BNE \$ 0311

A = 00 X = 04 Y = 03 P = 38 S = B6

被除数左移一次, 转 \$ 311 继续移, 直至移完四次即一位。

键入 * 6.9↵ 显示: 0006—00—09—62—50 可知被除数左移了一位。

继续运行: * S S S S S S S S↵

显示: 031C— 38 SEC

A = 00 X = 04 Y = 00 P = 3B S = B6

```

031D- A5 07 LDA $07
A = 09 X = 04 Y = 00 P = 39 S = B6
031F- E5 1A SBC $1A
A = 54 X = 04 Y = 00 P = B8 S = B6
0321- 85 07 STA $07
A = 54 X = 04 Y = 00 P = B8 S = B6
0323- A5 06 LDA $06
A = 00 X = 04 Y = 00 P = 3A S = B6
0325- E5 1B SBC $1B
A = 99 X = 04 Y = 00 P = B8 S = B6
0327- 85 06 STA $06
A = 99 X = 04 Y = 00 P = B8 S = B6
0329- 90 11 BCC $033C
A = 99 X = 04 Y = 00 P = B8 S = B6
被除数减除数，并判C = 0，不够减，转$33C。

```

继续运行：* S S S S S S S ✓

```

显示，033C- A5 07 LDA $07
A = 54 X = 04 Y = 00 P = 38 S = B6
033E- 65 1A ADC $1A
A = 09 X = 04 Y = 00 P = F9 S = B6
0340- 85 07 STA $07
A = 09 X = 04 Y = 00 P = F9 S = B6
0342- A5 06 LDA $06
A = 99 X = 04 Y = 00 P = F9 S = B6
0344- 65 1B ADC $1B
A = 00 X = 04 Y = 00 P = B9 S = B6
0346- 85 06 STA $06

```

A = 00 X = 04 Y = 00 P = B9 S = B6

键入 * 6.9 ↵ 显示: * 0006-00 09 62 50 可见 被除数又恢复。

继续运行: * S S ↵

显示: 0348- CA DEX

A = 00 X = 03 Y = 00 P = 39 S = B6

0349- F0 0B BEQ \$0356

A = 00 X = 03 Y = 00 P = 39 S = B6

X - 1, 计数一次, 由 Z = 0 知结果不为 0, 不转。

继续运行: * S S S S S S ↵

显示: 034B- A0 04 LDY # \$04

A = 00 X = 03 Y = 04 P = 39 = \$B6

034D- 06 EB ASL \$EB

A = 00 X = 03 Y = 04 P = 3A S = B6

034F- 26 EA ROL \$EA

A = 00 X = 03 Y = 04 P = 3A S = B6

0351- 88 DEY

A = 00 X = 03 Y = 03 P = 38 S = B6

0352- D0 F9 BNE \$034D

A = 00 X = 03 Y = 03 P = 38 S = B6

商左移一次, 转 \$34D 继续移, 直至移完四次。商左移了一位。

继续运行: * S ↵

显示: 0354- F0 B9 BEQ \$030F

A = 00 X = 03 Y = 00 P = 3A S = B6

转 \$30F 继续运算。

继续运行: * S S S S S S S S ↵

显示: 030F- A0 04 LDY # \$04

```

A = 00  X = 03  Y = 04  P = 38  S = B6
      0311- 06 09      ASL  $09
A = 00  X = 03  Y = 04  P = B8  S = B6
      0313- 26 08      ROL  $08
A = 00  X = 03  Y = 04  P = B8  S = B6
      0315- 26 07      ROL  $07
A = 00  X = 03  Y = 04  P = 38  S = B6
      0317- 26 06      ROL  $06
A = 00  X = 03  Y = 04  P = 3A  S = B6
      0319- 88      DEY
A = 00  X = 03  Y = 03  P = 38  S = B6
      031A- D0 F5      BNE  $0311

```

A = 00 X = 03 Y = 03 P = 38 S = B6
 循环移四次使被除数继续左移一位。键入 * 6.9,
 显示: 0006-00┐96┐25┐00

可知被除数又左移了一位。

继续运行: * S S S S S S S S ✓

```

显示: 031C- 38      SEC
A = 00  X = 03  Y = 00  P = 3B  S = B6
      031D- A5 07      LDA  $07
A = 96  X = 03  Y = 00  P = B9  S = B6
      031F- E5 1A      SBC  $1A
A = 41  X = 03  Y = 00  P = 79  S = B6
      0321- 85 07      STA  $07
A = 41  X = 03  Y = 00  P = 79  S = B6
      0323- A5 06      LDA  $06
A = 00  X = 03  Y = 00  P = 7B  S = B6

```

```

0325- E5 1B      SBC  $1B
A = 00  X = 03  Y = 00  P = 3B  S = B6
0327- 85 06      STA  $06
A = 00  X = 03  Y = 00  P = 3B  S = B6
0329- 90 11      BCC  $033C
A = 00  X = 03  Y = 00  P = 3B  S = B6

```

被除数减除数，判 C = 1，够减，不转。

继续运行：* S S S S ↵

```

显示：032B- A5 EB      LDA  $EB
A = 00  X = 03  Y = 00  P = 3B  S = B6
032D- 69 00      ADC  # $00
A = 01  X = 03  Y = 00  P = 38  S = B6
032F- 85 EB      STA  $EB
A = 01  X = 03  Y = 00  P = 38  S = B6
0331- 90 06      BCC  $0339
A = 01  X = 03  Y = 00  P = 38  S = B6

```

商加 1，判低位未加满，转 \$ 339。

继续运行：* S ↵

```

显示：0339- 4C 1C 03      JMP  $031C
A = 01  X = 03  Y = 00  P = 38  S = B6

```

转 \$ 31C 重复被除数减除数。余下过程基本重复上述步骤，这里略去，直至 X 减为 0，计算结束。键入：* EA、EB ↵
显示：00EA-01↵75 可见结果正确。

(4) 简单的开方程序 用汇编语言编制求平方根的程序是较复杂的。这里我们介绍一个简单的方法，用这方法编制的程序实用性不很大，但作为一种方法对我们学习编程还是有用的。

我们知道任何正整数都可用下式表示:

$$N^2 = 1 + 3 + 5 + \dots + (2N - 1)$$

N个奇整数

我们可以把一个正整数连续减去 1, 3, 5 …… (2N - 1) 直到结果为 0 或不够减时为止。所减去奇整数的个数就是这个正整数的平方根。为简单起见, 设 N^2 不大于 255, 方根值也取整数。将正整数存放在 \$ 06 单元, 方根值存放在 \$ 1A 单元, 程序见 P22。

0300-	D8	CLD	
0301-	A9 00	LDA	# \$ 00
0303-	85 1A	STA	\$ 1A
0305-	A5 06	LDA	\$ 06
0307-	38	SEC	
0308-	E5 1A	SBC	\$ 1A
030A-	90 0A	BCC	\$ 0316
030C-	E6 1A	INC	\$ 1A
030E-	E5 1A	SBC	\$ 1A
0310-	F0 04	BEQ	\$ 0316
0312-	B0 F4	BCS	\$ 0308
0314-	C6 1A	DEC	\$ 1A
0316-	00	BRK	

程序说明:

0300: 清10进制标志位

0301: 累加器清零

0303: 结果单元清零

0305: 取正整数

0307: C置1

0308: 减奇整数

030A: 判 $C = 0$, 不够减, 转结束

030C: (1A) 又作为已减的个数

030E: 减奇整数

0310: 判结果为 0, 转结束

0312: 够减转 \$ 308 继续

0314: 不够减, 结果减 1

0316: 结束

上机实习:

设 (06) = 64, 则开方结果 (1A) 应为 0A。键入 P22, 并输入数据, 然后单步执行:

* 6: 64 ✓

* 300 S S S ✓

显示: 0300- D8 CLD

A = 00 X = 60 Y = 0D P = 30 S = D6

0301- A9 00 LDA # \$00

A = 00 X = 60 Y = 0D P = 32 S = D6

0303- 85 1A STA \$1A

A = 00 X = 60 Y = 0D P = 32 S = D6

D 置 0, (1A) 清零。

继续运行: * S S S S ✓

显示: 0305- A5 06 LDA \$06

A = 64 X = 60 Y = 0D P = 30 S = D6

0307- 38 SEC

A = 64 X = 60 Y = 0D P = 31 S = D6

0308- E5 1A SBC \$1A

A = 64 X = 60 Y = 0D P = 31 S = D6

030A- 90 0A BCC \$0316

A = 64 X = 60 Y = 0D P = 31 S = D6

取正整数64至累加器中，并减奇整数，因第一次减，(1A)为0，故够减，C = 1，不转。

继续运行：* S S S S S ↙

显示：030C- E6 1A INC \$1A

A = 64 X = 60 Y = 0D P = 31 S = D6

030E- E5 1A SBC \$1A

A = 63 X = 60 Y = 0D P = 31 S = D6

0310- F0 04 BEQ \$0316

A = 63 X = 60 Y = 0D P = 31 S = D6

0312- B0 F4 BCS \$0308

A = 63 X = 60 Y = 0D P = 31 S = D6

减去一个奇整数1，同时也记了一个，由于Z = 0，C = 1故转\$308继续。

继续运行：* S S S S S S S ↙

显示：0308- E5 1A SBC \$1A

A = 62 X = 60 Y = 0D P = 31 S = D6

030A- 90 0A BCC \$0316

A = 62 X = 60 Y = 0D P = 31 S = D6

030C- E6 1A INC \$1A

A = 62 X = 60 Y = 0D P = 31 S = D6

030E- E5 1A SBC \$1A

A = 60 X = 60 Y = 0D P = 31 S = D6

0310- F0 04 BEQ \$0316

A = 60 X = 60 Y = 0D P = 31 S = D6

0312- B0 F4 BCS \$308

A = 60 X = 60 Y = 0D P = 31 S = D6

两次减 (1A) 正好减去第二个奇整数 3, 同时记了 2 个。由于 $Z = 0$, $C = 1$ 转 \$ 308 继续, 不断重复上述步骤, 减 5, 减 7 …… 直至减为 0 结束。键入 * 1A ↵ 显示: 001A - 0A 可见结果正确。

5. 调用BASIC运算符程序

上面我们介绍了一些机器语言算术运算符程序。实际上, 在 BASIC 解释程序中, 也固化有可供调用的运算符程序, 包括标准函数和四则运算 (加、减、乘、除) 及乘方等。在这一节, 我们将介绍一下如何调用 BASIC 运算符程序。

(1) 数据存放形式 在 BASIC 中, 有整型数和实型数两种类型。整型数为两个字节, 以补码表示数值。实型数则采用浮点数形式表示数值。一个浮点数由两部分组成: 一部分称为尾数, 它表示数的有效部分, 一般规定为一个小于 1 的带符号的小数; 另一部分称为阶码, 它只能是一个带符号的整数, 阶码指示尾数中的小数点应当向左或者向右移动的位数。这样, 随着阶码数值的变化, 我们可以很方便地“浮动”一个数中小数点的位置。在浮点数中, 尾数的符号称为数符, 阶码的符号称为阶符。

在中华机中, 实型数是采用浮点规格化的形式分五个字节存放的, 其中尾数 4 个字节, 阶码 1 个字节。阶码用偏移码表示, 它把 128 加到实际指数上, 这样阶码的正负就不需要用符号位来表示了。偏移阶码从 1—255, 表示的实际指数为 -127 到 127, 偏移阶码 0 (实际指数是 -128) 表示浮点数零。尾数用原码表示。浮点数规格化后, 小数的最高位必定是 1, 所以此位不存入, 而当进行算术运算时, 计算机

会自动补上这一位。例如实型数据 5 的 2 进制浮点数表示为 0.101×2^{11} ，其实际指数为 3(11)，加上 128 (10000000)，偏移码为 131 (\$83)，尾数 10100000，其中最高位 1 不存入，故最高字节为 00100000 (\$20)，其余 3 字节均为 0。所以实型数据 5 表示为浮点规格化存贮形式：\$83 \$20 \$00 \$00 \$00。

运算子程序采用了浮点累加器 (FAC) 和浮点暂存器 (ARG) 来进行运算。浮点累加器所用的地址为 0 页区 \$9D—\$A3，共 7 个字节，见图 1。其中前 5 个字节为浮点数形式，第 6 个字节为数符，\$0 表示正数，\$FF 表示负数，第 7 个字节固定为 0。要注意的是，浮点累加器是直接

阶 码	\$ 9D
尾数 1	\$ 9E
尾数 2	\$ 9F
尾数 3	\$ A0
尾数 4	\$ A1
数 符	\$ A2
0 0	\$ A3

图12.1

用于运算的，所以浮点数送入浮点累加器后尾数 1 最高位已自动加 1。因此，实型数据 5 在浮点累加器中就成了 \$83、\$A0、\$00、\$00、\$00、\$00、\$00。

浮点暂存器 (ARG) 所用地址为 0 页区 \$A5—\$AB，也是 7 个字节。存放格式与浮点累加器相同。

另外，为了便于存放计算过程中的中间计算结果，运算

子程序还用到三个数据临时存放区，每个区有 5 个字节，它们是：

临时存放区 1 (TEMP1) : \$ 93—\$ 97

临时存放区 2 (TEMP2) : \$ 98—\$ 9C

临时存放区 3 (TEMP3) : \$ 8A—\$ 8E

这三个区都是以浮点规格化形式存放数据的。

(2) 数据转换、传送子程序 由于运算符程序是采用浮点数运算的，因此对于整型数来说，必须先转换成浮点数后才能参加运算，同样，运算后也可以转换为整数并存放在指定单元中。

下面的子程序可以实现这样的转换。

① 名称 SNGFLT 入口地址 \$E301 其功能为将 Y 寄存器中的不带正负号的整数转换成浮点数并存放在 FAC 中。

例 1：运行下列程序

```
0300-   A0   FF           LDY    # $FF
0302-   20   01   E3     JSR     $E301
0305-   00                BRK
```

键入 *9D·A3↵

显示 009D·88□FF□00□00□00□00□00

可知整数 \$FF (255) 已被转换为浮点数形式。

② 名称 FLOAT 入口地址 \$EB93 其功能为将 A 累加器中的带正负号的整数转换成浮点数并放入 FAC 中。

例 2：运行下列程序

```
0300-   A9   8A           LDA    # $8A
0302-   20   93   EB     JSR     $EB93
```

键入 * 9D·A3↵

显示 009 D-87┐EC┐00┐00┐00┐FF┐00

可知负整数 \$ 8A (-118) 已被转换为浮点数形式。

③ 名称 GIVAYF; 入口地址 \$ E2F2, 其功能为将 A、Y 中的带正负号的整数转换成浮点数并放入 FAC 中。

例 3: 运行下列程序

```
0300-    A9    7F          LDA    # $7F
0302-    A0    FF          LDY    # $FF
0304-    20    F2 E2      JSR     $E2F2
0307-    00              BRK
```

键入 * 9D·A3↵

显示 009D-8F┐FF┐FE┐00┐00┐00┐00

可知正整数 \$ 7FFF (32767) 已被转换为浮点数。同样, 将 \$ 301 内容改为 8F 并运行, 键入 * 9D、A3↵

显示 009D-8F┐E0┐02┐00┐00┐FF┐00

可知负整数 \$ 8FFF (-28673) 已被转换为浮点数。

④ 名称 CONINT 入口地址 \$ E6FB 其功能为将 FAC 中的内容转换成一个字节的值放入 X 及 \$ A1 中。

例 4: 运行下列程序, 假设 FAC 中已有 8B┐0F┐00┐00┐00┐00┐00。

```
0300-    20    FB E6      JSR     $E6FB
0303-    00              BRK
```

键入 * A1↵

显示 00A1- 0F

可知已转换过来。调用此子程序时应注意 FAC 中的内容其数值不能大于 1 个字节所能表达的数 (即不能大于 \$ FF) ,

否则会出错。

⑤ 名称 AYZNT 入口地址 \$E10C 其功能为将 FAC 中的内容转换成两字节的整数并放入 \$A0 和 \$A1 中。

例 5：设 FAC 中已有 8F□E0□02□00□00□FF□00，运行下列程序：

```
0300-    20    0C    E1        JSR    $E10C
0303-    00                                BRK
```

键入 *A0·A1↵

显示：00A0- 8F FF

可知已转换。

需要说明的是，上述 5 个转换子程序所转换的浮点数不是规格化的浮点数，我们称其为 FAC 形式（包括 ARG）。这在数据传送子程序中可以看得更清楚。

下面的子程序可以实现数据传送。

⑥ 名称 MOVFM 入口地址 \$EAF9 其功能为将寄存器 Y、A 指定地址的浮点数据（5 个字节）传送给浮点累加器 FAC。

例 6：在 008A 开始的单元中有浮点数据 87□6A□00□00□00，运行下列程序：

```
0300-    A0    00                LDY    # $00
0302-    A9    8A                LDA    # $8A
0304-    20    F9    EA        JSR    $EAF9
0307-    00                                BRK
```

键入 *9D·A3↵

显示：009D-87□EA□00□00□00□6A□00，

可见浮点数据已传送到 FAC 中。浮点数据的尾数最高位 1 由于规格化的原因被舍去，成为 \$6A，而 FAC 形式的尾数最

高位则没有规格化而保留，成为\$EA，数符中存入了\$6A是由于调用传送程序时放入的，不影响到数据符号。

⑦ 名称CONUPK 入口地址\$E9E3 其功能为将寄存器Y、A指定地址的浮点数据传送给浮点暂存器ARG。调用方法与上相同，不再举例。

⑧ 名称MOVFA 入口地址\$EB53 其功能为将浮点暂存器ARG的内容传送到浮点累加器FAC中。

⑨ 名称MOVAF 入口地址\$EB63，其功能为将浮点累加器FAC的内容传送到浮点暂存器ARG中。

⑩ 名称MOVMF 入口地址\$EB2B 其功能为将浮点累加器FAC中的浮点数据传送到由寄存器Y、X指定地址的单元中。

⑪ 名称MOVML 入口地址\$EB23 其功能为将浮点累加器FAC的内容传送到由X指定的0页单元中。

例7：FAC中有87┐AA┐00┐00┐00┐00┐00。运行下列程序。

```
0300-   A2  8A           LDX    # $8A
0302-   20  23  EB      ISR    $EB23
0305-   00              BRK
```

键入 * 8A · 8E↵

显示：008A-87┐2A┐00┐00┐00，

可知数据已传送至由X指定的008A开始的单元中。

⑫ 名称MOVIF 入口地址\$EB21 其功能为将浮点累加器FAC的内容传送到0页暂时存放区1 (TEMP1)；\$93—\$97

⑬ 名称MOV2F 入口地址\$EB1E 其功能为将浮点累加器FAC的内容传送到0页暂时存放区2 (TEMP2)；

\$ 98—\$ 9C

在调用运算符程序时，还要用到一些比较、判别正负和输出子程序，下面也一起作个介绍。

⑭ 名称 FCOMP 入口地址 \$ EBB2 其功能为将寄存器 Y、A 所指定地址的浮点数据与 FAC 中的浮点数据相比较，若 $(Y, A) < FAC$, $A = \$ 01$ ；若 $(Y, A) = FAC$, $A = \$ 00$ ；若 $(Y, A) > FAC$, $A = \$ FF$ 。

例 8：设 008A—008E 中有浮点数据 87□2A□00□00□00（浮点数 \$ 55），FAC 中有浮点数据 87□EA□00□00□00□00□00，（浮点数 \$ 75）运行下列程序：

```
0300-    A0  00          LDY    # $ 00
0302-    A9  8A          LDA    # $ 8A
0304-    20  B2  EB      JSR     $ EBB2
0307-    00              BRK
```

根据 $A = \$ 01$ 可知 $FAC > (Y, A)$ 。将上两个数据互换一下，（注意 FAC 与规格化浮点数的区别）再运行，A 应为 \$ FF。

⑮ 名称 SIGN 入口地址 \$ EB82 其功能为根据 FAC 之值来设定 A，若 FAC 为正， $A = \$ 01$ ，若 FAC 为 0， $A = \$ 00$ ，若 FAC 为负， $A = \$ FF$ 。

例 9：将负数 8F (-113) 传送至 FAC，运行下列程序，可看到 $A = \$ FF$ 。

```
0300-    A9  8F          LDA    # $ 8F
0302-    20  93  EB      JSR     $ EB93
0305-    20  82  EB      JSR     $ EB82
0308-    00              BRK
```

⑯ 名称 FOUT 入口地址 \$ ED34 其功能为根据

FAC 之值产生一个字符串放在内部缓冲区中，调用后 Y、A 指向该字符串的首址，字符串以 0 作为结束标志。

例10：运行下列程序：

```
0300-   A9  7F           LDA    # $7F
0302-   20  93  EB      JSR     $EB93
0305-   20  34  ED      JSR     $ED34
0308-   00              BRK
```

可知 A = 00，Y = 01，因此字符串放在 100 开始的单元中，以 0 结束。键入 * 100、107 ↵

显示：0100-31┐32┐37┐00┐30┐30┐30┐30。

\$ 7F 的 10 进制数值为 127，其 ASCII 码正好为 31┐32┐37。

⑰ 名称 STROUT 入口地址 \$DB3A 其功能为将由 Y、A 所指定的字符串印出，字符串必须以 0 或单引号作结束。此子程序与上一个子程序联用正好可将 FAC 的值印出在屏幕上。

例11：用例10与本子程序联用：

```
0300-   A9  7F           LDA    # $7F
0302-   20  93  EB      JSR     $EB93
0305-   20  34  ED      JSR     $ED34
0308-   20  3A  DB      JSR     $DB3A
030B-   00              BRK
```

可看到屏幕上输出 127。

⑱ 名称 OUT 入口地址 \$ED24 其功能为将 A、X 寄存器中的 4 位 16 进制数转换为 10 进制数后在屏幕上输出。

例12：运行下列程序

```
0300-   A9  0E           LDA    # $0E
0302-   A2  0A           LDX    # $0A
0304-   20  24  ED      JSR     $ED24
```


可看到屏幕输出3594, 即 \$0E0A 转换为 10进制数3594。

(3) 四则运算子程序 下面再介绍四则运算子程序, 这些子程序能完成浮点累加器(FAC)和浮点暂存器(ARG)的加、减、乘、除运算, 其运算结果仍放在浮点累加器FAC中。

- ① 名称FADDT 入口地址\$E7C1 加法
- ② 名称FSUBT 入口地址\$E7AA 减法
- ③ 名称FMULTT 入口地址\$E982 乘法
- ④ 名称FDIVT 入口地址\$EA69 除法

除法(减法)子程序中 ARG 是被除(减)数, 其他两个子程序中FAC是被乘数、被加数。

例13: 计算 $100 + 255$

程序P23如下:

0300-	A0	FF	LDY	# \$FF
0302-	20	01 E3	JSR	\$E301
0305-	20	63 EB	JSR	\$EB63
0308-	A9	64	LDA	# \$64
030A-	20	93 EB	JSR	\$EB93
030D-	20	C1 E7	JSR	\$E7C1
0310-	20	34 ED	JSR	\$ED34
0313-	20	3A DB	JSR	\$DB3A
0316-	00		BRK	

程序说明:

300: 取255至 Y

302: 转换至FAC

305: FAC→ARG

308: 取100至 A

30A, 转换至FAC
 30D, FAC + ARG
 310, 转换为字符串
 313, 输出
 316, 中断

运行后可看到屏幕输出355。

其他运算可参照本程序进行。这里我们再举一个综合运算的例子。

例14: 计算 $(150 + 200 - 50) \times 5 \div 15$

程序P24如下:

0300-	A0	C8		LDY	#\$C8
0302-	20	01	E3	JSR	\$E301
0305-	20	63	EB	JSR	\$EB63
0308-	A0	96		LDY	#\$96
030A-	20	01	E3	JSR	\$E301
030D-	20	C1	E7	JSR	\$E7C1
0310-	20	63	EB	JSR	\$EB63
0313-	A9	32		LDA	#\$32
0315-	20	93	EB	JSR	\$EB93
0318-	20	AA	E7	JSR	\$E7AA
031B-	20	21	EB	JSR	\$EB21
031E-	A0	05		LDY	#\$05
0320-	20	01	E3	JSR	\$E301
0323-	A9	93		LDA	#\$93
0325-	A0	00		LDY	#\$00
0327-	20	E3	E9	JSR	\$E9E3
032A-	20	82	E9	JSR	\$E982
032D-	20	63	EB	JSR	\$EB63
0330-	A0	0F		LDY	#\$0F

0332-	20	01	E3	JSR	\$E301
0335-	20	69	EA	JSR	\$EA69
0338-	20	34	ED	JSR	\$ED34
033B-	20	3A	DB	JSR	\$DB3A
033E-	00			BRK	

程序说明:

300: 取200至Y
 302: 转换至FAC
 305: FAC→ARG
 308: 取150至Y
 30A: 转换至FAC
 30D: FAC+ARG→FAC
 310: FAC→ARG
 313: 取50至A
 315: 转换至FAC
 318: ARG-FAC→FAC
 31B: FAC→TEMP1
 31E: 取5至Y
 320: 转换至FAC
 323: 取TEMP1指针
 325: 取TEMP1指针
 327: (Y, A)→ARG
 32A: FAC*ARG→FAC
 32D: FAC→ARG
 330: 取15至Y
 332: 转换至FAC
 335: ARG/FAC→FAC
 338: 转换成字符串
 33B: 输出结果

33E: 中断, 结束
运行, * 300G ↙ 显示100。

除上述 4 个四则运算子程序外, 还有 4 个传送数据后再进行四则运算的子程序。实质上这 4 个子程序只比上述 4 个子程序各多一条传送指令 JSR \$E9E3。它们能将寄存器 Y、A 指定地址的浮点数据 (5 个字节) 传送给浮点暂存器, 然后再分别调用加、减、乘、除子程序。

⑤ 名称 FADD 入口地址 \$ E7BE (Y、A) → ARG调加法

⑥ 名称 FSUB 入口地址 \$ E7A7 (Y、A) → ARG调减法

⑦ 名称 FMULT 入口地址 \$ E97F (Y、A) → ARG调乘法

⑧ 名称 FDIV 入口地址 \$ EA66 (Y、A) → ARG调除法

例15: 计算 $32500 \div 650 + 15000$

程序P25如下:

0300-	A9	7E		LDA	#\$7E
0302-	A0	F4		LDY	#\$F4
0304-	20	F2	E2	JSR	\$E2F2
0307-	20	21	EB	JSR	\$EB21
030A-	A9	02		LDA	#\$02
030C-	A0	8A		LDY	#\$8A
030E-	20	F2	E2	JSR	\$E2F2
0311-	A9	93		LDA	#\$93
0313-	A0	00		LDY	#\$00
0315-	20	66	EA	JSR	\$EA66
0318-	20	1E	EB	JSR	\$EB1E

031 B-	A9	3A	LDA	# \$ 3A
031 D-	A0	98	LDY	# \$ 98
031 F-	20	F2 E2	JSR	\$ E2F2
0322-	A9	98	LDA	# \$ 98
0324-	A0	00	LDY	# \$ 00
0326-	20	BE E7	JSR	\$ E7BE
0329-	20	34 ED	JSR	\$ ED34
032 C-	20	3A DB	JSR	\$ DB3A
032 F-	00		BRK	

程序说明:

300: 取32500至

A、Y寄存器

304: 转换至FAC

307: FAC→TEMP1

30 A: 取650至

A、Y寄存器

30 E: 转换至FAC

311: 取TEMP1指针

315: (Y、A)→ARG, ARG/FAC→FAC

318: FAC→TEMP2

31 B: 取15000

31 D: 至 A、Y寄存器

31 F: 转换至FAC

322: 取TEMP2指针

326: (Y、A)→ARG, FAC+ARG→FAC

329: 转换成字符串

32 C: 输出结果

32 F: 中断

运行: *300G↙显示结果: 15050

(4) 标准函数子程序 除了四则运算符程序外, BASZC解释程序中还有12个标准函数子程序, 这些函数都是单一参数的, 调用前, 必须将自变量参数存放在该点累加器FAC中, 调用后的计算结果也还存放在FAC中。

下面我们逐个介绍:

① 指数函数EXP 入口地址为\$EF09 函数EXP(X)是用来求 e^x 的, $e = 2.71828\cdots$ 。

例1: 计算 e 的10次方, 即 $Y = e^{10}$, 求Y?

程序如下: (P26)

```
0300-    A9    0A            LDA    # $0A
0302-    20    93    EB      JSR    $EB93
0305-    20    09    EF      JSR    $EF09
0308-    20    34    ED      JSR    $ED34
030B-    20    3A    DB      JSR    $DB3A
030E-    00                    BRK
```

程序说明:

300: 取10至A

302: 转换至FAC

305: 计算EXP

308: 转换为字符串

30B: 输出结果

30E: 结束

运行后显示结果: 22026.4658。

② 对数函数LOG 入口地址为\$E941 对数函数LOG(X)是求X的以 e 为底的自然对数的。

例2: 计算 $Y = \text{LOG}(96)$

程序P27如下:

```
0300-    A9    60            LDA    # $60
```

0302-	20	93	EB	JSR	\$EB93
0305-	20	41	E9	JSR	\$E941
0308-	20	34	ED	JSR	\$ED34
030B-	20	3A	DB	JSR	\$DB3A
030E-	00			BRK	

程序说明:

300: 取96至A

302: 转换至FAC

305: 计算LOG

308: 转换为字符串

30B: 输出结果

30E: 结束

运行后显示结果: 4.56434819

LOG(X)是求自然对数的, 如果要求以10为底的常用对数的话, 可用换底公式, 我们举一例说明。

例3: 计算 $Y = \lg 10000$

程序P28如下:

0300-	A9	27		LDA	# \$27
0302-	A0	10		LDY	# \$10
0304-	20	F2	E2	JSR	\$E2F2
0307-	20	41	E9	JSR	\$E941
030A-	A2	FA		LDX	# \$EA
030C-	20	23	EB	JSR	\$EB23
030F-	A9	0A		LDA	# \$0A
0311-	20	93	EB	JSR	\$EB93
0314-	20	41	E9	JSR	\$E941
0317-	A9	EA		LDA	# \$EA
0319-	A0	00		LDY	# \$00
031B-	20	66	EA	JSR	\$EA66

031E-	20	34	ED	JSR	\$ED34
0321-	20	3A	DB	JSR	\$DB3A
0324-	00			BRK	

程序说明:

A、Y寄存器

300- 取10000至

304: 转换至FAC

307: 计算LOG10000

30A: 指定0页EA为首地址的单元

30C: FAC→(0,X)

30F: 取10至A

311: 转换至FAC

314: 计算LOG10

317- 设定地址

319: 至A、Y

31B: (Y,A)→ARG, ARG/FAC→FAC

31E: 转换为字符串

321: 输出结果

324: 结束

运行后显示结果: 4。

③ 正弦函数SIN 入口地址为\$EFF1 在计算正弦函数值时, 自变量须以弧度为单位。

例4: 计算 $Y = \sin \pi/6$

程序P29如下:

0300-	A9	82	LDA	# \$82
0302-	85	EA	STA	\$EA
0304-	A9	49	LDA	# \$49
0306-	85	EB	STA	\$EB
0308-	A9	0F	LDA	# \$0F

030A-	85	EC	STA	\$EC
030C-	A9	DA	LDA	#\$DA
030E-	85	ED	STA	\$ED
0310-	A9	A2	LDA	#\$A2
0312-	85	EE	STA	\$EE
0314-	A9	06	LDA	#\$06
0316-	20	93	JSR	\$EB93
0319-	A9	EA	LDA	#\$EA
031B-	A0	00	LDY	#\$00
031D-	20	66	JSR	\$EA66
0320-	20	F1	JSR	\$EFF1
0323-	20	34	JSR	\$ED34
0326-	20	3A	JSR	\$DB3A
0329-	00		BRK	

程序说明:

300-313: 取 π 的浮点
规格化数据
分别存放在
\$EA—\$EE
五个单元中

314: 取6至A

316: 转换至FAC

319-31C: 取地址指
针至Y、A

31D: (Y、A)→ARG, ARG/FAC→FAC

320: 计算 $\sin \pi/6$

323: 转换成字符串

326: 输出结果

329: 结束

运行程序，显示结果：0.5。

④ 余弦函数COS 入口地址为\$EFEA

例5：计算 $Y = \cos \pi/6$

将P29程序\$320调用SIN函数改为调用COS函数即可。程序P30如下：

0300-	A9	82	LDA	#\$82	
0302-	85	EA	STA	\$EA	
0304-	A9	49	LDA	#\$49	
0306-	85	EB	STA	\$EB	
0308-	A9	0F	LDA	#\$0F	
030A-	85	EC	STA	\$EC	
030C-	A9	DA	LDA	#\$DA	
030E-	85	ED	STA	\$ED	
0310-	A9	A2	LDA	#\$A2	
0312-	85	EE	STA	\$EE	
0314-	A9	06	LDA	#\$06	
0316-	20	93	EB	JSR	\$EB93
0319-	A9	EA	LDA	#\$EA	
031B-	A0	00	LDY	#\$00	
031D-	20	66	EA	JSR	\$EA66
0320-	20	EA	EF	JSR	\$EFEA
0323-	20	34	ED	JSR	\$ED34
0326-	20	3A	DB	JSR	\$DB3A
0329-	00		BRK		

运行程序，显示结果0.866025404

⑤ 正切函数TAN 入口地址为\$F03A

例6：计算 $Y = \tan \pi/4$

将P30程序修改一下，即可调用，见P31。

0300-	A9	82	LDA	#\$82
-------	----	----	-----	-------

0302-	85	EA		STA	\$EA
0304-	A9	49		LDA	#\$49
0306-	85	EB		STA	\$EB
0308-	A9	0F		LDA	#\$0F
030A-	85	EC		STA	\$EC
030C-	A9	DA		LDA	#\$DA
030E-	85	ED		STA	\$ED
0310-	A9	A2		LDA	#\$A2
0312-	85	EE		STA	\$EE
0314-	A9	04		LDA	#\$04
0316-	20	93	EB	JSR	\$EB93
0319-	A9	EA		LDA	#\$EA
031B-	A0	00		LDY	#\$00
031D-	20	66	EA	JSR	\$EA66
0320-	20	3A	FO	JSR	\$F03A
0323-	20	34	ED	JSR	\$ED34
0326-	20	3A	DB	JSR	\$DB3A
0329-	00			BRK	

运行后显示结果：01。

⑥ 反正切函数ATN，入口地址为\$F09E。

例7：计算 $Y = \text{ATN}1$

程序P32如下：

0300-	A9	01		LDA	#\$01
0302-	20	93	EB	JSR	\$EB93
0305-	20	9E	F0	JSR	\$F09E
0308-	20	34	ED	JSR	\$ED34
030B-	20	3A	DB	JSR	\$DB3A
030E-	00			BRK	

程序说明：

300: 取 1 至 A
 302: 转换至FAC
 305: 计算ATNI
 308: 转换为字符串
 30 B: 输出结果

运行后显示结果: 0.785398163

⑦ 开平方函数SQR 入口地址为\$EE8D

例 8: 计算 $Y = \sqrt{32400}$

程序P33如下:

0300-	A9	7E	LDA	# \$7E
0302-	A0	90	LDY	# \$90
0304-	20	F2 E2	JSR	\$E2F2
0307-	20	8D EE	JSR	\$EE8D
030A-	20	34 ED	JSR	\$ED34
030D-	20	3A DB	JSR	\$DB3A
0310-	00		BRK	

程序说明:

300-303: 取32400送至

A、Y

304: 转换至FAC

307: 开方

30 A: 转换成字符串

30 D: 输出结果

310: 结束

运行后显示结果: 180

⑧ 乘方 入口地址为\$EE97 调用乘方子程序时, 底数放入ARG中, 而幂则送入FAC, 计算后的结果仍在FAC中。

例 9：计算 $Y = 10^4$

程序P34如下：

0300-	A9	0A		LDA	#\$0A
0302-	20	93	EB	JSR	\$EB93
0305-	20	63	EB	JSR	\$EB63
0308-	A9	04		LDA	#\$04
030A-	20	93	EB	JSR	\$EB93
030D-	20	97	EE	JSR	\$EE97
0310-	20	34	ED	JSR	\$ED34
0313-	20	3A	DB	JSR	\$DB3A
0316-	00			BRK	

程序说明：

300：取底数

302：转换至FAC

305：FAC→ARG

308：取幂

30A：转换至FAC

30D：计算 10^4

310：转换为字符串

313：输出结果

316：结束

运行程序后，显示结果：10000。

⑨ 求绝对值ABS 入口地址为\$EBAF

例10：求 (-103) 的绝对值

程序P35如下：

0300-	A9	99		LDA	#\$99
0302-	20	93	EB	JSR	\$EB93
0305-	20	AF	EB	JSR	\$EBAF
0308-	20	34	ED	JSR	\$ED34

```

030B-    20 3A DB    JSR    $DB3A
030E-    00          BRK

```

程序说明:

```

300: 取 (-103)
302: 转换至FAC
305: 求绝对值
308: 转换成字符串
30B: 输出结果
30E: 结束

```

运行程序, 显示结果: 103。

⑩ 取整函数INT 入口地址为\$EC23

例11: 计算 $Y = \text{INT}(3.141592654)$

将 π 的浮点数据放入 0 页 EA—EE 单元, 计算程序 P36 如下:

```

0300-    A9 EA          LDA    # $EA
0302-    A0 00          LDY    # $00
0304-    20 F9 EA      JSR    $EAF9
0307-    20 23 EC      JSR    $EC23
030A-    20 34 ED      JSR    $ED34
030D-    20 3A DB      JSR    $DB3A
0310-    00          BRK

```

程序说明:

```

300-303: 取地址指针
          放入 Y、A
304: 转换至FAC
307: 取整
30A: 转换为字符串
30D: 输出结果
310: 结束

```

运行程序后显示结果： 3 。

⑪ 符号函数SGN 入口地址为\$EB90 当函数值 $X > 0$ 时, $SGN(X) = 1$; $X = 0$ 时, $SGN(X) = 0$; $X < 0$ 时, $SGN(X) = -1$ 。

例12: 求100的符号。

程序P37如下:

0300-	A9	64		LDA	# \$64
0302-	20	93	EB	JSR	\$EB93
0305-	20	90	EB	JSR	\$EB90
0308-	20	34	ED	JSR	\$ED34
030B-	20	3A	DB	JSR	\$DB3A
030E-	00			BRK	

程序说明:

300: 取100至A

302: 转换至FAC

305: 求SGN (100)

308: 转换为字符串

30B: 输出结果

30E: 结果

运行程序后显示结果: 1。可知符号为正。将\$301改为F2 (-14), 运行程序后显示结果为-1, 可知符号为负。

⑫ 随机函数RND 入口地址为\$EFAE 运行程序P38, 每次运行后结果都会不同:

0300-	20	AE	EF	JSR	\$EFAE
0303-	20	34	ED	JSR	\$ED34
0306-	20	3A	DB	JSR	\$DB3A
0309-	00			BRK	

程序说明:

300: 求RND

303: 转换为字符串

306: 输出结果

309: 结束

(5) 综合举例

例 1: 计算 $\frac{52 \times 63 + 88}{47 \times 3 - 18}$

程序P39如下:

0300-	A0	34		LDY	# \$ 34
0302-	20	01	E 3	JSR	\$ E301
0305-	20	63	EB	JSR	\$ EB63
0308-	A0	3F		LDY	# \$ 3F
030A-	20	01	E 3	JSR	\$ E301
030D-	20	82	E 9	JSR	\$ E982
0310-	20	63	EB	JSR	\$ EB63
0313-	A0	58		LDY	# \$ 58
0315-	20	01	E 3	JSR	\$ E301
0318-	20	C1	E 7	JSR	\$ E7C1
031B-	20	21	EB	JSR	\$ EB21
031E-	A0	2F		LDY	# \$ 2F
0320-	20	01	E 3	JSR	\$ E301
0323-	20	63	EB	JSR	\$ EB63
0326-	A0	03		LDY	# \$ 03
0328-	20	01	E 3	JSR	\$ E301
032B-	20	82	E 9	JSR	\$ E982
032E-	20	63	EB	JSR	\$ EB63
0331-	A0	12		LDY	# \$ 12
0333-	20	01	E 3	JSR	\$ E301

0336-	20	AA	E7	JSR	\$E7AA
0339-	A9	93		LDA	# \$93
033B-	A0	00		LDY	# \$00
033D-	20	66	EA	JSR	\$EA66
0340-	20	34	ED	JSR	\$ED34
0343-	20	3A	DB	JSR	\$DB3A
0346-	00			BRK	

程序说明:

300: 取52至Y
 302: 转换至FAC
 305: FAC→ARG
 308: 取63至Y
 30A: 转换至FAC
 30D: FAC*ARG→FAC
 310: FAC→ARG
 313: 取58至Y
 315: 转换至FAC
 318: +88
 31B: FAC→TEMP1
 31E: 取47至Y
 320: 转换至FAC
 323: FAC→ARG
 326: 取3至Y
 328: 转换至FAC
 32B: FAC*ARG→FAC
 32E: FAC→ARG
 331: 取18至Y
 333: 转换至FAC
 336: ARG—FAC→FAC

339-33C, 取TEMP1

指针

33D: 计算整个结果

340: 转换成字符串

343: 输出结果

346: 结束

运行后得到结果: 27.3495935。

例 2: 计算 $\frac{6 \times 2 \times 10^2}{5^4}$

程序P40如下:

0300-	A9	06		LDA	# \$ 06
0302-	20	93	EB	JSR	\$ EB93
0305-	20	63	EB	JSR	\$ EB63
0308-	A9	02		LDA	# \$ 02
030A-	20	93	EB	JSR	\$ EB93
030D-	20	82	E9	JSR	\$ E982
0310-	20	63	EB	JSR	\$ EB63
0313-	A9	0A		LDA	# \$ 0A
0315-	20	93	EB	JSR	\$ EB93
0318-	20	82	E9	JSR	\$ E982
031B-	20	63	EB	JSR	\$ EB63
031E-	A9	0A		LDA	# \$ 0A
0320-	20	93	EB	JSR	\$ EB93
0323-	20	82	E9	JSR	\$ E982
0326-	A2	EA		LDX	# \$ EA
0328-	20	23	EB	JSR	\$ EB23
032B-	A9	05		LDA	# \$ 05
032D-	20	93	EB	JSR	\$ EB93
0330-	20	63	EB	JSR	\$ EB63

0333-	A9	04		LDA	# \$04
0335-	20	93	EB	JSR	\$EB93
0338-	20	97	EE	JSR	\$EE97
033B-	A9	EA		LDA	# \$EA
033D-	A0	00		LDY	# \$00
033F-	20	66	EA	JSR	\$EA66
0342-	20	34	ED	JSR	\$ED34
0345-	20	3A	DB	JSR	\$DB3A
0348-	00			BRK	

程序说明:

300: 取 6 至 A
 302: 转换至FAC
 305: FAC→ARG
 308: 取 2 至 A
 30A: 转换至FAC
 30D: 计算 6×2
 310: FAC→ARG
 313: 取10至 A
 315: 转换至FAC
 318: 12×10
 31B: FAC→ARG
 31E: 取10至 A
 320: 转换至FAC
 323: 120×10
 326: 取 0 页单元指针
 328: 送往 0 页单元存贮
 32B: 取 5 至 A
 32D: 转换至FAC
 330: FAC→ARG

333: 取 4 至 A
 335: 转换至FAC
 338: 计算 5^4
 33B-33E: 取 0 页指针
 33F-33E: 计算整个式子
 342: 转换为字符串
 345: 输出结果
 348: 结束

运行后显示结果: 1.92。

例 3: 计算 $\sec 30^\circ + \csc 30^\circ$

程序P41如下:

0300-	A9	01	LDA	# \$ 01
0302-	20	93	JSR	\$ EB93
0305-	A2	1A	LDX	# \$ 1A
0307-	20	23	JSR	\$ EB23
030A-	A9	82	LDA	# \$ 82
030C-	85	EA	STA	\$ EA
030E-	A9	49	LDA	# \$ 49
0310-	85	EB	STA	\$ EB
0312-	A9	0F	LDA	# \$ 0F
0314-	85	EC	STA	\$ EC
0316-	A9	DA	LDA	# \$ DA
0318-	85	ED	STA	\$ ED
031A-	A9	A2	LDA	# \$ A2
031C-	85	EE	STA	\$ EE
031E-	A9	06	LDA	# \$ 06
0320-	20	93	JSR	\$ EB93
0323-	A9	EA	LDA	# \$ EA
0325-	A0	00	LDY	# \$ 00

0327-	20	66	EA	JSR	\$EA66
032A-	A2	EA		LDX	#\$EA
032C-	20	23	EB	JSR	\$EB23
032F-	20	EA	EF	JSR	\$EFEA
0332-	A9	1A		LDA	#\$1A
0334-	A0	00		LDY	#\$00
0336-	20	66	EA	JSR	\$EA66
0339-	A2	1A		LDX	#\$1A
033B-	20	23	EB	JSR	\$EB23
033E-	A9	EA		LDA	#\$EA
0340-	A0	00		LDY	#\$00
0342-	20	F9	EA	JSR	\$EAF9
0345-	20	F1	EF	JSR	\$EFF1
0348-	A2	EA		LDX	#\$EA
034A-	20	23	EB	JSR	\$EB23
034D-	A9	01		LDA	#\$01
034F-	20	93	EB	JSR	\$EB93
0352-	20	63	EB	JSR	\$EB63
0355-	A9	EA		LDA	#\$EA
0357-	A0	00		LDY	#\$00
0359-	20	F9	EA	JSR	\$EAF9
035C-	20	69	EA	JSR	\$EA69
035F-	A9	1A		LDA	#\$1A
0361-	A0	00		LDY	#\$00
0363-	20	BE	E7	JSR	\$E7BE
0366-	20	34	ED	JSR	\$ED34
0369-	20	3A	DB	JSR	\$DB3A
036C-	00			BRK	

程序说明:

300: 取 1 至 A
 302: 转换至FAC
 305-309: 送往 0 页
 指定单元
 30 A-31D: 将 π 的浮点
 数送往 0 页
 EA—EE单元
 31 E: 取 6 至 A
 320: 转换至FAC
 323-326: 取 0 页单元
 指针
 327: $\pi/6$ 转换为弧度
 32 A-32E: 送回 0 页
 单元
 32 F: 计算 $\cos \frac{\pi}{6}$
 332-335: 取0页指针
 336: 计算 $\sec \frac{\pi}{6}$
 339-33D: 将中间结果
 暂存
 33 E-344: 取 $\pi/6$ 至
 FAC
 345: 计算 $\sin \frac{\pi}{6}$
 348-34C: 暂存 0 页
 34D: 取 1 至 A
 34 F: 转换至FAC
 352: FAC→ARG
 355-35B: 取 $\sin \frac{\pi}{6}$ 至FAC

35 C: 计算 $\csc \frac{\pi}{6}$

35 F-365: 取中间

结果

计算整个式子

366: 转换成字符串

369: 输出结果

36 C: 结束

运行程序后显示结果: 3.15470054

例 4: 计算 $|\sqrt{\sin^2 60^\circ + \cos^2 60^\circ}|$

程序 P42 如下:

0300-	A9	82	LDA	#\$82
0302-	85	EA	STA	\$EA
0304-	A9	49	LDA	#\$49
0306-	85	EB	STA	\$EB
0308-	A9	0F	LDA	#\$0F
030A-	85	EC	STA	\$EC
030C-	A9	DA	LDA	#\$DA
030E-	85	ED	STA	\$ED
0310-	A9	A2	LDA	#\$A2
0312-	85	EE	STA	\$EE
0314-	A9	03	LDA	#\$03
0316-	20	93	JSR	\$EB93
0319-	A9	EA	LDA	#\$EA
031B-	A0	00	LDY	#\$00
031D-	20	66	JSR	\$EA66
0320-	A2	EA	LDX	#\$EA
0322-	20	23	JSR	\$EB23
0325-	20	F1	JSR	\$EFF1

0328-	20	63	EB	JSR	\$EB63
032B-	A9	02		LDA	#\$02
032D-	20	93	EB	JSR	\$EB93
0330-	20	97	EE	JSR	\$EE97
0333-	A2	1A		LDX	#\$1A
0335-	20	23	EB	JSR	\$EB23
0338-	A9	EA		LDA	#\$EA
033A-	A0	00		LDY	#\$00
033C-	20	F9	EA	JSR	\$EAF9
033F-	20	EA	EF	JSR	\$EFEA
0342-	20	63	EB	JSR	\$EB63
0345-	A9	02		LDA	#\$02
0347-	20	93	EB	JSR	\$EB93
034A-	20	97	EE	JSR	\$EE97
034D-	A9	1A		LDA	#\$1A
034F-	A0	00		LDY	#\$00
0351-	20	BE	E7	JSR	\$E7BE
0354-	20	8D	EE	JSR	\$EE8D
0357-	20	AF	EB	JSR	\$EBAF
035A-	20	34	ED	JSR	\$ED34
035D-	20	3A	DB	JSR	\$DB3A
0360-	00			BRK	

程序说明:

300-313: 取 π 浮点数

放至0页

单元

314: 取3至A

316: 转换至FAC

319-31F: 将 $\pi/3$, 转换为弧度

320-324: 送回 0 页

暂存

325: 计算 $\sin^2 \frac{\pi}{3}$

328: FAC \rightarrow ARG

32B: 取 2 至 A

32D: 转换至FAC

330: 计算 $\sin^2 \frac{\pi}{3}$

333-337: 送 0 页暂存中间结果

338-33E: 取 $\pi/3$ 弧度至FAC

33F: 计算 $\cos \frac{\pi}{3}$

342: FAC \rightarrow ARG

345: 取 2 至 A

347: 转换至FAC

34A: 计算 $\cos^2 \frac{\pi}{3}$

34D-350: 取中间结果

351: 计算 $\sin^2 \frac{\pi}{3} + \cos^2 \frac{2\pi}{3}$

354: 计算开方

357: 计算绝对值

35A: 转换为字符串

35D: 输出结果

360: 结束。

运行程序后显示结果: 1。

例 5: 计算 $\frac{(2\sin 45^\circ)^4}{\ln 5}$

程序P43如下:

0300-	A9	82		LDA	#\$82
0302-	85	EA		STA	\$EA
0304-	A9	49		LDA	#\$49
0306-	85	EB		STA	\$EB
0308-	A9	0F		LDA	#\$0F
030A-	85	EC		STA	\$EC
030C-	A9	DA		LDA	#\$DA
030E-	85	ED		STA	\$ED
0310-	A9	A2		LDA	#\$A2
0312-	85	EE		STA	\$EE
0314-	A9	04		LDA	#\$04
0316-	20	93	EB	JSR	\$EB93
0319-	A9	EA		LDA	#\$EA
031B-	A0	00		LDY	#\$00
031D-	20	66	EA	JSR	\$EA66
0320-	20	F1	EF	JSR	\$EFF1
0323-	20	63	EB	JSR	\$EB63
0326-	A9	02		LDA	#\$02
0328-	20	93	EB	JSR	\$EB93
032B-	20	82	E9	JSR	\$E982
032E-	20	63	EB	JSR	\$EB63
0331-	A9	04		LDA	#\$04
0333-	20	93	EB	JSR	\$EB93
0336-	20	97	EE	JSR	\$EE97
0339-	A2	EA		LDX	#\$EA
033B-	20	23	EB	JSR	\$EB23
033E-	A9	05		LDA	#\$05
0340-	20	93	EB	JSR	\$EB93

0343-	20	41	E9	JSR	\$E941
0346-	A9	EA		LDA	# \$EA
0348-	A0	00		LDY	# \$00
034A-	20	66	EA	JSR	\$EA66
034D-	20	34	ED	JSR	\$ED34
0350-	20	3A	DB	JSR	\$DB3A
0353-	00			BRK	

程序说明:

300-313: 取 π 浮点数

放0页单元

314: 取4至A

316: 转换至FAC

319-31F: 将 $\pi/4$ 转

换为弧度

320: 计算 $\sin \frac{\pi}{4}$

323: FAC \rightarrow ARG

326: 取2至A

328: 转换至FAC

32B: 计算 $2 * \sin \frac{\pi}{4}$

32E: FAC \rightarrow ARG

331: 取4至A

332: 转换至FAC

336: 计算 $\left(2 \sin \frac{\pi}{4}\right)^4$

339-33D: 暂存中间

结果

33E: 取5至A

340: 转换至FAC

343: 计算LOG5

346-349: 取中间结果

34 A: 计算整个式子

34 D: 转换为字符串

350: 输出结果

运行程序后显示结果: 2.48533974。

附录1 显示ASCII码表

字 符	ASCII 码	反相方式		闪烁方式		控制字符		正常方式	
		16 进制	10 进制	16 进制	10 进制	16 进制	10 进制	16 进制	10 进制
@		00	0	40	64	80	128	C 0	192
A		01	1	41	65	81	129	C 1	193
B		02	2	42	66	82	130	C 2	194
C		03	3	43	67	83	131	C 3	195
D		04	4	44	68	84	132	C 4	196
E		05	5	45	69	85	133	C 5	197
F		06	6	46	70	86	134	C 6	198
G		07	7	47	71	87	135	C 7	199
H		08	8	48	72	88	136	C 8	200
I		09	9	49	73	89	137	C 9	201
J		0 A	10	4 A	74	8 A	138	CA	202
K		0 B	11	4 B	75	8 B	139	CB	203
L		0 C	12	4 C	76	8 C	140	CC	204
M		0 D	13	4 D	77	8 D	141	CD	205
N		0 E	14	4 E	78	8 E	142	CE	206
O		0 F	15	4 F	79	8 F	143	CF	207
P		10	16	50	80	90	144	D 0	208
Q		11	17	51	81	91	145	D 1	209
R		12	18	52	82	92	146	D 2	210
S		13	19	53	83	93	147	D 3	211
T		14	20	54	84	94	148	D 4	212
U		15	21	55	85	95	149	D 5	213
V		16	22	56	86	96	150	D 6	214
W		17	23	57	87	97	151	D 7	215
X		18	24	58	88	98	152	D 8	216

续表

字 符	方 式 ASCII 码	反相方式		闪烁方式		控制字符		正常方式	
		16 进制	10 进制	16 进制	10 进制	16 进制	10 进制	16 进制	10 进制
	Y	19	25	59	89	99	153	D9	217
	Z	1A	26	5A	90	9A	154	DA	218
	[1B	27	5B	91	9B	155	DB	219
	\	1C	28	5C	92	9C	156	DC	220
]	1D	29	5D	93	9D	157	DD	221
	^	1E	30	5E	94	9E	158	DE	222
	_	1F	31	5F	95	9F	159	DF	223

字 符	方 式 ASCII 码	反相方式		闪烁方式		正常方式	
		16进制	10进制	16进制	10进制	16进制	10进制
		20	32	60	96	A0	160
!		21	33	61	97	A1	161
"		22	34	62	98	A2	162
#		23	35	63	99	A3	163
\$		24	36	64	100	A4	164
%		25	37	65	101	A5	165
&		26	38	66	102	A6	166
'		27	39	67	103	A7	167
(28	40	68	104	A8	168
)		29	41	69	105	A9	169
*		2A	42	6A	106	AA	170
+		2B	43	6B	107	AB	171
,		2C	44	6C	108	AC	172

续表

字 符	ASCII 码	反相方式		闪烁方式		正常方式	
		16进制	10进制	16进制	10进制	16进制	10进制
-		2D	45	6D	109	AD	173
.		2E	46	6E	110	AE	174
/		2F	47	6F	111	AF	175
0		30	48	70	112	B0	176
1		31	49	71	113	B1	177
2		32	50	72	114	B2	178
3		33	51	73	115	B3	179
4		34	52	74	116	B4	180
5		35	53	75	117	B5	181
6		36	54	76	118	B6	182
7		37	55	77	119	B7	183
8		38	56	78	120	B8	184
9		39	57	79	121	B9	185
:		3A	58	7A	122	BA	186
;		3B	59	7B	123	BB	187
<		3C	60	7C	124	BC	188
=		3D	61	7D	125	BD	189
>		3E	62	7E	126	BE	190
?		3F	63	7F	127	BF	191

附录2 10进制与16进制转换表

H	h	h0	h00	h000
0	0	0	0	0
1	1	16	256	4096
2	2	32	512	8192
3	3	48	768	12288
4	4	64	1024	16384
5	5	80	1280	20480
6	6	96	1536	24576
7	7	112	1792	28672
8	8	128	2048	32768
9	9	144	2304	36864
A	10	160	2560	40960
B	11	176	2816	45056
C	12	192	3072	49152
D	13	208	3328	53248
E	14	224	3584	57344
F	15	240	3840	61440

说明:

根据附录 2 可以把任何一个二字节的16进制数转换成10进制数。

如AC38可以表示成:

$$AC38 = A000 + C00 + 30 + 8$$

然后查出对应的10进制数, 相加起来就是10进制值。

$$\begin{aligned}
 (AC38)_{16} &= (A000)_{16} + (C00)_{16} + (30)_{16} + (8)_{16} \\
 &= (40960)_{10} + (3072)_{10} + (48)_{10} + (8)_{10} \\
 &= (44088)_{10}
 \end{aligned}$$

又如:

$$\begin{aligned}
 (39A5)_{16} &= (3000)_{16} + (900)_{16} + (A0)_{16} + (5)_{16} \\
 &= (12288)_{10} + (2304)_{10} + (160)_{10} + (5)_{10} \\
 &= (14757)_{10}
 \end{aligned}$$

附录3 6502指令助记符

助记符	功 能	意 义
ADC	相 加	ADd memory to accumulator with Carry
AND	相 与	"AND" memory with accumulator
ASL	左 移 位	Arithmetic Shift Left
BCC	C = 0 转移	Branch on Carry Clear
BCS	C = 1 转移	Branch on Carry Set
BEQ	Z = 1 转移	Branch on result Zero
BIT	数元校验	BITs test in memory with accumulator
BMI	N = 1 转移	Branch on result Minus
BNE	Z = 0 转移	Branch on result Not Zero
BPL	N = 0 转移	Branch on result PLus
BRK	中 断	force BReaK
BVC	V = 0 转移	Branch on oVerflow Clear
BVS	V = 1 转移	Branch on oVerflow Set
CLC	清C标志	Clear Carry Flag
CLD	清D标志	Clear Decimal mode
CLI	清I标志	Clear Interrupt disable bit
CLV	清V标志	Clear Overflow flag
CMP	比 较 A	Compare memory and accumulator
CPX	比 较 X	Compare memory and Index X
CPY	比 较 Y	Compare memory and Index Y
DEC	减 去 1	Decrement memory by One
DEX	X减去1	Decrement register X by One
DEY	Y减去1	Decrement register Y by One
EOR	异或运算	"Exclusive OR" memory with accumulator
INC	加 上 1	Increment memory by One
INX	X加上1	Increment Index X by One
INY	Y加上1	Increment Index Y by One
JMP	转 向	Jump to New Location

助记符	功 能	意 义
JSR	调子程序	Jump to SubRoutine
LDA	存 入 A	Load Accumulator with memory
LDX	存 入 X	Load X register with memory
LDY	存 入 Y	Load Y register with memory
LSR	右 移 位	Logical Shift Right one bit
NOP	空 操 作	No Operation
ORA	或 运 算	"OR" memory with Accumulator
PHA	A 入 栈	Push Accumulator on stack
PHP	P 入 栈	Push Processor status on stack
PLA	退栈到 A	Pull Accumulator from stack
PLP	退栈到 P	Pull Processor status from stack
ROL	左 转 位	ROtate one bit Left
ROR	右 转 位	ROtate one bit Right
RTI	中断返回	Return from Interrupt
RTS	子程序返回	Return from Subroutine
SBC	相 减	Subtract Memory from accumulator with borrow
SEC	置 C 标志	Set Carry flag
SED	置 D 标志	Set Decimal mode
SEI	置 I 标志	Set Interrupt disable status
STA	A 送入内存	Store Accumulator in memory
STX	X 送入内存	Store X register in memory
STY	Y 送入内存	Store Y register in memory
TAX	A 送入 X	Transfer Accumulator to X register
TAY	A 送入 Y	Transfer Accumulator to Y register
TSX	S 送入 X	Transfer Stack pointer to X register
TXA	X 送入 A	Transfer X register to Accumulator
TXS	X 送入 S	Transfer X register to Stack pointer
TYA	Y 送入 A	Transfer Y register to Accumulator

附录4 6502指令码表

助记符	操作码	字节数	寻址方式	指令格式	汇编格式	影响的P位元	周期	功能
ADC	69	2	立即式	69 MM	ADC # \$ MM	NVZC	2	A + MM + C → A, C
	65	2	零页	65 YY	ADC \$ YY		3	A + (00YY) + C → A, C
	75	2	零页, X	75 YY	ADC \$ YY, X		4	A + (00YY + X) + C → A, C
	6D	3	绝对	6D YY XX	ADC \$ XXYY		4	A + (XXYY) + C → A, C
	7D	3	绝对, X	7D YY XX	ADC \$ XXYY, X		4*	A + (XXYY + X) + C → A, C
	79	3	绝对, Y	79 YY XX	ADC \$ XXYY, Y		4*	A + (XXYY + Y) + C → A, C
	61	2	(间接, X)	61 YY	ADC (\$ YY, X)		6	A + ((00YY + X)) + C → A, C
	71	2	(间接), Y	71 YY	ADC (\$ YY), Y		5*	A + (((00YY)) + Y) + C → A, C
	29	2	立即	29 MM	AND # \$ MM	NZ	2	A ∧ MM → A
	25	2	零页	25 YY	AND \$ YY		3	A ∧ (00YY) → A
AND	35	2	零页, X	35 YY	AND \$ YY, X		4	A ∧ (00YY + X) → A

续表

助记符	操作码	字节数	寻址方式	指令格式	汇编格式	影响的P位元	周期	功能
	2D	3	绝对	2D YY XX	AND \$XXYY		4	$A \wedge (XXYY) \rightarrow A$
	3D	3	绝对, X	3D YY XX	AND \$XXYY, X		4*	$A \wedge (XXYY + X) \rightarrow A$
	39	3	绝对, Y	39 YY XX	AND \$XXYY, Y		4*	$A \wedge (XXYY + Y) \rightarrow A$
	21	2	(间接, X)	21 YY	AND (\$YY, X)		6	$A \wedge ((00YY + X)) \rightarrow A$
	31	2	(间接), Y	31 YY	AND (\$YY), Y		5*	$A \wedge (((00YY)) + Y) \rightarrow A$
ASL	0A	1	隐含	0A	ASL	NZC	2	A左移位
	06	2	零页	06 YY	ASL \$YY		5	(00YY)右移位
	16	2	零页, X	16 YY	ASL \$YY, X		6	(00YY + X)左移位
	0E	3	绝对	0E YY XX	ASL \$XXYY		6	(XXYY)左移位
	1E	3	绝对, X	1E YY XX	ASL \$XXYY, X		7	(XXYY + X)左移位
BCC	90	2	相对	90 MM	BCC \$XXYY	无	2**	C = 0 转移
BCS	B0	2	相对	B0 MM	BCS \$XXYY	无	2**	C = 1 转移
BEQ	F0	2	相对	F0 MM	BEQ \$XXYY	无	2**	Z = 1 转移

助记符	操作码	字节数	寻址方式	指令格式	汇编格式	影响的P位元	周期	功能
BIT	24	2	零页	24 YY	BIT \$YY	无	2**	A^(00YY), M ₇ →N, M ₆ →V
	2C	3	绝对	2C YY XX	BIT \$XXYY	NVZ	3	A^(XXYY), M ₇ →N, M ₆ →V
BMI	30	2	相对	30 MM	BMI \$XXYY	无	2**	N = 1 转移
BNE	D0	2	相对	D0 MM	BNE \$XXYY	无	2**	Z = 0 转移
BPL	10	2	相对	10 MM	BPL \$XXYY	无	2**	N = 0 转移
BRK	00	1	隐含	00	BRK	BI	7	中断
BVC	50	2	相对	50 MM	BVC \$XXYY	无	2**	V = 0 转移
BVS	70	2	相对	70 MM	BVS \$XXYY	无	2**	V = 1 转移
CLC	18	1	隐含	18	CLC	C	2	0 → C
CLD	D8	1	隐含	D8	CLD	D	2	0 → D

续表

助记符	操作码	字节数	寻址方式	指令格式	汇编格式	影响的P位元	周期	功能
CLI	58	1	隐含	58	CLI	I	2	0 → I
CLV	B8	1	隐含	B8	CLV	V	2	0 → V
CMP	C9	2	立即	C9 MM	CMP # \$ MM	NZC	2	A - MM
	C5	2	零页	C5 YY	CMP \$ YY		3	A - (00YY)
	D5	2	零页, X	D5 YY	CMP \$ YY, X		4	A - (00YY + X)
	CD	3	绝对	CD YY XX	CMP \$ XXYY		4	A - (XXYY)
	DD	3	绝对, X	DD YY XX	CMP \$ XXYY, X		4*	A - (XXYY + X)
	D9	3	绝对, Y	D9 YY XX	CMP \$ XXYY, Y		4*	A - (XXYY + Y)
	C1	2	(间接, X)	C1 YY	CMP (\$ YY, X)		6	A - ((00YY + X))
	D1	2	(间接), Y	D1 YY	CMP (\$ YY), Y		5*	A - (((00YY)) + Y)
	E0	2	立即	E0 MM	CPX # \$ MM	NZC	2	X - MM
	E4	2	零页	E4 YY	CPX \$ YY		3	X - (00YY)
CPX	EC	3	绝对	EC YY XX	CPX \$ XXYY		4	X - (XXYY)

助记符	操作码	字节数	寻址方式	指令格式	汇编格式	影响的P位元	周期	功能
CPY	C0	2	立即	C0 MM	CPY # \$ MM	NZC	2	Y - MM
	C4	2	零页	C4 YY	CPY \$ YY		3	Y - (00YY)
	CC	3	绝对	CC YY XX	CPY \$ XXYY		4	Y - (XXYY)
	C6	2	零页	C6 YY	DEC \$ YY	NZ	5	(00YY) - 1 → (00YY)
DEC	D6	2	零页, X	D6 YY	DEC \$ YY, X		6	(00YY + X) - 1 → (00YY + X)
	CE	3	绝对	CE YY XX	DEC \$ XXYY		6	(XXYY) - 1 → (XXYY)
	DE	3	绝对, X	DE YY XX	DEC \$ XXYY, X		7	(XXYY + X) - 1 → (XXYY + X)
	CA	1	隐含	CA	DEX	NZ	2	X - 1 → X
DEY	88	1	隐含	88	DEY	NZ	2	Y - 1 → Y
EOR	49	2	立即	49 MM	EOR	NZ	2	A ← MM → A
	45	2	零页	45 YY	EOR		3	A ← (00YY) → A
	55	2	零页, X	55 YY	EOR		4	A ← (00YY + X) → A
	4D	3	绝对	4D YY XX	EOR		4	A ← (XXYY) → A

续表

助记符	操作码	字节数	寻址方式	指令格式	汇编格式	影响的P位元	周期	功能
	5D	3	绝对, X	5D YY XX	EOR		4*	$A \leftarrow (XXYY + X) \rightarrow A$
	59	3	绝对, Y	59 YY XX	EOR		4*	$A \leftarrow (XXYY + Y) \rightarrow A$
	41	2	(间接, X)	41 YY	EOR		6	$A \leftarrow ((00YY + X)) \rightarrow A$
	51	2	(间接), Y	51 YY	EOR		5*	$A \leftarrow (((00YY)) + Y) \rightarrow A$
INC	E6	2	零页	E6 YY	INC \$YY	NZ	5	$(00YY) + 1 \rightarrow (00YY)$
	F6	2	零页, X	F6 YY	INC \$YY, X		6	$(00YY + X) + 1 \rightarrow (00YY + X)$
	EE	3	绝对	EE YY XX	INC \$XXYY		6	$(XXYY) + 1 \rightarrow (XXYY)$
	FE	3	绝对, X	FE YY XX	INC \$XXYY, X		7	$(XXYY + X) + 1 \rightarrow (XXYY + X)$
INX	E8	1	隐含	E8	INX	NZ	2	$X + 1 \rightarrow X$
INY	C8	1	隐含	C8	INY	NZ	2	$Y + 1 \rightarrow Y$
JMP	4C	3	绝对	4C YY XX	JMP \$XXYY	无	3	转向XXYY
	6C	3	间接	6C YY XX	JMP(\$XXYY)		5	转向(XXYY)
JSR	20	3	绝对	20 YY XX	JSR \$XXYY	无	6	调子程序

助记符	操作码	字节数	寻址方式	指令格式	汇编格式	影响的P位元	周期	功能
LDA	A9	2	立即	A9 MM	LDA # \$MM	NZ	2	MM \rightarrow A
	A5	2	零页	A5 YY	LDA \$YY		2	(00YY) \rightarrow A
	B5	2	零页, X	B5 YY	LDA \$YY, X		4	(00YY + X) \rightarrow A
	AD	3	绝对	AD YY XX	LDA \$XXYY		4	(XXYY) \rightarrow A
	BD	3	绝对, X	BD YY XX	LDA \$XXYY, X		4*	(XXYY + X) \rightarrow A
	B9	3	绝对, Y	B9 YY XX	LDA \$XXYY, Y		4*	(XXYY + Y) \rightarrow A
	A1	2	(间接, X)	A1 YY	LDA(\$YY, X)		6	((00YY + X)) \rightarrow A
	B1	2	(间接), Y	B1 YY	LDA(\$YY), Y		5*	((00YY) + Y) \rightarrow A
	A2	2	立即	A2 MM	LDX # \$MM	NZ	2	MM \rightarrow X
	A6	2	零页	A6 YY	LDX \$YY		2	(00YY) \rightarrow X
LDX	B6	2	零页, X	B6 YY	LDX \$YY, X		4	(00YY + Y) \rightarrow X
	AE	3	绝对	AE YY XX	LDX \$XXYY		4	(XXYY) \rightarrow X
	BE	3	绝对, Y	BE YY XX	LDX \$XXYY, Y		4*	(XXYY + Y) \rightarrow X

助记符	操作码	字节数	寻址方式	指令格式	汇编格式	影响的P位元	周期	功能
LDY	A0	2	立即	A0 MM	LDY # \$ MM	NZ	2	MM \rightarrow Y
	A4	2	零页	A4 YY	LDY \$ YY		3	(00YY) \rightarrow Y
	B4	2	零页, X	B4 YY	LDY \$ YY, X		4	(00YY + X) \rightarrow Y
	AC	3	绝对	AC YY XX	LDY \$ XXYY		4	(XXYY) \rightarrow Y
	BC	3	绝对, X	BC YY XX	LDY \$ XXYY, X		4*	(XXYY + X) \rightarrow Y
LSR	4A	1	隐含	4A	LSR	NZC	2	A 右移位
	46	2	零页	46 YY	LSR \$ YY		5	(00YY) 右移位
	56	2	零页, X	56 YY	LSR \$ YY, X		6	(00YY + X) 右移位
	4E	3	绝对	4E YY XX	LSR \$ XXYY		6	(XXYY) 右移位
	5E	3	绝对, X	5E YY XX	LSR \$ XXYY, X		7	(XXYY + X) 右移位
NOP	EA	1	隐含	EA	NOP	无	2	空操作
ORA	09	2	立即	09 MM	ORA # \$ MM	NZ	2	A/MM \rightarrow A
	05	2	零页	05 YY	ORA \$ YY		3	A/(00YY) \rightarrow A

助记符	操作码	字节数	寻址方式	指令格式	汇编格式	影响的P位元	周期	功能
	15	2	零页, X	15 YY	ORA \$YY, X		4	$A \vee (00YY + X) \rightarrow A$
	0D	3	绝对	0D YY XX	ORA \$XXYY		4	$A \vee (XXYY) \rightarrow A$
	1D	3	绝对, X	1D YY XX	ORA \$XXYY, X		4*	$A \vee (XXYY + X) \rightarrow A$
	19	3	绝对, Y	19 YY XX	ORA \$XXYY, Y		4*	$A \vee (XXYY + Y) \rightarrow A$
	01	2	(间接, X)	01 YY	ORA (\$YY, X)		6	$A \vee (00YY + X) \rightarrow A$
	11	2	(间接), Y	11 YY	ORA (\$YY), Y		5*	$A \vee (((00YY)) + Y) \rightarrow A$
PHA	48	1	隐含	48	PHA	无	3	A 入栈
PHP	08	1	隐含	08	PHP	无	3	P 入栈
PLA	68	1	隐含	68	PLA	NZ	4	A 出栈
PLP	28	1	隐含	28	PLP	所有位元	4	P 出栈
ROL	2A	1	隐含	2A	ROL	NZC	2	A 左移位

助记符	操作码	字节数	寻址方式	指令格式	汇编格式	影响的P位元	周期	功能
	26	2	零页	26 YY	ROL \$ YY		5	(00YY)左转位
	36	2	零页, X	36 YY	ROL \$ YY, X		6	(00YY + X)左转位
	2E	3	绝对	2E YY XX	ROL \$ XXYY		6	(XXYY)左转位
	3E	3	绝对, X	3E YY XX	ROL \$ XXYY, X		7	(XXYY + X)左转位
ROR	6A	1	隐含	6A	ROR	NZC	2	A右转位
	66	2	零页	66 YY	ROR \$ YY		5	(00YY)右转位
	76	2	零页, X	76 YY	ROR \$ YY, X		6	(00YY + X)右转位
	6E	3	绝对	6E YY XX	ROR \$ XXYY		6	(XXYY)右转位
RTI	7E	3	绝对, X	7E YY XX	ROR \$ XXYY, X		7	(XXYY + X)右转位
	40	1	隐含	40	RTI	所有位元	6	中断返回
RTS	60	1	隐含	60	RTS	无	6	返回
	E9	2	立即	E9 MM	SBC # \$ MM	NVZC	2	A - MM - C → A

助记符	操作码	字节数	寻址方式	指令格式	汇编格式	影响的P位元	周期	功能
	E5	2	零页	E5 YY	SBC \$ YY		3	$A - (00YY) - \overline{C} \rightarrow A$
	F5	2	零页, X	F5 YY	SBC \$ YY, X		4	$A - (00YY + X) - \overline{C} \rightarrow A$
	ED	3	绝对	ED YY XX	SBC \$ XXYY		4	$A - (XXYY) - \overline{C} \rightarrow A$
	FD	3	绝对, X	FD YY XX	SBC \$ XXYY, X		4*	$A - (XXYY + X) - \overline{C} \rightarrow A$
	F9	3	绝对, Y	F9 YY XX	SBC \$ XXYY, Y		4*	$A - (XXYY + Y) - \overline{C} \rightarrow A$
	E1	2	(间接, X)	E1 YY	SBC (\$ YY, X)		6	$A - ((00YY + X)) - \overline{C} \rightarrow A$
	F1	2	(间接), Y	F1 YY	SBC (\$ YY), Y		5*	$A - (((00YY)) + Y) - \overline{C} \rightarrow A$
SEC	38	1	隐含	38	SEC	C	2	$1 \rightarrow C$
SED	F8	1	隐含	F8	SED	D	2	$1 \rightarrow D$
SEI	78	1	隐含	78	SEI	I	2	$1 \rightarrow I$
STA	85	2	零页	85 YY	STA \$ YY	无	3	$A \rightarrow (00YY)$
	95	2	零页, X	95 YY	STA \$ YY, X		4	$A \rightarrow (00YY + X)$
	8D	3	绝对	8D YY XX	STA \$ XXYY		4	$A \rightarrow (XXYY)$

续表

助记符	操作码	字节数	寻址方式	指令格式	汇编格式	影响的P位元	周期	功能
	9D	3	绝对, X	9D YY XX	STA, \$XXYY, X		5	A \rightarrow (XXYY + X)
	99	3	绝对, Y	99 YY XX	STA, \$XXYY, Y		5	A \rightarrow (XXYY + Y)
	81	2	(间接, X)	81 YY	STA (\$YY, X)		6	A \rightarrow ((00YY + X))
	91	2	(间接), Y	91 YY	STA (\$YY), Y		6	A \rightarrow (((00YY)) + Y)
STX	86	2	零页	86 YY	STX \$YY	无	3	X \rightarrow (00YY)
	96	2	零页, Y	96 YY	STX \$YY, Y		4	X \rightarrow (00YY + Y)
	8E	3	绝对	8E YY XX	STX, \$XXYY		4	X \rightarrow (XXYY)
	84	2	零页	84 YY	STY \$YY	无	3	Y \rightarrow (00YY)
STY	94	2	零页, X	94 YY	STY \$YY, X		4	Y \rightarrow (00YY + X)
	8C	3	绝对	8C YY XX	STY, \$XXYY		4	Y \rightarrow (XXYY)
	AA	1	隐含	AA	TAX	NZ	2	A \rightarrow X
TAY	A8	1	隐含	A8	TAY	NZ	2	A \rightarrow Y

助记符	操作码	字节数	寻址方式	指令格式	汇编格式	影响的P位元	周期	功能
TSX	BA	1	隐含	BA	TSX	NZ	2	S → X
TXA	8A	1	隐含	8A	TXA	NZ	2	X → A
TXS	9A	1	隐含	9A	TXS	无	2	X → S
TYA	98	1	隐含	98	TYA	NZ	2	Y → A

说明:

表中MM代表一个00—FF的立即数, XX代表一个内存地址的高位, YY代表地址的低位。功能栏中单括号表示取该地址单元中的数, 双括号表示取该间接地址所指单元中的数。

* 不发生转移时按表中给出的数, 转移到本页内周期加1, 转移到另一页周期加2。

** 转移到另一页周期加1。

附录5 6502操作码表

操作码	汇 编 格 式	操作码	汇 编 格 式	操作码	汇 编 格 式
00	BRK	2 A	ROL	51	EOR (\$YY), Y
01	ORA (\$YY, X)	2 C	BIT \$XXYY	55	EOR \$YY, X
05	ORA \$YY	2 D	AND \$XXYY	56	LSR \$YY, X
06	ASL \$YY	2 E	ROL \$XXYY	58	CLI
08	PHP			59	EOR \$XXYY, Y
09	ORA # \$MM	30	BMI \$XXYY	5D	EOR \$XXYY, X
0A	ASL	31	AND (\$YY), Y	5E	LSR \$XXYY, X
0D	ORA \$XXYY	35	AND \$YY, X		
0E	ASL \$XXYY	36	ROL \$YY, X	60	RTS
		38	SEC	61	ADC (\$YY, X)
10	BPL \$XXYY	39	AND \$XXYY, Y	65	ADC \$YY
11	ORA (\$YY), Y	3D	AND \$XXYY, X	66	ROR \$YY
15	ORA \$YY, X	3E	ROL \$XXYY, X	68	PLA

操作码	汇 编 格 式	操作码	汇 编 格 式	操作码	汇 编 格 式
16	ASL \$ YY, X	40	RTI	69	ADC # \$ MM
18	CLC	41	EOR (\$ YY, X)	6A	ROR
19	ORA \$ XXYY, Y	45	EOR \$ YY	6C	JMP (\$ XXYY)
1D	ORA \$ XXYY, X	46	LSR \$ YY	6D	ADC \$ XXYY
1E	ASL \$ XXYY, X	48	PHA	6E	ROR \$ XXYY
20	JSR \$ XXYY	49	EOR \$ MM	70	BVS \$ XXYY
21	AND (\$ YY, X)	4A	LSR	71	ADC (\$ YY), Y
24	BIT \$ YY	4C	JMP \$ XXYY	75	ADC \$ YY, X
25	AND \$ YY	4D	EOR \$ XXYY	76	ROR \$ YY, X
26	ROL \$ YY	4E	LSR \$ XXYY	78	SEI
28	PLP	50	BVC \$ XXYY	79	ADC \$ XXYY, Y
29	AND # \$ MM	AA	XTA	7D	ADC \$ XXYY, X
7E	ROR \$ XXYY, X	AC	LDY \$ XXYY	D1	CMP (\$ YY), Y
				D5	CMP \$ YY, X

续表

操作码	汇 编 格 式	操作码	汇 编 格 式	操作码	汇 编 格 式
81	STA (\$YY,X)	AD	LDA \$XXYY	D6	DEC \$YY,X
84	STY \$YY	AE	LDX \$XXYY	D8	CLD
85	STA \$YY			D9	CMP \$XXYY,Y
86	STX \$YY	B0	BCS \$XXYY	DD	CMP \$XXYY,X
88	DEY	B1	LDA (\$YY),Y	DE	DEC \$XXYY,X
8A	TXA	B4	LDY \$YY,X		
8C	STY \$XXYY	B5	LDA \$YY,X	E0	CPX #\$MM
8D	STA \$XXYY	B6	LDX \$YY,Y	E1	SBC (\$YY,X)
8E	STX \$XXYY	B8	CLV	E4	CPX \$YY
		B9	LDA \$XXYY,Y	E5	SBC \$YY
90	BCC \$XXYY	BA	TSX	E6	INC \$YY
91	STA (\$YY),Y	BC	LDY \$XXYY,X	E8	INX
94	STY \$YY,X	BD	LDA \$XXYY,X	E9	SBC #\$MM
95	STA \$YY,X	BE	LDX \$XXYY,Y	EA	NOP
96	STX \$YY,Y			EC	CPX \$XXYY

操作码	汇 编 格 式	操作码	汇 编 格 式	操作码	汇 编 格 式
98	TYA	C 0	CPY # \$ MM	ED	SBC \$ XXYY
99	STA \$ XXYY, Y	C 1	CMP (\$ YY, X)	EE	INC \$ XXYY
9 A	TX	C 4	CPY \$ YY		
9 D	STA \$ XXYY, X	C 5	CMP \$ YY	F 0	BEQ \$ XXYY
A 0	LDY # \$ MM	C 6	DEC \$ YY	F 1	SBC (\$ XX), Y
A 1	LDA (\$ YY, X)	C 8	INY	F 5	SBC \$ YY, X
A 2	LDX # \$ MM	C 9	CMP # \$ MM	F 6	INC \$ YY, X
A 4	LDY \$ YY	CA	DEX	F 8	SED
A 5	LDA \$ YY	CC	CPY \$ XXYY	F 9	SBC \$ XXYY, Y
A 6	LDX \$ YY	CD	CMP \$ XXYY	FD	SBC \$ XXYY, X
A 8	TAY	CE	DEC \$ XXYY	FE	INC \$ XXYY, X
A 9	LDA # \$ MM	D 0	BNE \$ XXYY		

附录6 BASIC函数运算 入口地址表

函数名	地址	函数名	地址	函数名	地址
ABS	EBAF	INT	EC23	RIGHT \$	E 686
ASC	E 6 E 5			RND	EFAE
ATN	F 0 9 E	LEFT \$	E5AA		
		LEN	E6D6	SGN	EB90
CHR \$	E 646	LOG	E 941	SIN	EFF1
COS	EFEA			STR \$	E3C5
		MID \$	E 691	SQR	EE8D
EXP	EF09				
		PEEK	E 764	TAN	F03A
FRE	E2DE	POS	E2FF		
				VAL	E 707

附录7 零页地址大全

地址	用 途
\$ 00—\$ 02	存放着由监控转到APPLESOFT BASIC的指令。通常开机时设置为\$ 4 C、\$ 3 C、\$ D 4 (JMP \$ D4 3C)。在监控下键入0G↵便返回BASIC状态。
\$ 00—\$ 01	磁盘I/O口矢量低位和高位暂存单元
\$ 03—\$ 05	存放着转去输出某一字符串的指令 (JMP \$ DB3A)
\$ 06—\$ 09	用户可以选用
\$ 0A—\$ 0C	存放着转移至USR函数的指令 (由用户设置)

\$0D—\$0E	用于扫描字符串, REM内容等的开始和结束标志
\$0F	工作单元, 常用于计数, 如统计一个程序行或字符串的长度, 在把关键字转为内码时记代号或记数组的维数
\$10	用于DIM时, 置入数组各首字符的ASCII码, 用来检查一个数组是否重复定义
\$11	用于计算表达式, 如字符串型变量, 置FF; 如果算术型变量, 置0
\$12	区分整型数和实型数, 前者置80, 后者置00
\$13	处理键盘输入时, 若是冒号, 置0; 若是DATA, 置49; 其它置入4
\$14	标志是否允许用整型数, 允许置0; 否则置80
\$15	用于INPUT、READ和GET执行中, 三者分别为0, 98和40
\$16	处理关系表达式时, 存放关系运算符代码
\$17—\$19	用户可以使用
\$1A—\$1B	高分辨率清屏时, 放图形首地址, 清完下一行后, \$1B中数加1, 直至完全清屏。用图形表作图时, \$1A—\$1B先被置入图形表首址
\$1C	高分辨率颜色标志
\$1D—\$1F	用户可以使用
\$20	文本窗口左边位置单元, 屏幕从左至右共40列, 列号为0—39, 但不能大于40, 否则输出的文字资料将全部或部分不被显示在屏幕上, 从而损失程序或数据的某些部分
\$21	文本窗口宽度单元, 其值必须在1到40之间选择, 它等于屏幕左边框与右边框相距的列数。注意不要在\$21单元存放0, 否则将使系统BASIC解释程序不能正常工作

\$ 22	文本窗口上边框单元, 机器将屏幕分成24行, 行号是0—23, \$ 22单元置0则文本从顶端开始, 置23则在屏幕底端
\$ 23	文本窗口下边框单元, 其值应在0—23之间选取, 使用时注意不要将上边框的值与下边框的值颠倒了
\$ 24	光标所在水平位置 (列计数)
\$ 25	光标所在垂直位置 (屏幕字符行计数)
\$ 26—\$ 27	低 (高) 分辨图形地址; 段落缓冲区地址(ROM); 临时使用空间 (RWST)
\$ 28—\$ 29	光标所在行最左一个字符在屏幕存储器地址
\$ 2A—\$ 2B	屏幕滚动输出时记滚动行最左一个字符在屏幕存储器的地址
\$ 2C	低分辨率画水平线时终止位置
\$ 2D	低分辨率画垂直线时终止位置
\$ 2C—\$ 2D	反汇编左、右助记符代码暂存单元
\$ 2E	控制低分辨率图形屏幕颜色; 写、读磁带时“检查和”存储单元; 反汇编寻址方式代码
\$ 2F	检查读磁带信号单元, 平时 $D_7 = 0$; 反汇编操作数字节计数单元
\$ 30	低分辨率颜色暂存单元; 高分辨时坐标除以7的余数相对应的值
\$ 31	监控键盘命令中“:”、“.”、“+”、“-”、“.”方式存储单元
\$ 32	字符显示方式 (\$ FF正常, \$ 7F闪烁, \$ 3F反白)
\$ 33	存放提示符
\$ 34—\$ 35	寄存器Y暂存单元
\$ 36—\$ 37	指出输出字符的外部设备地址, 通常为\$ FDF0 (指向显示器)
\$ 38—\$ 39	指出输入字符的外部设备地址, 当RESET之后, 它

	会自动指向键盘工作单元 \$FD1B
\$3A—\$3B	存放并控制6502CPU的程序计数器PC的内含值，改变这两个单元的值，即等于执行一条JMP指令
\$3C—\$3D	读写磁带首地址；操作数地址缓冲区（A1）；装置特征表地址（RWTS）
\$3E—\$3F	读写磁带末地址，操作数地址缓冲区（A2）
\$40—\$41	操作数地址缓冲区（A3）
\$42—\$43	操作数地址缓冲区（A4）
\$44—\$45	操作数地址缓冲区（A5）
\$45—\$49	依次存放累加器A，寄存器X，寄存器Y，状态寄存器P和堆栈S的值
\$4A—\$4B	INTEGER BASIC的变量存放起始地址的指示器，由LOMEM设定，通常是\$0800
\$4C—\$4D	INTEGER BASIC程式存放最终地址的指示器，由HIMEM设定，通常是\$C000
\$4E—\$4F	键盘计数地址的值；随机数产生器
\$50—\$51	行号暂存单元，POKE，PEEK，CALL命令地址存储单元
\$52—\$5D	多用途指针
\$5E—\$5F	经常存放被移地址
\$60—\$61	处理关系表达式时存放数据；修改程序区内容时，用作被移动位置地址
\$62—\$66	存放上次进行浮点数运算乘除的结果
\$67—\$68	APPLESOFT BASIC程序区首指针，通常被置为\$801
\$69—\$6A	APPLESOFT BASIC程序变量区首指针
\$6B—\$6C	APPLESOFT BASIC程序数组区首指针
\$6D—\$6E	APPLESOFT BASIC程序自由区首指针
\$6F—\$70	APPLESOFT BASIC程序字符串区首指针

\$ 71—\$ 72	当前字符串首址
\$ 73—\$ 74	APPLESOFT BASIC程序最高可用地址 (HIMEM)
\$ 75—\$ 76	正在执行的程序行的行号, 若程序不再执行时, 存放FF)
\$ 77—\$ 78	由CTRL—C或STOP或END建立的行号
\$ 79—\$ 7A	下一个被执行的程序行首字节的地址
\$ 7B—\$ 7C	READ语句正在执行DATA语句的行号
\$ 7D—\$ 7E	READ语句正在执行DATA语句的存贮地址
\$ 7F—\$ 80	执行INPUT时指向 \$ 201; 在执行 READ 时指向DATA语句所在地址上
\$ 81—\$ 82	刚刚用到的变量名
\$ 83—\$ 84	正在处理的变量的数据地址
\$ 85—\$ 86	为一个变量存放数据的首址
\$ 87—	当前处理的算术运算符号 (关系符 除外) 的一个代码
\$ 88—	未使用
\$ 89—	放正在处理的关系运算符号的一个代码
\$ 8A—\$ 8B	函数数据首址
\$ 8A—\$ 8E	浮点暂存器 3 (TEMP 3)
\$ 8F	打扫内存空间时作指针用
\$ 90—\$ 92	放一条JMP指令指向被处理函数的相应入口
\$ 93—\$ 97	浮点暂存器 1 (TEMP 1)
\$ 98—\$ 9C	浮点暂存器 2 (TEMP 2)
\$ 9D—\$ A3	主浮点累加器 (FAC)
\$ A4	浮点数运算时暂存器
\$ A5—\$ AB	次浮点累加器 (或称浮点暂存器ARG), 用于浮点数运算
\$ AB—\$ AC	在移动字符串子程序时, 指向该字符串首地址
\$ AD—\$ AE	在处理字符串时, 指向该字符串首地址

\$ AF—\$ B0	APPLESOFT BASIC程序区末指针
\$ B1—\$ C8	扫描子程序
\$ B8—\$ B9	读得字符的指针
\$ C9—\$ CD	上次RND产生的随机数
\$ CE—\$ CF	未使用
\$ D0—\$ D5	高分辨率图形指标表
\$ D6	DOS的程序保护标志
\$ D7	未使用
\$ D8	错误标志, ONERR使用时放\$ 80
\$ D9	未使用
\$ DA—\$ DB	存放产生错误的行号
\$ DC—\$ DD	ONERR错误地址
\$ DE	错误代码存贮单元
\$ DF	错误时堆栈值
\$ E0—\$ E1	高分辨率图形的水平位置坐标 (X) 值
\$ E 2	高分辨率图形的垂直位置坐标 (Y) 值
\$ E 3	未使用
\$ E 4	高分辨率图形颜色代码, 0 = 0, 42 = 1, 85 = 2 127 = 3, 128 = 4, 170 = 5, 213 = 6, 255 = 7
\$ E5	水平坐标除以7的整数部分
\$ E6	高分辨率图形页面设定单元, \$ 20为HGR, \$ 40为 HGR2
\$ E 7	造型表放大的倍数 (SCALE值)
\$ E 8 — \$ E9	造型表首址
\$ EA	高分辨图形碰撞计数器
\$ EB—\$ EF	用户可以使用
\$ F 0	低分辨率图形使用暂存单元
\$ F 1	显示文本字符的速度值
\$ F 2	TRACE存在的标志, 最高位为1, 表示跟踪

\$ F 3 输出闪烁单元时标志
\$ F 4 — \$ F 5 ONERR GOTO的指针
\$ F 6 — \$ F 7 ONERR GOTO的行号
\$ F 8 每一行执行前的堆栈值
\$ F 9 造型表角度 (ROT) 值
\$ FA — \$ FF 用户可以使用