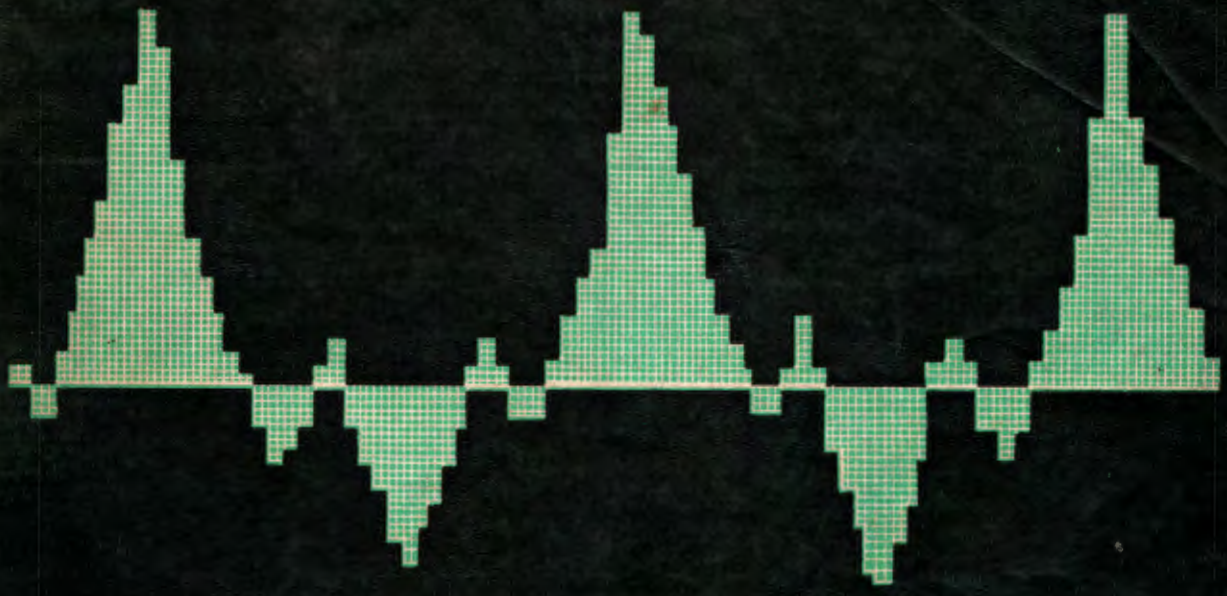
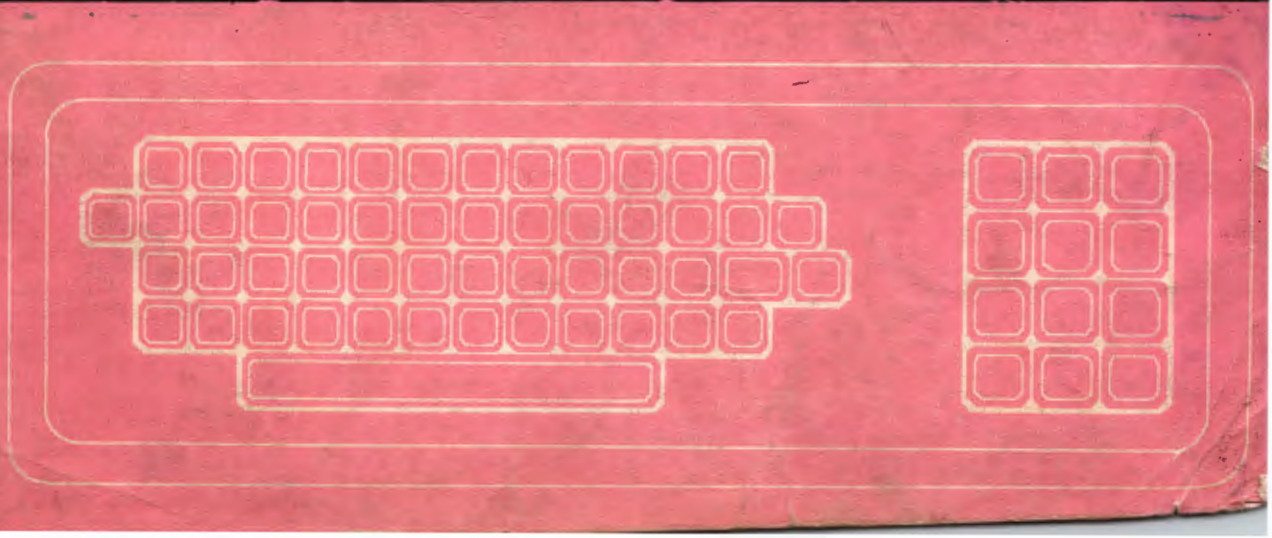


微计算机实用手册

WEI JISUAN JI
SHI YONG SHOU CE

计 声 编译

海 洋 出 版 社



责任编辑：王小南

统一书号：13193·0119

定　价：6.40 元

微计算机实用手册

计 声 编译

海 洋 出 版 社

1982年·北 京

内　容　简　介

本书是介绍微计算机软、硬件比较系统比较全面的一本选辑。文章选自期刊《微计算机应用》80年1—6期，内容经过重新校订和修改。主要内容是以Z—80为CPU的TRS—80微型机的有关语言、系统软件、硬件的使用和用户手册。第一部分是软件部分，包括LEVEL Ⅱ BASIC；FORTRAN包；磁盘操作系统和磁盘BASIC；编辑／汇编程序；监视、纠错和调试程序等手册。第二部分是硬件部分，有硬件手册；扩展接口硬件及操作员手册；RS—232—C接口等。对于同类型机器BC3—80，MDR—Z80，SEED—Z80，PS—80，EG3003，YEE—8100等，语言和软件是通用的。

本书主要对象是使用该系统的用户以及从事微计算机设计和应用的科研人员、工程技术人员和管理人员、大专院校师生等。

微计算机实用手册

计　声编　译

*

海　洋　出　版　社　出　版

（北京复兴门外海贸大楼）

国　防　科　工　委　印　刷　厂　印　刷

新华书店北京发行所发行　　各地新华书店经售

*

1982年7月第1版　　1985年8月第3次印刷

开本：787×1092　1/16　印张：30 1/8插页：4

字数：800,000　　印数：40001—64000

统一书号：13193·0119　定价：7.50元

目　　录

第一部分　软 件 部 分

第一篇　TRS-80 Level Ⅰ BASIC 使用手册……（ 1 ）

前　　言……（ 1 ）

一、概述……（ 3 ）

二、命令……（ 9 ）

三、输入／输出语句……（ 13 ）

四、程序语句……（ 23 ）

五、字符串……（ 37 ）

六、数组……（ 45 ）

七、数学函数……（ 50 ）

八、特殊功能和逻辑运算……（ 53 ）

九、编辑……（ 61 ）

十、扩展接口……（ 66 ）

十一、节省时间和空间……（ 69 ）

附　　录

1. LEVEL Ⅱ一览表……（ 70 ）

2. 错误代码……（ 80 ）

3. 控制代码、ASCII 代码和图示代码……（ 81 ）

4. LEVEL Ⅱ TRS-80 内存图……（ 83 ）

5. 导出函数……（ 86 ）

6. 基本变换表……（ 87 ）

7. 程序举例……（ 88 ）

第二篇　TRS-80 FORTRAN 使用手册……（ 96 ）

一、概述……（ 96 ）

二、TRS-80 FORTRAN 编译器……（101）

三、TRS-80 FORTRAN 盘文件……（103）

四、错误信息……（104）

第三篇　FORTRAN-80 参考手册……（107）

一、引言……（107）

二、FORTRAN 程序格式……（107）

三、数据表示／存贮格式……（111）

四、FORTRAN 表达式……（115）

五、置换语句……（118）

六、说明语句……（119）
七、FORTRAN 控制语句……（124）
八、输入／输出……（130）
九、函数和子程序……（145）
附　录
1．语言的扩充和限制……（154）
2．I/O 接口……（155）
3．子程序连接法……（160）
4．ASCII 字符的代码……（162）
5．FORTRAN—80 子程序库……（163）
第四篇　LINK-80 连接装配程序参考手册……（165）
第五篇　EDIT-80 用戶指南……（170）
一、EDIT-80 操作……（170）
二、行间编辑……（172）
三、行内编辑—变更方式……（175）
四、寻找和替换命令……（178）
五、分页……（179）
六、由 EDIT-80 出口……（181）
附　录
1．命令摘要……（183）
2．变更方式子命令摘要……（184）
3．记号摘要……（184）
4．EDIT—80 专用字符……（185）
5．错误信息……（185）
6．输出文件格式……（186）
第六篇　TRS-80 磁盘操作系统和磁盘 BASIC 参考手册……（188）
一、概述……（188）
二、小型磁盘操作……（191）
三、TRS 磁盘操作系统概述……（198）
四、TRS 磁盘操作系统命令……（203）
五、扩展实用程序……（219）
六、TRSDOS 的技术资料……（223）
七、磁盘 BASIC……（232）
八、附　录……（283）
第七篇　TRS-80 编辑／汇编程序使用手册……（299）
一、前言……（299）
二、本手册中所用符号的说明……（299）
三、编辑／汇编程序……（300）
四、汇编语言……（314）
五、错误信息……（317）

第八篇 TRS-80 监视、纠错和调试程序使用手册……（323）
一、引言……（323）
二、送入和使用 TBUG (LEVEL Ⅰ)……（324）
三、送入和使用 TBUG (LEVEL Ⅱ)……（325）
四、TBUG命令……（327）
五、有关变换的一些考虑……（332）
六、附　录

第二部分　硬 件 部 分

第一篇 TRS-80 硬件手册……（363）
一、序言……（363）
二、系统的结构与工作原理……（363）
三、机器的调整和故障分析……（392）
四、关于 TRS-80 的应用方法……（413）
五、BASIC Ⅱ用的ROM……（423）
第二篇 TRS-80 扩展接口操作员手册……（425）
第三篇 TRS-80 扩展接口硬件手册……（431）
一、安装……（431）
二、外部设备……（435）
三、技术资料……（439）
四、故障检查……（449）
第四篇 TRS-80 RS—232—C 接口……（453）
一、引言……（453）
二、RS—232—C 的安装方法……（456）
三、工作原理……（458）

第一部分　软 件 部 分

第一篇　TRS-80 Level Ⅱ BASIC使用手册

前　　言

1. 连接视频显示器和键盘

1) 将视频显示器的电源线接到120V、60Hz交流电源上。要注意，该插头的插柱要比别的粗，应把粗头插入交流插座上最大的孔中。应尽可能使用墙上的(110V)市电。(译注：有110V，220V两种电源，千万不要把110V的插入220V电源中，下同)

2) 将电源变压器的电源线接到120V、60Hz、交流电源上。

3) 将视频显示器前面的灰色电缆接到键盘（系统）背面的VIDEO插孔上。要注意正确按管脚的的顺序插入（插头的配接只有一种方法）。

注意：在做下一项工作之前，键盘背后的电源POWER开关必须是关着的（按出）。

4) 将电源变压器的灰色电缆接到键盘（系统）背后的POWER插孔。再次注意正确接线。

2. 连接盒式磁带机

注意：除非要记录程序或把磁带程序送入TRS-80，否则不必连接盒式磁带机。

1) 将CTR-41记录器接到120V交流电源上（对于TRS-80所用的记录器，不提倡使用电池）。

2) 将短电缆一端的组合电缆插头，（另一端是三个分散的插头），接入键盘背后的TAPE插孔。插头的配接一定要正确。

3) 该线另一端的三个插头是连接CTR-41的：

(1) 将黑色插头插入CTR-41边上的EAR插孔。它给TRS-80提供了CTR-41的输出信号（可将磁带程序送入TRS-80）。

(2) 将大的灰插头插入CTR-41边上的AUX插孔。它提供了TRS-80的输出信号以便把程序记录到CTR-41的磁带上。
同时，应将短路插头（CTR-41备有）插入MIC插孔[注]（这使得往磁带上记录时，切断内部的传声器，使它不能拾取周围的噪声）。

(3) 将小的灰插头插入CTR-41边上的REM插孔。这使TRS-80能自动控制CTR-41的马达（在记录或放送时，控制马达的开和关）。

注：对于随机器提供的CTR—80磁带机，不需也不能将短路插头接入MIC插孔！因为AUX插孔的插入已使内部传声器切断，再插入MIC插孔反而使TRS—80的输出信号也被开路，结果什么也记录不上。—译者

3. 使用磁带机的注意事项

Radio—shack TRS—80 的盒式磁带机是快速和精确地存储数据和程序的设施。有些事项在使用盒式磁带机时必须知道：

1) 放送磁带（将程序送入 TRS-80）时，必须将 CTR-41 的音量控制放在中间偏高位置（约 4—6 之间），然后按下 CTR-41 的 PLAY 键。接着在 TRS-80 上打 CLOAD，再发 ENTER 命令，于是磁带开始运动，在显示器的右上角将出现一个*号，第二个*号的不断闪烁表示程序正在进入。程序输完后，TRS-80 自动关掉 CTR-41，并在荧光屏上显示 READY。这时，你就可以运行程序（打入 RUN并按 ENTER）。

2) 要把 TRS-80 上的程序记录到磁带上，则同时按下 CTR-41 的 RECORD 和 PLAY 两键，然后在键盘上打入 CSAVE，后面接一个由引号括着一个字母的“文件名”（例如“A”—译者），再按 ENTER 命令。当程序被记录下来后，TRS-80 自动关闭 CTR-41，并在显象管上显示R EADY，于是你的程序就记在磁带上了(它也依然在TRS—80中）。许多计算机用户要在磁带上做第二个甚至第三个记录，他们可以深信他们将获得很好的记录。

3) 应用 CTR-41 的磁带计数器可帮助你在磁带上寻找程序。

4) 要得到最好的结果，请用 Radio shack 的特殊的每一面 10 分钟的计算机磁带盒(是专门为记录计算机程序设计的）。如果用标准的音频磁带盒，一定要用最高质量的，如真正的超级磁带。记住，音频盒带的两头都有引导带（蓝色的非磁性涤纶薄膜），不能在磁带的导带位置上记录，在记录程序前要把引导带走完。

5) 当你不想用 CTR-41 来送入或记录程序时，不要按下 RECORD 键或 PLAY 键（按 STOP 键）。

6) 要倒带或快速走带，按下磁带机的 REWIND 或 FAST-Forward 键，然后再打入 CLOAD 及按 ENTER [注]。当磁带到达所需位置时，按清除钮一它在与扩展接口连接插板的旁边（TRS—80 键盘的后左面）。（还可以把遥控插头从其插孔上拆去，代替使用 CLOAD／复位步骤，但是反复地插入／拆去会使插头损坏，因此不提倡这么做）

7) 如果要永久保存磁带上的程序，请把磁带盒后边的消磁防护片折断(见 CTR-41手册）。

8) 存在磁带上的计算机数据和程序是高保真度记录，应小心处置。必须保存在相对无尘的地方（建议放磁带盒内），并防止高温。磁场和电场也会改变记录信息，因此不要将磁带暴露在磁场中，要避免放在强电源（如变压器，电视机，家用电器设备等）附近。

9) 要检查磁带上是否记有程序，可以从 EAR 插孔上拆去插头（也拆去 REM 插头，于是能用键来控制 CTR-41），放送磁带，用扬声器听有否程序的声音。

10) 为使计算机从录音机上获得最好结果，必须保持盒式录音机处于最佳状况，要定期对它进行保养，必须保持录音机的磁头、磁带传动机构十分干净。脏的磁头会使多达50%的信息丢失，长期使用使磁头磁化由此可能产生失真。因此，建议每工作 4 小时以后要清洁磁头和压辊，磁头要定期消磁。新的磁带机在第一次使用前总是干净的。如果使用不同的录音机，应在放送时拆去 AUX（灰色）插头，在记录时拆去EAR（黑色）插头，这样使进入的

注：对于随机器提供的 CTR—80 磁带机，不需再打入 CLOAD 及按 ENTER—译者

噪声减到最小。

特别注意：

在把磁带上的程序输入计算机以前，一定要将磁带倒回到该程序前的空白部位。如果你从前一个程序的中间开始输入，就可能使计算机“挂起来”（在这种情况下，要按RESET纽并重新开始）。如果在输入程序的过程中，也使用复位(RESET)按钮使盒式机回到手控状态。在使用CLOAD？命令比较带程序与计算机内存贮的程序时，也有相类似的情形。

4. 盒式磁带程序输入的窍门

有许多因素影响着盒式系统的性能，最重要的一个是音量。音量太低会引起一些信息的丢失，音量太高会产生畸变，并把背景噪声作为有用的信息输入。这两种情况都会产生错误。但是，稍加谨慎，就可毫无忧虑地使用盒式磁带机。

由盒式磁带机输入时，建议把音量旋纽放在：〔注〕

	用户产生的	Radio SHack 预录的
LEVEL Ⅱ	4—6	$5\frac{1}{2}$——$6\frac{1}{2}$

由Radio Shack买来的软件程序在输入时音量要稍高些，是因为这些带子是在使用工业音频设备的条件下高速产生的，其记录音量也比用户产生的低些。

当信息开始从磁带上输入的时候，荧光屏右上角将出现两个星号，右边的一个会闪烁（一会儿亮一会儿不亮）每亮一次就读进了一行新的程序或数据。如果星号不出现或者右边一个不闪，那末就是没有正确输入，这几乎总是因为音量不合适。若不出现星号，降低音量，另一个办法是不插EAR（黑色）插头，倾听程序的声音，它将准确地告诉你程序从何处开始。如果出现两个星号，但右边的一个不闪，则增大音量。若较高的音量仍不能解决问题，那末就要清洁磁头。

5. 处置程序输入时的错误

有一种极希罕的情况：在程序输入时只产生一个小错误而不会打印出错误信息。检查的最佳方法是列出程序清单。如果程序看上去没错，可使用CLOAD？命令比较磁带上的程序和已输入的程序，若它们不完全一样，将打印出“BAD”信息。这种情况，只要稍微调正音量（通常稍增大些）就能正常地纠正。

一、概　　述

本章介绍LEVEL Ⅱ BASIC的概况：有哪些特点，与LEVEL Ⅰ有什么差别，以及操作须知。另外章后附有简短词汇表。

1. 打开电源

如前面介绍的那样连接好键盘一计算机，视频显示器和电源变压器。显示器和电源变压器要接入120V交流市电。如果已将磁带机接上TRS—80，在接通电源时，它一定要在

注：数字可参照（Radio Shack）音量旋纽上刻的数字，从1—10（全程量）。不同的录音机，所建议的数字或许不合适，可以做一些试验。

STOP 状态（不是 PLAY，REWIND 等状态），这将避免 TRS-80 内部的磁带机控制继电器的不必要的损坏。

打开视频显示器的电源，让显象管预热一会儿，按下键盘背后的 POWER 钮。

荧光屏将显示出：MEMORY SIZE?

使用特殊命令 SYSTEM，使你有机会保留一部分内存以便输入机器语言的程序。一般使用时，不必保留任何内存，只要按 ENTER 键而不打入任何数字。这样将允许你使用计算机的全部内存容量（对 4 K LEVEL Ⅱ 机器为 3284 字节，16K LEVEL Ⅱ 机器为15572 字节）来书写 BASIC 程序。

注意：通常，无论什么时候你通过键盘打入一些东西，你想让计算机按你的输入来“运行”的话，你必须首先敲 ENTER 键。还有一些方法可使计算机对你敲的键（不用 ENTER）立即响应，但将在后面介绍。

当荧光屏上显示出：

```
RADIO SHACK LEVEL Ⅱ BASIC
READY
>—
```

你就能使用 LEVEL Ⅱ BASIC 了。

2. 操作方式

有四种操作方式：命令、执行、编辑和管理。在命令状态，计算机立即响应进入的命令。这是用来书写程序和直接进行计算的状态。无论什么时候，显示屏上出现>—，就已处在命令状态。

执行方式通常输入 RUN，它使 BASIC 程序被执行。在进入命令 RUN 时，LEVEL Ⅱ首先使所有变量清零，并置所有字符串为零。

编辑方式是 LEVEL Ⅱ 的实时存储特性。它允许你编辑（修改、添加和删除）程序语句的内容。你只需改变你所要改变的部分，而不必重新输入整个程序。

注意：无论何时，计算机在执行过程中发觉句法错误，它就把该语句转入到编辑方式。要从编辑方式退出，打 Q（不要引号）〔注〕。

管理方式，可将机器语言“目标文件”输入内存。这些程序或数据能够与你的 BASIC 程序往来，或者它们可以完全是独立的程序。

3. 特殊功能键

这些功能键的功能取决于计算机处在什么方式中。

命令方式：

ENTER	用作回车，计算机“注视”刚输入的一行语句并按它运行。如果刚打入的一行语句没有语句标号，计算机将对全行的所有语句进行解释和执行。如果有语句标号，计算机把这行语句存入程序内存。
←	光标退格，并删去刚打入的字符。

注：修改以后，要用E—译者

SHIFT ←	删去你正输入的一行，并使光标回到该行的开头。
↓	移行，将光标向下移到显示器的下一个自然行。
:	分离在同一逻辑行上的BASIC语句，允许一行中有多个语句。
→	将光标移到下一个表站（格式区—译注），表站的位置是 0，8，16，24，32,40，48和56。
SHIFT →	将显示器转换到每行32字符的格式。
CLEAR	清除显示器，并使它回到每行64字符的格式。

执行方式：

SHIFT @	暂停。停止程序的执行，按任何键能使执行继续进行。在LIST过程中，清单是不断向上快速移动着的，因此按SHIFT @可使显示器上清单的运动停止（即所谓“冻结”），便于你检查程序。
BREAK	停止执行。要继续执行时须打入CONT。
ENTER	当计算机正在等待从键盘输入时，ENTER使计算机“注视”你输入的内容。

编辑方式的特殊功能键参阅第九节。

4. 变量名称

变量名称必须由一个字母(A—Z)开头，后边可以跟随另一个字母或一个数字(0—9)，因此以下都是正确的代表不同变量的名称：

A、A2、AA、AZ、G9、GP、M、MU、ZZ、Z1

变量名称可以比两个字符长，但是仅开头的两个字符被计算机用来区分不同的变量。如“SU”“SUM”和“SUB”将被LEVEL Ⅱ当作同一个变量来处理。

可见，在LEVEL Ⅱ中可供使用的变量名称是丰富的（900左右）。然而，不能使用BASIC语言中具有特殊意义的词汇（也不能包含）作变量名称。例如“XON”不能作变量名使用，因为它含有BASIC键盘词汇“ON”。不能用作变量名的全部“保留词汇”清单在本手册的附录A中。

5. 变量类型

LEVEL Ⅱ中有四种变量类型：整型、单精度型、双精度型和字符串型（或简称串型）变量。前三个变量用于存放不同精度的数值，最后一种存放字符的串(段)，如字母、空格、数字和特殊符号，可长达255个字符。LEVEL Ⅱ允许使用任何变量名作串型变量，只要简单地在变量名后加一个表明为串型的符号$。其它变量类型也有声明符，下边为全部清单：

变量类型	声明符	例子	存入的典型数值
整型（大于－32769和小于＋32768的全部数字）	%	A%, B%	－30, 123, 3, 5001
单精度型（6位有效数）	!	A!, AA!, Z1!	1, －50, .123456, 354321
双精度型	#	A#, ZZ#, C#	－300.12345678,

(16位有效数)			3.141592653589， 1.000000000000001
双精度科学表示法 (在输入常数或输出很大和很小的数时)	D	"A#=1.2345678901 D+12"	$1.2345678901\times10^{12}$
串型（最多 255 字符）	$	A$,GT$,HJ$	"JOHN Q.DOE"， "WHISTLE—STOP"， "1+2=?"

相同的变量名称可以用于不同的变量类型，计算机仍能区分它们的，因为有类型声明符，例如 A$，A%，A!，A# 区分了不同变量名。

不带声明符的变量就被认为是单精度的，但这个假设可用 DEF 语句改变（第四节）。

6. 数　组

在LEVEL Ⅱ BASIC中，任何正确的变量名都可用作数组的名称。数组并不限于一维。用 DIM 语句在程序开头定义数组。根据使用的变量类型，数组可以存放串型、整型、双精度型等等，本手册有一整节介绍数组。

例如：A$(X,Y,Z)是一个存放串型值的三维数组。

G3(I,J) 是一个存放单精度数值的二维数组。

G#(I) 是一个双精度的一维数组。

7. 算术运算符

LEVEL Ⅱ 使用的算术运算符有：

+（加），　　-（减），　　*（乘），　　／（除）以及

↑（指数：2 ↑ 3 = 8）

例如：要计算 $6*2^{1/3}$，可写成：PIRNT 6 * 2 ↑(1/3)

注意：有些TRS—80用一个[符号代替箭头↑。

8. 关系算符

<（小于），　　>（大于），　　=（等于），　　<>（不等于）

<=（小于或等于），　　>=（大于或等于）

这些运算通常用于 IF……THEN 语句，及用于逻辑运算。

例如：100　IF　C<=0　THEN　C=127

9. 逻辑算符

在 LEVEL Ⅱ 中有逻辑运算的与、或、非，直接使用算符 AND，OR 和 NOT。

例如：

```
50  IF Q=13   AND   R2=0   THEN   PRINT   "READY"
100 Q=(G1<0)  AND  (G2<L)  若两表达式都为真 Q=-1，否则 Q=0
200 Q=(G1<0)  OR   (G2<L)  若任一表达式为真 Q=-1，否则 Q=0
300 Q=NOT (C>3)            若表达式为假 Q=-1，  表达式为真 Q=0
```

```
400 IF NOT (P AND Q)THEN PRINT"P AND Q ARE NOT BOTH EQUAL
    TO-1"
500 IF NOT (P OR Q) THEN PRINT "NEITHER P NOR Q EQUALS-1"
```

10. 字符串运算符

在 LEVEL Ⅱ 中，字符串可以比较和连接“串在一起”。本手册有一整节介绍串的操作。

符 号	意 义	举 例
<	按字母序例(ABC……)比较，次序在前	“A” < “B”
>	按字母序例比较，次序在后	“JOE” > “JIN”
=	相等	B$= “WIN”
<>	不相等	IF A$<>B$ THEN PRINT A$
<=	在前或相等	IF A$<=AZ$ PRINT “DONE”
>=	在后或相等	IF L1$>= “SMITH” PRINTL1$
+	连接两个字符串	A$=C$+C1$ A$= “TRS-” + “80”

11. 运算的优先等级

首先执行最内层括号中的运算，然后一层层往外执行。同一括号中的运算按下面优先级别执行：

↑ (指数：A↑B)
- (取负值：-X)
*,／ (同一优先等级，运算自左至右，下同)
+,-
<,>,=,<=,>=,<>
NOT
AND
OR

12. 内部函数

LEVEL Ⅱ 内具有使用时很快、很精确和很方便的内部函数（参阅第七节）。

13. 图 示

LEVEL Ⅱ 有 SET, RESET, 和 POINT 功能，用于开、关图块和检查图块的开或关（参阅第八节）。

LEVEL Ⅱ 的一大特点是可选择显示器—或者每行 64 字符（c/l），或者每行 32字符。机器一开时它就处在 64c/l 状态，同时按 SHIFT 键和→键就改变为 32c/l。任何时候，只要一执行 CLS 或 NEW，或者按CLEAR键，显示器就回到6 4c/l。执行PRINT CHR $(23)也能使显示器成为32c/l。更多内容将在第五节介绍。

14. 错误信息

LEVEL Ⅱ 使用一组错误代码（见附录2），给出了关于错误类型的专用信息，也指出了犯错误的程序的语句标号，以便寻找程序中的错误。

也有一些难以发现的错误：〔注〕

[SHIFT] 字符和对应的非 [SHIFT] 字符总是不能互换的，例如，PRINT@ 中如果使用了，[SHIFT] @，则不能工作，但是你在荧光屏上看起来是不错的。如果对一行给出句法错误信息的语句，找不出任何错误的话，再重打入这一行，特别警惕 [SHIFT] 键。

空格在 LEVEL Ⅱ BASIC 中有时是重要的。下行语句是错误的：

```
IF  D>0  D=0
```

因为其中的 0D 说明了“双精度零”的意思。应加上空格并改成：

```
IF  D<0  THEN  D=0
```

另外，如果当你按某些键时，经常出现按一次键却输入两个相同字符的话，则取下塑料键帽，用一片硬纸小心地清洁触点：在触点之间插入纸片，按下键使触点压在纸上，再把纸片拉出来。

15. 缩　写

LEVEL Ⅱ 允许极少数缩写，使你在一个较小的内存中仍能输进较多的程序。

缩写有：

? 作 PRINT

' 作 REM

· 代表最后一行。在打印清单，编辑或删除等命令中代替最后一行的语句标号。

16. 键盘的连打

对 LEVEL Ⅰ TRS-80（及其他许多计算机），在计算机允许输入另一个键以前，你必须松开前一个键。LEVEL Ⅱ 允许你在放开第一个键之前就按下第二个键。当你是一个熟练的打字员时这对于你一定是愉快的。

LEVEL Ⅱ BASIC 词汇表

（内容略——译者）

address	地　址	hexadecimal number	十六进制数
alphanumerics	字　符	intrinsic function	内部函数
argument	自变量	logical expression	逻辑表达式
array	数　组	machine language	机器语言
ASCII	信息交换美国标准代码	port	出/入　口
assembler	汇编程序	RAM	随机存储器

注：本段译自“TRS—80 LEVEL Ⅱ 注意事项”之7、9两节—译者。

BASIC	初学者通用符号指令代码的第一个字母缩写	ROM	只读存储器
		routine	程　序
baud	波特——信息传输速率	statement	语　句
binary number	二进制数	string	字符串（简称串）
bit	比特——二进制位	subroutine	子程序
byte	字节——最小存储单元	variable	变　量
decimal number	十进制数	variable name	变量名称
expression	表达式		
file	文　件		

二、　命　　令

无论何时，一出现>一，就表明计算机已处在命令状态，你可以打入命令，并按 ENTER，计算机将立即响应。本节介绍用于控制计算机的命令：改变状态、开始输入和输出程序、修改程序内存等等。所有这些命令—除 CONT—也可用在你的程序内作为语句，在某些情况中，这是很有用的，其它时间它们只作特殊的应用。

本节介绍的命令有：

AUTO	CONT	LIST	TROFF
CLEAR	CSAVE	NEW	TRON
CLOAD	DELETE	RUN	
CLOAD?	EDIT	SYSTEM	

1. AUTO　语句标号，增量

它打开一个执行自动编写语句标号的功能，使用户输入程序时更为方便——不必为每一行语句一一手编语句标号，只要输入实际的程序语句。可以指定一个开始的语句标号和两相邻语句标号之间的增量。如果只简单地输入 AUTO 及按 ENTER，那末语句标号从10开始，增量为10。你每按一次 ENTER，计算机就显示出下一条语句标号。

例如：	自动给出的语句标号
AUTO	10，20，30，……
AUTO5，5	5，10，15，……
AUTO 100	100，110，120，……
AUTO 100，25	100，125，150，……

要停止执行 AUTO 功能，按 BREAK 键。（注意：当 AUTO 给出一个已被使用的语句标号时，将在该语句标号旁边出现一个星号。如果你不想重编该语句，则按BREAK键，停止执行 AUTO 功能）

2. CLEARn

当不带自变量的时候（即输入 CLEAR 和按 ENTER），这个命令把所有数字变量

清零，所有字符串变量也置零。当带有自变量n时（例如CLEAR100），这个命令除完成上述功能外还执行第二个功能：它开辟了指定数目(n)个字节作字符串存贮单元。

例如：CLEAR 100，为字符串开辟了100字节的存贮单元。

在你一打开机器时，计算机自动执行 CLEAR 50。

3. CLOAD "文件名"

命令从盒式磁带机上输入一个 BASIC 程序。此时应将盒式磁带机置 play 状态（必须正确的连接好并将盒带倒回适当位置）。

注意：在 LEVEL Ⅱ 中，CLOAD 和 CSAVE 操作以 500 波特的传输速率进行，两者都比 LEVELl 的传输速率快一倍，因此在 CLOAD 过程中使用的音量旋纽应当相应低些。例如，如果你使用 Radio Shack的CTR-41 盒式磁带机，那么当你把存放在磁带上的程序或数据输入时，要把音量控制放在 4—6 之间。若输入 Radio Shack 预录的程序，音量要稍高些。也可作一些试验。

执行CLOAD 命令时（不带"文件名"—译者），它将启动盒式机的马达，并把遇到的第一个程序输入计算机。也可在CLOAD语句中规定一个所需要的"文件名"。例如：CLOAD "A"，将使计算机只输入一个文件名为"A"的程序，而对带上的其它程序都不予理采。因此，不论文件"A"存在磁带的什么地方，你都可以从磁带的开头开始放送，文件"A" 将被从磁带上的所有程序中找出来并输入到计算机中去。在计算机检索文件"A"的过程中，遇到的每一个文件名将出现在显示器的右上角并伴随一个闪烁的*号。

在CLOAD,CLOAD? 和CSAVE命令中，计算机只使用文件名中的第一个字符。

从磁带上输入一个程序，将自动地清除以前存贮在计算机中的程序。（参阅 CSAVE）

4. CLOAD? "文件名"

命令计算机比较存储在磁带上的程序与现在计算机里的程序。当你往磁带上转贮一个程序（见 CSAVE）并希望核对传输是否成功时，这个命令是非常有用的。如果在你 CSAVE 一个程序时已标明文件名，那末应使用命令：CLOAD? "文件名"。相反，若你没有规定文件名 那末只核对碰到的第一个程序。在 CLOAD? 过程中，带上的程序和内存中的程序被一个字一个字地进行比较，若有任何差异（指转储不好），将给出信息，"BAD"，这时，你应重新 CSAVE 程序。（CLOAD? 和 CLOAD 不同，它不清除程序内存）

5. CONT

当程序执行过程中产生暂停时（由于按了 BREAK 键或遇到了程序中的 STOP语句），要从停止或中断的地方继续往下执行，打入 CONT 和按 ENTER。在执行过程中的这种暂停或中断期间，你可以检查变量的值（用 PRINT）或改变这些数值，然后打入 CONT 和按 ENTER，程序将以新的变量值继续进行运算。与STOP语句和 BREAK 键一样，CONT 命令主要用于调试程序。

注意：在用 EDIT 编辑程序或用其他方式改变程序以后，不能使用 CONT。在执行正

常结束以后，CONT也无效。

6. CSAVE “文件名”

将机内程序存到盒式磁带上去。(在输入CSAVE命令以前，盒式磁带机必须正确连接上，放上磁带，放在记录状态)，这一命令必须带有规定的文件名。文件名可以是除双引号(“)以外的任何一个字符符号。于是，程序就按规定的文件名存入磁带，因此，当用CLOAD命令去访问各文件时，就能按这个文件名找到它。为了便于以后使用，应该在磁带盒上写上相应的文件名。

例如：

CSAVE “1”　　转储机内程序并加上标记“1”

CSAVE “A”　　转储机内程序并加上标记“A”

7. DELETE 语句标号—语句标号

从内存中删除程序行。你可以删除一行或一段，方法如下：

DELETE 语句标号　　删除指定的程序行

DELETE 语句标号—语句标号　　删除指定的两语句标号之间的所有程序行(包括该两行，下同)

DELETE—语句标号　　删除指定语句标号之前的所有程序行

以上删除的语句标号必须是现在已使用的那些号码。

例如：

DELETE 5　　从内存删除语句标号为5的程序行（若标号5没用过，将产生错误)

DELETE 11—18　　删除包括语句标号11、18在内的两者之间的所有程序行

如果你刚输入或编辑一行，要删除这行，只要简单地使用DELETE。（用一个句点代替该语句标号)

8. EDIT 语句标号

使计算机进入编辑方式，以修改存贮在内存中的程序。你的程序越长、越复杂，那末你越会感到EDIT的重要及便利。在EDIT方式中可选择它自己的子命令，这将在第九节讲述。

9. LIST 语句标号—语句标号

命令计算机显示现在存贮于内存中的所有程序行。

如果LIST不带自变量（语句标号—译者注)，则全部的程序将连续地在荧光屏上向上推移着显示出来。要停止这种自动的推移，则同时按下SHIFT键和@键，可使屏幕上移

动着的内容停下来（即所谓“冻结”）。再按任何键，即可解除“暂停”并继续自动地向上推移。

要检查一下某行，则将所需的语句标号规定为LIST命令的自变量。

要检查某一段程序，则指定该段的第一行和最后一行的语句标号。

例如：

LIST 50	显示行 50
LIST 50—150	显示行 50至行 150 之间的每一行
LIST 50—	显示语句标号大于 50 的所有各行。
LIST .	显示现在行（刚输入或刚编辑的行）
LIST —50	显示行 50 以前的所有行（包括行 50）

10. NEW

清除所有程序，置数字变量为零，置串变量为零。但它不能改变由前面的 CLEARn 语句所规定的串空间。

11. RUN 语句标号

命令计算机执行内存中存放的程序。若不规定语句标号，那末从最小语句标号的程序行开始执行。如果指定语句标号，就从该语句标号开始执行（若指定一个未被使用的语句标号，将出错）。一旦执行 RUN 命令，计算机同时执行 CLEAR 命令。

例：		
	RUN	从最小数目的语句标号行开始执行
	RUN100	从语句标号 100 开始往下执行

RUN 可以用在程序中作为一个语句，对于象游戏这样的循环程序，这是一个重复地清除和开始运行的方便办法。

12. SYSTEM

使计算机进入管理方式，它允许你输入目标文件(机器语言程序或数据)。Radio Shack 提供几种机器语言软件程序库，例如内存信息系统。(IN—MEMORY INFORMATION SYSTEM)。你也可以使用 TRS—80 编辑程序／汇编程序—其本身是一个目标文件—来建立你个人的目标文件。

要输入目标文件：打 SYSTEM 和按 ENTER 键。

?出现时，就输入文件名（不要加引号），磁带开始输入，当输入完成时，显示出另一个?，这时输入一个斜号／，随后输入你希望由此开始执行的地址（十进制），或者，可以简单地打入斜号和 ENTER，而不要任何地址。这种情况，执行从目标文件指定的地址开始。

13. TROFF

关闭执行跟踪功能，参看 TRON 命令。

14. TRON

打开执行跟踪功能，使你跟踪程序流程以调试程序和分析执行的情形。当程序每执行一条新的程序行时，其语句标号就显示在一对括号里。

例如，输入下列程序

```
10  PRINT  "START"
20  PRINT  "GOING"
30  GOTO   20
40  PRINT  "GONE"
```

现在打入TRON，[ENTER]和RUN，[ENTER]。

你将从显示器上看到，此程序进入了一个无底的循环：

```
〈10〉   START
〈20〉   GOING
〈30〉     〈20〉  GOING
〈30〉     〈20〉  GOING
          ⋮
```

（同时按[SHIFT]和@，可暂停执行并“冻结”显示器，如果再按任何键仍能继续往下执行）

括号内的数字向你精确地表示正在执行什么（要停止执行，敲[BREAK]键）。

要关闭跟踪功能，则进入 TROFF 命令。

TRON 和 TROFF 可以写在程序中，以便帮助你了解某些给定语句的执行情况。

例如：

```
50  TRON
60  X = X*3.14159
70  TROFF
```

这有助于了解行 60 的执行情况（假设执行过程中没有绕过 50 而直接跳到60），每一次执行这三行程序时，＜60＞＜70＞ 将显示出来。没有 TRON，你就不知道程序是否确实执行了行 60。在程序故障（错误）被排除以后，TRON 和 TROFF 这两语句就可以删掉。

三、 输入 / 输出语句

本章所述的语句使你能把数据从键盘送入计算机，或从计算机送到显示器，以及在计算机与盒式磁带机接口间进行数据交换。它们在程序中主要用于输入数据，输出结果和信息。

本章涉及的语句有：

```
PRINT
        @          (PRINT 的修饰项)
        TAB        (PRINT 的修饰项)
        USING      (PRINT 的格式项)
```

INPUT
DATA
READ
RESTORE
PRINT ＃　　（输出给盒式磁带机）
INPUT ＃　　（由盒式磁带机输入）

1. PRINT 项目清单

在显示器上打印出一个项目或许多项目。项目既可以是串常数（信息放在引号里），串变量，数字常数（数目），数字变量，也可以是包含有上述项目的表达式。要打印的项目可以用逗号（，）或分号（；）把它们隔开。如果使用逗号，则在打印下一个项目以前，光标自动地进到下一个打印区。如果使用分号，则在显示器上两条打印项目之间不插入空格。

例如：

```
50    X=5
100   PRINT25; "IS  EQUAL  TO"; X↑2
RUN
△25△IS△EQUAL△TO△25〔注〕
```

……

```
10    A$="STRING"
20    PRINT A$; A$, A$; "△"; A$
RUN
STRINGSTRING△△△△STRING△STRING
↑                    ↑
0                    16
```

打印出的正数前面有一个空格（代替＋号），打印出的所有数字后边有一个空格。而字符串的前后都不插入空格（如20，可将空格放引号里）。

```
10  PRINT "ZONE 1",  "ZONE 2",  "ZONE 3", "ZONE 4",
    "ZONE 1 ETC"
RUN
ZONE△1          ZONE△2          ZONE△3          ZONE△4
↑               ↑               ↑               ↑
0               16              32              48
ZONE△1△ETC
↑
0
```

每行分成四个打印区，每个区占16个字符的宽度

……

```
10 PRINT "ZONE1", , "ZONE3"
RUN
```

注：本章举例中，译者用△代表空格，并在有些输出信息下边标注出数字（例如$\overset{\uparrow}{16}$）表示显示器X轴上TAB的位置。—译者

```
ZONE△1                                  ZONE△3
↑                                       ↑
0                                       32
```

每遇到一个逗号，光标就移到下一个打印区。

……

```
10  PRINT "PRINT    STATEMENT#10";
20  PRINT "PRINT    STATEMENT#20"
RUN
PRINT△STATEMENTE#10PRINT△STATEMENT#20
```

用分号结尾使光标不回车换行，因此下一条 PRINT 语句就接着上一条的结尾处开始往下打印（见语句 10）。如果 PRINT 语句不用标点结尾，光标就自动回车换行，处于下一行的起始点。

2. PRINT @ 位置，项目表

精确地按规定位置开始打印信息。修饰项@必须紧随 PRINT，规定的位置必须是0—1023的数字〔注1〕

```
100  PRINT@550,  "LOCATION550"
```

运行此程序，找一找位置为 550 的那一点。

无论何时，如果你在显示器的底线上用了 PRINT @，则将因为自动的回车换行，使显示的每行内容都向上移动一行。要防止这种现象产生，可以在语句结尾使用一个分号。

如：

```
100  PRINT@  1000,  1000;
```

3. PRINT TAB （表达式）

将光标移到本行的指定位置(如果你指定的TAB位置大于63的话，将移到下一行)。〔注2〕TAB 可以在一条 PRINT 中多次使用，表达式的值必须在 0 和 255 之间。

例如：

```
10  PRINT  TAB(5) "TABBED 5"; TAB(25) "TABBED 25"
```

这条语句打出的结果是：〔注3〕

```
△△△△△TABBED△5          TABBED△25
↑    ↑                  ↑
0    5                  25
```

在修饰项 TAB 后边不需要加任何标点。

```
5    X = 3
10   PRINT  TAB(X) X;  TAB(X ↑ 2) X ↑ 2;  TAB(X ↑ 3) X↑3
```

输出的结果是：

```
△△△△3          △9          △27
↑   ↑          ↑           ↑
0   3          9           27
```

注 1：位置及项目可以是常数，变量，表达式。项目之间可用逗号或分号隔开—译者

注 2：实际上不换行，打印的位置在（指定 TAB 位置减 64）上一译者

注 3：为帮助读者理解 TAB 的功能，本例及下例的结果是译者添加的—译者

可以用数学表达式来指定TAB的位置，这在利用TAB来作数学函数的曲线，图表时非常方便。

TAB不能用来使光标往左移动。如果光标已超过指定位置，这时TAB也不起作用了（相当；号的作用一译注）。

4. PRINT USING 字符串；项目清单

PRINT USING—这一语句使你可以为打印字符串和数字变量指定某种格式。这语句有很多的用途，例如可用来打印报告标题、帐单、发票或者需要特定打印格式的其它场合。

PRINT USING 语句用下格式：

PRINT USING 字符串；数值

字符串和数值可以表达为变量或常数。这语句把包含在字符串中的表达式打印出来，并把分号右边的数值按照字段说明符插入上述字符串表达式中。

在字符串中可使用下列字段说明符；〔注〕

\# 这符号规定数值中每一位数的位置。用几个#号就表示打印的数有几位（数字段），若打印的数字段大于数值的实际位数，那末多余的位数将在数的左边以空格表示，小数点的右边将以零表示。

小数点可以放在由#组成的数字段的任何地方。当小数点右边的值被抑制的时候，将四舍五入。

当逗号放在第一位数与小数点之间的任何位置时，打印中将在每三位的左边出现一个逗号。逗号在字段中占一位。

* * 在字段的开头放双星号，将使小数左边所有没有数值的位置用星号填充。双星号在字段中占两位。

$ $ 两个美元符号放在字段的开头，表示浮动的美元符号，也就是在打印时将把一个美元符号放在数值的紧前边。

* * $ 如果在字段的开头使用这样三个符号，那末$仍在数字的紧前边，而它左边空着的位置将充满*号。

\+ 当一个+号放在字段的开头或结尾时，那末按它所在位置，正数则打印+号，负数打印-号。

\- 当字段的末尾放一个-号时，那末所有的负数后边将打印一个负号，而正数为一个空格。

%空格% 如果要打印一个字符串，字符的个数多于一个字符，则可使用此%空格%，字符串段的长度等于百分号之间的空格数目再加2。

! 使计算机只打印紧随着的字符串的第一个字母。

下面的程序将有助于说明这些格式规定：

```
10 INPUT A$, A
20 PRINT USING A$; A
30 GOTO 10
```

注：其中缺少指数格式↑↑↑↑，参阅附录1的5. PRINT USING 格式规定符—译者。

运行此程序试验各种规定，给 A$ 赋以字符串，而给 A 赋以不同的值。

例如：

```
RUN
    ? ##.##,12.12
    12.12
    ? ###.##,12.12
    △12.12
    ? ##.##,121.12
    %121.12
```

如果字段的长度小于被打印的数值的位数，将自动印出%号，在它后边能把小数点左边的全部数字打印出来。

```
    ? ##.##,12.127
    12.13                    注意，这个数已舍入成两位小数
    ? +##.##,12.12
    +12.12
    ? +##.##,-12.12
    -12.12
    ? ##.##+,12.12
    12.12+
    ? ##.##+,-12.12
    12.12-
    ? ##.##-,12.12
    12.12
    ? ##.##-,-12.12
    12.12-
    ? **##, 12.12
    **12
    ? **##.##, 1212.12
    1212.12
    ? $$##.##, 12.12
    △$12.12
    ? "##, #####", 12121.2
    △12,121
    ? "#####,#", 12121.2
      12,121
    ? ###, 1212
    %1212
```

使用 PRINT USING 语句的另一个方法是把字符串段规定为“!”和“%空格%”。

例如：

```
PRINT USING“！”；字符串
PRINT USING “%△△△%”；字符串
```

“！”号只允许印出字符串的第一个字母。

“%空格%”允许打印的字母个数为空格数加2。另外，字符串和规定符可以表达成串变量。下边的程序将说明这一特点：

```
10 INPUT A$, B$
20 PRINT USING A$; B$
30 GOTO 10
RUN
? !, ABCDE
△A
? %%, ABCDE
△AB
? %△△%, ABCDE
△ABCD
```

用这些规定符能把多个字符串或串变量连接起来。但“！”号允许把每个字符串的第一个字母打印出。例如：

```
10 INPUT A$, B$, C$
20 PRINT USING“!”; A$; B$; C$
RUN
? ABC, DEF, GHI
△ADG
```

使用多个“！”号时，只是在打印出的每个字符串的第一个字母之间插入空格而已，插入的空格数与“！”之间插入的空格数相等。为说明这一特点，对上面的程序作如下修改：

```
20 PRINT USING “!△!△!”; A$, B$, C$〔注〕
RUN
? ABC, DEF, GHI
△A△D△G
```

现在在字母ADG之间出现了空格，它相应于三个“！”号之间的空格数。

可以试一试把20语句中的“！！！”改成，“%%”，然后运行此程序。

下边的程序说明了PRINT USING语句的一种可能的应用：

```
10 CLS
20 A$ =“* * $##, ########.##DOLLARS”
30 INPUT “WHAT IS YOUR FIRST NAME”; F$
40 INPUT “WHAT IS YOUR MIDDLE NAME”; M$
50 INPUT “WHAT IS YOUR LAST NAME”; L$
60 INPUT “ENTER THE AMOUNT PAYABLE”; P
70 CLS: PRINT “PAY TO THE ORDER OF”;
```

注：项目清单中的项目之间用逗号与用分号无区别一译者。

```
80  PRINT USING "!!△!!△"; F$; "."; M$; ".";
90  PRINT L$
100 PRINT: PRINT USING A$; P
110 GOTO 110
```

运行此程序。要记住，为节省编程序的时间，可用“？”号代替 PRINT。上述程序将在显示器上出现如下内容：

```
WHAT IS YOUR FIRST NAME? JOHN
WHAT IS YOUR MIDDLE NAME? PAUL
WHAT IS YOUR LAST NAME? JONES
ENTER THE AMOUNT PAYABLE? 12345.6
PAY TO THE ORDER OF J. P. JONES
* * * * * * * $12,345.60 DOLLARS
```

如果你想使用一个大于 999,999 的数值又不四舍五入，或者又不采用科学表示法，那末只要简单地在语句 60 和语句 100 的变量 P 后边加上双精度符号(#)，于是你用的数值可长达 16 位有效数。

5. INPUT 项目清单

使计算机暂停执行，等待你通过键盘输入规定的数值。INPUT 语句可以规定一列字符串变量或数字变量清单作为输入，清单中的每一项之间必须用逗号隔开。

```
100 INPUT X$, X1, Z$, Z1
```

这语句要求你的输入次序为：一串文字，一个数值，再一串文字和一个数值。当计算机遇到 INPUT 语句时就显示：

```
? -
```

你可以一次输入所有的值，也可以一次只输入一个值。要一次输入所有的值，则要在每个值之间用逗号把它们隔开。（如果在一串文字中有开头的空格或者有冒号、逗号等，则必须用引号将这串字符括起来）

例如，上述语句 100 在执行时，计算机等待着你的输入，你可输入：

```
JIM, 50, JACK, 40        (ENTER)
```

计算机将按下列方法赋值：

```
X$ = "JIM"  X1 = 50  Z$ = "JACK"  Z1 = 40
```

如果一次只输入一个值（打一个值就按 ENTER），那末计算机将显示出：

```
? ? -
```

等待着输入后几个数据，要继续输入数据，直到所有的变量都赋给了值，计算机才进入程序的下一条语句。

输入数值的类型一定要与 INPUT 语句中规定的类型相一致。例如：你不能将一个字符串值输入给一个数字变量，假如你这样作了，计算机将显示出：

```
? REDO   (重新输入)
? -
```

让你重新输入正确类型的数值，但要从INPUT语句清单中规定的第一个值起开始输入。

注意：不能将一个表达式输入给一个数字量，而必须输入一个简单的数字常数。

例如：

```
100  INPUT X1, Y1$
200  PRINT X1, Y1$
RUN
? -（你打入：）7+3（按 ENTER 键）
? REDO
? -（你打入：）10   （按 ENTER 键）
? ? -（你打入：）"THIS IS A COMMA:,"
   10          THIS  IS A COMMA:,
```

在"THIS IS A COMMA:,"上必须加引号，因为字符串中包含有逗号。

如果你输入的数据的个数多于INPUT语句所规定的个数，那末计算机将显示如下信息：

```
? EXTRA  IGNORED      (多余的数据被忽略)
```

并继续往下正常地执行程序。

在INPUT 语句中也可以包含"提示信息"，这使得数据能更正确地输入。提示的信息必须紧跟在INPUT 后面并放在引号内，后边再跟随一个分号。

例如：

```
100  INPUT"ENTER YOUR NAME AND AGE (NAME,AGE)"; N$,
 (RUN)
 ENTER YOUR NAME AND AGE(NAME,AGE)? -
```

6. DATA 数据清单

给你在程序中存放数据，以便由 READ 语句读取。读取数据时，是从第一条 DATA语句中的第一个数据开始，以最后一条 DATA 语句中的最后一个数据为结尾，按顺序往后读取。DATA 中的数据可以是字符串也可以是数字常数，而不允许是表达式。如果字符串值中包含有开头的空格或者含有冒号及逗号，则必须将这样的字符串值用引号把它们全部括起来。

注意：DATA 语句中数据的类型与 READ 语句中变量的类型必须相符。

DATA 语句可以放在程序中的任何地方，通常把它们顺序地放在一起，但并不是必须如此。

例如：

```
 500  READ N1$, N2$, N1, N2
1000  DATA "SMITH,J.R.", "WILSON, T.M."
2000  DATA 150, 175
```

参阅〈READ〉，〈RESTORE〉。

7. READ 变量清单

命令计算机由 DATA 语句读取数值，并把这些数值赋给指定变量。第一次执行一个READ时，将使用第一条DATA语句中的第一个数值，第二次读取该 DATA 语句的第二个数值。当第一条 DATA 语句中的全部数值被读完以后，下一个 READ 将使用第二条DATA语句中的第一个数值，……。（如果READ 要读取的数据的数目超过了 DATA 所包含的数目，则显示出错误信息 OD—数据出界）下列说明了 READ／DATA 语句的应用：

```
 50 PRINT "NAME",   "AGE"
100 READ N$
110 IF N$ = "END" PRINT "END OF LIST": END
120 READ AGE
130 IF AGE<18 PRINT N$, AGE
140 GOTO 100
150 DATA "SMITH, JOHN", 30, "ANDERSON, T.M.", 20
160 DATA "JONES, BILL", 15, "DOE, SALLY", 21
170 DATA "COLLINS, W.P.", 17, END
RUN
NAME                AGE
JONES, BILL          15
COLLINS, W.P.        17
END OF LIST
READY
>-
```

程序从供给的数据中判明并打印出所有未成年者的姓名、年龄。注意，其中使用了一个END 字符串，以便知道数据已被读完。

参阅〈DATA〉，〈RESTORE〉。

8. RESTORE

使下一个 READ 语句在执行时由第一条 DATA 语句中的第一个数据开始重新读取。它使程序能够重复使用相同的DATA语句。

例如：

```
100 READ X
110 RESTORE
120 READ Y
130 PRINT X, Y
140 DATA 50, 60
```

```
RUN
△50              50
READY
>-
```

由于 RESTORE 语句，第二条 READ 语句仍从 DATA 语句的第一个数据重新开始读起。

参阅〈READ〉，〈DATA〉。

9. PRINT#-1, 项目清单

把指定变量的数值写到盒式磁带上。（当执行这一语句时，必须正确连接好磁带机并处在记录状态）。PRINT#语句一定要规定一个设备号，这是因为 TRS—80 实际上能输入／输出给两台盒式机，这将在后面的第十章附加扩展接口中进行介绍。对于只连接一台盒式磁带机的正常使用情况，设备号必须是－1，也就是PRINT#－1（然后是逗号及项目清单）。

例如：

```
5        A1 = -30.334 : B$ = "STRING-VALUE"
10       PRINT#-1, A1, B$, "THAT' S ALL"
```

这样就把 A1 和 B$ 的当时数值以及字符串文字 "THAT' S ALL" 存入了磁带。以后可以应用 INPUT# 语句由磁带上把这些值输入。INPUT#语句和 PRINT# 语句两者的清单在项目的个数和项目的类型上必须完全一致，参阅〈INPUT#〉。

特别注意：

项目清单中表示的数值总共不能超过 255 个字符，否则在 255 个以后的所有字符将被截断。例如，PRINT#－1，A#，B#，C#，D#，E#，F#，G#，H#，I#，J#，A$ 如果 A$ 长于 75 个字符，那末就可能超过允许的最大记录长度，在这种情况下，将不记录A$，而当你试图输入（INPUT#－1）这些数据时就产生数据出界（OD）错误。

因此，如果你需要记录（PRINT#）的清单很长，为不丢失最大记录长度以后的信息，你应该将此清单拆成两条或更多的PRINT#语句。[注1]

10. INPUT#-1 项目清单

从盒式磁带上输入规定的数值，并把它们赋给指定的变量。

和 PRINT# 语句一样，INPUT# 语句也要求你给指定一个设备号（当你加了扩展接口，并使用两台盒式磁带机系统时，这更加重要，参看第十节）。在没有扩展接口的常规应用中，使用设备号为－1，即INPUT#－1，项目清单。

例如：

```
50  INPUT#-1, X, P$, T$
```

当执行这一语句时，计算机启动盒带机的马达，按规定次序输入数值，然后关闭盒带机马达，继续执行下一条语句。如果在 INPUT 清单要求一个数字时，却遇到了一个字符串，

注1：本段根据附页："LEVEL I TRS—80 使用须知" 编写的。关于最大记录长度，须知说每条 PRINT# 不能多于248 字节（byte）。

那末将产生数据文件不合适(FD)的错误。如果在磁带上没有足够的数据以“满足”INPUT语句的需要，那末产生OD错误。

INPUT 清单必须与建立磁带数据块的 PRINT 清单完全一致（相同的变量数目，相同的变量类型以及相同的排列次序）。

有些 TRS-80 在执行 INPUT#-n 以后，不能正确地用 READ 语句从 DATA 语句读取数据，而在每次 READ 之前自动执行 RESTORE 语句，于是总是只读取第一个 DATA 项。如果你的 TRS-80 也如此工作（由于一些集成电路由同一个电源供电）。对此有一个简单的调整方法：在每条INPUT#-n语句后直接插入语句：〔注〕

```
POKE16553,255
```

程序举例：

用在 PRINT# 说明中（本节第九小节）提供的两条程序产生一个简短的数据文件，然后把磁带倒回到数据文件的开头，作所有必要的连接，并将盒式磁带机置 PLAY 状态，运行下列程序。

```
10  INPUT#—1, A1, B$, L$
20  PRINT A1, B$, L$
30  IF L$ = "THAT' S ALL" END
40  GOTO  10
```

这个程序将不管数据文件有多么长，只要：

①由PRINT# 语句产生的文件必须和语句 10 在形式上一致。

②在最后的三个一组的数据中，最后一个数据必须是“THAT' S ALL”。

四、 程 序 语 句

LEVEL Ⅰ BASIC作了一些有关程序运行的假设，如：

* 没有类型说明的变量都假设为单精度变量。
* 为字符串和数组自动地开辟了一部分内存一而不论你是否全部使用它们。
* 程序按顺序执行，从程序的第一条语句开始到最后一条为止。

本章介绍的语句，使你能越过这些假设，给了程序更大的灵活性，以便适应各种情况。

注意：所有的 LEVEL Ⅰ 语句，除 INPUT 和 INPUT# 以外，都能够在命令方式及执行方式下使用。

本章介绍的语句有：

类 型 定 义	赋值或分配	执 行 次 序	试验（条件语句）
DEFINT	CLEAR n	END	IF
DEFSNG	DIM	STOP	THEN
DEFDBL	LET	GOTO	ELSE
DEFSTR		GOSUB	
		ON……GOTO	

注：本段译自附页“LEVEL Ⅰ TRS—80 使用须知”—译者。

ON……GOSUB
FOR…NEXT…STEP
ERROR
ON ERROR GOTO
RESUM
REM

本章还包括 LEVEL Ⅱ BASIC 中的数据转换的讨论，这使你能预测和控制表达式、常数等产生的结果，这些结果将作为整数、单精度和双精度存储起来。

1. DEFINT 字母范围

用它指定范围内的任何字母开头的变量，都将作为整型变量存储和处理，除非在变量名后附加上别的类型说明符，按说明符处理。使用整型变量可节省内存，因为整型值占有的内存比其他数字类型少，整型运算比单精度和双精度型运算要更快。但是，定义为整型的变量只能在 －32768与＋32767 范围内取值。

例如：

```
10  DEFINT A, I, N
```

在行 10 以后，所有以 A，I 或 N 开头的变量都作为整型处理，如 A1，AA，I3 和 NN 就都是整变量。而 A1#，AA#，I3#，仍然作为双精度变量，因为类型说明符总是不受 DEF 语句的限止。

```
10  DEFINT  I—N
```

使得以 I, J, K, L, M 和 N 开头的变量都作为整型变量处理。

DEFINT 可以放在程序中任何地方。由于它会改变那些没有类型说明符的变量的类型属性，因此，通常把它放在程序的开头。

参阅〈DEFSNG〉，〈DEFDBL〉和第一节的“变量类型”。

2. DEFSNG 字母范围

指定某个范围内的字母开头的变量作为单精度变量存储和处理，除非变量名后加有其他类型的说明符。单精度变量和常数以 7 位精度存储而以 6 位精度印出。由于所有的数字变量：除了定义为其它类型的以外，都被假设为单精度变量，因此，DEFSNG 语句主要用于新定义那些前面已被定义或双精度型成整型的变量。

例：

```
100  DEFSNG I, W—Z
```

使得字母 I 或任何 W 至 Z 之间的字母开头的变量都作为单精度变量处理。但是 I% 仍是整变量，I#是双精度变量，因为它们使用了类型说明符。

参阅〈DEEINT〉，〈DEFDBL〉和第一节的“变量类型”。

3. DEFDBL 字母范围

指定某个范围内的任何字母开头的变量作为双精度变量存储和处理，除非加有其它类型

的说明符。双精度型允许 17 位精度，但双精度变量被输出时只显示出 16 位。

例如：

```
100  DEFDBL  S—Z, A—E
```

使得以字母 S 至 Z 或 A 至 E 之一开头的变量作为双精度变量。

DEFDBL 语句一般用在程序的开头，因为它会改变没有类型说明符的变量的类型属性。

参阅〈DEFINT〉，〈DEFSNG〉和第一节“变量类型”。

4. DEFSTR 字母范围

指定某个范围内的字母开头的变量作为串变量存储和处理，除非变量名后加有其他类型的说明符。如果你已经 CLEAR（开辟了）足够的串存储空间，那末每一字符串最多能够存储 255 个字符。

例如：

```
10  DEFSTR  L—Z
```

使得以字母 L 至Z之间的任何字母开头的变量为串变量，除非加有其他类型的说明符。在行 10 执行后，赋值 L1 =“WASHINGTON”是有效的。

参阅〈CLEARn〉，第一节“变量类型”和第五节。

5. CLEAR n

当使用一个自变量 n 时（n 可以是常数或表达式），这一语句使计算机开辟了 n 字节的内存作字符串存储用，另外使所有变量清零。在 TRS—80 一打开时，自动地为字符串开辟了 50 字节的内存。

用 CLEAR 开辟的字符串存储的数量必须等于或大于字符串变量在执行过程中需要存储字符的最大数目，否则将产生字符串空间出界（OS）错误。

例如：

```
10  CLEAR  1000
```

为字符串存储开辟了 1000 字节的内存空间。

实际上需要多少字符串存储量，便按这个数量去开辟内存，这样使你的程序能更有效地使用内存。例如，对一个不应用串变量的程序，应包含有 CLEAR0 语句。CLEAR 的自变量必须是非负数，否则将产生错误。

6. DIM 数组名称（维 1，维 2，……维k）

为一个数组或一组数组设置“长度”〔每一维允许的元素数目〕。如果不用 DIM 语句，那末每个数组的每一维允许使用的长度为11（下标 0—10）。

例如：

```
10  DIM  A(5), B(2, 3), C$(20)
```

它设置了一个具有下标单元 0 到 5 的一维数组 A，一个下标（0，0）到（2，3）的二维数组B，以及一个下标 0 到 20 的一维字符串数组 C$。除非前边作了其它的定义，否则数组

A 和 B 将为单精度值。

DIM语句可以放在程序中任何地方，长度的规定可以是数字或数学表达式。

例如：

```
40  INPUT "NUMBER OF NAMES"; N
50  DIM  NA(N,Z)
```

要重新定义一个数组，必须首先使用 CLEAR 语句，可以带也可以不带自变量，否则将产生错误。

例如，程序：

```
10  AA(4) = 11.5
20  DIM  AA(7)
RUN
? DD  ERROR  IN 20
```

参阅第六节。

7. LET 变量=表达式

可以用来给一个变量赋值。Radio Shack LEVEL Ⅰ可以不用 LET 作赋值语句，但是也可以使用它，以保证与那些需要用此语句的一类 BASIC 版本相兼容。

例如：

```
100  LET  A$ = "A ROSE IS A ROSE"
110  LET  B1 = 1.23
120  LET  X = X - Z1
```

在这种情况下，等号右边的表达式或常数值赋给了等号左边的变量。

8. END

正常地结束程序的执行（不给出 BREAK 信息）。有些类型的 BASIC 版本需用 END 作为程序的最后一条语句，在 LEVEL Ⅰ 中并没有这样的要求。END 主要用于强迫执行在某点终止，而不是在程序的逻辑结尾。

例如：

```
10  INPUT  S1,  S2
20  GOSUB  100
    ⋮
99  END
100 H = SQR (S1 * S1 + S2 * S2)
110 RETURN
```

语句 99 的END防止了程序控制“擅自闯入”子程序，这样，语句 100 只用作调用子程序的语句，如20 GOSUB 100，才能入口。

9. STOP

中断程序的执行，并打印出信息：BREAK IN 语句标号。STOP 主要是一个调试手段。在执行中强迫程序在某处中断，你可以检查或修改变量的数值，然后使用命令CONT，使执行从刚才停止的地方重新开始往下执行。（如果程序本身在中断过程中被修改了，那末不能用 CONT 继续执行）

例如：

```
10  X=RND(10)
15  STOP
20  GOSUB  1000
RUN
BREAK IN 15
READY
>-
```

假设我们想检查一下传送到以语句标号 1000 开头的子程序中去的变量 X 是什么值，在这中断过程中，我们能用 PRINTX 来检查X值（在程序调试完以后可以删去语句15）。

10. GO TO 语句标号

把程序控制转移到指定的语句标号上去执行。单独使用时，GOTO 语句标号是一个无条件转移，但是，试验条件语句可以放在 GOTO 前边，作为一个条件转移。

例如：

```
200  GOTO  10
```

当执行语句 200 时，控制就无条件地跳到语句10。

可以在命令状态应用 GOTO 作为 RUN 的另一种命令。GOTO语句标号，使得计算机从指定的语句标号开始执行，并且不自动执行CLEAR，这使你能把命令方式中赋给的值在执行方式传送到变量中去。

参阅〈IF〉，〈THEN〉，〈ELSE〉，〈ON……GOTO〉。

11. GOSUB 语句标号

把程序控制转移到以指定语句标号开头的子程序中去执行。当计算机遇到子程序中的 RETURN 语句时，它就使控制回到 GOSUB 下边的一条语句上去执行。GOSUB 和GOTO 一样，在它的前边可以放试验条件语句。

参阅〈IF〉，〈THEN〉，〈ELSE〉，〈ON……GOSUB〉。

例如程序：

```
100  GOSUB 200
110  PRINT "BACK FROM SUBROUTINE" : END
200  PRINT "EXECUTING THE SUBROUTINE"
210  RETURN
```

```
(RUN)
EXECUTING THE SUBROUTINE
BACK FROM SUBROUTINE
```

控制从语句100转移到语句标号200开头的子程序，语句210使计算机回到GOSUB紧下边的语句即标号为110的语句上去执行。

12. RETURN

子程序的结束语句，使控制回到这一次调用它的GOSUB紧下边的语句，如果机器遇到了RETURN而找不到调用它的GOSUB语句，则产生错误。

13. ON n GOTO 语句标号，……，语句标号

这是一个多分支的转移语句，它受一个试验变量或表达式的控制。其一般的格式为：

ON 表达式 GOTO 第1语句标号，第2语句标号，……，第K语句标号

表达式的结果必须是0和255之间的数。

当ON……GOTO执行时，首先计算表达式，然后取其整数部分—INT（表达式）。若整数部分为J，计算机就在语句标号清单上数到第J项，然后转移到第J项规定的语句标号。如果没有第J项（即J>K—见上述一般格式），那末控制就执行程序中的下一条语句。（若表达式等于0，也执行下一条语句—译注）

如果试验表达式或数小于0，则产生错误。在清单中语句标号的数目不受限制。

例如：

```
100  ON MI GOTO 150, 160, 170, 150, 180
```

计算变量MI，

若其整数部分等于1，则转移到语句 150

若其整数部分等于2，则转移到语句 160

若其整数部分等于3，则转移到语句 170

若其整数部分等于4，则转移到语句 150

若其整数部分等于5，则转移到语句 180

若其整数部分不等于1至5的任何数，则程序就执行下一条语句。

应用ON n GOTO的一个程序例子：

```
100  INPUT "ENTER A NUMBER"; X
200  ON SGN(X) + 2 GOTO 220, 230, 240
220  PRINT "NEGATIVE": END
230  PRINT "ZERO": END
240  PRINT "POSITIVE": END
```

SGN(X)对于X小于0得－1，对于X等于0得0，对于X大于0得＋1，再加上2以后，表达式就根据X为负数、零或正数分别得到1、2、3，于是控制就转移到相应的语句上去。

14. ON n GOSUB 语句标号，……，语句标号

功能与ON n GOTO一样，只是控制转移到一个由语句标号清单中规定中某个子程序上去。

例如：

```
100  INPUT "CHOOSE 1, 2 OR 3": I
110  ON I GOSUB 200, 300, 400
120  END
200  PRINT "SUBROUTINE# 1": RETURN
300  PRINT "SUBROUTINE# 2": RETURN
400  PRINT "SUBROUTINE# 3": RETURN
```

试验项 n 可以是数字常数，变量或表达式。它必须是非负数值，否则会出错。

见〈ON n GOTO〉。

15. FOR 变量名称=表达式 TO 表达式 STEP 表达式 NEXT变量名称

程序进入一个循环圈，使得一段程序语句重复执行规定的次数。循环语句的一般形式是（括号中的内容是可有可无的）：

```
语句标号# FOR 计数变量=初值 TO 终值 〔STEP 增量〕
    ⋮          [程序语句]
语句标号# NEXT [计数变量]
```

在 FOR 语句中，初值、终值和增量可以是常数，变量或表达式。第一次执行 FOR 语句时，先求得这三个量的值并把它们存储起来，如果这三个变量被循环改变了（这里的三个变量是指 FOR 语句中代表初值、终值和增量的那三个变量，它们在循环中可以改变而不影响循环的执行，可参阅本节举例一译者注），并不影响循环的执行。但是，计数变量不得改变，否则会使循环不能正常进行。

FOR—NEXT—STEP 循环语句是这样工作的：第一次执行 FOR语句时，计数变量等于初值，执行继续进行到遇到 NEXT 语句，这时计数变量就加上 STEP 规定的增量（如果增量是个负数值，那末计数变量实际上是减小）。如果不用 STEP 增量，那末增量假设为 1。接着计数变量与 FOR 语句中规定的终值进行比较，如果计数变量大于终值，那末循环结束，继续执行NEXT语句下边的语句（如果增量是个负数，那末当计数变量小于终值时，循环结束）。如果计数变量的值还没有超过终值，那末程序又执行 FOR 语句后的第一条语句。

程序举例：

```
10  FOR  I=10 TO 1 STEP-1
20  PRINT  I;
30  NEXT
```

```
RUN
  10   9   8   7   6   5   4   3   2   1
READY
>-
```

```
10  FOR  K=0  TO1 STEP  .3
20  PRINT K;
30  NEXT
RUN
  0   .3   .6   .9
READY
>-
```

在K=.9 以后再加上 .3，K=1.2，这时就大于终值 1，因此，循环结束，不再打印出该值。

```
10  FOR  K=4  TO  0
20  PRINT K;
30  NEXT
RUN
4
READY
>-
```

没有规定STEP，因为假设 STEP 为 1，在K第一次增加以后，K的值为 5，由于 5 比终值 0 大，所以循环结束。

```
10  J=3 : K=8 : L=2
20  FOR I=J TOK+1 STEP L
25  J=0 : K=0 : L=0
30  PRINT  I,
40  NEXT
RUN
  3             5             7             9
READY
>-
```

在语句 20 中对变量和表达式作了一次计算，于是对于 FOR—NEXT—STEP循环，这些值变成了常数，所以后边改变了三个变量的值并不影响循环。

FOR—NEXT 循环可以“嵌套起来”：

```
10  FOR  I=1  TO 3
```

```
20  PRINT "OUTER LOOP"
30    FOR  J=1  TO 2
40    PRINT "△△ INNER LOOP"
50    NEXT J
60  NEXT I
RUN
OUTER  LOOP
   INNER    LOOP
   INNER    LOOP
OUTER  LOOP
   INNER    LOOP
   INNER    LOOP
OUTER  LOOP
   INNER    LOOP
   INNER    LOOP
```

注意，每个 NEXT 语句规定了各自的计数变量，这恰好有助于程序编制者留意嵌套次序。计数变量可以从 NEXT 语句中省略。但是，如果你使用了计数变量，那末你必须按正确次序使用它们，也就是最内层循环的计数变量必须首先出现在 NEXT 语句中。

当你的程序允许转移到 FOR—NEXT 循环外边的程序语句去执行时，用 NEXT 语句规定计数变量也是合理的。

另一个嵌套 NEXT 语句的方法是使用计数变量清单。

从上述程序中删去 50，行 60 改为：

```
60  NEXT J, I
```

循环还可以嵌套 3 层，4 层等等，能嵌套多少层是取决于内存数量的大小。

16. ERROR 代　码

让你在程序执行中“模拟”一个指定的错误。这个语句的主要应用是调试 ON ERROR GOTO 程序。

当碰到ERROR 代码语句时，计算机将精确地按这种错误产生时那样处理。参考附录 2：错误代码及其意义。

程序举例：

```
100  ERROR  1
RUN
? NF ERROR
READY
>-
```

错误代码 1 是“没有匹配的 FOR 语句，试图执行 NEXT 语句”。

参阅〈ON ERROR GOTO〉，〈RESUME〉。

17. ON ERROR GOTO 语句标号

当计算机在你的程序中碰到任何一种错误时，通常，它就中断执行并印出错误信息。用ON ERROR GOTO，使你能够设置一个捕获错误程序，它允许你的程序因错误而“再生”，并继续执行而一点不中断执行。通常，当你使用ON ERROR GOTO语句时，你已经考虑到会产生某一种特殊类型的错误。例如，假设你的程序作一些除法运算，并且没有排除被零除的可能性，你可能想写一个处理被零除的错误的程序，那末使用ON ERROR GOTO，就能使得此错误产生时转移到那个处理程序中去。

例如：

```
 5  ON ERROR GOTO 100
10  C=1/0
```

在这个有明显错误的程序中，当计算机试图执行语句10时，就产生被零除的错误，但是，由于语句5，计算机将根本不管语句10，而转移到100开头的错误处理程序中去。

如果在错误捕获程序中使用了ON ERROR GOTO 0语句，那么BASIC将按正常的方法处理该错误。

错误处理程序必须用RESUME语句作结尾。

参阅〈RESUME〉。

18. RESUME 语句标号

结束错误处理程序，使程序返回到指定的语句恢复正常执行。

不带语句标号的RESUME和RESUME 0将使计算机回到产生错误的那条语句。

带有语句标号的RESUME将使计算机转移到指定的语句标号。

RESUME NEXT将使计算机转移到产生错误的那条语句下边一条语句上去执行。

应用错误处理程序的例程序：

```
 5  ON ERROR GOTO 100
10  INPUT "SEEKING SQUARE ROOT OF"; X
20  PRINT SQR (X)
30  GOTO 10
100 PRINT "IMAGINARY ROOT:"; SQR(-X); "*I"
110 RESUME 10
```

运行此程序，并试验一下输入一个负数值。

19. REM

通知计算机忽略该程序行的其余部分。它允许你在程序中插入注释作为对程序的说明。于是，当你（或别人）看你的程序清单时，能够很容易了解它。如果REM用在多语句程序行中时，它必须是该行的最后一条语句。

程序举例：

```
10  REM * * THIS REMARK  INTRODUCES THE PROGRAM * *
20  REM * * AND POSSIBLY THE PROGRAMMER, TOO.    * *
30  REM * *                                       * *
40  REM * * THIS REMARK EXPLAINS WHAT THE        * *
50  REM * * VARIOUS VARIABLES REPRESENT:         * *
60  REM * * C = CIRCUMFERENCE  R = RADIUS        * *
70  REM * * D = DIAMETER                         * *
80  REM
90  INPUT "RADIUS" ; R : REM THIS IS FIRST EXECUTABLE  LINE
```

上面的程序显示了 REM 语句的一些概貌，在 REM 语句中可以包含字符符号，它的最大的长度和其他语句一样：不超过 255 个字符。

在 LEVEL Ⅱ BASIC 中，可以使用一个省字符号 ′（SHIFT 7）作为 REM 的缩写，如：

```
100′THIS TOO IS A REMARK
```

20. IF （真/假）表达式 动作条款

命令计算机对跟随着的逻辑或关系表达式进行试验，如果表达式为“真”，控制将处理紧随表达式后的“动作”；如果表达式为“假”，控制将转移到对应的 ELSE 语句（如果有的话）或者转移到下一条程序上去执行。

在数值项中，如果表达式具有非零值，它总是等价于逻辑“真”。

例如：

```
100  IF X>127 PRINT"OUT OF RANGE" : END
```

如果 X 大于 127，控制就传送到 PRINT 语句，然后到 END 语句。但是，如果 X 不大于 127，控制转移到程序的下一行中去执行，跳开了 PRINT 和 END。

```
100  IF  0<=X  AND X<=90  THEN Y=X+180
```

如果两个表达式都是真的，那末 Y 被赋值为 X + 180，否则控制转移到下一条程序行，跳过了 THEN 后的赋值语句。

注意：在上例或相似的语句中，THEN 是可有可无的，但是，有时必须用 THEN 来消除意义不明确，例如：400 IF Y = M THEN M = 0，如果没有 THEN 就不能工作。

```
500  INPUT A$ : IF A$ = "YES" THEN 100
600  INPUT A$ : IF A$ = "YES" GOTO 100
```

这两条语句有相同的作用。在语句 500 和其它 IF 表达式 THEN 语句标号 的语句中，THEN 是必须有的。

```
100  IF A>0  AND B>0  PRINT "BOTH POSITIVE"
```

试验表达式可以由逻辑算子如 AND 和 OR 连接一些关系表达式所组成。

参阅〈THEN〉，〈ELSE〉。

21. THEN 语句标号

在IF—THEN 类型的语句中引入一个“动作条款”。THEN 是可有可无的，除非要利

用它转移到规定的某条语句标号上去，如 IF A<0 THEN 100。在 IF-ELSE 语句中也必须用 THEN。

22. ELSE 语句或语句标号

用在 IF 的后边，以便在 IF 试验失败的情况下，指定一个替换动作（在不用 ELSE 语句时，控制在试验失败后将转移到下一条程序行上去执行）。

例如：

```
100 INPUT A$ : IF A$ = "YES" THEN 300 ELSE END
```

在语句 100，如果A$等于“YES”，那末程序转移到语句 300，但是，如果 A$不等于“YES”，程序就跳到 ELSE 语句，使计算机执行结束。

```
200 IF A<B PRINT "A<B" ELSE PRINT "B<=A"
```

如果A小于B，那么计算机打印出这个结果，然后处理下一个程序行，跳过了 ELSE 语句。如果 A 不小于 B，计算机直接跳到 ELSE 语句并打印出指定信息，然后控制转到程序的下一条语句上去执行。

```
200 IF A>.001 THEN B=1/A : A=A/5 : ELSE 260
```

如果A>.001是“真”，那末执行后面的语句，赋新值给 B 和 A，然后程序转移到下一行去，跳过了 ELSE 语句，但是如果 A>.001 是“假”，程序直接跳到 ELSE 语句，它使得转移到语句 260 上去。注意，在 ELSE 后边不需要GOTO。

IF—THEN—ELSE 语句可以嵌套，但是必须小心地配对 IF—ELSE。

```
10 INPUT "ENTER TWO NUMBERS"; A, B
20 IFA<=B THEN IF A<B PRINT A;:ELSE PRINT
   "NEITHER"; :ELSE PRINT B;
30 PRINT "IS SMALLER"
```

运行这个程序，输入一对不同的数，程序将把你输入的任何两数中的较小的一个检出并打印出来。注意，在语句 20 中可省略去 THEN 语句及冒号。

23. 数 据 转 换

在执行过程中，使用的每个数都必须属于某种类型：整型、单精度型或双精度型。这种分类常常包含着数从一种类型到另一种类型的转换。它可能产生意外的混乱的结果—除非你了解这种自动分类和类型转换所遵循的规则。

24. 类 型 转 换

常数是 LEVEL Ⅱ BASIC 执行过程中实际使用的数字（不是变量名称）。它们可以出现在你的程序中（如X = 1/3 的等式右边），或者它们可能是在计算同一个表达式过程中产生的临时的（中间的）常数，下面的规则决定了怎样确定一个常数的类型：

1）如果一个常数包含 8 位以上数字，或者用 D 表示成指数形式，那末此数作为双精度型存储。加有一个说明符#的常数也按双精度存储。

2）如果一个数不是双精度型，但是它超过了 -32768 到 32767 的范围，或者它包含有小数点，那末此数作为单精度存储。如果一个数用指数符号E来表示，此数也为单精度数。

3）如果常数既不是Ⅰ也不是Ⅱ的情况，那末它作为整型存储。

例如：

```
10  PRINT 1.234567, 1.2345678
RUN
 1.23457          1.2345678
READY
>-
```

第一个常数含有 7 位，因此根据规则Ⅰ和Ⅱ，它成为单精度数，其最小有效位四舍五入。但第二个常数有 8 位，因此根据规则Ⅰ，它成为双精度数，内部存储为1.2345678000000000，打印出的数字为 8 位有效数，后边的零不打印出来。

25. 常数的分类

当用一个或两个数进行运算时，其结果必然被分类为整型，双精度型或单精度型。

在进行 +，-或 * 运算时，其结果的精度与最高精度运算量的精度相同。例如，如果一个运算量是单精度的，另一个是双精度的，其结果也是双精度的。只有两个运算量都是整型时，其结果才是整型。如果一个整型数经 *，-或 + 运算后其结果超出了整型的范围，那末运算将以单精度进行，其结果也以单精度存储。

除法也遵守与加，减和乘法一样的规则，只有整型除外：当运算量都是整型时，除法运算以单精度进行，并得到单精度结果。

在比较运算中（<，>，=等等），运算量在比较以前都转换成相同类型，低精度类型总是转换成高精度类型。

如果你应用逻辑算符作按位（bit）运算或布尔（Boolean）运算（见第八节逻辑运算符），请阅读下一节，否则就跳过去，不去读它。

逻辑算符 AND,OR 和 NOT,首先将它们的运算量转换成整数形式。如果有一个运算量超出了整数的允许范围（-32768到+32767），就产生溢出错误。逻辑运算的结果总是整数。

26. 类型转换对精度的影响

当一个数转换成整型时，它是被“舍入”，就是取不大于此数的最大整数（这与求该数的 INT 函数获得的结果一样）。

当一个数从双精度转换成单精度时，它四舍五入（当最低有效位后的尾数大于等于5时，最低有效位加 1，否则不变）。

在下述举例中，请记住：单精度变量是以 7 位精度存储的，但只印出 6 位（作适当的四舍五入）。同样，双精度变量以 17 位存储，但只印出 16 位。

程序举例：

```
10  A#=1.666666666666667
20  B!=A#
```

```
30  C% = A#
40  PRINT B!, C%
RUN
△1.66667              1
READY
>-
```

……

当一个单精度数转换成双精度数时，只有 7 个有效位是精确的。如果单精度数没有 7 位有效数，要警惕！（会引入一定的误差，见下例—译者注）

例：

```
10  A! = 1.3
20  A# = A!
30  PRINT A#
RUN
△1.29999952316284
READY
>-
```

……

```
10  A# = 2/3
20  PRINT A#
RUN
△.6666666865348816
READY
>-
```

2 / 3被转换成单精度常数（整数的除法以单精度进行—译者注），因此 A#只有前边的 7 位是精确的。

……

```
10  A# = 2 / 3#
20  PRINT A#
RUN
△.6666666666666667
READY
>-
```

由于表达式 2 / 3 #是按双精度常数计算的，所以 A#的所有 16 位都是精确的，最低有效位已正确地作了四舍五入。

当给一个双精度变量赋予常数时，一定要包括尽可能多的有效位（直到17位），如果常数低于 7 位有效数，最好使用单精度。

例如：

```
10  PI# = 3.1415926535897932
20  E# = 2.7182818284590452
```

五、 字 符 串

“计算机若没有处理字符串的能力，它只是一个超级计算器”，这种夸张的说法有部分道理。你越是使用 LEVEL Ⅱ 的字符串本领，就越是体会到这种说法的正确。

在LEVEL Ⅱ 中,所有的变量名称都可以用来装字符串值,只要对变量名称使用DEFSTR语句或加上类型说明符就行。每个字符串最多能够容纳 255 个字符。

在 LEVEL Ⅱ 中，你甚至能够对字符串进行比较，例如按字母次序表排列它们，还可以将字符串分离和串起来（连接在一起）。

本章的参考材料，见第一节“变量类型”和“词汇表”，第四节“DEFSTR”。

本章涉及的内容和语句有：

“字符串输入/输出”	FRE（字符串）	RIGHT$
“字符串比较”	INKEY$	STR$
“字符串运算”	LEN	STRING$
ASC	LEFT$	VAL
CHR$	MID$	INSTRING 子程序

1. 字符串输入/输出

字符串常数——字符序列——完全可以像数字常数那样，使用 INPUT,READ/DATA 和 INPUT#（由盒式机输入）输入给程序，通常它们可以不用引号输入：

```
10  INPUT “YES  OR  NO”; R$
20  IF R$ = “YES” PRINT “THAT ’S BEING POSITIVE!” : END
30  PRINT “WHY  NOT? ”
RUN
YES  OR  NO? —  [你打入] YES  ENTER
THAT ’S  BEING  POSITIVE!
READY
>-
```

但是，输入一个含有逗号，冒号或开头空格的串常数时，字符串必须放在引号里。

```
10    INPUT  “LAST  NAME, FIRST  NAME”; N$
20    PRINT  N$
RUN
LAST NAME,  FIRST NAME? -[你打入] “SMITH, JOHN”  ENTER
SMITH, JOHN
```

```
READY
>-
```

以上的规则同样适用于由DATA 语句和 INPUT# 语句来输入字符串值。

```
10    READ  T$, N$,. D$
20    PRINT  T$, N$, D$
30    DATA "TOTAL△IS:△", "ONE THOUSAND, TWO HUNDRED"
40    DATA  DOLLARS
```

T$因为有冒号，需要用引号。

N$因为有逗号，需要用引号。

2. 字符串比较

字符串可以比较是否相同（相等），或者判断字母次序的先后。当认为它们相等时，必须每个字符，包括头或尾的空格都相同，否则就表明二个串不相等。

```
600  IF  Z$ = "END" THEN999
```

字符串是从左至右一个字符一个字符地进行比较的。实际上是比较字符的 ASCII代码，代码数较低的字符就在代码数比它大的其他字符之前（见附录 3， ASCII代码）。

例如，常数"A! "在常数"A#"的前面，因为"!"（ASCII代码为十进制的 33）在"#"（ASCII代码为十进制 35）的前边。当不同长度的字符串进行比较时，如果两者的字符相同，那末短的字符串在前（优先），例如"A"排在"A△"之前。

下列关系符号可以用来比较字符串：

=, <>, <, <=, >, >=

注意：无论什么时候，一个字符串常数用于比较表达式中或者用于赋值语句中时，串常数必须括在引号里：

```
A$ = "CONSTANT"
IF  A$ = "CONSTANT" THEN  PRINT  A$
```

（以上两称情况都必须加引号）

3. 字符串运算

本段不包括以后讲述的那些函数，字符串运算只有一种——连接，用加号(+)表示。

例如：

```
10    CLEAR 75
20    A$ = "A ROSE"
30    B$ = "IS A ROSE"
40    C$ = A$ + B$ + B$ + B$ + "."
50    PRINT C $
RUN
A  ROSE  IS  A  ROSE  IS  A  ROSE  IS  A  ROSE.
```

```
READY
>—
```

在语句40，字符串被连接——串在一起了。

```
10    T$ = "100"
20    SUB$ = "5"
30    CODE$ = "32L"
40    LC$ =T$ +"."+SUB$ +CODE$
50    PRINT  LC$
RUN
100.532L
READY
>—
```

4. ASC （字符串）

求指定字符串第一个字符的ASCII代码（以十进制表示）。串自变量必须放在括号里。如果串自变量为0，将产生错误。

```
100    PRINT  ASC("A")
110    T$ ="AB" : PRINT  ASC(T$)
```

语句100和110将打印出相同数字。

自变量可以是包含串运算和函数的表达式：

```
200  PRINT  ASC (RIGHT$(T$, 1))
```

参阅附录C的ASCII代码表。注意：小写字母ASCII代码[注]等于大写字母ASCII代码加上32。所以，ASC函数可以用于将大写值转换成小写值，对于你有小写字母的行式打印机并有合适的硬件/软件接口的情况，这是有用的。

ASC也可以用于建立编码/译码程序（见本节末的例子）。

5. CHR$ （表达式）

执行ASC函数的反转换：求一个字符串，它把指定的ASCII代码，控制代码或图示代码转换成字符。自变量可以是0到255之间的任何数字，或者数值在此范围内的任何变量表达式。自变量必须放在括号内。

```
100  PRINT  CHR$(35)     打印出一个"磅"的符号#。
```

使用CHR$，你甚至能将引号（通常它作为串定义符）赋给串。引号（"）的ASCII代码是34，因此A$ =CHR$(34)就将值（"）赋给了A$。

```
100  A$ =CHR$(34)
110  PRINT "HE  SAID, " ; A$; "HELLO."; A$
RUN
```

注：同时按压 SHIFT 键和字母键。虽然在显示器上出现大写字母，实际上给出了小写字母的ASCII代码—译者。

```
HE SAID, "HELLO."
READY
>—
```

CHR$ 也可以用于显示 64 个图示符号的任何一个（见附录 3，图示代码）

```
10    CLS
20    FOR I=129 TO 191
30    PRINT I; CHR$(I)
40    NEXT
50    GOTO 50
```

（运行此程序，看看不同的图示符号）

代码 0—31 是显示控制代码，用它们得到的是控制符号，而不是实际的显示符号。当 PRINT 一个控制符号时，就能完成某种功能。例如，23 是每行 32 个字符格式的代码，因此，用命令 PRINT CHR$(23)就能使显示格式转换到每行32个字符的格式(敲 CLEAR 键，或者执行 CLS，将返回到每行 64 个字符格式）。

6. FRE （字符串）

当把一个串变量或串常数作自变量时，FRE 得到现在用作字符串存储的空间数量。自变量必须放在括号内。

```
500    PRINT FRE(A$), FRE(L$), FRE("Z")
```

都得到相同的值，因为括号中使用的串变量没有意义，它是一个虚变量[注]。

参阅第四节，CLEAR n。

7. INKEY$

在键盘选通脉冲的瞬间从键盘上得到一个输入的一个字符的字符串。如果在选通脉冲期间没有敲键，那末得到了零串（长度为 0）。这是一个很有用的功能，因为它允许你在计算机执行中不使用 ENTER 键输入数值。流行的电视游戏，如高炮射击，走迷宫，打网球等等，都是用 INKEY$ 功能模拟的（当然，要加上许多其他的程序）。

给 INKEY$ 打入的字符不在荧光屏上自动地显示出来。因为在选通脉冲周期的很短，持续时间只是微秒级，可把 INKEY$ 放在某个循环里，使程序反复地搜索键盘。

程序举例：

```
10    CLS
100   PRINT@540, INKEY$ : GOTO 100
```

运行这个程序，注意：在你第一次敲一个键以前，荧光屏一直保持空白，在你敲另一个键以前，刚敲入的字符一直保持在荧光屏上。（无论什么时候，在键盘择通脉冲期间，你敲

注：FRE 求得的串存储的内存空间数量，完全取决于命令 CLEAR n，不论 FRE 的自变量是什么，FRE（串）总等于n，在本例中都等于 50，因为程序未用 CLEAR 语句，开机时自动执行 CLEAR50—译者。

键不成功，那末在540位置打印零串即“没有字符”，这对于在540上显示的字符毫无影响）

INKEY$ 可以用在循环段里，以允许用户建立一个长的字符串。

例如：

```
90    PRINT "ENTER  THREE  CHARACTERS"
100   A$=INKEY$ : IF A$="" THEN  100 EL SE PRINT A$
110   B$=INKEY$ : IF B$="" THEN  110 ELSE  PRINT B$
120   C$=INKEY$ : IF C$="" THEN  120 ELSE  PRINT C$
130   D$=A$+B$+C$
```

于是一个三个字符的字符串D$，现在可以通过键盘输入而不用 ENTER 键。注意：语句 IF A$=""是将A$与零串进行比较。

8. LEFT$(串，n)

求字符串的前n个字符。自变量必须放在括号里。串可以是串常数或串表达式，n 可以是数学表达式。

例如：

```
10    A$="TIMOTHY"
20    B$=LEFT$(A$, 3)
30    PRINT  B$; "--THAT'S  SHORT FOR"; A$
RUN
TIM--THAT'S  SHORT  FOR  TIMOTHY
READY
>—
```

9. LEN (串)

求指定字符串的字符长度。串变量，串表达式，或串常数必须放在括号里。

例如：

```
10    A$=""
20    B$="TOM"
30    PRINT  A$, B$, B$+B$
40    PRINT  LEN(A$), LEN(B$), LEN(B$+B$)
RUN
                TOM                 TOMTOM
  0              3                    6
READY
>—
```

10. MID$（串，P，n）

求字符串的子串，其长度为 n，开始位置是 P。串的名称，长度和起始位置都必须放在括号里。串可以是串常数或串表达式，n 和 P 可以是数学表达式或常数。例如，MID$（L$，3，1）得到 L$ 的第 3 个字符开始的（长度）1 个字符的串。

程序举例：

本地电话号码的开头 3 位有时称作“交换”号。下面这个程序将等待输入一个完整的电话号码（地区代码，交换号，最后 4 位）并检出交换号。（在美国电话号码有 10 位，如 212—456—7890—译者注）

```
10    INPUT "AREA CODE AND NUMBERS (NO HYPHENS, PLEASE)";
      PH$
20    EX$=MID$(PH$, 4, 3)
30    PRINT "NUMBER IS IN THE"; EX$; "EXCHANGE
```

如果在自变量中不指定长度n，则得到从位置 P 开始的全部字符串。

11. RIGHT$（串，n）

求字符串的最后 n 个字符。串和 n 必须放在括号里。串可以是串常数或串变量，n 可以是数字常数或变量[注]。如果 LEN（串）小于或等于 n，则得到全部字符串。

RIGHT$（ST$，4）得到 ST$ 的最后 4 个字符。

12. STR$（数学表达式）

将数学表达式或常数转换（表示）成字符串。数学表达式或常数必须放在括号里。例如，STR$(A) 把 A 的数值转换成字符串，如果 A=58.5，那末 STR $（A）就等于串“△58.5”（注意：在“58.5”之前插入了一个开头空格，是 A 的符号位。但是，数学运算要用 A 来进行，串“△58.5”只能进行串运算和用在串函数（功能）中。

PRINT STR$(X) 打印出的 X 没有最后一个空格，而 PRINT X 打印出的 X 最后有一个空格。

程序举例：

```
10    A=58.5  :B=-58.5
20    PRINT   STR$(A)
30    PRINT   STR$(B)
40    PRINT   STR$(A+B)
50    PRINT   STR$(A)+STR$(B)
RUN
△58.5
-58.5
```

注：也可以是表达式—译者注。

△0
△58.5—58.5
READY
>—

注意：在 STR$(B)中，开头的空格已放上负号。

13. STRING$（n，字符或数字）

得到一个有 n 个字符符号组成的串。例如，STRING$（30，“*”）得到 30 个星号的串：“*********”

STRING$ 在制图、制表等方面是有用的。

字符也可以是 0—255 之间的一个数，在这种情况下，把它作为一个 ASCII 代码，控制代码或图示代码处理。

例如：

STRING$(64，191)得到一个由 64 个图块（191 代码表示的图块）组成的串。

14. VAL （串）

执行 STR$ 功能的反转换：把串自变量中表示字符转换成相应的数字。例如，如果 A$ = “12”，B$ = “34”，那末 VAL（A$ + “.” + B$）就得到数值 12.34，VAL（A$ + “E” + B$）得到数值 12E34 即 12×10^{34}。

VAL 对于含有数字，及数字后边有字符符号的混合串的操作上有一些不一样。在这种情况下，只有前边的数字在确定 VAL 中被使用，后面其余的字符都被忽略。

例如：　　VAL（“100 DOLLARS”）得到 100，

用这种方法能方便地获取地址。

例如：

```
10   REM “WHAT  SIDE  OF  THE  STREET？”
15   REM  EVEN=NORTH. ODD=SOUTH
20   INPUT “ADDRESS: NUMBER  AND  STREET”; AD$
30   C=INT (VAL(AD$)/2) * 2
40   IF C=VAL(AD$) PRINT “NORTH  SIDE”  :GOTO20
50   PRINT “SOUTH  SIDE”  :GOTO  20
```

运行此程序，输入像“1015　SEVENTH　AVE”这样的街道地址。

15. 一个说明编码 / 译码的程序

```
 5   CLS : PRINT  CHR$(23)
10   CLEAR  1000
20   INPUT “ENTER  MASSAGE”; M$
30   FOR  K=1  TO  LEN(M$)
40   T$ =MID$(M$, K, 1)
60   CD=ASC(T$)+5 : IF CD>255  CD=CD-255
```

```
 70   NU$=NU$+CHR$(CD)
 80   NEXT
 90   PRINT "THE CODED MASSAGE IS"
100   PRINT NU$
110   FOR K=1 TO LEN(NU$)
120   T$=MID$(NU$, K, 1)
130   CD=ASC(T$)-5 : IF CD<0 CD=CD+255
140   OLD$=OLD$+CHR$(CD)
150   NEXT
160   PRINT "THE DECODED MESSAGE IS"
170   PRINT OLD$
```

运行此程序。语句30—80和语句110—150说明了怎样“截取”一个字符的串中的字符。语句60和130说明了ASCII代码的操作。

16. INSTRING 子程序

使用内部的字符串函数MID$和LEN，很容易建立一个非常灵巧的字符串处理子程序INSTRING。它的功能是取两个串自变量，判断其中一个是否包含着另一个。当你在一大批数据或文字中要检索一个特殊的词、短语或一段数据时，它就非常有用。

下边是这一子程序：

```
 999   END'THIS IS A PROTECTIVE END—BLOCK
1000   FOR I=1 TO LEN(X$)—LEN(Y$)+1
1010   IF Y$=MID$(X$, I, LEN(Y$)) RETURN
1020   NEXT : I=0 : RETURN
```

要使用这个子程序，首先要给X$赋一个长字符串值（即“检索范围”），给Y$赋一个需要的值，然后用GOSUB调用子程序。子程序将得到一个I值，它告诉你Y$在长字符串X$中的起始位置，如果Y$不是X$的一个子串，那末I取值为0。

下边是一个应用INSTRING子程序的程序举例(将上面的子程序999—1020加在后边)：

```
 5    CLEAR 1000 : CLS
10    INPUT "ENTER THE LONGER STRING"; X$
20    INPUT "NOW ENTER THE SHORTER STRING"; Y$
30    GOSUB 1000
40    IF I=0 THEN 70
50    PRINT Y$; "IS A SUBSTRING OF"; X$
55    PRINT "STARTING POSITION: ", I,
60    PRINT "ENDING POSITION: "; I+LEN(Y$)-1
65    PRINT: PRINT: GOTO 10
70    PRINT Y$; "IS NOT CONTAINED IN"; X$
80    GOTO 10
```

运行此程序，输入被检索的字符串，然后输入需要的子串。

六、 数　　组

一个数组只不过是一张数值的序列表。

在LEVEL Ⅱ 中，这些数值既可以是数字也可以是字符串，取决于数组如何定义或者如何分类。一个数组提供了处理大量数据的快速、有效的方法。为说明数组的功能，本节描述了一个数组在存储支票簿数据：支票号码、书写日期和每次签发支票的金额数量等方面的情况。

另外，在本节末尾列出了一些矩阵运算子程序。它们能作数组的加，乘，转置和其它运算。

注意：在整个这一节里，为了简单起见，一般不考虑0下标单元。但是必须记住，为了更有效地利用内存，它们是有用的并应该使用的。例如，在DIMA(4)以后，数组A包含5个元素：A(0)，A(1)，A(2)，A(3)，A(4)。

有关数组的参考材料，见第四节 DIM，第一节“数组”。

1. 一个支票簿数组

考虑下列支票簿信息：

支 票 号	书 写 日 期	(金额) 数　量
025	1—1—78	10.00
026	1—5—78	39.95
027	1—7—78	23.50
028	1—7—78	149.50
029	1—10—78	4.90
030	1—15—78	12.49

注意：表中的每一项可以简单地由两个参考数字确定：行数和列数。例如，（行3，列3）确定了数量23.50，因此，这对数字（3，3）可称作23.50的“下标地址”。

让我们根据支票簿信息设置一个数组CK。由于表包括6行3列，所以数组CK需要2维：一个作行数，一个作列数。我们可以这样描述数组：

```
A(1, 1)=025      A(1, 2)=1.0178      A(1, 3)=10.00
    :                 :                  :
A(6, 1)=030      A(6, 2)=1.1578      A(6, 3)=12.49
```

注意：日期信息是以mm.ddyy形式记载的，其中mm=月，dd=日，yy=年的最后两位。因为CK是一个数字数组，因此我们不能将字符符号，如破折号（长划），这样的数据存入。

假设我们已给数组元素赋了适当的值。除非我们使用DIM语句，否则计算机将假设数组的每一维长度为10，也就是计算机开辟的内存单元还有CK(7,1)，CK(7，2)，……

CK(9,1), CK(9, 2), CK(9, 3)。在目前情况下，我们不需要开辟这么多的内存空间，所以我们在程序的开头使用 DIM 语句：

```
10  DIM CK(6,3)'SET UP A 6 BY 3 ARRAY (EXCL.ZERO SUBSCRIPTS)
```

现在，让我们加上把数据读入数组 CK 的程序：

```
20    FOR  ROW=1 TO 6
30       FOR COL=1  TO  3
40         READ CK(ROW, COL)
50    NEXT COL, ROW
90    DATA 025, 1.0178,  10.00
91    DATA 026, 1.0578,  39.95
92    DATA 027, 1.0778,  23.50
93    DATA 028, 1.0778, 149.50
94    DATA 029, 1.1078,   4.90
95    DATA 030, 1.1578,  12.49
```

我们已给数组各元素赋了值，能应用它的内部结构了。例如，我们想把所有支票签发的金额数加起来，用下列程序：

```
100   FOR ROW=1  TO  6
110     SUM=SUM+CK (ROW, 3)
120   NEXT
130   PRINT "TOTAL OF CHECKS WRITTEN";
140   PRINT USING "$$###·##"; SUM
```

现在，再加上在一给定日期内签发的所有支票信息都打印出来的程序：

```
200   PRINT "SEEKING CHECKS WRITTEN ON WHAT DATE (MM.
      DDYY)";
210   INPUT DT
230   PRINT : PRINT "ANY CHECKS WRITTEN ARE LISTED
      BELOW:"
240   PRINT "CHECK#", "AMOUNT" : PRINT
250   FOR ROW=1  TO  6
260   IF CK (ROW, 2)=DT PRINT CK(ROW, 1), CK(ROW, 3)
270   NEXT
```

把程序推广到处理全年 12 个月和其他年份的支票是很容易的，要修改的地方只是按需要增加每一维的大小（或“长度”）。假设我们的支票簿包括的支票号码是 001 到 300，我们要存入全部支票簿的记录，那末，作如下修改：

```
10  DIM (300,3) 'SETUP A 300 BY 3 ARRAY
20  FOR ROW=1 TO 300
```

并加上支票号码从 001 到 300 的数据 (DATA) 行，你可以在每一 DATA 行中装入更多些的数据。另外，所有循环语句中的行计数器的终值都应改变：

```
100  FOR ROW=1 TO 300
      :
      :
250  FOR ROW=1 TO 300
```

2. 数组的其他类型

请记住，在LEVEL Ⅱ 中，一个数组能够具有多少维数（以及数组的大小或长度），只受到可利用的内存大小的限止。

另外，可以使用字符串数组。例如，C$ (X) 将被自动地解释为串数组。

如果在程序开头使用了 DEFSTR A，那末任何以A作名字开头的数组也都是串数组。

串数组的一个明显的用途是用串运算程序存储文字材料。

```
10  CLEAR 1200
20  DIM TXT$(10)
```

将建立一个能存储 10 行文字的串数组。总共开辟了 1200 字节，允许存放 10 行每行 60 个字符，另外 600 字节的串变量存储空间可供它与其它串变量进行串运算时使用。

3. 数组 / 矩阵运算子程序

1) 矩阵输入子程序（二维）

主程序必须给出 N_1（行）和N_2（列）的值

```
30100  REM MATRIX INPUT SUBROUTINE (2 DIMENSION)
30110  FOR  I=1  TO N1
30120  PRINT "INPUT ROW"; I
30130     FOR J=1 TO N2
30140     INPUT A (I, J)
30160  NEXT J, I
30170  RETURN
```

2) 矩阵读数子程序（三维）

主程序必须给出N_1（第 1 维），N_2（第 2 维），N_3（第 3 维）的值

```
30200  REM MATRIX READ SUBROUTINE (3 DIMENSION)
30205  REM REQUIRES DATA STMTS.
30210  FOR K=1  TO N3
30220     FOR  I=1  TO N1
30230     FOR  J=1  TO N2
30240     READ  A (I, J, K)
30270  NEXT J, I, K
30280  RETURN
```

3) 零矩阵子程序（三维）

主程序要给出 N_1, N_2, N_3 的值

```
30300  REM MATRIX ZERO SUBROUTINE ( 3 DIMENSION )
30310  FOR K=1  TO N3
30320    FOR J=1  TO N2
30330     FOR I=1  TO N1
30340      A ( I, J, K ) =0
30370  NEXT I, J, K
30380  RETURN
```

4) 矩阵打印子程序（三维）

主程序要给出 N_1, N_2, N_3 的值

```
30400  REM MATRIX PRINT SUBROUTINE ( 3 DIMENSION )
30410  FOR K=1  TO N3
30420    FOR  I=1 TO N1
30430     FOR  J=1 TO N2
30440     PRINT A ( I, J, K )
30450     NFXT  J : PRINT
30460    NEXT  I : PRINT
30470  NEXT K : PRINT
30480  RETURN
```

5) 矩阵输入子程序（三维）

主程序要给出 N_1, N_2, N_3 的值

```
30500  REM MATRIX INPUT SUBROUTINE ( 3 DIMENSION )
30510  FOR K=1  TO N3
30520  PRINT "PAGE"; K
30530    FOR  I=1  TO N1
30540    PRINT "INPUT ROW"; I
30550     FOR  J=1  TO N2
30560     INPUT A ( I, J, K )
30570     NEXT J
30580    NEXT I
30590  PRINT : NEXT K
30595  RETURN
```

6) 标量乘：乘以简单变量（三维）

```
30600  FOR K=1  TO N3 : 'N3=3RD DIMENSION
30610    FOR  J=1  TO N2 : 'N2=2ND DIMENSION
30620     FOR  I=1  TO N1 : 'N1=1ST DIMENSION
```

```
30630      B(I, J, K)=A(I, J, K)*X
30640      NEXT I
30650    NEXT J
30660  NEXT K
30670  RETURN
```

矩阵 A 的每个元素乘以 X 得到了矩阵 B

---

7) 矩阵转置（二维）

```
30700  FOR  I=1  TO N1
30710    FOR  J=1  TO N2
30720    B(J, I)=A(I, J)
30730    NEXT J
30740  NEXT I
30750  RETURN
```

矩阵 A 转置成矩阵 B

---

8) 矩阵加（三维）

```
30800 FOR K=1 TO N3
30810    FOR  J=1  TO N2
30820      FOR  I=1  TO N1
30830      C(I, J, K)=A(I, J, K)+B(I, J, K)
30840      NEXT I
30850    NEXT J
30860  NEXT K
30870  RETURN
```

---

9) 数组元素相乘

```
30900  FOR K=1 TO N3
30910     FOR  J=1  TO N2
30920       FOR  I=1  TO N1
30930       C(I, J, K)=A(I, J, K)*B(I, J, K)
30940       NEXT I
30950     NEXT J
30960  NEXT K
```

数组 A 的每个元素乘以数组 B 中的与它相对应的元素

---

10) 矩阵乘（二维）

```
40000  FOR  I=1  TO N1
40010      FOR  J=1  TO N2
```

```
40020      C(I, J)=0
40030       FOR K=1  TO N3
40040      C(I, J)=C(I, J)+A(I, K)*B(K, J)
40050        NEXT K
40060      NEXT J
40070  NEXT I
```

矩阵 A 必须是 $N_1 \times N_3$ 的矩阵，B 必须是 $N_3 \times N_2$ 的矩阵，得到的矩阵 C 是 $N_1 \times N_2$ 的矩阵。因此，A，B，C 必须相应地定义其维数大小。

七、 数学函数

LEVEL Ⅱ 提供了多种内部（“机内”）函数，以执行数学运算及特殊运算。特殊运算函数将在下节讲述。

本节讲述的所有常用数学函数都获得单精度值的解，精确到 6 位十进制数。ABS，FIX 和 INT 取得的值，其精度取决于自变量的精度。转换函数（CINT，CDBL等）取值的精度取决于个别的函数。对所有的函数，自变量必须放在括号里。自变量既可以是数字变量，表达式也可以是常数。

三角函数的自变量或反三角函数的解都以弧度表示，而不是角度。本手册给出了弧度—角度的变换。〔注〕

本节介绍的函数有：

ABS	COS	INT	SGN
ATN	CSNG	LOG	SIN
CDBL	EXP	RANDOM	SQR
CINT	FIX	RND	TAN

1. ABS(X)

取自变量的绝对值。X大于或等于 0 时，ABS(X)=X；X小于 0 时，ABS(X)=－X。

```
100  IF ABS(X)<1E-6 PRINT "TOO SMALL"
```

2. ATN(X)

取自变量的反正切（以弧度表示）；即 ATN(X) 求得的角，其正切等于 X。要获得以（角）度表示的反正切，必须将 ATN(X) 乘以 57.29578。

```
100  Y=ATN(B/C)
```

3. CDBL(X)

取自变量的双精度表示。得到的值有 17 位，但只有包含自变量的那几位才有意义。

注：本节译自附页：TRS—80 使用须知一译者。

当你想使一个运算，虽然其运算量是单精度的甚至是整型的，以双精度进行时，CDBL函数就很有用。例如，CDBL (I%)/J% 将得到一个具有17位精度的小数。

```
100  FOR I%=1 TO 25 : PRINT 1/CDBL (I%) : NEXT
```

4. CINT (X)

取不大于自变量的最大整数。例如，CINT (1.5) 得到1;CINT (-1.5) 得-2。对于CINT函数，自变量必须在-32768到+32767范围内。

假如你只对整数结果感兴趣，CINT可以用于加速一个涉及单精度或双精度运算量的运算而又不失去运算量的精度。

```
100  K%=CINT (Y#)+CINT (Y#)
```

5. COS (X)

取自变量的余弦（自变量必须是弧度）。若要得到X为（角）度时的余弦，可用COS (X *. 0174533)。

```
100  Y=COS (X+3.3)
```

6. CSNG(X)

取自变量的单精度表示。当自变量是双精度值时，得到6位有效数字，其最小有效位由四舍五入得到。因此，CSNG (.6666666666666667) 得到 .666667；CSNG (.3333333333333333)得.333333。

```
100  PRINT CSNG (A#+B#)
```

7. EXP(X)

取X的“自然指数”，即e^X。这是LOG函数的逆函数，因此，X=EXP (LOG (X))。

```
100  PRINT EXP (-X)
```

8. FIX (X)

取自变量的尾数被截断以后的整数，即把其小数点右边的所有位都简单地截掉，所以其结果是一个整数值。对于非负数X，FIX (X) =INT (X)，而对于负的X值，FIX (X) =INT (X) +1。例如，FIX (2.2) 得2，FIX (-2.2) 得-2。

```
100  Y=ABS (A-FIX (A))
```

得到的Y值是A的小数部分。

9. INT (X)

取自变量的整数表示，得到不大于自变量的最大整数。自变量不受-32768到+32767范围的限止。例如，INT (2.5) 得2；INT(-2.5)得-3；INT(1000101.23) 得1000101。

```
100  Z=INT ( A * 100 + .5 )/100
```

（对于非负的 A），得到的 Z 值就是四舍五入成两位小数的 A 值。

10. LOG (X)

取自变量的自然对数，即 $\log_e(X)$。它是 EXP 函数的逆函数，因此，X=LOG(EXP(X))。要求一个以 b 为底的对数时，可利用公式：

$$\log_b(x)=\log_e(x)/\log_e(b)$$

例如，LOG (32767)/LOG(2) 得到 32767 以 2 为底的对数。

```
100  PRINT LOG (3.3 * X)
```

11. RANDOM

RANDOM 与其说是函数，倒不如说它实际上是一个完整的语句。它打开一个随机数发生器。如果程序使用 RND 函数，那末需要把 RANDOM 放在程序的开头。这确保你每一次打开计算机，就得到一个不可预测的伪随机数序列，输入到程序中并对它进行运算。

```
10  RANDOM
20  C=RND( 6 ) + RND( 6 )
    ⋮
80  GOTO20 'RANDOM NEEDS TO EXECUTE JUST ONCE
```

12. RND (X)

利用已有的伪随机“种子数”（内部产生的，不受用户干涉的）产生另一个伪随机数。RND 可用于产生 0 到 1 之间的随机数，或产生大于 0 的随机整数，这取决于自变量的数值。

RND (0) 得到一个 0 到 1 之间的单精度值。

RND(整数)得到一个 1 和此整数之间的整数（自变量整数必须为正值并小于 32768）。例如，RND(55) 得到一个大于 0 小于 56 的伪随机整数。RND (55.5) 得到的数也在此范围内，因为 RND 使用自变量的整数部分。

```
100  X=RND( 2 ) : ON X GOTO 200, 300
```

13. SGN (X)

“符号”函数：X为负数时得 −1，X 为 0 时得 0，X 为正数时得 +1。

```
100  ON SGN (x) +2 GOTO 200, 300, 400
```

14. SIN (X)

取自变量的正弦（自变量必须以弧度表示）。当 X 以（角）度表示时，要得到 X 的正弦，可用SIN (X * . 0174533)。

```
100  PRINT SIN (A * B - B)
```

15. SQR(X)

取自变量的平方根。SQR (x) 和 X ↑ (1/2) 是相同的，但运算得更快些。

```
100  Y=SQR (X↑2 -H↑2)
```

16. TAN(X)

取自变量的正切（自变量以弧度表示）。当X以度表示时，要得到 X 的正切，可用TAN (X * .0174533)。

```
100  Z=TAN (2 * A)
```

八、 特殊功能和逻辑运算

值得特别强调的是，LEVEL Ⅱ 提供了一些异乎寻常的功能和操作，有些似乎高度专业化了些，但是，随着你对程序设计的深入了解和开始尝试机器语言程序，那时你将体会到它们具有的重要意义。本章中另一些功能是很有益处的还将经常使用它们(例如，图示功能)。另外，还有两个特点：INP 和 OUT，它们主要用于 TRS—80 的扩展接口。

本章介绍的函数、语句及操作有：

图示	错误程序功能	逻辑运算	其他功能和语句
SET	ERL	AND	INP
RESET	ERR	OR	MEM
CLS		NOT	PEEK
POINT			POKE
			POS
			OUT
			USR
			VARPTR

1. SET (x, y)

打开由坐标 x 和 y 指定的位置上的图示块。为了在显示器屏幕上作图示，把屏幕分成了 128（水平向）乘 48（垂直向）格光块，x 坐标从左往右数是 0 到 127，y 坐标从上往下数是 0 到 47，因此，(0，0)点在显示器左端的最上方，而 (127，47) 点在右下角。

自变量 x 和 y 可以是数字常数、变量或表达式，它们不需要是整数，因为 SET (x，y) 中取 x 和 y 的整数部分。SET (x，y) 的有效范围：

$$0 <= x < 128$$

$$0 <= y < 48$$

注：用上面的函数可以建立大量的其他函数。参阅附录 5 “导出的函数”。

例如：

```
100  SET ( RND(128) - 1, RND(48) - 1 )
```

在显示器上描出一个随机亮点。

```
100  INPUT  X, Y  : SET (X, Y)
```

输入一坐标值，看看图块在哪儿。

2. RESET (x, y)

关闭由坐标 x 和 y 指定的位置上的图示块。这个功能的参量及限制与SET(x,y)相同。

```
200  RESET (X, 3)
```

3. CLS

“清除荧光屏”——关闭显示器上的全部图块，并将光标移到最左上角。像图示块一样。所有字符都被擦去。无论什么时候，你只想在显示器上显示感兴趣的东西时，可先使用CLS。

```
5    CLS
10   SET ( RND(128) - 1, RND(48) - 1 )
20   GOTO  10
```

4. POINT (x, y)

测试一个指定的图示块是“开着”还是“关着”。如果图示块是“开着”的（也就是它已经SET），那末 POINT 将取“真”（二进制数，在 LEVEL Ⅱ BASIC 中为－1），如果图示块是“关着”的，那末 POINT 将取“假”(二进制数 0)。具有代表性的是，将 POINT 试验放在 IF——THEN 语句里边。

```
100  SET(50, 28) : IF  POINT(50, 28)THEN PRINT"ON"ELSE PRINT"OFF"
```

这行语句总是打出信号，“ON”，因为 POINT (50, 28)总得二进制的真，因此执行 THEN 条款。假如试验失败，POINT 将得二进制的假，使得执行跳到 ELSE 语句。

5. ERL

取错误产生时的语句标号。这个功能主要用于错误处理程序中，错误处理程序由 ON ERROR GOTO 语句转入。在使用ERL以后，如果没有产生错误，那末得到的语句标号为 0，但是，从打开电源以后，如果产生了错误，ERL 就得到产生错误行的语句标号。如果错误以直接的方式产生，就得到 65535（由 2 个字节表达的最大数）。

应用 ERL 的程序举例：

```
5    ON  ERROR  GOTO  1000
10   CLEAR  10
20   INPUT "ENTER  YOUR  MESSAGE" ;  M$
30   INPUT "NOW ENTER A NUMBER" ;  N : N=1/N
```

```
40    REM REST OF PROGRAM BEGINS HERE
      :
      :
999   END
1000  IF ERL=20 THEN 1010 ELSE IF ERL=30 THEN1020
1005  ON ERROR GOTO 0
1010  PRINT "TRY AGAIN——KEEP MESSAGE UNDER 11 CHARACTERS"
1015  RESUME20
1020  PRINT "FORGOT TO MENTION : NUMBER MUST NOT BE ZERO"
1025  RESUME 30
```

运行此程序，先输入一个长的信息，当程序请求一个数时，输入一个 0 试试。注意，在语句 1000 中使用 ERL，是为了判断哪儿产生了错误，并作适当的处理。

6. ERR/2+1

功能与 ERL 相似，只是 ERR 得到与错误代码有关的值，而不是产生错误的语句标号。通常用于由 ON ERROR GOTO 语句规定的错误处理程序中。参阅附录 2 中列的“错误代码”。

ERR / 2 + 1 = 真正的错误代码

（真正的错误代码 − 1） * 2 = ERR

程序举例：

```
10    ON ERROR GOTO 1000
20    DIM A(15) : I=1
30    READ A(I)
40    I=I+1 : GOTO 30
50    REM REST OF PROGRAM
      :
      :
100   DATA 2,3,5, 7, 1, 13
999   END
1000   IF ERR/2+1=4 RESUME50
1010   ON ERROR GOTO 0
```

注意，语句 1000 中，4 是数据出界 (OD) 的错误代码。

7. INP(入口)

从指定的入口取 1 个字节的数值。要有效地使用 INP，就需要 TRS—80 的扩展接口板（可与用户供给的外围硬件相联）。总共允许 256 个入口，编号为 0 —255。

例如：

```
100    PRINT INP(50)
```

由入口 50 输入一个字节，并打印出这个字节的十进制值。

8. MEM

取内存中还未用的和未保留的字节数。这功能可用于命令方式，看看程序占去了多少内存空间或者程序还可以使用多少内存空间，也可用在程序中，通过分配较少的串空间或定义(DIM)较小的数组等来避免内存溢出(OM)错误。MEM不需要自变量。

例如：

```
100   IF  MEM<80  THEN  900
110   DIM  A(15)
 ⋮
```

输入命令PRINT MEM（在命令方式），可得到尚未用于存储程序、变量、字符串、堆栈、或为目标文件保留等等的内存数量。

9. OUT 出口，值

从指定的出口输出1个字节的数值。OUT不是一个函数，但是，它本身是一个完整的语句。它需要2个自变量，中间用逗号隔开（没有括号）：指定的出口处的代码和要送的字节值。

例如：

```
OUT  250,  10
```

指数值10送到出口250。

两个自变量都限于0—255范围内。

OUT和INP一样，当你与TRS—80扩展接口联接时，这语句是很有用的。参阅〈INP〉。

10. PEEK（地址）

取一个存在指定地址的数值（以十进制形式）。要使用这一功能，需参考后面的两节附录：内存图〔由此你将知道(PEEK)从哪儿取数值〕和功能表、ASCII代码和图示代码(于是，你将知道数值代表什么）。

如果你应用PEEK来检查目标文件，你也需要微处理器指令手册（有一本包含有TRS—80编辑/汇编指令手册）

PEEK能连接机器语言程序和LEVEL Ⅱ BASIC程序。机器语言程序能在某内存中存储信息，PEEK能用在BASIC程序中以便取得这些信息。例如，

```
A=PEEK (17999)
```

得到地址17999中存储的值，并将此值赋给了变量A。

PEEK也可用于取回由POKE语句存储的信息。使用PEEK和POKE允许你建立非常密集的按字节排列的存储系统。要确定这种存储类型能存储在什么地方，可参考附录中的内存图。见〈POKE〉，〈USR〉

11. POKE 地址，值

将一个值输送到指定的内存地址中。POKE不是一个函数，但是它本身是一个完整的

语句。它需要两个自变量：一个是地址（以十进制形式）和一个值。数值必须是0到255之间的数。参考附录中的内存图，看看你希望用 POKE 把数存进到什么地址中。

要 POKE（或 PEEK）的地址超过32767时，应用下面的公式：

－1＊（65536—所需的地址）＝POKE 或 PEEK 的地址

例如：要 POKE（存入）地址为32769，可应用 POKE－32767，值。

POKE 对 LEVEL Ⅱ 的图示是有用的。在视频显示器工作图上，有1024个PRINT位置，每一个位置中有6个小格。如果我们令每个 PRINT 位置用1个字节，那末每小格就由一个二进制位表示，我们知道每个字节有8位，因此，另外2位作什么用？其中一位用于表示该字节为图示代码还是ASCII代码，另一位没有用，余下的6位容纳 ASCII 代码，图示代码或者控制代码。

我们能够用 POKE 同时打开视频显示器整个 PRINT 位置（即6个小格）。当我们用 SET 时，只能打开一格。因此，POKE 比 SET 大约快6倍。下述程序证明了这个速度。

```
10    CLS
20    FOR  X=15360  TO  16383
30    POKE  X, 191
40    NEXT
50    GOTO  50
```

运行程序，看看荧光屏怎样迅速地“画”白。（191是“所有6格都打开”的代码，15360—16383是显示器的内存地址）

由于 POKE 能将信息存在内存的任何地方，因此它很重要，特别当我们把图示放在显示器地址的范围内时，更能显出它的方便和灵活。如果我们 POKE 不限于这个范围，可以把字节存储在内存的任何位置中。我们可以利用 POKE 把数据插入到程序中去，或者插入到甚至像堆栈这样最不利的位置。乱用 POKE 会产生不幸的后果，你将不得不清除或关闭电源，并重新开始，除非你清楚地知道你使用 POKE 去把数据插入到什么地方去了。

参考〈PEEK〉，〈USR〉，〈SET〉和第四节〈CHR $〉

12. POS（X）

获得一个0到63之间的数，它代表光标正处在显示器上的位置。POS 需要一个“虚自变量”（任何数学表达式）。

```
100    PRINT  TAB(40)  POS( 0 )
```

在位置40打印出数字40（注意，在“4”前面插入了一个空格用于安放正负号；因此“4”实际上是在位置41）。0在 POS(0)中是虚自变量。

```
100    PRINT "THESE" TAB(POS(0)+5) "WORDS"
       TAB(POS(0)+5)"ARE";
110    PRINT  TAB (POS( 0 )+5) "EVENLY"
       TAB (POS( 0 )+5) "SPACED"
RUN
THESE△△△△△WORDS△△△△△ARE△△△△△EVENLY△△△△△SPACED
```

READY

>-

13. USR (X)

调用一个机器语言子程序，并将自变量传送给子程序（你可能不需要自变量，这种情况下它是一个虚自变量）。这种子程序可以从磁带机输入，或者用 POKE 将微处理器指令存入适当的内存地址中去。要使用USR，你应该熟悉机器语言程序设计（在 TRS—80编辑/汇编指令手册中或任何 Z—80 程序设计手册中都有说明）。在使用 USR 时，对于你已经存在 TRS—80 中的一些程序可能会产生破坏，因此，你在使用它以前，应学习点有关知识。

在 LEVEL Ⅱ BASIC 中，只允许使用一个 USR 。在 LEVEL Ⅱ DISK BASIC 中可用10 个：USR0 到 USR9。

例如：

```
100    X=USR(N)
```

将使计算机转移到子程序。　　子程序的起始地址由POKE事先存入到地址 16526—16527 中去。N是 2 个字节整数。在子程序返回时，变量 X 将得到子程序送回的数值。如果不送回值，X 被赋予自变量 N 的值。

N 必须是 －32768 到 32767 之间的一个整数。

要建立一个用 USR 入口的机器语言子程序，你必须在内存的高端保留一块内存。（见附录 4，“内存图”）。首先，要确定你的子程序需要多少字节，然后从计算机的最高内存地址减去这个数目（取决于你的 TRS—80 的内存是 4K 还是 16K 字节），其差数就是你要保留内存的起始地址。打开TRS—80，回答问题MEMORY SIZE？就用键盘输入需要保留内存的起始地址，于是，这个数字以上的内存就被保留，以便给机器语言程序和数据使用。

应用 POKE 或者用 SYSTEM 命令（见第二节，“SYSTEM”），通过盒式机接口，输入机器语言程序。然后，在你的 BASIC 程序中希望转移到机器语言程序的地方，插入调子语句 USR（0）。

例如：

```
50    PRINT  USR(N)
```

或

```
50    A=USR (I%) +B
```

要将自变量传送给子程序，在一进入子程序中必须立即执行CALL 0A7F（16 进制），即调用由 2687（十进制）开始的内部子程序，取来的自变量放在 HL 寄存器中。

有两种从机器语言子程序返回 BASIC 程序的方法：

（1）如果你不希望将任何值从子程序送回到BASIC 程序，可以用机器语言指令 RET。

（2）要在返回时带回值，可以把值以 2 个字节带有符号的整数送入 HL 寄存器对中，并执行一个 JUMP 转移到地址 0A9A（16 进制）〔即 2714（十进制）〕上去，HL 将作为一个有符号的 2 个字节整数返回。

最后需做的工作是，告诉 BASIC 程序你的机器语言程序的入口地址。这个 2 个字节的地址必须用 POKE 输入到内存地址 16526 和 16527。把低位地址值存入到内存地址的低位，（16526），例如，如果你的子程序由 32000 开始，用 16 进制表示是 7D00，因此，我们用

POKE 把 00（16 进制）存入到 16526，把 7D（16 进制）存入 16527。由于 POKE 需 要的自变量是十进制形式，所以，我们用：

POKE 16526, 0 : POKE 16527, 125

($125_{十进制}=7D_{十六进制}$)

在执行上面这行以后，使用 USR(0)功能时，计算机便能转移到存储在 32000地 址开始的子程序中去。

注：除非用 POKE 修改，通常地址 16526—16527 包含着非法功能调子程序的地址。

USR 程序被自动地分配给多达 8 个堆栈级或 16 个字节（每个堆栈级都占有 2 个字节的内存）如果个需要更多的堆栈空间， 那末你可以保存 BASIC 堆栈指示器和建立 自 己 的堆栈，但是，你要知道，你这样做就复杂化了。

参阅第二节〈SYSTEM〉和本节〈PEEK〉，〈POKE〉。

14. VARPTR （变量名称）

取一个地址值，它将有助于你寻找变量名及其值存放在内存中的什么地点。如果你指定的变量名没有被赋过值，使用这个函数时就产生 FC（非法调出函数）错误。

如果VARPTR(整变量)得到地址 K：一个整变量占有二个字节，即地址为K，K + 1。

地址K　　包含整数（补码形式）的最低有效位字(LSB)

地址K + 1　　包含整数的最高有效位字（MSB)

如果VARPTR(单精度变量)得到地址K：一个单精度变量占有四个字节，即地K—K + 3

(K)[注] =值的 LSB

(K + 1) =次最高有效位(NEXT MSB)

(K + 2) =最高有效位(MSB)

(K + 3) =值的指数

如果 VARPTR（双精度变量）得到K：

(K)　=值的LSB

(K + 1) =NEXT MSB

(K + ⋯)=NEXT MSB

(K + 6) = MSB

(K + 7)=值的指数

如果VARPTR（串变量）得到K：

(K)　=串的长度

(K + 1)=串值起始地址的LSB

(K + 2)=串值起始地址的MSB

对于单精度和双精度变量，数是以规正化的指数形式存储的，因此， 小数点在 MSB 之前，指数已加上了 128，而且，在 MSB 中的最高一位是用作符号位的。见下述例子。

例如：

A! =4　　将如下存储：

注：(K)表示“地址K的内容”。

4＝100 二进制，归一化为.1E3

因此，A 的指数为 128＋3＝131，A 的 MSB 是 10000000，然而，由于值是正的，高位变成 0 。因此，A! 这样存储：

指 数	MSB	NEXT MSB	LSB
131	0	0	0

A! ＝－.5 这样存储：

指 数	MSB	NEXT MSB	LSB
128	128	0	0

A! ＝7 这样存储：

指 数	MSB	NEXT MSB	LSB
131	96	0	0

A! ＝－7：

指 数	MSB	NEXT MSB	LSB
131	224	0	0

0 值简单地存储为：0 指数，其它字节没有意义。

15. 逻 辑 运 算

在第一节，我们介绍过 AND，OR 和 NOT 怎样用于关系式表达，例如，

```
100  IF A=C AND NOT(B>40) THEN 60  ELSE 50
```

AND，OR 和 NOT 也可用于位 (bit) 操作，按位比较和进行布尔运算。在本节，我们将介绍在使用LEVEL Ⅱ BASIC时，这些运算如何执行。但是，我们不准备介绍布尔代数，十进制—二进制转换，二进制算术等，如果你需要学习这些有关内容 Radio Shack 的“了解数字计算机”（目录号62—2027）将是一本较好的入门书。

AND，OR和NOT 首先把它们的自变量转换成 16 位的，在 －32768 到 ＋32767 范围内的，带符号的补码整数。然后对它们执行指定的逻辑运算，并得到相同范围内的结果。如果自变量不在此范围内，将产生“FC”（非法调用）错误。

运算是以逐位方式进行的，它的意思是：结果的每一位是以每个自变量的相同位置上的那一位值运算而得到的。

下面的真值表显示了位之间的逻辑关系：

算 符	自 变 量1	自 变 量2	结 果
AND	1	1	1
	0	1	0
	1	0	0
	0	0	0
算符	1	1	1
OR	1	0	1
	0	1	1
	0	0	0

算 符	自 变 量	结 果
NOT	1	0
	0	1

举例：

（下边的所有例子中，二进制数前边的0都不写出，凡注有(2)的数都是二进制数，不注的为十进制）

63AND16＝16　　因为$63=111111_{(2)}$，　$16=10000_{(2)}$

所以结果为$10000_{(2)}=16$

15AND14＝14　　因为$15=1111_{(2)}$，　$14=1110_{(2)}$

所以与的结果为$1110_{(2)}=14$

－1AND8＝8　　$-1=1111111111111111_{(2)}$，　$8=1000_{(2)}$

所以结果为$1000_{(2)}=8$

4AND2＝0　　$4=100_{(2)}$，　$2=10_{(2)}$结果为$0_{(2)}$

因为两个自变量中没有一位能相与得$1_{(2)}$

4OR 2＝6　　$100_{(2)}$和$10_{(2)}$相或，结果为$110_{(2)}=6$

10OR10＝10　　$1010_{(2)}$和$1010_{(2)}$相或，结果为$1010_{(2)}=10$

－1OR－2＝－1　　$1111111111111111_{(2)}$和$1111111111111110_{(2)}$

相或，得$1111111111111111(2)=-1$

NOT　0＝－1　　16位$0_{(2)}$的反码是16个$1_{(2)}$即－1，

所以 NOT－1＝0

NOT　X＝－(X＋1)　　这是因为要形成一个数的16位补码，则取它的反码并加$1_{(2)}$

NOT　1＝－2　　1的16位反码是$1111111111111110_{(2)}$

它就等于－(1＋1)即－2

逐位运算的一个典型应用是对TRS—80的入口处的位进行测试，以便反映某外部设备的状态。这需要有TRS—80扩展接口。

第7位是一个字节的最高有效位，而第0位是最低有效位。

例如，假设输入/输出(I/O)接口5的第1位是0表示房间X的门关着，如果门开着则为1，那末下面的程序在门开着的时候将打印出信息“Intruder Alert”

```
10 IF INP(5) AND 2 THEN PRINT "INTRUDER ALERT" : GOTO100
20 GOTO 10
```

参阅第一节“逻辑算符”。

九、编　　辑

LEVEL Ⅱ的编辑特点使你不必因微小修改程序而从头开始用键盘输入一行程序。事实上，它修改程序行是如此的容易，你或许能够用多语句行，复杂的表达式等作很多的实验。

本章描述的命令、子命令和特殊功能键：

EDIT	L	nD
[ENTER]	X	nC
n空格键	I	nSc
n←	A	nKc
[SHIFT] ↑	E	
	Q	
	H	

1. EDIT 语句标号

这个命令使你进入编辑方式。你必须指定你要编辑的语句，用下面的两个方法之一：

EDIT 语句标号 [ENTER]　让你编辑指定行，如果不用语句标号，将产生 FC（非法调用）错误。

EDIT .　让你编辑本程序行一刚输入或修改的一行，这行是发生错误的行。

例如，打入下行程序并按 [ENTER] 键：

```
100 FOR I=1 TO 10 STEP.5 :PRINT I, I↑2, I↑3 :NEXT
```

这行程序将用来练习下述所有的编辑子命令。

现在打入 EDIT . 并按 [ENTER] 键，计算机将显示：

```
100 —
```

于是，你已处在编辑方式，可以开始编辑行100。

2. ENTER 键

在编辑方式时，按 [ENTER] 键，将使计算机执行你对本行所作的修改并回到命令方式。

3. n 空 格 键

在编辑方式中，敲空格键可使光标向右移动一格，并显示出前面位置上存储的任何字符。例如，应用上面输入的行100，使计算机处在编辑方式。于是显示器显示：

```
100 —
```

现在按空格键，光标将移过一格，并显示出程序行的第一个字符。如果这个字符是空格，则显示出空格，敲空格键直到出现第一个非空格字符：

```
100 F—
```

要一次移过更多的格，首先打入需要的格数，然后敲空格键。例如，打入 5 并敲空格键，那末显示器将出现如下内容（根据你在一行中输入的空格数可改变此内容）：

```
100 FOR I= -
```

再打入 8 并按空格键，那末光标向右移过 8 格，显示出另外 8 个字符。

4. n← （退 格）

把光标向左移 n 格。如果不指定数目 n，光标退回一格。当光标向左移时，在它退回的

途径上把所有字符将从显示器上擦去，但是它们并没有从程序行中删除。例如，假设你再使用了n空格键，显示器上会重新出现：

```
100  FOR  I=1  TO  10—
```

打入8并敲←键，显示器将显示出如下内容：

```
100  FOR  I=—          (语句100中的空格数目可改变的)
```

5. SHIFT ↑

同时按 SHIFT 键和↑键的作用是从任何的插入子命令（X，I和H）中退出。在退出一个插入子命令后，仍处在编辑方式，光标仍然在它现在的位置（退出这些插入子命令的另一个办法是按 ENTER 键）[注1]。

6. L （开一行清单）

当计算机处在编辑方式并且还没有执行一个插入子命令时，敲L键可使程序行的剩余部分显示出来，并且在显示器的下一行重新给出该行的语句标号，光标移到该行的第一个位置。

例如，当显示器出现：

```
100  —
```

敲L键（别敲 ENTER ）键，于是行100将显示出来：

```
100  FOR  I=1  TO  10  STEP.5 : PRINT  I, I↑2, I↑3 : NEXT
100  —
```

它使你在编辑的时间，看一看该行的目前内容[注2]。

7. X （移到行的末尾和插入）

使得本行的剩余部分显示出来，光标移到行的末尾，并使计算机处在插入子命令状态，于是你能在行的末尾添加内容。例如，应用行100，当显示器出现：

```
100  —
```

敲X键（别敲 ENTER 键），整个行将显示出来，注意，光标现在是在行的最后字符的后边：

```
100  FOR  I=1  TO 10  STEP.5 : PRINT  I, I↑2, I↑3 : NEXT—
```

这时，我们就能在此行添加其他语句，或者用←键从此行删除一些东西。例如，在行的末尾打上：PRINT "DONE"，然后敲 ENTER 键。这时候，如果你打入，LIST100，那么显示器出现如下内容：

```
100  FOR I=1  TO  10  STEP.5 : PRINT I, I↑2, I↑3 : NEXT : PRINT
     "DONE"
```

注1：用 ENTER 键退出子命令不同于SHIFT ↑，它将回到命令状态，不能再继续修改该行程序，所以只能在程序修改完后使用。一译者

注2：若经过检查没有发现错误，则按 ENTER 键，把该行程序送入内存。一译者

8. I （插　入）

允许你在此行中从光标所在的位置起插入某些内容（在此状态下，敲←键将把字符从行中实际删除）。例如，打入 EDIT100 命令，然后用空格键移到行 100 的小数点处，显示器出现：

```
100  FOR  I=1  TO 10 STEP.—
```

假设你想把增量由 .5 改成 .25，那末敲 I 键（别敲 ENTER 键），于是计算机就允许你在现在位置插入内容，这时敲 2，显示器出现：

```
100  FOR  I=1  TO  10  STEP.2—
```

你做完必要的修改以后，同时按 SHIFT ↑，退出插入子命令。现在再敲 L 键，显示出这行的剩余部分并且光标移到该行的开头（在下一行位置—译者注）。

```
100  FOR  I=1  TO  10  STEP.25 : PRINT I, I↑2, I↑3 : NEXT :
     PRINT "DONE"
100  —
```

你也可以敲 ENTER 键来退出插入子命令和保存全部的修改，并回到命令方式。

9. A （取消修改并重新开始）

把光标移回到程序行的开头并取消已做的修改。例如，你已经在行中增添、删除或修改了一些内容，而你又希望回到这一行的开头并取消已做的修改，那末你首先敲 SHIFT ↑（从任何正执行的子命令中退出），然后敲 A（光标将降到下一行，显示出语句标号，移到该行程序第一个字符位置上）。

10. E （保存修改和退出编辑方式）

使计算机结束编辑并保存已作的全部修改。当你敲 E 键的时候，必须是在编辑方式，而不能正在执行任何子命令。

11. Q （取消和退出）

使计算机结束编辑并取消这一次编辑期间已做的全部修改。如果你决定不修改程序了，敲 Q 就取消了修改并退出编辑方式。

12. H （切尾和插入）

使计算机删除该行程序中光标后的剩余部分，并且可以从光标的目前位置起插入某些内容。在这个方式中，敲←键将从程序行中实际删除一个字符。例如，应用前面列出的语句，100 进入编辑方式，用空格键移到最后一条语句PRINT "DONE" 之前，假设你想删除这一条语句并插入 END 语句，显示器出现：

100 FOR I=1 TO 10 STEP.25 : PRINT I, I↑2, I↑3 : NEXT : —

这时，敲H，然后打入 END，再敲 ENTER 键。

开列这行清单时将显示出：

100 FOR I=1 TO 10 STEP.25 : PRINT I, I↑2, I↑3 : NEXT : END

13. nD （删 除）

使计算机从光标的右边删除指定数目 n 个字符。被删除的字符将夹在两个警叹号之间，表示这些字符已被删除了。例如，应用行 100，用空格键移到 PRINT 语句之前：

100 FOR I=1 TO 10 STEP.25 : —

打入 19D，告诉计算机从光标右边删除 19 个字符。于是显示器显示出如下内容：

100 FOR I=1 TO 10 STEP.25 : ¡ PRINT I, I↑2, I↑3 : ¡ —

当你开列全行的清单时，你就会看到 PRINT 语句已被删除了。

14. nC （修 改）

在计算机上允许你由光标现在位置起修改指定数目（n 个）的字符。如果你打入 C 而前边没有数目，计算机就假设你想修改 1 个字符。当你输入了 n 字符后，计算机就回到编辑方式（于是，就不处在 nC 子命令了）。例如，应用行 100，假设你想改变 FOR—NEXT循环的终值，由“10”改为“15”。在编辑方式，把空格刚好移到 10 中的 0 那儿，

100 FOR I=1 TO 1—

然后打入 C，计算机就假设你只想修改 1 个字符，打入 5，再敲 L。当你开列此行清单时，你将看到修改已经完成了。

100 FOR I=1 TO 15 STEP.25 : NEXT : END

如果你按照本节中的编辑顺序做了，那末这就是目前的这一行程序。

15. nSc （检 索）

让计算机检索某个字符 (C) 第 n 次出现的地方，并把光标移到这个位置。如果不指定 n值，那末计算机将检索第 1 次出现的指定字符。如果字符 C 没有找到，光标就跑到程序行的末尾。注意：计算机只检索光标右边的字符。

例如，应用目前形式的行 100 打入 EDIT100（按 ENTER），然后打入2S:，让计算机检索第 2 个冒号出现的地方。显示器将出现：

100 FOR I=1 TO 15 STEP.25 : NEXT—

现在，你可以从光标目前的位置开始执行一个子命令。例如，你想在NEXT语句后边加上计数变量，那末打入 I 使转入插入子命令，再打入变量名 I，这就是你想插入的全部内容，于是，敲 SHIFT ↑ 使退出插入子命令。再开列程序清单时，就出现：

100 FOR I=1 TO 15 STEP.25 : NEXT I : END

16. nKc （检索和“抹除”）

让计算机把第 n 次出现的字符 C 的前面（与光标位置之间）的所有字符都删除，并把

光标移到这个位置。例如，应用目前形式的行100，假设我们要删除 END 语句前面的整个一行。打入 EDIT 100（按[ENTER]键），然后打入 2K：，这就让计算机把第2个冒号前边的所有字符都删除。于是显示器将出现：

```
100  !FOR  I=1  TO  15  STEP.25:NEXT  I! —
```

因为第2个冒号仍需要删除，所以敲D，于是显示器出现：

```
100  !FOR  I=1  TO  15  STEP.25:NEXT  I!!:! —
```

这时，敲[ENTER]键，再打入LIST 100（按[ENTER]）语句100就成了这样：

```
100  END
```

十、 扩展接口

在 TRS—80 计算机中有一个扩展接口板。这个接口允许接附加的输入/输出设备，它也是一个增加 RAM 存储器的设备。允许接到 TRS—80 上的四个主要附加设备是：

1） 一个附加的盒式磁带机，

2） 一个 TRS—80 行式打印机，

3） 多至四个小型磁盘，

4） 多至 48K 字节的 RAM 存储器（32 K在扩展接口内）。

若要把扩展接口和任何外部设备相联，请参阅扩展接口说明书。

当扩展接口接上 TRS—80 后，计算机可接一个小型磁盘。小型磁盘将允许使用后面列出的附加命令和语句。若你没有小型磁盘，计算机都认为你有(因为磁盘控制器已经有了)，并试图由磁盘控制器输入特殊指令，因此，要使用没有小磁盘的接口，你在打开TRS—80时同时要按着[BREAK]键。无论什么时候，你需要按[RESET]纽的时候，也必须同时按着[BREAK]键。在接有扩展接口时，要清除计算机，也同时按压[BREAK]键和[RESET]纽，这样就又回到了询问 MEMORY SIZE问题的初始状态，存储器中的任何 BASIC 程序都被清除掉了。

1. 双盒式记录器

使用两台盒式磁带机将使数据存储到磁带上去时更有效、更方便。例如，如果你要更新存在带上的数据，那末把信息从盒式机#1以一次一条的方式读入一条，再对它进行修改或增添，然后又一次一条地写入盒式机#2。这个程序可如下这样编写：

```
10  INPUT#-1, A, B, C, D
20  PRINT "MAKE CORRECTIONS HERE:RETYPE LINE"
30  INPUT A, B, C, D
40  PRINT "THE LINE NOW READS:"A, B, C, D
50  PRINT "STORING ON TAPE #2…"
60  PRINT#-2, A, B, C, D
70  GOTO 10
```

这是一个非常简单的应用程序，然而可以编制一个非常有用的程序，以允许同时使用两

个磁带机输入，输出数据。

命令也可以给设备号，以允许你控制盒式机，与盒式机有关的命令有CLOAD,CSAVE和CLOAD? 等。在给定盒式机号（1或2）后，就启动指定的盒式机。例如：

CLOAD#-1，“A”

（这是非常重要的），将由盒式机 #1 输入文件“A”。

CSAVE#-2，“A”

将把机内程序转储到盒式机 #2 的磁带上。

参阅第二章CLOAD，CLOAD?，CSAVE。

2. 行式打印机

行式打印机能够把 TRS—80 产生的信息打印出来。有一些命令适用于各种型号的行式打印机。

LLIST

功能和 LIST 一样，不过是输出给打印机：

LLIST	（在行式打印机上）开列目前程序清单
LLIST 100—	开列行 100 到末尾的程序清单
LLIST 100—200	开列行 100 到行 200 的程序清单
LLIST .	开列目前行（刚输入或编辑的一行）清单
LLIST -100	开列行 100 以前的全部清单

参考第二节LIST。

LPRINT

这个命令或语句允许你把信息输出给行式打印机。例如，LPRINT A 将在行式打印机上打印出 A 的值。所有 PRINT 适用的语句除 PRINT@ 以外，都能用 LPRINT 替代。

例如：

LPRINT 变量或表达式	（在行式打印机上）打印变量或表达式的值
LPRINT USING	按规定格式打印信息
LPRINT TAB	按 TAB 表达式指定的位置向右移动行打滑架

在 TRS—80 没有接行式打印机（或行式打印机关闭着）时[注]，你执行 LPRINT 或 LLIST 的话，计算机将“冻结”，这时，你应打开行式打印机，或者复位计算机—如果没有行式打印机。

由于 LPRINT 语句能打印最大的表格为 63、所以行式打印机不能打印出超过 63 列的表格。对此有一个简单的消除限止的方法，即用 STRING $ 功能去模拟超过 63 列的表格。

例如：

LPRINT TAB(5) “NAME” TAB(30) “ADDRESS” STRING $(63, 32) “BALANCE” 这样，行式打印机就在第 5 列打印出“NAME”，在第 30列打印出“ADDRESS”，在第 100 列打印出“BALANCE”。

注：本段和下面两段（代码之前），译自附页：TRS—80 使用须知—译者

代　码

有一些代码可用于控制行式打印机。代码及其功能列出于下，可用函数 CHR $ 来调用这些功能代码。

代　码	功　　能
10	滑架回车、换行
11	滑架移到下一页顶端（即换页）
12	同　上
13	滑架回车、换行

注意．在使用的时候，换页代码应放在 LPRINT 语句的开头，因为在它前边的任何项都将被忽略，而回车/换行代码不能放在LPRINT语句的开头，否则它们后边的项目都被忽略。

行式打印机每英寸打印 6 行，每页打印 66 行。如果这个格式不合适，那末每页要打的行数可以改变，用 POKE 把新的行数放入内存地址 16424。

例如：

```
POKE    16424, 40
POKE    16425, 0
```

这命令行式打印机每页打印 39 行，并将行计数器清除为 0 。

3. 小型磁盘—(LEVEL Ⅱ DISK BASIC)

TRS—80 小型磁盘系统是一种小型的软磁盘。磁盘比磁带的存取时间快，并且有巨大的文件存储空间。磁盘 0 的文件空间约有 80000 字节，每增加一个磁盘就增加 89600字节的文件空间。磁盘系统有它自己的一套命令，进行文件操作，扩大了对文件应用的能力。在 TRS—80 小型磁盘系统中既允许顺序存取也可随机存取。使用磁盘后增加了一些BASIC 命令和函数。

命　令

CLOSE	KILL	OPEN	REST
FIELD	LSET	PRINT	MERGE
GET	NAME	PUT	LOAD
			SAVE

I/O 功　能：

CVD	EOF	MKD $	DSKF
CVI	LOC	MKI $	
CVS	LOF	MKS $	

对 LEVEL Ⅱ 的补充：

10 个 USR—USR 0 到 USR9

&H（16进制常数）

&O（8 进制常数）

INSTR（执行 INSTRING 子程序的功能—参阅第四节）

DEFUSR　　TIME$（日期和 24 小时实时时钟）
LINE INPUT
MID$（在方程式左边）　　DEF FN（用户定义函数）
为说明这些命令功能，请见 TRS—80 磁盘操作系统手册。

4. RAM 存储器的扩展

TRS—80 扩展接口已提供了附加的随机（RAM）存储器。它是用增加 RAM 存储器芯片来达到的。最多可增加 32768 字的额外的内存。

十一、 节省时间和空间

多数 LEVEL Ⅱ 程序要比 LEVEL Ⅰ 的相同程序的运算速度快，并少占内存空间。但是，除了它更有效率的特点之外，你在设计程序时，若用下列一些简单技巧作指导，还能进一步改进 LEVEL Ⅱ。

1. 节约内存空间

1） 当你的程序已能正确执行时，从你的上机程序中删除所有不必要的 REM 语句。

2） 在语句，算符等之间不要使用不必要的空格。

3） 尽可能使用多语句程序行（在每两条语句之间用一个冒号隔开）。因为，你每加入一个新的语句标号要耗费 5 个字节。

4） 在一切可能的时候，要使用整变量，例如：

```
FOR I%=1 TO 10
```

因为整数只占 5 个字节，单精度数占 7 个字节，双精度数占 11 个字节。

5） 如果一个运算在不同的地方几次用到，使用子程序，可节约程序空间。如果一个程序总是从相同的地方被调用，用无条件转移（GOTO）语句。因为每个 GOSUB 将占 6 个字节，而在运行时，GOTO一点不占内存。

6） 在你的计算结构中，要尽可能少用括号（参考第一章“算术运算符”）。处理括号要占 4 个字节，因为要先做括号中的运算，所以每个括号内表达式的解都必须存储起来（这要占 12 个字节）。

7） 定义数组要有节制。当你建立一个矩阵时，计算机为每一维保留 11 个下标地址，即使用不到11个元素，但计算机还是如此。另外，要使用 0 下标元素，因为它们都是可用的。

8） 当你用非单精度值（字符串，整数和双精度数）工作时，最好使用 DEF 语句。一个 DEF 语句虽然要占 6 个字节，但是，因为你不需要在变量名称后边使用类型说明符，所以编制时既快，又能节省内存。

2. 加快执行（节省时间）

一个程序的处理速度取决于运算的复杂性和指令的数目。对多数的简单的程序，速度不是一个因素，它似乎在你进入 RUN 的瞬间就得到了结果。但是，随着你开始书写更长、更复杂的程序，速度就成了重要因素。下面有一些建议可指导你设计更快的程序。

1） 删除所有没有必要的程序行（REM语句等）。

2） 在实际应用中，联成多语句成序行。

3） 在运算中要使用变量而不使用常数（非常重要）。通常，TRS-80的运算是用浮点数进行的。它取一个变量所用的时间比把一个常数转换成浮点数少得多。例如，如果你在程序中要大量应用 π，那么把 π 定义为一个变量(PI=3.14159)并在运算中使用变量 PI。

4） 使用 POKE 作图示，它能使你的图示显示比用 SET 方法显示快 6 倍。

5） 首先定义最常用的变量。因为第一个定义的变量，它被放在变量表的顶上，第二个在就它的下边。在取变量的时候，计算机是从上往下搜索变量表以寻找某变量，因此，把频繁使用的变量放在表的顶端(首先定义它们)可节省时间，计算机不用寻找很远就能找到它们。

6） 尽可能使用整型变量，特别在 FOR—NEXT 循环里，这是最重要的。

附 录 1. LEVEL II 一 览 表

1. 特殊字符和缩写

命 令 状 态

ENTER	回车和解释命令
←	光标退格并清除刚打的字符
SHIFT ←	光标退回本行开头；删除全行
↓	换 行
:	语句定义符；用于同为行的两条语句之间
→	光标移到下一格式区。格式区的位置：0，8，16，24，32，48，56
SHIFT →	显示器转换为每行32字符
CLEAR	清除显示器并转换为每行64字符

执 行 状 态

SHIFT	暂停执行；在 LIST 过程中“冻结”显示器
BREAK	停止执行
ENTER	解释从键盘输入的数据（用于INPUT语句）

缩 写

?	代 替 PRINT
'	代 替 :REM
.	“本行”；代替LIST，EDIT等中的行号

类 型 说 明 符

符 号	类 型	举 例
$	字 符 串	A$，ZZ$
%	整 数	A1%，SUM%

!	单 精 度 数	B!，N1!
#	双 精 度 数	A#，1/3#
D	双 精 度 数（指数表示）	1.23456789D—12
E	单 精 度 数（指数表示）	1.23456E+30

字符串算符

+	连 接（串起来）	"2"+"2"="22"

算术算符

+	加	-	减	*	乘	/	除
↑	指 数（即：2↑3=8）						

关系算符

符 号	数学表达式中意义	串表达式中意义
<	小 于	超 前（按字母排列顺序，在前面）
>	大 于	落 后（按字母排列顺序，在后面）
=	等 于	相 等（相同）
<=或=<	小 于 等 于	超前或相等
>=或=>	大 于 等 于	落后或相等
<>或><	不 等 于	不相等

运算等级（同一行上的算符有相同优先权）

↑ （指 数）
- （求负数）
*，/
+，-
关系算符(<，>，=，<=，>=，<>)
NOT
AND
OR

2. 命 令

命 令	功 能	举 例
AUTOmm，nn	打开自动行数编号	AUTO
	从mm开始，增量	AUTO 10
	是nn（行数即语句标号）	AUTO 5,5
		AUTO，10
CLEAR	置数字变量初值为0	CLEAR
	串为零	
CLEAR n	同CLEAR，但为字符串	CLEAR 500

	开辟n字节内存	CLEAR MEM/4
CLOAD#—d，“文件名”	从盒式机 d （1或2）	CLOAD#-1，“T”
	输入该“文件名”的BASIC程序，若省略#—d	CLOAD“T”
	则假设为盒式机 1	CLOAD
CLOAD#-d,？“文件名”	输入该“文件名”程序并与机内程序比较，若不同，打印“BAD”错误	CLOAD#-1,?“R” CLOAD？ “R” CLOAD?
CONT	在BREAK或STOP 以后续继执行	CONT
CSAVE#-d，“文件名”	将把BASIC程序以“文件名”为名存在盒式机d上	CSAVE#-1，“S” CSAVE “S”
DELETEmm-nn	删除由行 mm到行nn全部程序行	DELETE 100 DELETE10—50 DELETE.
EDITmm	行mm进入编辑状态	EDIT 100 EDIT.
LIST mm—nn	给出行mm到行nn的全部程序行清单	LIST LIST30—60 LIST30— LIST—90 LIST.
NEW	清除整个程序，所有变量，指示字等都置 0	NEW
RUN mm	从行 mm执行程序，或者从最小行数起执行（若无mm）	RUN RUN55
SYSTEM	进入由盒式机输入机器语言文件的管理状态	见第二节
TROFF	关闭跟踪	TROFF
TRON	打开跟踪	TRON

3. 编辑状态子命令和功能键

子命令/功能鍵	功　　能
ENTER	结束编辑和回到命令状态
SHIFT ↑	退出子命令并保持编辑状态
n 空格鍵	光标向右移 n格
n ←	光标向左移 n格
L	列出程序行的剩余部分，光标回到下一行的开头并给出该行语句标号
X（子命令）	列出程序行的剩余部分，光标移到行的末尾、并开始插入子命令
I （子命令）	从光标所在位置起插入字符段、用 SHIFT ↑ 退出作此子命令
A	取消修改，光标回到行的开头，重新开始修改
E	结束编辑，保存全部修改，回到命令状态
Q	结束编辑、取消所作的全部修改，回到命令状态
H （子命令）	删除行的剩余部分，插入字符段，用 SHIFT ↑ 作其出口

nD	从光标所在位置起删除 n个字符
nC	用输入的n 个字符代替从光标位置起的n个字符
nSc	从光标所在位置算起，将光标移到指定的字符C第 n 次出现的地方
nKc	从光标所在位置起，到指定字符C第n次出现处之间的全部字符都被删除

4. 输入／输出语句

句 语	功 能	举 例
PRINT 表达式	将表达式的值输出给显示器，表达式可以是数字或串表达式，或常数，或以上项目的组合	PRINT A$ PRINT X+3 PRINT"D="D
	逗号是PRINT修饰项，它使打印一个数值后(逗号前的),光标移到下一个打印区，打印后一个数值（逗号后的）	PRINT 1,2,3,4 PRINT "1", "2" PRINT 1,,2
	分号是PRINT修饰项，它使打印的数字项后插入了一个空格，打印字符串后无空格。分号放在PRINT清单的末尾时，制止自动回车	PRINT X; "=ANSWER" PRINT X; Y; Z PRINT "ANSWERIS";
PRINT@n	PRINT修饰项；在指定的显示器位置n处打印	PRINT@540, "CENTER" PRINT@N+3, X*3
TABn	PRINT修饰项；将光标移到显示器的指定位置n处打印	PRINT TAB(N) N
PRINT USING串；表达式	打印格式规定：按串字段规定的形式输出表达式	PRINT USINGA$; X PRINT USING "#.#"; Y+Z
INPUT "信息"；变量	打印出信息并等待从键盘输入	INPUT"ENTER NAME"; A$ INPUT "VALUE"; X INPUT A, B, C, D$
PRINT#-1	输出给盒式机 #1	PRINT# -1,A,B,C,D $
INPUT#-1	从盒式机 #1输入	INPUT#-1A, B, C, D$
DATA项目清单	容纳数据，由 READ语句读取	DATA22,23,11, 1.2345 DATA"HALL", "SMITH"
READ变量清单	给指定变量赋值	READ A,A1,A2,A3
RESTORE	把数据指示器恢复到第一条DATA 语句的第一项	RESTORE

5.PRINT USING 语句的格式规定符

数字符号	功 能	举 例	
#	数字段（每个#是一位）	####	
.	小数点位置	## .###	
+	打印符号位，可放在数值的前头或末尾（正数打	+#.###	#.###+

	印加号，负数打印减号）	-#.### #.###-
-	如果打印的数值是负数，打印出负号，只打印在数值的末尾	###.##-
**	开头空格用星号填充	**###.##
$$	把美元号紧放在头一位的左边	$$#####·##
**$	美元号在头一位左边而开头空格用星号填充	**$####.##
↑↑↑↑	指数格式，其小数点左边有一个有效数	#.#######↑↑↑↑

串符号	功能	举例
!	一个字符	!
% %	一串字符，其长度等于两个%之间的空格数再加2	% %

6. 程序语句

语句	功能及举例
（类型定义）	
DEFDBL字母表或范围	定义为双精度型，所有以指定字母或指定范围内的字母作开头的变量都是双精度型。 例：DEFDBL J　DEFDBL X,Y,A-E,J
DEFINT字母表或范围	所有以指定字母或指定范围内的字母作开头的变量都被定义为整型。 例：DEFINT A　DEFINT C, E, G DEFINT A-K
DEFSNG字母表或范围	所有以指定字母或指定范围内的字母作开头的变量都被定义为单精度型。 例：DEFSNG L　DEFSNG A—L,Z,P
DEFSTR字母表或范围	所有以指定字母或指定范围内的字母作开头的变量都被定义为串变量。 例：DEFSTR A,B,C　DEFSTR S, X—Z,M
（赋值和分配）	
CLEAR	为字符串存储开辟指定数目n字节内存。 例：CLEAR 750　CLEAR MEM/10　CLEAR 0
DIM数组（维数1，……，维数K）	为K维数组分配内存，数组的每一维具有指定长度：维数1，……，维数K，DIM后的数组表可用逗号隔开。 例：DIM A(2,3)　DIM A1(15), A2(15) DIM B(X+2), C(J,K)　DIM T(3,3,5)
LET　变量=表达式	把表达式的值赋给变量，在LEVELⅡBASIC中LET可有可无。 例. LET A $="CHARLIS"　LET A%=I# LBT B1=C1　D=X*Y/Z
（执行的顺序）	
END	结束执行，回到命令状态。 例：99　END
STOP	停止执行，打印出有关中断行语句标号的中断信息。 例：100　STOP

GOTO 行号　　转移到指定行号。　　例：GOTO 100

GOSUB 行号　　转移到在指定行号开始的子程序。

例：GOSUB 3000

RETURN　　子程序出口，转移到最近执行的 GOSUB 下边的语句。

ON表达式GOTO行号1，……行号K　　计算表达式，如果表达式的整数部分等于1至 K 之间的一个数，则转移到相应的行号，否则到下一条语句。

例：ON K+1 GOTO 100, 200, 300

ON表达式GOSUB行号1,……行号K　　同上，根据表达式的整数值转移到相应的行号开始的子程序。

例：ON J GOSUB 330, 7000

FOR 变量=表达式 TO 表达式 STEP 表达式　　打开 FOR—NEXT 循环，STEP 可有可无，若不用时增量为1。

例：FOR I=1 TO 50 STEP 1.5

FOR M%=J% TO K-1%

NEXT 变量　　关闭FOR—NEXT循环，变量可省略，要关闭嵌套的循环可用变量表。

例：NEXT　NEXT I　NEXT I, J, K

ERROR 代码　　模拟用代码指定的错误。

例：ERROR 14

ON ERROR GOTO行号　　在以后的程序行中如果产生错误，转移到该行数开始的错误处理程序。

例：ON ERROR GOTO 999

RESUMEn　　由错误处理程序回到指定行n，如果n等于0或沒有，则回到含有错误的语句，如果n是"NEXT"则回到错误语句的下一条语句。

例：RESUME RESUME 0 RESUME 100 RESUME NEXT

RANDOM　　开启随机数发生器。　　例：RANDOM

REM　　注释说明：BASIC 解释程序将忽略程序行其余部分。

例：REM A IS ALTITUDE

（条 件 语 句）

IF 表达式1 THEN 语句1 ELSE 语句2　　测试表达式1：如果是真，执行语句1，然后跳到下一程序行(除非语句1是GOTO)；如果是假，直接跳到ELSE语句并执行后边的语句。

例：IF A=0 THEN PRINT "ZERO" ELSE PRINT "NOT ZERO"

（图 示 语 句）

CLS　　清除电视显示器并成为每行64字符。

例：CLS

RESET(x,y)　　关闭水平坐标x垂直座标y的图示块，0<=x<128，0<=y<48。

例：RESET (8+B, 11)

SET (x,y)　　打开坐标为x,y的图示块，自变量限止同上。

例：SET (A*2, B+C)

（特 殊 语 句）

POKE地址，值　　将值输入到指定地址的内存单元中。

例：POKE 15635, 34　POKE 17770, A+N

两自变量都以十进制形式表示，0<= 值 =<255

OUT 出口，值	将值送到出口（两自变量都在0—255之间）。 例：OUT 255，10　　OUT 55,A

7. 串函数

函数	运算及举例
ASC(串)	取串自变量第一个字符的ASCII代码。串可以是串变量，表达式或常数，下同——原注。 例：ASC(B$)　　ASC（"H"）
CHR$(代码表达式)	取回由代码规定的一个字符的串，如果代码规定一个控制功能，那末实施这功能。 例：CHR$(34)　　CHR$(11)
FRE(串)	取得可用于串存储的内存数量，自变量是一个虚变量。 例：FRE（A$） 译注：函数取决于CLEAR n的数n。
INKEY$	选通键盘，得到一个在选通脉冲期间所按键输入的字符（如果不按键，则为零串）。 例：100 PRINT@ 540，INKEY$：GOTO 100
LEN（串）	求串的长度（零串为0）。 例：LEN（A$+B$）　　LEN（"HOURS"）
LEFT$(串，n)	取串的开头n个字符。 例：LEFT$(A$，1)　　LEFT$(L1$+C$，8) LEFT$(A$，M+L)
MID$(串，p，n)	取串的子串，其长度为n，在串中的起始位置是p。 例：MID$(M$，5，2)　　MID$(M$+B$，P,L-1)
RIGHT$(串，n)	取串的最后n个字符。 例：RIGHT$(NA$，7)　　RIGHT$(AB$，M2)
STR$(数学表达式)	计算自变量并把其值表示为串。 例：STR$(1.2345)　　STR$(A+B*2)
STRING$(n，字符)	得到一个字符串，由n个规定的字符所组成。 例：STRING$(30，"·")　　STRING$(5，C$)
VAL(串)	STR$函数的逆函数：根据数值串求数值。 例：VAL("1"+A$+"."+C$)　　VAL(G1$)

8. 数学函数

函数	运算（除非有其他说明，函数自变量限于：−1.7E+38<=表达式<=1.7E+38）及举例
ABS(表达式)	取绝对值。表达式可以是任何数学表达式或常数，下同。——原注 例：ABS(L*7)　　ABS(SIN（X）)
ATN(表达式)	求反正切（弧度表示）。 例：ATN(2.7)　　ATN(A*3)
CDBL(表达式)	取表达式的双精度表示。 例：CDBL(A)　　CDBL(A+1/3#)

CINT(表达式)　　取不大于表达式的最大整数。

限制：－32768＜＝表达式＜＋32768

例：CINT(A#＋B)

COS(表达式)　　求表达式的余弦，表达式用弧度表示。

例：COS(2＊A)　　COS(A/57.29578)

CSNG(表达式)　　取单精度表示，当表达式为双精度数时，其最小有效小数四舍五入。

例：CSNG(A#)　　CSNG(.33＊B#)

EXP(表达式)　　取自然指数，即 e 表达式＝EXP(表达式)。

例：EXP(34.5)　　EXP(A＊B＊C—1)

FIX(表达式)　　取表达式被截尾后的整数（截尾：把表达式的小数部分砍掉）。

例：FIX(A—B)

INT(表达式)　　取不大于表达式的整数。自变量不受－32768——＋32767范围的限制。

例：INT (A＋B＊C)

LOG(表达式)　　取表达式的自然对数（以 e 为底）。限制：表达式必须是正数。

例：LOG (12.33)　　LOG (A↑B＋B)

RED(0)　　取 0.000001 到 0.999999 之间的伪随机数。

例：RND (0)

RND(表达式)　　取 1 到INT(表达式) 之间的伪随机数。

限制：1＜＝表达式＜32768

例：RND(40)　　RND(A＋B)

SGN(表达式)　　符号函数：表达式为负数时得－1；零时得 0；正数时得＋1。

例：SGN(A＊B＋3)　　SGN(COS (X))

SIN(表达式)　　求表达式的正弦。表达式用弧度表示。

例：SIN(A／B)　　SIN(90/57.29578)

SQR(表达式)　　求表达式的平方根。限制：表达式必须是非负数。

例：SQR(A＊A－B＊B)

TAN(表达式)　　求表达式的正切，表达式是弧度。

例：TAN(X)　　TAN(X＊.0174533)

注：其他三角函数可参见附录 5，导出函数－译者

9. 特 殊 函 数

函　数	运 算 和 限 制	举　例
ERL	取产生错误的行号。	
	例：ERL	
ERR	取与错误代码有关的值（如果产生错误的话）。	
	ERR＝（错误代码－1）＊2，也就是：(ERR/2)＋1＝错误代码。	
	例：ERR/2＋1	
INP（入口）	从指定入口输入一个数值。自变量和结果都在 0 到 255 范围内。	
	例：INP (55)	
MEM	取未用的和未保留的内存字节总数。	
	例：MEM　　IF MEM＜800 THEN 90	

PEEK（地址）　　取回存在指定内存的一个字节的数值，地址必须是十进制形式的有效内存地址。

例：PEEK（15370）

POINT(x，y)　　检查由水平坐标 x 和垂直坐标 y 指定的图示块：如果图示块“开着”，得真(－1)；如果“关着”，得假(0)。

限制：0<=x<128，　0<=y<48

例：10 SET（50，28）：IF POINT(50，28)　THEN PRINT “ON” ELSE PRINT “OFF”

POS(0)　　取一个表明光标目前位置的数，自变量 0 是虚变量。

例：POS(0)

USR(n)　　转移到机器语言子程序。对于 LEVEL Ⅱ BASIC，n 是二个字节的整数，－32768≤n≤32767。

例：USR(0)

VARPTR（变量）　　取一个地址，该处存着指定变量的名称、数值和指数。变量必须是有效的变量名。如果变量没有被赋值，则得 0。

例：VARPTR(A$)　　VARPTR (NI)

10. LEVEL Ⅰ保留词汇

@	CLEAR	CDBL	DEFINT
ABS	CLOSE	CVD	DEFSNG
AND	CLS	CVI	DEFUSR
ASC	CMD	CVS	DEFSTR
ATN	CONT	DATA	DELETE
CHR$	COS	DEFDBL	DIM
CINT	CSNG	DEFFN	EDIT
ELSE	KILL	NOT	SET
END	LEFT$	ON	SGN
EOF	LET	OPEN	SIN
ERL	LSET	OUT	SQR
ERR	LEN	PEEK	STEP
ERROR	LINE	POINT	STOP
EXP	LIST	POKE	STRING$
FIELD	LOAD	POS	STR$
FIX	LOC	PRINT	TAB
FOR	LOF	PUT	TAN
FRE	LOG	RANDOM	THEN
GET	MEM	READ	TIME$
GOSUB	MERGE	REM	TROFF
GOTO	MID$	RESET	TRON
IF	MKD$	RESTORE	USING
INKEY$	MKI$	RESUME	USR
INP	MKS$	RETURN	VAL

INPUT	NAME	RIGHT $	VARPTR
INSTR	NEW	RND	
INT	NEXT	SAVE	

注：这些词汇中有好些在 LEVEL I BASIC 里并没有函数，它们是为 LEVEL I DISK BASIC 使用而保留的。所有这些词汇都不能用作变量名种。

11. 程序限制和常用内存数量

范　围

整　数	-32768 到 +32767
单精度数	-1.701411E+38 到 +1.701411E+38
双精度数	-1.701411834544556E+38 到 +1.701411834544556E+38
字 符 串	最多 255 个字符
语句标号	0 到 65529
程序行长度	最多 255 个字符

常用内存数量

程序行最少需 5 个字：

语句标号——2 个字

行指示符——2 个字

回车——1 个字

另外，每个保留词、算符、变量名、特殊符号和常数符号都需要 1 个字

动态（运行时）内存分配

整变量：　每个 5 个字节（数值 2 个字，变量名 3 个字）

单精度变量：　每个 7 个字节（数值 4 个字，变量名 3 个字）

双精度变量：　每个11个字节（数值 8 个字，变量名 3 个字）

字符串变量：　至少 6 个字节（变量名 3 个字，堆栈和变量指示符 3 个字，每个字符 1 个字）

数组变量：　至少12个字节（变量名 3 个字，大小 2 个字，维数 1 个字，每一维 2 个字，数组中每一元素 2，3，4 或 8 个字［取决于数组类型］）

每个有效的 FOR—NEXT 循环：需 16 个字节

每个有效的（还没有返回）GOSUB：需 6 个字节

每重括号需 4 个字节加每个中间值的 12 个字节

附录 2. 错 误 代 码

代码	简写	错 误	代码	简写	错 误
1	NF	有NEXT无 FOR	13	TM	类型不符合
2	SN	句法错误	14	OS	字符串空间出界
3	RG	有Return无 GOSUB	15	LS	字符串太长
4	OD	数据出界	16	ST	字符串公式太复杂
5	FC	非法调用函数	17	CN	不能继续
6	OV	溢出	18	NR	无RESUME
7	OM	内存出界	19	RW	有RESUME无错误
8	UL	未定义行	20	UE	不能印出的错误
9	BS	数组下标出界	21	MO	遗漏运算量
10	DD	重复定义数组（维数）	22	FD	文件数据不好
11	/0	被0除	23	L3	只用于 Disk（磁盘） BASIC
12	ID	非法指示			

错误信息的解释

NF　有NEXT无FOR：使用了NEXT而没有对应的 FOR语句。这个错误也可能发生在一个嵌套的循环里保留着 NEXT 变量的语句。

SN　句法错误：这通常是由于不正确的标点符号，不闭合的括号，错误的字符或者拼写错误的命令等引起的。

RG　有 RETURN 无 GOSUB：在对应的 GOSUB 被执行以前碰到了 RETURN语句。

OD　数据出界：执行 READ 或 INPUT# 语句而可用的数据不够，DATA 语句可能已经丢失或者所有数据已经从磁带或 DATA 读完。

FC　非法调用函数：企图执行用非法参量的运算，例如：负自变量的平方根，负的矩阵维数，LOG 自变量是负的或 0 等等，或者调用 USR 没有首先 POKE 进入口地点。

OV　溢出：输入或导出的数值对于计算机来说太大或太小了。

OM　内存出界：所有可用的内存已经用完或保留。这可能产生在很大的矩阵维数，嵌套像 GOTO，GOSUB 这样的转移和 FOR—NEXT 循环等情况。

UL　未定义的行：企图涉及或转移到不存在的行。

BS　下标超出范围：企图赋值给下标超出维数定义范围的矩阵元素。

DD　重复定义数组：企图定义一个前边已经用 DIM 或不用语句[注]定义过的数组。将所有DIM语句放在程序开头是个好主意。

/0　被零除：企图用 0 值作分母。

ID　非法指示：将 INPUT 作直接命令使用。

TM　类型不符：企图把字符串赋给非字符串变量或者反其道而行之。

OS　串空间出界：超出了分配给的字符串（存储）空间总数。

注：不用语句定义是指自动按规定数组每维10个单元一译者

LS　字符串太长：赋给串变量的字符串值的长度超过了255个字符。

ST　字符串公式太复杂：要处理的串运算太复杂，可分成短的段运算。

CN　不能继续：CONT用于不能继续的程序处，例如在程序结束或编辑之后。

NR　无 RESUME：程序到达错误捕获状态，在处理完错误以后找不到出口。

RW　有RESUME无 ERROR：在执行 ON　ERROR GOTO 以前碰到了 RESUME。

UE　不能印出的错误：企图用 ERROR 语句以无效的代码产生错误。

MO　遗漏运算量：没有提供需要的运算量企图作运算。

FD　不合适的文件数据：从外部源（即磁带）输入的数据是不正确的或是以不合适的顺序，等等。

L3　只用于DISK BASIC：企图使用只有在通过扩展接口联接了 TRS—80小型磁盘时才适用的语句、函数或命令。

附录 3. 控制代码、ASCII代码和图示代码

控制代码：　1—31（十进制）

代码	功　能	代码	功　能
0—7	无	23	转换到32字符格式
8	退格并删去该字符	24	退格，光标←
9	无	25	进格，光标→
10	换行／回车	26	向下，换行↓
11	换页（至下页开头）（有行式打印机时）	27	向上，换行↑
12	换页（至下页开头）（有行式打印机时）	28	复位，光标回到显示器位置(0，0)
13	换行／回车	29	光标移到行的开头
14	打开光标	30	删去行的末尾
15	关闭光标	31	清除帧的末尾
16—22	无		

ASCII 字符代码 32—128 （十进制）

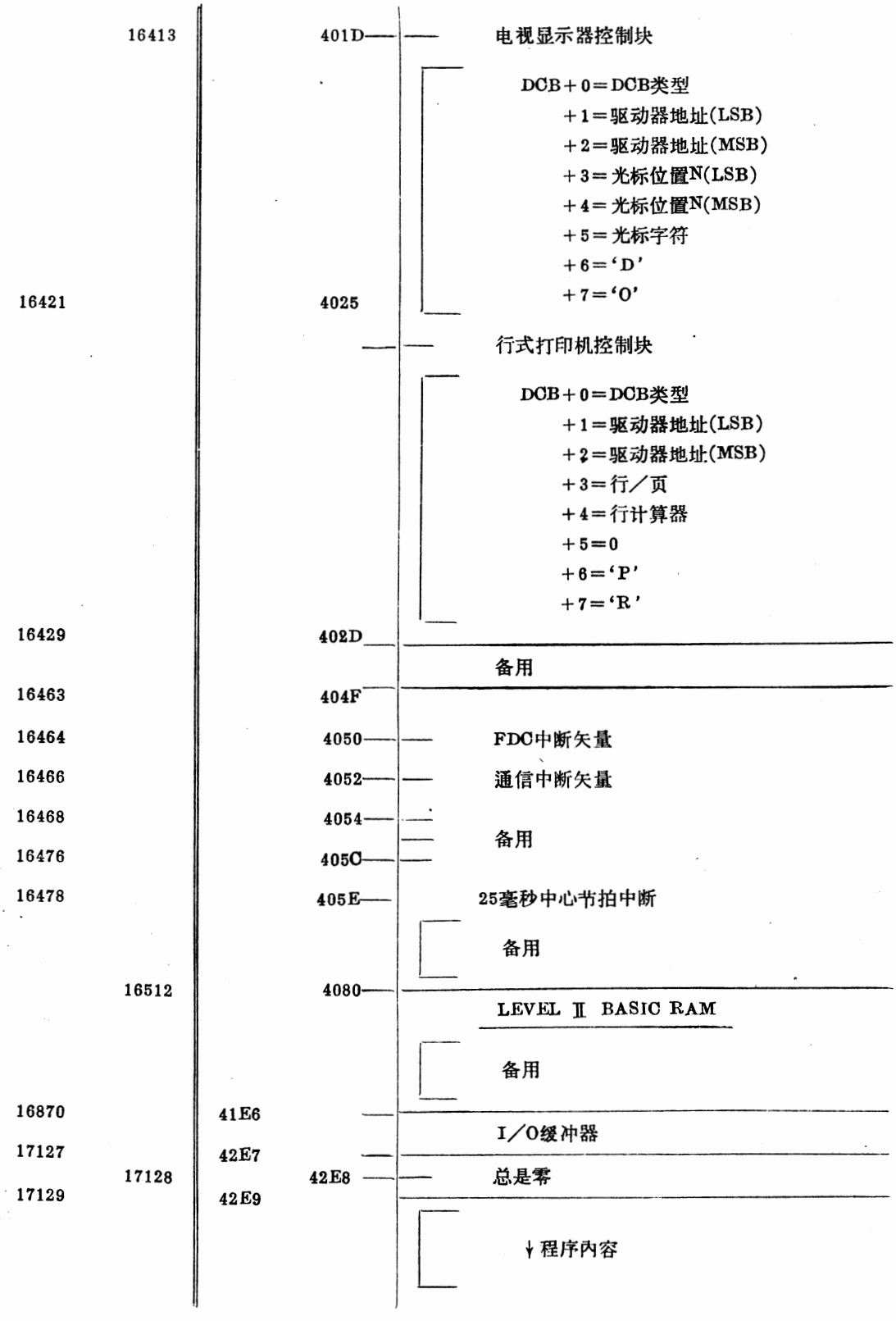
代 码	字 符	代 码	字 符
32	空 格	65	A
33	!	66	B
34	"	67	C
35	#	68	D
36	$	69	E
37	%	70	F
38	&	71	G
39	'	72	H
40	(	73	I
41	)	74	J
42	*	75	K
43	+	76	L
44	,	77	M
45	—	78	N
46	.	79	O
47	/	80	P
48	0	81	Q
49	1	82	R
50	2	83	S
51	3	84	T
52	4	85	U
53	5	86	V
54	6	87	W
55	7	88	X
56	8	89	Y
57	9	90	Z
58	:	91	↑或[
59	;	92	↓
60	<	93	←
61	=	94	→
62	>	95	—
63	?	96—127	小写代码（对应64—95）
64	@	128	空格

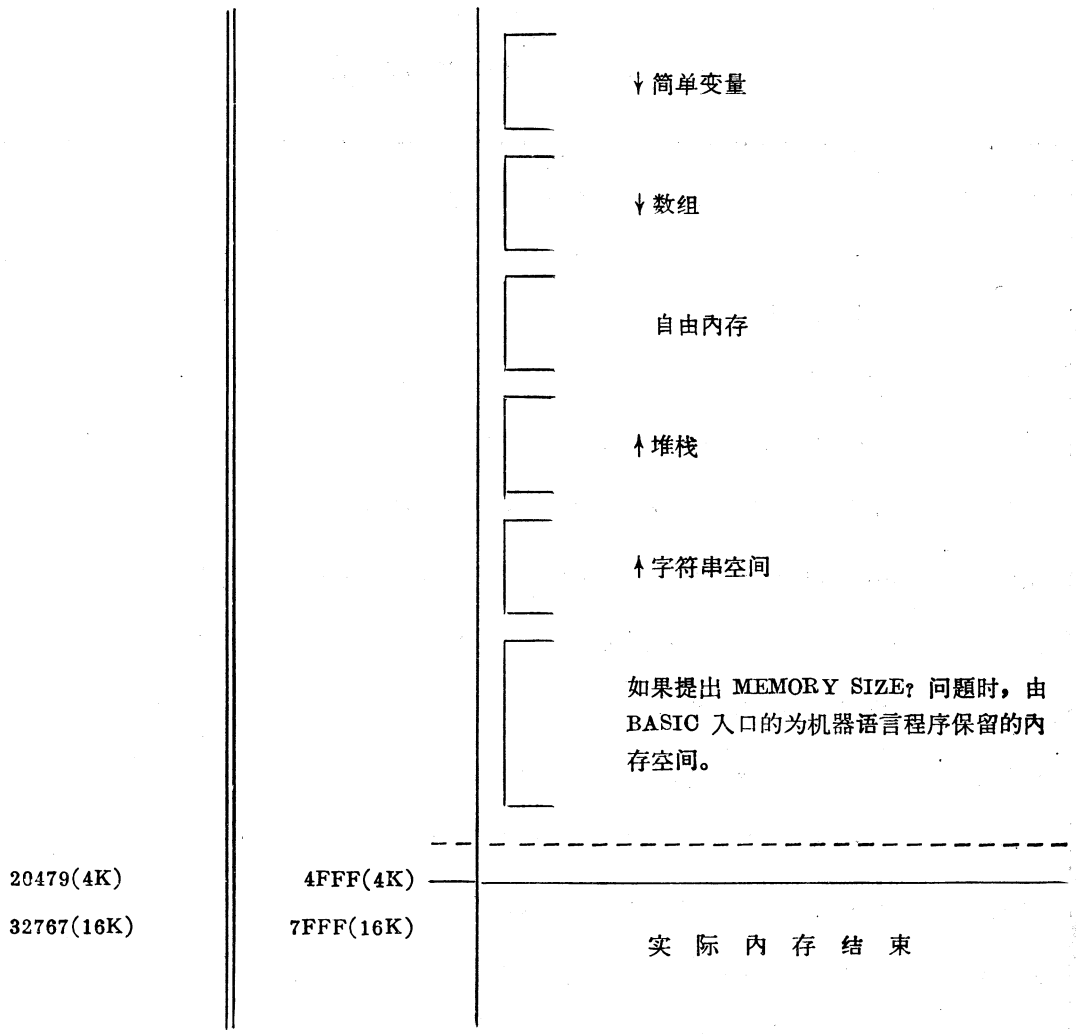
图示代码： 129—191　　**空间压缩代码：** 192—255

附录 4. LEVEL Ⅱ TRS—80内存图

地 址

十 进 制	十六进制	
0	0000	LEVEL Ⅱ BASIC ROM
12288	3000	备用
14302	37DE	通信状态地址
14303	37DF	通信数据地址
14304	37E0	中断锁存器地址
14305	37E1	磁盘驱动选择锁存器地址
14308	37E4	盒式机选择锁存器地址
14312	37E8	行式打印机地址
14316	37EC	软盘控制器地址
14336	3800	TRS—80键盘内存
15360	3C00	TRS—80电视显示器内存
16383	3FFF	
16384	4000	LEVEL Ⅱ BASIC 固定RAM 矢量（RST 1到7）
16402	4012	
16405	4015	键盘设备控制块 DCB+0 =DCB类型 DCB+1 =驱动器地址 +2 =驱动器地址 +3 = 0 +4 = 0 +5 = 0 +6 ='K' +7 ='I'


16413
401D
电视显示器控制块
DCB+0=DCB类型
+1=驱动器地址(LSB)
+2=驱动器地址(MSB)
+3=光标位置N(LSB)
+4=光标位置N(MSB)
+5=光标字符
+6='D'
+7='O'
16421
4025
行式打印机控制块
DCB+0=DCB类型
+1=驱动器地址(LSB)
+2=驱动器地址(MSB)
+3=行/页
+4=行计算器
+5=0
+6='P'
+7='R'
16429
402D
备用
16463
404F
16464
4050
FDC中断矢量
16466
4052
通信中断矢量
16468
4054
备用
16476
405C
16478
405E
25毫秒中心节拍中断
备用
16512
4080
LEVEL Ⅱ BASIC RAM
备用
16870
41E6
I/O缓冲器
17127
42E7
17128
42E8
总是零
17129
42E9
↓程序内容



简写的注释——译注：

ROM——只读存储器

RAM——随机存储器

DCB——数据控制块

LSB——最低有效位（二进制）

MSB——最高有效位（二进制）

FDC——软盘控制器

I/O——输入／输出

附录 5. 导出函数

函　数	用LEVEL Ⅱ BASIC的函数来表示
正割	SEC (X) =1／COS(X)
余割	CSC (X) =1／SIN(X)
反正弦	ARCSIN (X) =ATN〔X／SQR (-X*X+1) 〕
反余弦	ARC COS (X) = -ATN〔X／SQR (-X*X+1) 〕+1.5708
反正割	ARCSEC (X) =ATN〔SQR (X*X-1) 〕+(SGN (X) -1) *1.5708
反余割	ARCCSC (X) =ATN〔1／SQR (X*X-1) 〕+〔SGN (X) -1〕*1.5708
余　切	COT (X) =1／TAN (X)
反余切	ARCCOT (X) = -ATN (X) +1.5708
双曲正弦	SINH (X) =〔EXP (X) -EXP (-X) 〕／2
双曲余弦	COSH (X) =〔EXP (X) +EXP (-X) 〕／2
双曲正切	TANH (X) = -EXP (-X) ／〔EXP (X) +EXP (-X) 〕*2+1
双曲正割	SECH (X) =2／〔EXP (X) +EXP (-X) 〕
双曲余割	CSCH (X) =2／〔EXP (X) -EXP(-X) 〕
双曲余切	COTH (X) =EXP (-X) ／〔EXP (X) -EXP (-X) 〕*2+1
反双曲正弦	ARGSINH (X) =LOG〔X+SQR (X*X+1) 〕
反双曲余弦	ARCCOSH (X) =LOG〔X+SQR (X*X-1) 〕
反双曲正切	ARGTANH (X) =LOG〔 (1+X) ／ (1-X) 〕／2
反双曲正割	ARGSECH (X) =LOG{〔SQR (-X*X+1) +1〕／X}
反双曲余割	ARGCSCH (X)=LOG {〔SGN (X) *SQR (X*X+1) +1〕／X}
反双曲余切	ARGCOTH (X) =LOG〔 (X+1) ／ (X-1) 〕／2

附录 6. 基 本 变 换 表

二进制	八进制	十六进制	十进制	二进制	八进制	十六进制	十进制
00000000	000	00	0	01101000	150	68	104
00000001	001	01	1	01110000	160	70	112
00000010	002	02	2	01111000	170	78	120
00000011	003	03	3	10000000	200	80	128
00000100	004	04	4	10001000	210	88	136
00000101	005	05	5	10010000	220	90	144
00000110	006	06	6	10011000	230	98	152
00000111	007	07	7	10100000	240	A0	160
00001000	010	08	8	10101000	250	A8	168
00001001	011	09	9	10110000	260	B0	176
00001010	012	0A	10	10111000	270	B8	184
00001011	013	0B	11	11000000	300	C0	192
00001100	014	0C	12	11001000	310	C8	200
00001101	015	0D	13	11010000	320	D0	208
00001110	016	0E	14	11011000	330	D8	216
00001111	017	0F	15	11100000	340	E0	224
00010000	020	10	16	11101000	350	E8	238
00011000	030	18	24	11110000	360	F0	240
00100000	040	20	32	11111000	370	F8	248
00101000	050	28	40	11111001	371	F9	249
00110000	060	30	48	11111010	372	FA	250
00111000	070	38	56	11111011	373	FB	251
01000000	100	40	64	11111100	374	FC	252
01001000	110	48	72	11111101	375	FD	253
01010000	120	50	80	11111110	376	FE	254
01011000	130	58	88	11111111	377	FF	255
01100000	140	60	96				

附录 7. 程 序 举 例

1. 宇宙飞船着陆

这个程序是模似宇宙飞船在地球，月亮、火星和 Vesta 小行星等四个行星之一着陆的游戏。在每 10 秒钟“燃烧”前的间隔时间，应给出下列信息：

经过时间（秒）

高度（千米）

速度（千米/小时—— 负数表示飞离行星）

剩余燃料（千克）

应用这些信息，选择“燃烧速率”（千克燃料／秒）例如，10kg／sec 燃烧速率就是在 10 秒钟燃烧期间耗费了 100kg 燃料。燃烧速率必须在 0—100kg／sec 范围内（超过100kg／sec将使“加速度”力太大，对人体有危害）。

注意：　负的速度表示燃烧了太多燃料和正在离开行星。

燃料可以包含小数部分（如15.5kg／sec)。

由于消耗了燃料，飞船变轻，因此，以后的燃烧的影响越大。

每个行星的着陆条件是不同的，因为它们有各自的重力：地球＝980cm／sec^2，

月球＝162cm／sec^2，　　火星＝372cm／sec^2，

Vesta行星＝17.5cm／sec^2。

在印出的清单上，向上的箭头是以左括号〔出现的，请记住将它们作为向上的箭头用作指数进入程序。

指挥员，祝你好运气！

```
100  CLS
110  PRINT@20, "***LANDER***"
120  PRINT : PRINT "TYPE 1 FOR EARTH, 2 FOR MOON, 3 FOR MARS 4 FOR VESTA"
140  INPUT X : ON X GOTO 500, 600, 700, 800
145  CLS
147  PRINT@980, A$;
150  G2=G1/36
160  G3=SQR (G2) *100 : G3=FIX (G3) : IF G3<175 THEN G3=175
170  G4=G3*55 : G4=FIX (G4) : IF G4<10000 THEN G4=10000
180  G5=G4* (LOG (G1) /20) +10000
190  A1= -6400 : A2=5000 : A3=15000 : A4=10
200  B4=A4 : B2=A2: N3=G3 : N4=G4
205  PRINT@0, "ELAPSED ALTITUDE VELOCITY REMAINING INPUT FUEL" ;
206  PRINT@64, "TIME (KM)     (KM/HR) FUEL        BURN, (KG/SEC)";
210  PRINT@ 128+Q, T1; TAB (10) N3; TAB (24) B2; TAB (39) N4; TAB (53); :INPUT F
250  IF  F=0  GOTO  280
260  IF  F<0  OR F>100  GOTO  320
270  T=N4/F : IF  T<10  THEN  B4=T
```

```
280  N4=N4-(F*B4)
285  V1=B3
286  T1=T1+B4
290  B5=(G2+((G2*N3)/(A5*-2)))-((F*G5)/(A3+N4))
295  B3=B2+(B5*B4)
298  N5=N3
300  N3=N3+(((B3+B2)/A1)*B4)
305  B2=B3
307  IF N3<0 GOTO 450
310  IF N4<=0 GOTO 400 : GOTO 210
312  Q=Q+64 : IFQ +128>960 THFN Q=832
315  GOTO 205
320  PRINT "- ->> ILLEGAL FUEL BURN—DUMMY! - -TRY AGAIN (0 TO 100)"
     : GOTO 210
400  V2=SQR (B2〔2+N3*G2*5650) : PRINT "OUT OF FUEL AT; T1; "SECONOS"
410  V3=ABS (V2) *10000/3600
420  T1=T1+LOG((V3*N3*10000)/G1)
430  GOTO 1000
450  V2=SQR (ABS (N5/(26*B5))) * (26*B5) +V1 : GOTO 1000
460  T1=T1-(10-B4)
500  G1=980.7 : A5=6371 : A$="EARTH" : GOTO 145
600  G1=162 : A5=1738 : A$="MOON" : GOTO 145
700  G1=374 : A5=3380 : A$="MARS" : GOTO 145
800  G1=17.5 : A5=195 : A$="VESTA" : GOTO 145
1000 PRINT : PRINT "YOU HAVE" ;
1010 IF V2<20 PRINT "LANDED" : GOTO 1100
1020 IF V2<100 PRINT "CRASHED" : GOTO 1140
1030 IF V2<250 PRINT "BEEN OBLITURATED" : GOTO 5000
1040 IF V2<5000 PRINT" MADE A NEW CRATER" : GOTO 5000
1050 IF V2<4999 PRINT "BORED A HOLE INTO THE PLANET" : GOTO 5000
1100 IF V2<1 PRINT "NICE TOUCH- -VERY GOOD" : GOTO 5000
 110 IF V2<5 PRINT "NOT TOO BAD" : GOTO 5000
1120 PRINT "KIND OF ROUGH" : GOTO 5000
1140 IF V2<30 PRINT "YOU WILL NOT BE ABLE TO TAKE OFF" : GOTO 5000
1150 IF V2<45 PRINT "YOU ARE INJURED, THE LANDER IS ON FIRE" : GOTO 5000
1160 PRINT " THERE ARE NO SURVIVORS"
5000 PRINT "VELOCITY AT IMPACT***" TAB(40); ABS(V2); "KM/HR"
5010 PRINT "ELAPSED TIME******" TAB(40); T1; "SECONDS"
5020 END
```

2. 通 信 录

这个程序允许你将有关客人的姓名，地址及电话号码记录在磁带上以备参考。它可以用

“命令表”规定的动作调出，修改等等。这是一个建立通信／电话表的灵便方法。

```
 10 CLEAR 1000 : CLS : DIM N$(50) : DIM A(50) : DIM P$(50)
 20 CLS:PRINT@ 10, "**MENU**" : PRINT : PRINT
 30 PRINT "TO BUILD A FILE TYPE 1"
 40 PRINT "TO SEE THE ENTIRE FILE TYPE 2"
 50 PRINT "TO SEE AN INDIVIDUAL NAME TYPE 3"
 60 PRINT "TO MAKE CORRECTIONS TYPE 4"
 70 PRINT "TO SAVE THE CURRENT FILE ON TAPE TYPE 5"
 80 PRINT"TO INPUT A FILE FROM TAPE TYPE 6"
 90 INPUT "Q : ON Q GOTO 100, 200, 300, 400, 500, 600
100 INPUT "WHEN READY, HIT ENTER(TO CLO SE THE FILE TYPE 9999 FOR NAME)"; X
110 FOR I=1 TO 50 : CLS : PRINT "ENTER YOUR NAME (LAST FIRST, NO
    COMMAS PLEASE)
112 PRINT "THEN HIT THE ENTER KEY"; : INPUT N$(I)
115 IF N$(I)="9999" THEN P1=I : GOTO 150
120 INPUT "ENTER YOUR ADDRESS (NO COMMAS)"; A$(I)
130 INPUT "ENTER YOUR PHONE #"; P$(I)
135 IF FRE (X$)<100 GOTO 150
140 NEXT
150 PRINT "FILE CLOSED--" : INPUT "TO SEE THE MENU, HIT ENTER"; X
160 GOTO 20
200 CLS : FOR I=1 TO P1 : PRINT N$(I), A(I), P$(I) : NEXT
210 INPUT "TO SEE THE MENU, HIT ENTER"; X : GOTO 20
300 CLS : INPUT "ENTER THE NAME, LAST FIRST (NO COMMAS)"; N$
310 FOR I=1 TO P1 : IF N$(I)=N$THEN 330
315 NEXT
320 PRINT "NAME NOT IN FILE" : GOTO 340
330 PRINT N$(I), A$(I), P$(I)
340 PRINT : PRINT "FOR ANOTHER NAME TYPE 1, OTHERWISE 0"; : INPUT X
350 IF X=1 GOTO 300 ELSE 20
400 CLS : PRINT "ENTER THE NAME FOR THE LINE YOU WISH TO CHANGE
    (NO COMMAS)"
405 INPUT N$
410 FOR I=1 TO P1 : IF N$=N$ (I)GOTO430
415 NEXT
420 PRINT "NAME NOT IN FILE" : GOTO460
430 PRINT "ENTER THE CORRECTED INFO. : NAME, ADDRESS, PHONE"
440 INPUT N$(I), A$(I), P$(I)
450 PRINT "THE LINE NOW READS : " : PRINT N$(I), A$(I), P$(I)
460 INPUT "FOR ANOTHER CORRECTION TYPE 1, OTHERWISE 0"; X
470 IF X=1 GOTO 400
480 GOTO 20
```

```
500 CLS : INPUT "MAKE PREPARATIONS FOR CASSETTE, WHEN READY HIT ENTER"; X
510 PRINT "COPYING…"
520 PRINT #-1, P1
530 FOR I=1 TO P1 : PRINT#-1, N$(I), A$(I), P$(I) : NEXT
540 PRINT "COMPLETE- -NOTE TAPE LOCATION"
550 INPUT "TO SEE THE MENU, HIT, ENTER"; X : GOTO 20
600 CLS : INPUT "WHEN READY HIT ENTER"; X
610 PRINT "INPUTING……"
620 INPUT #-1, P1
630 FOR I=1 TO P1 : INPUT#-1, N$(I), A$(I), P$(I) : NEXT
640 PRINT "COMPLETE" : INPUT "TO SEE MENU, HIT ENTER"; X : GOTO 20
```

3. 有图示的三角运算

这个程序说明数学函数及图示的应用。这是研究三角几何的好方法（适合于中学生）。注意：在此程序清单中，向上箭头↑=(。

```
 10 CLS
100 PRINT "THIS PROGRAM CALCULATES THE AREA OF A TRIANGLE"
110 PRINT "GIVEN 3 PARAMETERS AND DRAWS THE TRIANGLE TO SCALE"
120 PRINT : PRINT "FOR 3 SIDES TYPE : SSS, FOR 2 SIDES AND 1 ANGLE TYPE : SAS, "
130 PRINT "FOR 1 SIDE AND 2 ANGLES TYPE : ASA"
140 INPUT A$ : IF A$="SAS" GOSUB 300
150 IF A$="ASA" GOSUB 400
200 'SSS
210 PRINT "ENTER 3 SIDES, (LONGEST SIDE FIRST) : "
220 INPUT L1, L2, L3
225 IF L2>L1 OR L3>L1 PRINT "***LONGEST FIRST PLEASE……" : PRINT : GOTO 210
230 S=(L1+L2+L3)/2
235 IF S<=L1 PRINT "***NOT A TRIANGLE***" : PRINT : GOTO 210
240 Y3=2 * SQR(S * (S-L2) * (S-L1) * (S-L3))/L1
250 A=Y3/L2 : A=ATN (A/SQR ( -A*A+1))
260 X3=COS (A) *L2
270 AR=(L1 * Y3)/2
280 GOTO500
300 'SAS
310 PRINT "ENTER 2 SIDES AND 1 ANGLE : AB, AC, THETA : (LARGEST SIDE FIRST)"
320 INPUT L1, L2, T
325 T=(T * 3.14159)/180
330 Y3=L2*SIN(T)
340 X3=COS(T)*L2
350 AR=(L1*Y3)/2
360 GOTO 500
400 'ASA
```

```
410 PRINT "ENTER 2 ANGLES AND 1 SIDE··THETA1, THETA2, AB:"
420 INPUT T1, T2, L2
425 T1=(T1 * 3.14159)/180:T2=(T2 * 3.14159)/180
430 Y3=L2 * SIN(T1)
440 B1=COS(T1) * L2
450 B2=Y3/TAN(T2)
460 L1=B1+B2 : X3=B1 : IF L2>L1 THEN X=L1 : L1=L2 : L2=X
470 AR=(L2*Y3)/2
500 CLS : F=1 : IF L1>50 OR Y3>30 OR L2>50 THEN GO SUB 700
510 VC=(3.14159 * (L1 * F-X3 * F) * (Y3 * F) [2)/3
520 VS=3.14159 * (X3 * F) * (Y3 * F) [2)/3 : VT=VC+VS
525 IF F=6 GOTO 610
530 S1=Y3/X3 : S2=Y3/(X3-L1)
532 IF INT(X3)=0 THEN 1100
533 IF INT(X3)=INT(L1) THEN 1000
534 IF X3<0 THEN 1299
535 IF X3>L1 THEN 1199
537 IF X3=L2 THEN 1000
540 FOR Y=20 TO L1*2+20 STEP 2 : SET(Y, Y3+5) : NEXT
550 FOR X=0 TO X3 : SET(X*2+20, S1*(X3-X)+5) : NEXT
560 FOR X=X3 TO L1 : SET(X*2+20, Y3+(S2 * (L1-X)+5)): NEXT
590 PRINT@64 * INT((Y3+5)/3)+69, "A(0, 0)"; TAB(L1); "B<"; L1 * F; ", 0>"
600 PRINT@ (X3+20)/2, "C<"; X3 * F; ", "; Y3 * F; ">";
610 PRINT@ 832, "AREA="; AR; "SQ. UNITS";
620 PRINT@ 896, "THE VOLUME OF THE SOLID CREATED BY REVOLVING THE
    TRIANGLE",
625 PRINT "ABOUT THE X AXIS (LINE AB)="; VT; "CUBIC UNTIS";
630 PRINT@ 768, "*"; : INPUT "TO RUN AGAIN, TYPE 1"; B6 : IF B6=1 THEN10
640 STOP : GOTO 10
700 IF L1<100 THEN F=2 : GOTO 750
710 IF L1<150 THEN F=3 : GOTO 750
720 IF L1<200 THEN F=4 : GOTO 750
730 IF L1<250 THEN F=5 : GOTO 750
740 PRINT "SORRY, SCALE TOO LARGE TO BE DRAWN"; : F=6 : GOTO 510
750 L1=L1/F : Y1=Y1/F : Y2=Y2/F : Y3=Y3/F : X1=X1/F : X2=X2/F : X3=X3/F
760 RETURN
1000 FOB Y=5 TO Y3+5 : SET (X3 * 2+20, Y) : NEXT : GOTO 540
1100 FOB Y=5 TO Y3+5 : SET (20, Y) : NEXT : GOTO 540
1199 IF X3>127 GOSUB 700
1200 FOR X=L1 TO X3 : SET (X*2+20, Y3+(S2*(L1-X)+5)) : NEXT : GOTO540
1299 IF X3<-10 GOSUB 700
1300 FOR X=X3 TO 0 : SET (X * 2+20, Y3+(S1 * (0-X)+5)) : NEXT : GOTO 540
```

4.高炮射击演习

这个程序使用 INKEY$ 函数模拟一个流行的“电视游戏”。它只需要极少几行。这个程序能容易地“编出”——要选择快速目标或慢速目标，进行计算，打印信息等等。要改变目标的速度，请修改行 40：用 RND(0)*S1 代替RND (10)／10。要慢速运动的目标，令 S1 小于 1；要快速目标，令 S1 大于 1。但是S1不能超过 1.5，否则目标跑到下一行。

```
 1  CLS : PRINT : PRINT CHR$ (23) ; "HIT 'Z' KEY TO AIM LEFT"
 2  PRINT  "HIT '/' KEY TO AIM RIGHT"
 3  PRINT  "HIT SPACE BAR TO FIRE"
 4  FOR  I=1  TO  5000  : NEXT
10  CLS : CA=928 : I=1 : PRINT@CA, "*"; : PRINT@ 991, "***";
20  F=0
30  IF  I>=15  PRINT @ 124,  "          " ; : I=1
40  PRINT@ 64+I *4, "       "; : I=I+RND(10)／10 : PRINT@64+I*4,  "- - >";
50  IF  F=0  THEN  200
60  RESET (MX, MY) : MX=MX-MD : MY=MY-8 : IF MX<=0 OR MX>=127 THEN 20
70  IF  MY>2  SET (MX, MY) : GOTO  30
80  IF  ABS (I*8-MX) >4  THEN  20
90  FOR J=1 TO 6:PRINT@64+4*I,  "****"; : FOR K=1 TO 50 : NEXT
95  PRINT@ 64+4*I, "       " : FOR K=1 TO 50 : NEXT K, J
100 GOTO  10
200 Y$=INKEY$
205 IF  F=1  STOP
210 IF  Y$<> "Z" THEN 250
220 IF  CA<922  THEN  30
230 PRINT@ CA,  "       "; : CA=CA-1 : GOTO  280
250 IF  Y$<>"／" THEN 300
260 IF  CA>934 THEN  30
270 PRINT@ CA, "       "; : CA=CA+1
280 PRINT@ CA,  "*"; : GOTO30
300 IF  Y$<> "     " THEN 30
310 F=1 : MD=928-CA : MY=40 : MX=64-3*MD : SET (MX, MY) : GOTO  30
311 END
```

5. 准备—瞄准—射击!

这个程序应用 INKEY$ 函数将 LEVEL I 中的弹球程序变成了一两个人玩的游戏。进入一个三位数字作为目标（因三位数取决于显示器X轴，因此它必须在001—126范围内，并且对一位或两位数，一定要在前边写 0），用火箭将此目标击毁。游戏中总是计算机先射击，然后 1 井游戏者射击。

```
5  DIM  N$(4)
6  CLS : INPUT  "ENTER  THE  NO . OF  PLAYERS";  X1 : PRINT  "ENTER";  X1;
   "1ST  NAMES : "
```

```
 7  FOR  XI=1  TO  X1 : INPUT N$(XI) : NEXT : XI=1
10  CLS
20  FOR  M=0  TO  127 : SET (M, 0):SET (M, 47) : NEXT
30  FOR  M=0  TO 47 : SET (0, M) : SET (127, M) : NEXT
35  FOR  X=1  TO  121  STEP  10 : RESET (X, 0) : NEXT
40  RANDOM : Y=RND(40)+1 : X=RND(110)+4
50  D=1 : Q=1 : Z=64
60  RESET(Z, Y-D) : RESET(X-Q*4, 24)
70  SET (Z, Y) : SET (X, 24) : GOSUB  500
80  Y=Y+D : X=X+Q
90  IF  X=123  OR  X=4  THEN GOSUB  700
100 IF  Y=47  THEN  120
105 IF  Y=0  GOSUB  900
110 IF  Y<>-1  OR X<>-1 THEN  60
120 Y=Y-2*D : D=-D : GOTO  60
500 IF  X=Z  OR  X=Q+Z  OR  X=2*Q+Z  OR  X=3*Q+Z  OR  X=Q*4+Z  THEN
    IF  Y=24  GO SUB  600
510 IF  Y=23  OR  Y=24  OR  Y=25 THEN IF  X=Z  GOSUB  600
520 RETURN
600 X=1
610 FOR  Z=1  TO  50 : PRINT@  550, "HIT! ! ! "; : NEXT
620 FOR  Z=1  TO  25 : PRINT@ 550, "       "; : NEXT
630 X=X+1:IF X<5 GOTO  910
640 GOTO  2000
700 X=X-2*Q : Q=-Q : RETURN
900 T$=INKEY$ : A$=" ":B$=" ":C$=" "
1000 A$=INKEY$ : IF LEN(A$)=0 THEN 1000
1005 PRINT@ 0,  A$;
1010 B$=INKEY$ : IF  LEN(B$)=0  THEN  1010
1015 PRINT@  1, B$;
1020 C$=INKEY$ : IF LEN(C$)=0 THEN  1020
1025 PRINT@ 2, C$;
1030 RESET(Z, 1) : X$=A$+B$+C$ : Z=VAL(X$) : IF Z>126 GOTO 1100
1033 PX=PX+1
1035 GOTO  120
1040 RETURN
1100 FOR X=1 TO 50 : PRINT@ 70, "TOO LARGE, TRY AGAIN" : NEXT
1110 PRINT@  70, "          " :Z=1 : GOTO  1000
2000 IF  PX=0  GOSUB 3000
2010 CLS : PRINT"      ***"; N$(XI); "***" : PRINT : PRINT
2017 PX(XI)=PX+PX(XI) : PH(XI)=PH(XI)+1
2020 PRINT,  "SHOTS        HITS       PERCENTAGE"
```

```
2030  PRINT : PRINT "THIS ROUND"; TAB(17)PX; TAB(28)"1"; TAB(42)(1/PX)*100
2035  IF PX(1)=0 THEN PX(1)=1
2040  PRINT : PRINT "TOTAL      "; TAB(17) PX(XI);
2042  PRINT TAB(28)PH(XI); TAB(42) (PH(XI)/PX(XI))*100
2045  FOR X=1 TO 2500 : NEXT
2050  XI=XI+1
2060  IF XI>X1 THEN XI=1
2065  PX=0
2070  GOTO10
2115  IF PX=0 GOSUB 3000
3000  PRINT@0, "WHAT LUCK ! ! ! " : PX=1 : RETURN
```

（周宝兴译　宋知用校）

第二篇　TRS-80 FORTRAN使用手册

一、概　　述

本手册讲述 Radio Shack 的 FORTRAN 包，要和 TRS-80 磁盘操作系统（TRSDOS）一起使用。它不教你如何编写 FORTRAN 程序，为此，你必须阅读有关参考书，在本节的第 2 小节中已列出一些书目。

FORTRAN 包包括四个模块：

1）编辑程序(EDIT/CMD)，用于写和编辑 FORTRAN 源程序。

2）编译程序(F80/CMD),用于读 FORTRAN 源程序并将它翻译为浮动目标程序。

3）链接装配程序(L80/CMD)，它用于装入和执行编译程序，并将它们存贮为 TRSDOS 命令文件

4）FORTRAN 子程序库（FORLIB/REL)，由装配程序用于链接用户的浮动 FORTRAN 文件。

在手册中，你会发现有些地方涉及另外两个程序模块，即 MACRO-80 汇编程序和 CREF-80 交叉参照设备，这些程序不包括在 FORTRAN 包中，也是使用 FORTRAN 所不必要的。因此，可以忽略所有涉及此两模块的地方。

FORTRAN 包的使用

三个程序存贮在两个磁盘上：

1)　磁盘 1# 有 F80/CMD 和 EDIT/CMD

2)　磁盘 2# 有 L80/CMD 和 FORLIB/REL

每个磁盘上也有全部 TRSDOS 和磁盘 BASIC 文件，因此，两者都可放入驱动器 0。

注：

(1) 在你使用 FORTRAN 包之前，你应当复制这两个 FORTRAN 盘片的付本。

(2) 千万不能把尚有打开文件的磁盘盘片从驱动器中取出。例如，你不应在编辑期间交换磁盘。在更换磁盘之前，一定要关闭所有文件。

1. 程序实例

这个程序实例给你一个机会去练习使用 FORTRAN 包。可使你看到各个部分如何配合在一起。假设这个系统有两个驱动器（驱动器 0 和 1），所以不用交换盘片。在尝试这个程序实例以前，单驱动器用户应当参阅 F80 编辑器手册，因为某些过程需要改变。

在显示 DOS READY 的时候：

第一步：在驱动器里放入 1# 盘，并敲入命令 EDIT。这个命令装入 EDIT-80 文本编辑器。EDIT-80 将响应

FILE:

如果你使用图 1 中的程序，则在敲入文件名 TEMP/FOR 后按一下 [BREAK] 键。如果用你自己的 FORTRAN 程序，则敲任意合法的 TRSDOS 文件名。当建立一新文件时，总是在文件名后按 [BREAK] 键，当阅读存在的文件时，总在文件名后按 [ENTER] 键。在 EDIT-80 打印出信息：

```
CREATING
VERSION ×.×
COPYRIGHT 1977, 78(C) BY MICROSOFT
CREATED: ××××
××××BYTES FREE
*
```

然后打入命令：　　I

EDIT-80 将打印出 00100，这是第一行的行号。

第二步：开始输入图 1 列出的 FORTRAN 程序(或输入你自己的 FORTRAN 程序)，每当你 [ENTER] 一行后，EDIT-80 将打印出下一行的行号。

当你在打入程序时，你可应用所有的 EDIT-80 的编辑功能，阅读 EDIT-80 用户指南，你将发现：插入和删除一行、修改文本、以及检索文本是如此之容易。

在 TRS-80 上写任何 FORTRAN 程序时，请使用本手册来检查所有的 FORTRAN 语句的句法和用法是否正确。

第三步：将程序全部打入以后，在下一个可用行号之后按 [BREAK] 键，即回到 EDIT-80 命令级。要退出编辑，输入命令：　　E

于是，你打入的程序已用文件名 TEMP/FOR 存贮起来（TEMP 是你在第一步里规定的文件名；而/FOR 是由编辑程序提供的空缺后缀）。TEMP/FOR 称为源文件，它已可以进行编译。

第四步：句法检查。

在进行下一步以前，最好检查程序的句法有无错误，消除句法错误可避免重新编译。要检查称为 TEMP/FOR 的源文件的句法，将 1# 磁盘放入盘驱动器并打入：

```
F80 =TEMP
```

F80 是编译程序文件名，=TEMP 是一个参量，它告诉编译器该文件需编译。由于没有提供后缀，F80 就使用空缺后缀/FOR。因为不规定目标文件和列表文件，所以编辑器不输出这两者。这就是所谓“干运行”，只看看有无错误产生。

如果有错误，请删除文件 TEMP/FOR，并小心地重复步骤 1 到 4（在此练习中，我们不准备使用编辑程序的方便的编辑命令，这些命令在 EDIT-80 用户指南中讲述）。

在处理过程中，将显示出 $MAIN，当编译完成以后，又显示出 DOS READY。

第五步：编译源文件

要编译源文件 TEMP/FOR 并产生目标文件和列表文件，则打入下列命令：

```
F80 TEMP, TEMP=TEMP
```

这时，除指定了源文件 （=TEMP 有空缺后缀/FOR）以外，还指定了输出浮动目标码文件和列表文件（显示源语句以及有关的编译器动作）。目标文件 TEMP 得到空缺后缀

/REL，列表文件得到空缺后缀/LST。有关句法的详细内容，请见本手册第二节。图2就是TEMP/FOR产生的列表文件TEMP/LST。

第六步：装入并执行程序

要将程序装入内存并执行程序，把2#磁盘放入盘驱动器并打入

L80 TEMP-G

这命令告诉TRSDOS装入并运行LINK-80，而LINK-80又将目标文件TEMP/REL（LINK-80提供了空缺后缀/REL）装入当前内存地址，检索系统库以消除无定义的参数，并执行程序。在此情况下，LINK-80不产生命令文件。图3是实例运行的结果。

第七步：存贮目标码程序

目标文件一旦由LINK-80装入，就是TRS-80计算机可执行的形式。要存贮这个文件，打入

L80 TEMP-N, TEMP-E

这命令建立一个命令文件，它可以在TRSDOS直接运行。TEMP-N告诉LINK-80该文件取名TEMP/CMD；TEMP-E告诉LINK-80装入目标文件TEMP/REL，其中/CMD和/REL都是空缺后缀。

于是，在DOS READY时，你打入

TEMP [ENTER]

就可以象TRSDOS命令文件一样装入并运行此程序。

TRS-80 FORTRAN包的功能远比本简短实例中描述的强得多。经常试验，你将惊奇地发现，TRS-80已增加了如此多的计算功能。

图1 FORTRAN 源程序

```
00100      C     CONVERT FAHRENHEIT TO CENTIGRADE
00200            INTEGER F
00300            WRITE(5.5)
00400      5     FORMAT (33H  FAHRENHEIT  CENTIGRADE)
00500            DO 20 F=20, 65, 5
00600            C=5./9. * (F—32)
00700            WRITE(5, 10)F, C
00800      10    FORMAT(12X, I2, 11X, F6.3)
00900      20    CONTINUE
01000            END
01100      $     ←(这是对 [BREAK] 键的响应)
```

图2 列表文件 TEMP/LST

```
1 .  FORTRAN-80 VER.3.2 COPYRIGHT 1978〈C〉BY MICROSOFT
2 .  BYTES: 3699
3 .  CREATED: 15-FEB-79
4 .  00100  C       CONVERT FAHRENHEIT TO CENTIGRADE
5 .  00200          INTEGER F
6 .  00300          WRITE(5,5)
7 .  *****  0000'  LD     BC, $$L
8 .  *****  0003'  JP     $INIT
```

```
 9.  *****  0006'  LD    DE, 5 L
10.  *****  0009'  LD    HL, [  05  00]
11.  *****  000C'  CALL  $W2
12.  00400   5     FORMAT(33H  FAHRENHEIT  CENTIGRADE)
13.  *****  000F'  CALL  $ND
14.  00500         DO 20 F=20, 65, 5
15.  00600         C=5./9.*(F-32)
16.  *****  0012'  LD    HL,0014
17.  *****  0015'  LD    (F), HL
18.  00700         WRITE(5,10)F,C
19.  *****  0018'  LD    HL, (F)
20.  *****  001B'  LD    DE, FFE0
21.  *****  001E'  ADD   HL, DE
22.  *****  001F'  LD    (T;000000), HL
23.  *****  0022'  LD    HL, [00 00 20 83]
24.  *****  0025'  CALL  $L1
25.  *****  0028'  LD    HL, [00 00 10 84]
26.  *****  002B'  CALL  $DB
27.  *****  002E'  LD    HL, (T; 000000)
28.  *****  0031'  CALL  $MA
29.  *****  0034'  LD    HL, C
30.  *****  0037'  CALL  $T1
31.  *****  003A'  LD    DE, 10L
32.  *****  003D'  LD    HL, [ 05 00 ]
33.  *****  0040'  CALL  $W2
34.  00800   10    FORMAT(12X, I2, 11X, F6.3)
35.  *****  0043'  LD    DE, F
36.  *****  0046'  LD    HL, [  01  00]
37.  *****  0049'  LD    A, 02
38.  *****  004B'  CALL  $I0
39.  *****  004E'  LD    DE, C
40.  *****  0051'  LD    HL, [  01  00]
41.  *****  0054'  LD    A,02
42.  *****  0056'  CALL  $I1
43.  *****  0059'  CALL  $ND
44.  00900   20    CONTINUE
45.  01000         END
46.  *****  005C'  LD    HL, (F)
47.  *****  005F'  LD    DE,0005
48.  *****  0062'  ADD   HL, DE
49.  *****  0063'  LD    A,41
50.  *****  0065'  SUB   L
51.  *****  0066'  LD    A,00
```

```
52.   *****  0068'  SBC   H
53.   *****  0069'  JP    P,0015'
54.   *****  006C'  CALL  $EX
55.   *****  006F'  0100
56.   *****  0071'  0500
57.   *****  0073'  00002083
58.   *****  0077'  00001084
59.
60. PROGRAM UNIT LENGTH=007B (123) BYTES
61. DATA AREA LENGTH=0040 (64) BYTES
62.
63. SUBROUTINES REFERENCED:
64.
65. $I1    $IO     $INIT
66. $W2    $ND     $L1
67. $DB    $MA     $T1
68. $EX
69.
70. VARIABLES:
71.
72.   F    00001"  C  0029"  T: 000000
73.
74. LABELS:
75.
76.   $$L   0006'  5L  0003"  20L  005C'
77.   10L   002F"
78.
```

图3　TEMP/FOR 程序的输出

FAHRENHEIT	CENTIGRADE
20	-6.667
25	-3.889
30	-1.111
35	1.667
40	4.444
45	7.222
50	10.000
55	12.778
60	15.556
65	18.333

2. TRS-80 FORTRAN 手册的注意事项

1) 在你使用 FORTRAN 包之前，你应当复制这两个 FORTRAN 盘片的付本。

2) FORTRAN-80 参考手册严格说是本 TRS-80 FORTRAN 语言的句法与语法参考

资料，它不能扩展为FORTRAN程序设计的指导书。如果你对FORTRAN语言是生疏的，并想借助学习一下语言的话，我们建议你参考：

〔1〕 "Guide to FORTRAN-IV Programming" by Daniel McCracken (Wiley, 1965)

〔2〕 "Ten Statement FORTRAN plus FORTRAN IV" by Michael Kennedy and Martin B. Solomon (Prentice-Hall, 1975, Seeond Edition)

〔3〕 "FORTRAN" by kenneth P. Seidel (Goodyear, 1972)

〔4〕 "FORTRAN IV, A Self-Teaching Guide" by Jehosua Friedmann, Philip Greenberg, and Alan Hoffbert (Jotn Wiley & Sons, Inc., 1975)

〔5〕 "FORTRAN, A Structured, Disciplined, Style" by Gordon B. Davis and Thomas R. Hoffman (McCraw-Hill Book Company, 1978)

3) LINK-80手册严格说是一本可用命令和开关的参考资料。

二、TRS-80 FORTRAN编译器

如果你按照程序实例进行过练习，你就开始熟悉TRS-80 FORTRAN包软件了。现在，让我们特别着重看看TRS-80 FORTRAN编译器。

1. 运行编译器

当你给出TRSDOS命令

F80

时（1#盘必须在盘驱动器上），你就把TRS-80 FORTRAN编译器运行起来了。FORTRAN编译器取来个FORTRAN程序（源文件）并编译它，产生一个浮动目标文件，也就是机器码文件。当编译器准备接收命令时，它发给用户一个星号。要退出编译器，使用 BREAK 键。

命令也可打在同一行上，这称作"命令行"。在程序实例中，当我们打入命令行

F80 =TEMP

时，实际上已经这样做了。在执行一个命令行后，编译器自动地返回到操作系统。

2. 命令格式

一个编译器命令将文件名以及你希望使用的选择项传送给你想编译的源文件，以下是编译命令格式（方括号表示选择项）：

〔目标文件名〕〔,列表文件名〕=源文件名〔-开关…〕

注：所有的文件名必须是TRSDOS文件名格式：

文件名〔/ext〕〔.Password〕〔:drive#〕

如果你使用编译器的空缺后缀，那就不必在编译命令中指定后缀。

让我们分别研究编译命令的每一部分：

1) 目标文件名

要产生一个浮动目标文件，必须有这部分命令。这仅仅是你希望调用的目标文件的名称。目标文件名的空缺后缀是/REL。

2） 列表文件名

要产生一个列表文件，必须有这部分命令。这仅仅是你希望调用的列表文件的名称。列表文件名的空缺后缀是/LST。

3） 源文件名

一个编译器命令总必须包含一个源文件名——也就是让编译器“知道”编译什么。这仅仅是一个你存在盘上的FORTRAN程序名。FORTRAN源文件名的空缺后缀是/FOR。在编译器命令中，源文件名前总有一个等号。

例如（*是由F80给出的）：

*=TEST

编译程序TEST/FOR而不产生目标文件或列表文件

*TEST, TEST=TEST

编译程序TEST/FOR，产生一个称为TEST/REL的浮动目标文件和一个称为TEST/LST的列表文件。

*, TEST.PASS=TEST.PASS

编译程序TEST/FOR.PASS并产生一个称为TEST/LST PASS的列表文件（不产生目标文件）。

*TESTOBJ=TEST

编译程序TEST/FOR并产生一个称为TESTOBJ/REL的目标文件（不产生列表文件）。

4） 开关

命令末端的开关指定一个在编译期间使用的特殊的参数。开关前面总有一短划线（—）。在同一个命令中可以使用多个开关。可使用的开关有：

开　关	动　作
O	以8进制形式打印所有的列表地址。
H	以16进制形式打印所有的列表地址（空缺条件）。
N	不开列产生的目标码。只开列FORTRAN源码。
P	每一个-P分配一个额外的100字节栈空间供编译期间使用。如果在编译期间出现栈溢出错误，则使用-P，否则不需要。
M	指定编译器，使其产生的码应具有能装到ROM里去的形式，当指定-M后，产生的码有以下两点不同于正常的码形： ①FORMAT　将放在程序区，在它们周围有一个“JMP”。 ②参数块（为了调用多于三个参数的子程序）将在运行时被初始化，而不是由装入程序来初始化。

举例：

*CT.ME,CT.ME=CT.ME-O

编译程序CT/FOR.ME，产生一个称为CT/LST.ME的列表文件和称为CT/REL.ME的目标文件。在列表文件中地址是8进制的。

*CT, CT=CT-N

编译程序CT/FOR.产生一称为CT/LST的列表文件。列表文件只包含FORTRAN

源语句，不产生目标码。

*MAX10=MAX10-P-P

编译程序 MAX10/FOR 并产生一个称为 MAX10/REL 的目标文件。有 200 个额外字节的栈空间分配给编译器。

注：如果 FORTRAN 程序是打算给 ROM 的，那么程序员应意识到如下几点：

(1) DATA 语句不应用于初始化 RAM。这种初始化是由装入程序来作的，所以在执行时将不出现。变量和数组在执行期间可通过赋值语句初始化或用 READ 读入它们。

(2) 在执行期间不读入 FORMAT。

(3) 如果使用标准的 I/O 库程序，若 LUN 不是 6，7，8，9，10，则盘文件打不开。如果盘 I/O 需要其他 LUN，那么，用指示盘驱动器程序的相应地址重新编译 $LUNTB。

在执行开始以前，库程序 $INIT 将栈指针置在可用内存的顶部（像操作系统所指出的那样）。

调用规则是

```
LXI  B,〈返回地址〉
JMP  $INIT
```

如果产生的码是为某个其他机器准备的，这个程序就可能要重写。标准的初始化程序的源程序在盘上，称为"INIT/.MAC"。其中，只有建立栈指针这部分程序须由用户修改。FORTRAN 库已包含了标准的初始化程序

3. 输入/输出设备名

在 FORTRAN I/O 语句 READ 及 WRITE) 中 LUN 1，3，4 和 5 代表控制台/键盘。LUN2 代表行式打印机。而 LUN6-10 代表盘驱动器。

三、TRS-80 FORTRAN 盘文件

请参阅 FORTRAN-80 参考手册第八节第 3 小节

1. 空缺盘文件名

TRS-80 FORTRAN 可以随机存取盘文件，也可以顺序存取盘文件。由 READ 或 WRITE 语句打开的任一盘文件都给了一个空缺文件名，这个空缺文件名依赖于 LUN。

LUN	空缺文件名
6	FORT 06/DAT
7	FORT 07/DAT
8	FORT 08/DAT
9	FORT 09/DAT
10	FORT 10/DAT

2. 调用 OPEN

代替 READ 或 WRITE，可通过调用 OPEN 子程序来打开盘文件。调用 OPEN 的格式是：

CALL OPEN (LUN, Filename, Reclen)

式中

LUN＝与文件有关的逻辑单元数（必须是1至10之间的整常数或整变量）。

Filename＝与文件有关的TRSDOS的ASCII名。Filename应当是一个Hollerith名或者文字常数名，或着变量名或者数组名。其中的变量或数组包含着ASCII名。

Filename应具有通常TRSDOS所需要的形式

filename/ext.password：drive#

而且，应当用一个非字母字符结束，最好用一个空格。

Reclen＝你希望指定作为记录长度的字节数（最多256个字节）。空缺记录长度是128个字节。Reclen必须是一整常数或整变量。如果Reclen等于零，则记录长度将是256个字节。

下面是正确的OPEN调用例子：

CALL OPEN (6,'TIME/DAT. JULY：1', 256)

CALL OPEN (7,'COUNT/NUM', 200)

CALL OPEN (1,'TESTQ/MIN：2', 100)

四、错 误 信 息

1. FORTRAN编译器错误信息

FORTRAN编译器检测两类错误：警告性错误和致命性错误。当发现一个警告性错误后，编译继续进行源程序的下一款项。当找到一个致命性错误后，编译器就不管余下的逻辑行，包括继续行在内，从而停止编译。警告错误前有一百分号(%)，致命性错误前有一问号(?)。接下来打印编辑器行号（如有的话）或者打印实际行号。跟着打印错误码或错误信息。例如：

? Line25：Mismatched Parentheses

%Line16：Missing Integer Variable

当任一类错误出现时，应当修改程序使它编译无错。显然，一个编译有错的程序是保证不能执行的。

致命性错误：

错误号	信息	错误号	信息
100	非法语句标号	107	无效DATA常数或重复因子
101	不认识的语句或拼写错误	108	不正确的DATA常数编号
102	非法语句结束	109	不正确的整常数
103	非法的DO嵌套	110	无效的语句标号
104	非法数据常数	111	不是一个变量名
105	遗漏名称	112	非法逻辑算符形式
106	非法过程名	113	数据区溢出

114	文字串太大
115	在I/O中有无效数据列表单元
116	不平衡的DO嵌套
117	标识符太长
118	非法操作符
119	括号不配对
120	操作符相连
121	不适当的下标句法
122	非法整型量
123	非法Hollerith结构
124	相反的DO循环
125	非法语句函数名
126	句中有非法字符
127	语句缺乏顺序
128	遗漏整型量
129	无效逻辑操作符
130	在INTEGER或REAL或LOGICAL后跟着非法项
131	在输入设备上过早结束文件
132	非法的混合型操作
133	函数调用没有参量
134	栈溢出
135	在逻辑IF后有非法语句

警告性错误：

0	重复语句标号
1	非法DO终止
2	块名=过程名
3	错用数组名
4	COMMON名用法错
5	下标错误编号
6	在一组内数组多重EQUIVALENCE
7	COMMON的多重EQUIVALENCE
8	COMMON基础较低
9	在BLOCK DATA中有非COMMON变量
10	无格式写语句没有清单
11	非整型表示式
12	操作方式和操作符不一致
13	不允许的混合操作方式
14	遗漏整变量
15	在FORMAT中遗漏语句号
16	零重复因子
17	零格式值
18	格式嵌套太深
19	语句号与FORMAT无关
20	使用了无效语句号
21	没有路径通向这个语句
22	遗漏的DO终止
23	在BLOCK DATA中有码输出
24	出现非定义标号
25	在主程序中有RETURN
26	在READ中有STATUS错误
27	使用了无效运算量
28	函数没有宗量
29	Hex常数溢出
30	被零除
31	要用数组名
32	将非法宗量给ENCODE/DECODE

2. FORTRAN运行时的错误信息

在运行FORTRAN程序时，可能出现下列的一个或多个错误。致命性错误引起执行停止。在警告性错误后继续执行。然而，在20个警告性错误后，执行停止。运行时的错误用星号围起来，如下所示。

FW

警告性错误

IB	超出输入缓冲器极限
TL	在 FORMAT 中左括号太多
OB	超出输出缓冲器极限
DE	十进制指数溢出（在输入数据流中有大于 99 的指数）
IS	整数太大
BE	二进制指数溢出
IN	输入记录太长
OV	算术溢出
CN	在实数到整数转换中转换溢出
SN	SIN 的宗量太大
A2	ATAN 2 中的两个宗量都是零
IO	非法 I/O 操作
BI	在二进制 I/O 时超出缓冲器尺寸
RC	在 FORMAT 中有负的重复计数

致命性错误

ID	非法的 FORMAT 描述符
FO	FORMAT 区域宽度为零
MP	在 FORMAT 中遗漏周期
FW	FORMAT 区域宽度太小
IT	I/O 传输错
ML	在 FORMAT 中遗漏左括号
DZ	实数或整数被零除
LG	LOG 函数中有非法宗量（负数或零）
SQ	SQRT 函数中有非法宗量（负数）
DT	数据类型和 FORMAT 清单不一致
EF	在 READ 中遇到 EOF

（陈品瓒　译　　周宝兴　校）

第三篇　FORTRAN-80

参考手册

一、引　言

FORTRAN 是一种为简化计算机程序的编制和调试而设计的用途广泛的面向问题的程序语言。FORTRAN 语言的名称是英语的公式翻译(FORmulaTRANslator)的字头缩写。

使用此语言的句法规则是严格的，并要求程序员将一个问题的各特性用一系列严谨的语句充分明确地表达出来。这些语句称为源程序，它们将由叫做 FORTRAN 处理器的系统程序翻译成计算机能够执行的程序，即机器语言形式的目的程序。

本手册规定了 8080 和 Z-80 微型计算机的 FORTRAN 源语言。它包括了美国国家标准 FORTRAN 语言（1966.3.7 通过的 ANSI 文件 X3.9—1966），并加了一些语言扩充和限制，这些语言扩充和限制在本手册的文本中有详细描述，并列在附录 1 中。

注：本 FORTRAN 与标准 FORTRAN 的差别在于它不包括复数数据类型。

在整个手册中有例子说明各语言元素的结构和使用。程序员应熟知语言的所有部分，以充分运用它的功能。

第二节描述 8080 FORTRAN 源程序的格式和各部分。第三节和第四节规定了数据类型和它们的表达关系式，第五节至第九节描述了各种语句类型的相应结构和用法。

二、FORTRAN 程序格式

8080 FORTRAN 源程序由一个称作主程序的程序单元和若干叫做子程序的程序单元所组成。有关子程序的类型、书写方法以及使用将在本手册的第九节讨论。

程序和程序单元是由语句的命令组构成的，这些语句精确地描述了问题的求解过程，也确定了在编译目的码过程中 FORTRAN 处理器使用的信息。每一条语句是按照下述格式用 FORTRAN 字符集书写的。

1. FORTRAN 字符集

为了简化引用和说明，FORTRAN 字符集分为四个小组，并各给它们一个名称。

1) 字母

A、B、C、D、E、F、G、H、I、J、K、L、M、N、O、P、Q、R、S、T、U、V、W、X、Y、Z、$

注：大写字母和小写字母是没有区别的，但希望你只用大写字母，以便明了易读。

2) 数字

0、1、2、3、4、5、6、7、8、9

注：表示数量的数字串通常解释为十进制数。但是，在某些语句中，用十六进制数说明，在这种情况下，字母 A、B、C、D、E、F 也用作十六进制数字。十六进制的用法在那些允许使用这种数制的语句的描述中进行规定。

3） 字母数字

由全部数字和全部字母组成的一个字符小组。

4） 特殊符号

	空格	/	斜道
=	等号	(	左括号
+	正号	)	右括号
-	负号	,	逗号
*	星号	.	小数点

注：(1) FORTRAN 程序行由 80 个字符位置或列组成，编号从 1 到 80，并分为四个区。

(2) 以下特殊符号按分类称作算术操作符，在算术表达式的语句中有意义。

+	加或取正值	*	乘	**	指数
-	减或取负值	/	除		

(3) 其他特殊符号在 FORTRAN 语言的句法表示和 FORTRAN 语句的结构中有特殊用途。

(4) 任何可打印的字符都可出现在 Hollerith 区域或文字区域中。

2. FORTRAN 行格式

FORTRAN 编码格式（图 2.1）的实例表明了 FORTRAN 程序行的格式，行的格式由 80 个字符位置或列组成，编号为 1 到 80 并分为四个区。

1） 语句标号（或编号）区　第 1 列至第 5 列（见语句标号的定义）

2） 连续行字符区　第 6 列

3） 语句区　第 7 列至 72 列

4） 标识区　第 73 列至 80 列

标识区可用于 FORTRAN 程序员所希望的任何用途，FORTRAN 处理器是不管的。

FORTRAN 语句行根据行类型放在第 1 列至 72 列格式内。四种行的类型、它们的定义以及列的格式是：

1） 注释行——用于源程序注解，以方便程序员阅读使用等。

(1) 第 1 列要有字母 C。

(2) 第 2 列至 72 列用于任何希望的格式来表达注释，或者是空格。

(3) 注释行后可以只跟一个初始行、END（结束）行，或其他注释行。

(4) 注释行对目的程序没有影响，并被 FORTRAN 处理器忽略不管，只有在开列程序清单时作显示用。

例如：

```
C    COMMENT LINES ARE INDICATED BY THE
C    CHARACTER C IN COLUMN 1.
```

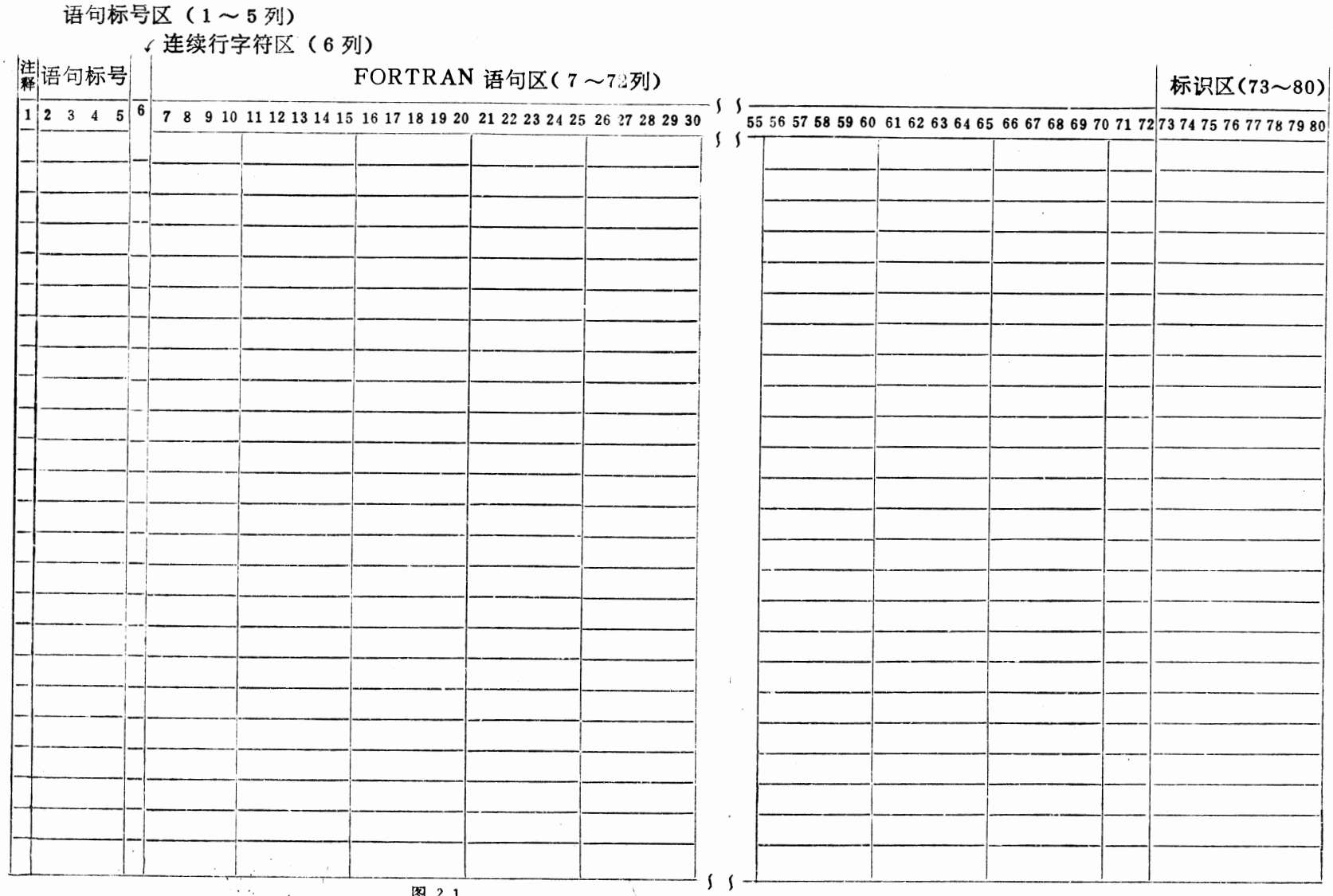

语句标号区（1～5列）
连续行字符区（6列）
注释
语句标号
FORTRAN 语句区(7～72列)
标识区(73～80)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

图 2.1

```
C     THESE ARE COMMENT LINES
```

2) END（结束）行——程序单元的最后一行。

(1) 第 1 列至 5 列可以有语句标号。

(2) 第 6 列必须是零或空格。

(3) 第 7 列至 72 列包含字符E、N、D，在此命令中，其前面、中间、或后面都可以加空格符号。

(4) 每个 FORTRAN 程序单元必须有一个 END 行作它的最后一行，告诉处理器它是程序单元的实际结束。

(5) END 行可以跟在任何其他类型之后。

例如：

```
      END
```

3) 初始行——每个语句的第一行或唯一行。

(1) 第 1 — 5 列可以有标识语句的语句标号。

(2) 第 6 列必须是零或空格。

(3) 第 7 —72 列包含所有或部分语句。

(4) 初始行可以在语句区的任何地方开始。

例如：

```
C     THE STATEMENT BELOW CONSISTS
C     OF AN INITIAL LINE
C
                A=.5*SQRT (3-2.*C)
```

4) 连续行——在需要添加程序行时使用，以完成由初始开始的语句。

(1) 第 1 — 5 列可以不管，除非第 1 列是 C。

(2) 如果第 1 列是 C，则它是注释行。

(3) 第 6 列必须有**非零或非空格**字符。

(4) 第 7 —72 列包含语句的继续部分。

(5) 为完成一条语句，可以按需要有许多连续行。

例如：

```
C     THE STATEMENTS BELOW ARE AN INITIAL LINE
C       AND 2 CONTINUATION LINES
C
 63             BETA (1, 2)=
     1    A6BAR**7-(BETA(2, 2)-A5BAR*50
     2    +SQRT(BETA(2, 1)))
```

语句标号放在 FORTRAN 语句初始行的第 1 — 5 列内，它用于被其他语句引用的目的。

在使用语句标号中应注意下列几点：

(1) 标号是 1 至 99999 的整数。

(2) 标号数值前面的零和空格是没有效的。

(3) 一个标号在程序单元中必须是唯一的。

(4) 连续行中的标号 FORTRAN 处理器是不管的。

例如：

```
C     EXAMPLES OF STATEMENT LABELS
C
      1
     1 01
   99999
    763
```

3. 语句

个别的语句是与程序单元中描述的过程的详细情况有关的，它可以分为可执行的和不可执行的两类。

可执行语句确定了动作，并使 FORTRAN 处理器产生目的程序指令。有三种可执行语句：

(1) 置换语句

(2) 控制语句

(3) 输入/输出语句

非执行语句在程序装入和执行过程中为处理器描述了数据的种类和排列，并为目的程序提供了有关输入/输出格式和数据初始化信息。非执行语句有五种：

(1) 说明语句

(2) 数据初始化语句

(3) 格式 (FORMAT) 语句

(4) 函数 (FUNCTION) 定义语句

(5) 子程序语句

各类语句的相应用法和结构将在第五节到第九节中叙述。

三、数据表示/存贮格式

FORTRAN 语言用名称和类型规定了区分 FORTRAN 程序中使用的数据的明确方法。

1. 数据名称和类型

1) 名称

(1) 常数——明显表示的已知数

(2) 变量——用符号表示的已知数

(3) 数组——一维，二维或三维形式的数据序列组

(4) 数组元素——数组数据中的一个数

2) 类型

(1) 整型——严格表示整数（正，负或零），精度为 5 位有效数字，范围为 -32768 至 $+32767$（即 -2^{15} 至 $2^{15}-1$），包括 -32768 和 $+32767$ 两者。

(2) 实型——实数的近似值（正，负或零），在计算机中以4个字节的浮点形式存贮，实型数据的精度为7+个有效数字，数值大小大致限于10^{-38}至10^{38}之间（即2^{-127}—2^{127}）

(3) 双精度型——实数的近似值（正，负或零），在计算机中以8个字节的浮点形式存贮。双精度数据的精度为16+个有效数字，其数值范围与实型数据相同。

(4) 逻辑型——表示真值"TRUE"（真）或"FALSE"(假)的一个字节，"FALSE"的内部表示规定为0，"TRUE"为－1，但是在逻辑IF语句中任何非零值都作为"TRUE"处理。另外，逻辑型还可以作为单字节带符号整数使用，数值范围为－128至＋127，包括－128和＋127在内。

(5) Hollerith型——是计算机字符集中的一些字符组成的字符串。所有字符，包括空格都是有效的。Hollerith型数据需要一个字节来存贮字符串中的每个字符。

2. 常数

FORTRAN常数用表示它们的实际数值来识别。正的常数值前面不需要加正号（+）。书写常数的格式如表3—1所示。

表 3—1　　常数格式

类型	格式和使等规则	举例
整数	1．1至5位十进制数	－763
	2．前面的正号(＋)或负号(－)是选择项	1
	3．不允许有小数点(·)或逗号(，)	－32768
	4．数值范围：－32768到＋32767	＋32767
	（即-2^{15}到$2^{15}-1$）	
实数	1．是一个具有7位精度的十进制数，	345.
	它有下述两种表达形式：	－.345678
	a．　＋或－.f，＋或－i.f	＋345.678
	b．　＋或－i.E＋或－e	＋.3E3
	＋或－.fE＋或－e	－73E4
	＋或－i.fE＋或－e	
	其中i、f和e分别是一串代表整数、小数和指数的数字	
	2．正号（＋）和负号（－）是选择项	
	3．在上面1b形式中，如果r表示E＋或－e前面的任何项（即rE＋或－e），那么常数的数值为r乘10^e，其中，$-38<=e<=38$	
	4．如果E＋或－e前面常数的有效位多于实数的精度所允许的位数，那末就发生截尾，只有有效数字范围内的数能表示出来。	
双精度数	具有16位精度的十进制数，所有的格式和	＋345.678
	规则与实数的一样，只是用D代替了E。注意，	＋.3D3
	一个实数除非有"D"指数，否则都假设是单精度的。	－73D4

逻辑型	.TRUE. 产生一个非零字节（十六进制的FF）， .FALSE. 产生一个所有位都是0的字节。如果 逻辑值用作单字节整数，那末它的使用规则和整型相 同，只是允许范围为－128至＋127，包括－128和＋127。	.TRUE. .FALSE.
文字型	在文字型中，任何数目的字符可以用单引号括起来，如以下 形式：　　'X1X2X3…Xn' 其中，每一个Xi是除'以外的任何字符。连续使用两个引号 可以表示字符串中的引号字符，也就是，如果X2是一个引 号字符，则字符串以下列形式出现： 'X1''X3…Xn'	
十六进制数	1.字母Z或X后面跟一个单引号及多至4 个十六进制数（0—9和A—F），然后又 一个单引号就识别为一个十六进数值 2.十六进制常数在它的存贮值中是向右对齐的。	Z'12' X'AB1F' Z'FFFF' X'1F'

3. 变量

变量数据在FORTRAN语句中用符号名称来标识，变量名称是由1至6个字母数字符组成的唯一字符串，它的第一个字符必须是字母。

注：系统变量名称和子程序名称与其它变量名称的区别在于它们用美元符号（$）开头。因此，为了避免冲突，极力建议在FORTRAN源程序中的符号名称用"$"以外的其他字母开头。

举例：I5, TBAR, B23, ARRAY, XFM79, MAX, A1$C

变量数据分成四类：整型，实型，双精度型和逻辑型。类型的标识以下列方法之一来完成：

(1) 以符号名称的第一个字母隐含形式规定整数型或实数型。除非明确规定类型（见第六节），否则，凡以字母I, J, K, L, M或N开头的符号名称都代表整型变量，而以上述字母以外的字母开头的符号名称都是实变量。例如：

整变量　ITEM, J1, MODE, K123, N2

实变量　BETA, H2, ZAP, AMAT, XID

(2) 变量可以明确规定类型，即可以给它们一个具体的类型而不是引用它们的名称的第一个字母。变量可以明确规定整型、实型、双精度型或逻辑型等类型。在明确规定数据类型中使用的说明语句将在第六节讲述。

变量在程序执行过程中或在初始的DATA语句中（第六节）接受赋给它们的数值。

Hollerith或文字数据可以赋给任何类型的变量。有关Hollerith数据存贮将在本节第6小节讨论。

4. 数组和数组元素

数组是由维数特性表征的数据序列集合。数组可以有1维、2维或3维，并用符号名称来标识和规定类型，方法与变量的相同，只是数组名称必须用一个"数组说明符"来说明。有关数组说明符的详细讨论将在本手册的第六节中进行。数组说明指明了数组的维数，也指

明了数组的大小。一个数组元素是组成数组的数据集合之一，在 FORTRAN 语句中引用一个数组元素是在数组名称上加下标。术语数组元素与某些 FORTRAN 文本和参考手册中使用的下标变量术语是同义词。

可以用 DATA 语句给任何数组元素赋初值，或者在程序执行过程中确定和导出它的值。

5. 下标

在数组名称后跟一个下标唯一地标识了一个数组元素，在使用中，FORTRAN 语句中的下标与熟知的代数表示法的下标具有相同的意义。

下标的使用规则如下：

1) 下标在一个括号内可以含有 1 个，2 个或 3 个下标表达式（请看以下之 4）。

2) 如果在括号内有两个或三个下标表达式，那么必须用逗号将它们隔开。

3) 下标表达式的数目必须与数组说明符规定的维数相同（见第六节）。

4) 下标表达式用下列形式之一书写：

K　　C*V　　V-K

V　　C*V+K　　C*V-K

V+K

其中，C 和 K 是整常数，V 是整变量名称（关于表达式求值的讨论请看第四节）。

5) 下标本身不可以是下标。

例如：　X(2*J, 7)　　A(I,J,K)　　I(20)　　C(L-2)　　Y(I)

6. 数据存贮分配

对 FORTRAN 数据的存贮分配是以存贮单位的数目进行的。一个存贮单位是存贮一实型数据值所需要的内存空间（4 个字节）。

十六进制数据可以（通过 DATA 语句）与任何类型数据相联。它的内存分配和相联数据的一样。

Hollerith 或文字数据可以使用 DATA 初始化语句（见第六节）和任何数据类型相联。

最多 8 个 Hollerith 字符就可以与双精度型存贮相联，与实型相联最多 4 个 Hollerith 字符，与整型最多 2 个，而与逻辑型存贮相联只要 1 个 Hollerith 字符。

表 3—2．各数据类型的存贮分配

类　型	分　配
整　型	2 个字节（1/2 个存贮单位）： S | 二进制值 负数是正数表示的 2 的补码
逻辑型	1 个字节（1/4 个存贮单位） 零（假）或非零（真） 非零值字节表示真（逻辑常数.TRUE.用十六进制值 FF 表示）。 零值字节表示假。 当作为算术值使用时，逻辑型数据就作为－128 到＋127 范围内的整数处理。

实　型　　4个字节（1个存贮单位）：

阶数	S	尾数	尾数（续）

第一个字节是表示超过200（八进制）计数法的阶数；即，数值200（八进制）对应二进制指数0，小于200（八进制）的数值对应负的指数，而大于200的数值对应正的指数。按照定义，如果阶数为零，则全部数字为零。

后面三个字节由尾数组成。尾数总是规格化的，其最高位限止数据实际存贮于该位，此最高位用以代替表示数字的符号，最高位为1表示负数，为0表示正数。尾数被假设为二进制小数，其二进制小数点在尾数的左边。

双精度型　　8个字节（2个存贮单位）

双精度数据的内部形式与实数相同，只是双精度数又使用了额外4个字节的尾数。

四、FORTRAN表达式

FORTRAN表达式由单个运算量或一串用运算符连起来的运算量组成。FORTRAN提供两种类型的表达式：算术表达式和逻辑表达式。两种类型表达式使用的运算量、运算符和规则将在下面各小节讲述。

1. 算术表达式

下列规则规定了所有允许的算术表达式的形式：

1）单独存在的常数，变量名称、数组元素或FUNCTION引用都是一个表达式。

例如：　S(I)　　JOBNO　　217　　17.26　　SQRT(A+B)

2）如果E是一个表达式，其第一个字符不是运算符，那么+E和-E称为带符号表达式。

例如：　-S　　+JOBNO　　-217　　+17.26　　-SQRT(A+B)

3）如果E是一个表达式，那么(E)表示E求值时的数量结果。

例如：　(-A)　　-(JOBNO)　　-(X+1)　　(A-SQRT(A+B))

4）如果E是一个无符号表达式，F是任何表达式，那么：F+E，F-E，F*E，F/E，F**E是全部表达式。

例如：　-(B(I,J)+SQRT(A+B(K,L)))　　1.7E-2**(X+5.0)

　　-(B(I+3，3*J+5)+A)

5）一个计算表达式可以是整型、实型、双精度型或逻辑型，其类型取决于表达式元素的数据类型。如果表达式元素不是同一类型的，那么表达式的类型取决于元素具有的最高类型。类型的等级如下（由最高至最低）：

双精度型、实型、整型、逻辑型

6）表达式可以含有嵌套括号的元素，例如：

A* (Z-((Y+X)/T)) **J

其中，Y＋X 是最里层元素，(Y＋X)/T 是第二层，Z-- (Y＋X)/T 是最外层。在这种表式中，要特别留神使左括号数目与右括号数目相等。

2. 表达式计算

算术表达式的计算遵循下列规则：

1) 先计算括号内的表达式元素，如果带括号元素是嵌套的，那么先计算最里层元素，然后由里往外直到整个表达式求得值。

2) 在括号内或者任何不用括号给出计算次序的地方，运算的等级按下列优先级：

(1) 函数 (FUNCTION) 求值

(2) 指数运算

(3) 乘、除

(4) 加、减

例如：表达式 A* (Z－((Y＋R)/T))**J＋VAL

以下列次序进行计算：

Y＋R＝e1

(e1)/T＝e2

Z－e2＝e3

e3**J＝e4

A*e4＝e5

e5＋VAL＝e6

3) 表达式 X**Y**Z 是不允许的，必须写成：

(X**Y)**Z 或 X**(Y**Z)

4) 数组元素的使用需要对它的下标进行计算。下标表达式的计算遵循与其他表达式相同的规则。

3. 逻辑表达式

逻辑表达式可以是下列任何一个：

(1) 单个逻辑常数 (即.TRUE.或.FALSE.)，逻辑变量，逻辑数组元素或逻辑函数 (FUNCTION) 引用 (参阅第九节 FUNCTION)。

(2) 用一个关系运算符隔开的两个算术表达式 (即，关系表达式)。

(3) 对逻辑常数、逻辑变量、逻辑数组元素、逻辑 FUNCTION，关系表达式或其他逻辑表达式起作用的逻辑运算符。

逻辑表达式的值总是.TRUE.或.FALSE.

1) 关系表达式

关系表达式的一般形式如下：

e1 r e2

其中，e1 e2 和是算术表达式，r 是关系运算符。

有六个关系运算符：

.LT. 小于　　.NE. 不等于

.LE.	小于或等于	.GT.	大于
.EQ.	等于	.GE.	大于或等于

如果运算符规定的条件满足，关系表达式的值为 .TRUE.，否则为 .FALSE.

例如：　　A .EQ. B

　　(A**J) .GT. (ZAP*(RHO*TAU−ALPH))

2) 逻辑算符

表 4 — 1 列出了逻辑运算，U 和 V 表示逻辑表达式

表 4 — 1　　逻辑运算

.NOT. U	此表达式的值为 U 的逻辑补码（即，二进制位 1 变成 0，而 0 变成 1）。
U.AND.V	此表达式的值是 U 和 V 的逻辑乘（即，只有在 U 和 V 的相应位都是 1 的时候才得到结果 1）
U .OR. V	此表达式的值是 U 和 V 的逻辑和（即，如果 U 或 V 的相应位是 1，或者 U 或 V 的相应位都是 1，则结果为 1）
U .XOR. V	此表达式的值为 U 和 V 的异或（即，如果 U 和 V 的相应位分别是 1 和 0，或者是 0 和 1，则结果为 1）。

例如：

若　U=01101100, V=11001001, 则

　.NOT. U=10010011

　U .AND. V=01001000

　U .OR. V=11101101

　U .XOR. V=10100101

以下是逻辑表达式结构的附加注意事项：

(1) 任何逻辑表达式都可以放在括号内，但是，有 .NOT. 算符的逻辑表达式如果含有两个或两个以上的元素，那末必须用括号括起来。

(2) 在运算等级中，可以使用括号规定表达式计算的次序。在括号内以及在不用括号决定计算次序的地方，遵循的次序如下：

① 函数 (FUNCTION) 引用

② 指数运算 (**)

③ 乘和除 (*和/)

④ 加和减 (+和−)

⑤ .LT., .LE., .EQ., .NE., .GT., .GE

⑥ .NOT.

⑦ .AND.

⑧ .OR., .XOR.

例如：

　表达式　X .AND. Y. OR. B(3, 2) .GT. Z

计算次序为　　e1=B(3, 2) .GT. Z

　　e2=X .AND. Y

　　e3=e2 .OR. e1

表达式　X .AND. (Y .OR. B (3, 2).GT.Z)

计算次序为　e1=B(3, 2) .GT. Z

e2=Y .OR. e1

e3=X .AND. e2

(3) 两个连续的逻辑算符是无效的除非第二个算符是 .NOT.

即 .AND. .NOT. 和 .OR. .NOT. 是允许的。

例如：　A.AND. .NOT. B　是允许的

A.AND. .OR.B　是不允许的

4. 表达式中的 Hollerith、文字和十六进制常数

允许在表达式中用 Hollerith、文字和十六进制常数代替整型常数，这些特殊的常数总是求得一整数值并限于 2 个字节的长度。其唯一的例外是：

(1) 长的 Hollerith 或文字常数可以作为子程序参量使用。

(2) 当和实变量相联时，在 DATA 语句中，Hollerith、文字或十六进制常数可以长达 4 个字节，当和双精度变量相联时，可以长达 8 个字节。

五、置 换 语 句

置换语句规定了计算，并同样地在正常的数学表示法中用于方程式，它具有如下形式：

V=e

其中，V 是任何变量或数组元素，e 是表达式。

FORTRAN 的语义定义等号(=)为置换而不是正常的等于。因此，在执行由置换语句产生的目标程序时，将计算等号右边的表达式，并将结果放入分配给等号左边的变量或数组元素的存贮空间内。

下列条件适用于置换语句：

1) V 和等号必须出现在同一行上。甚至这适用于语句为逻辑 IF 语句的一部分的时候(见第七节)

例如：

```
C IN A REPLACEMENT STATEMENT THE'='
C     MUST BE IN THE INITIAL LINE.
      A(5, 3)=
     1 B(7, 2)+SIN(C)
```

包含 V=的行必须是语句的初始行，除非这语句是逻辑 IF 语句的一部分，在这种情况下，V=的出现个可晚于 IF 结果后的第一行末尾。

2) 如果变量 V 和表达式 e 的数据类型不同，那末，由表达式确定的值将被转换成符合变量的类型（如果可能的话）。表 5 — 1 显示了可以和变量类型相等的表达式类型，Y 表示有效置换，N 表示无效置换，Y 的脚注表示转换考虑。

表 5—1　按类型的置换

变量类型	表达式类型 (e)			
	整型	实型	逻辑型	双精度型
整型	Y	Ya	Yb	Ya
实型	Yc	Y	Yc	Ye
逻辑型	Yd	Ya	Y	Ya
双精度型	Yc	Y	Yc	Y

其中 a：实型表达式值转换成整型，为符合整型数据的范围就要截尾。

b：符号被扩展到整个第二字节。

c：将表达式的整数值的近似数值赋给变量。

d：将整型表达式的截尾值赋给变量（使用低阶字节，忽略符号）。

e：将实型表达式的舍入值赋给变量。

六、说明语句

说明语句是非执行、非产生语句，它定义变量和数组的数据类型，规定数组的维数和大小，给数据分配存贮器，或者反过来给 FORTRAN 处理器提供指定信息。DATA 初始语句是非执行语句，但是产生目标程序数据和建立变量数据的初始值。

1. 说明语句

有六种说明语句：

类型、EXTERNAL 和 DIMENSION 语句

COMMON 语句

EQUIVALENCE 语句

DATA 初始语句

所有说明语句要集中放在程序单元开头并必须按上面的出现次序排好。说明语句的前边只可以出现 FUNCTION, SUBROUTINE, POGRAM 或者 BLOCK DATA 语句，而说明语句又必须放在语句函数和第一条可执行语句的前面。

2. 数组说明符

有三种说明语句可以规定数组说明符，这三种句为：

类型语句、DIMENSION 语句、COMMON 语句

其中，DIMENSION 语句只具有数组说明的唯一作用，而其他两个语句适用双重目的。这些语句将在本节的 3，5，6 小节定义。

数组说明符用于指定数组的名称、维数和大小。在程序单元中，一个数组只能说明一次。

数组说明符的形式为下列形式之一：

ui(k), ui(k1, k2), ui(k1, k2, k3)

其中，ui 是数组名称，称作说明符名称，而k 是整型常数。

数组的存储分配是根据数组说明符的外貌确定的，这种存储器由 FORTRAN 处理器线性分配，其中序列的上升是这样确定的：第一个下标变化最快，而最末的下标变化最慢。

例如，有一个数组说明符 AMAT(3, 2, 2)，则按以下序列分配 12 个单元：

AMAT(1, 1, 1), AMAT(2, 1, 1), AMAT(3, 1, 1), AMAT(1, 2, 1),
AMAT(2, 2, 1), AMAT(3, 2, 1), AMAT(1, 1, 2), AMAT(2, 1, 2),
AMAT(3, 1, 2), AMAT(1, 2, 2), AMAT(2, 2, 2), AMAT(3, 2, 2)

3. 类型语句

变量，数组和函数名称按“预先约定”的习惯将自动地分类成整型或实型，只有用类型语句才能改变。例如，如果一个项目的第一个字母是 I, J, K、L, M, N, 则类型为整型，否则其类型是实型。

类型语句可以用指定一个项目的类型的方法取代或确认按预先约定习惯规定的类型。另外，这些语句可以用于说明数组。

类型语句具有以下一般形式：

t V1, V2, ···, Vn

其中，t 代表 INTEGER, INTEGER＊1, INTEGER＊2, REAL, REAL＊4, REAL＊8, DOUBLE PRECISION, LOGICAL, LOGICAL＊1, LOGICAL＊2 或 BYTE 等项之一，每个 V 是数组说明符或变量、数组或函数名称。INTEGER＊1, INTEGER＊2, REAL＊4, REAL＊8, LOGICA＊1 和 LOGICAL＊2 等类型考虑到用其他 FORTRAN 的可读性和兼容性。BYTE, INTEGER＊1, LOGICAL＊1 和 LOGICAL 都是等价的；INTEGER＊2 ,LOGICAL＊2 是和 INTEGER 等价的；REAL 和 REAL＊4 等价，DOUBLE PRECISION 和 REAL＊8 等价。

例如：

REAL AMAT(3, 3, 5), BX, IETA, KLPH

注：(1) 重复地说明了 AMAT 和 BX 的类型。

(2) 无条件地说明了 IETA 和 KLPH 为实型。

(3) AMAT(3, 3, 5) 是一个数组说明符，指定了 45 个元素的数组。

例如：

INTEGER M1, HT, JMP(15), FL

注：这里对 M1 类型进行了重复说明，并取代了 HT 和 EL 按预先约定习惯规定的类型，由于它们出现在 INTEGER 语句中。JMP(15) 是数组说明符，它重复地说明了数组元素的类型为整型，并将数组的维数和需要的存贮器通知处理器。

例如：

LOGICAL L1, TEMP

注：所有需要规定为逻辑型的变量、数组、或函数都必须出现 在 LOGICAL 语句 中，因为用第一个字母表明类型的隐含规则不成立。

4. 外部（EXTERNAL）语句

EXTERNAL 语句具有如下形式：

EXTERNAL u1, u2, …, un

其中，每个 ui 是子例行（SUBROUTINE），数据块（BLOCK DATA）或者函数（FUNCTION）的名称，当子程序名称在子程序引用中作宗量时，它必须出现在 EXTFRNAL 语句的前边。

当数据块子程序包括在程序装入中时，它的名称必须出现在主程序单元中的 EXTERNAL 语句里。

例如，假设 SUM 和 AFUNC 是子程序名，它们在子程序 SUBR 中作宗量使用，则下列语句将出现在调用程序单元内：

```
      ⋮
EXTERNAL SUN, AFUNC
      ⋮
CALL SUBR(SUM, AFUNC, X, Y)
```

5. 维数（DIMENSION）语句

DIMENSION 语句具有以下形式：

DIMENSIONu1, u2, u3, …, un

其中，每一个 ui 是数组说明符。

例如：

```
DIMENSION RAT(5,5), BAR(20)
```

这语句说明了两个数组——25 个元素的数组 RAT 和 20 个元素的数组 BAR。

6. 公用（COMMON）语句

COMMON 语句是非执行、存贮器分配的语句，它给变量和数组分配了一个叫做 COMMON 存贮器的存储区域，为各程序单元共同使用同一个存储区域提供了方便。

COMMON 语句表示成以下形式：

COMMON /Y1/A1/Y2/A2/…/Yn/An

其中，每个 Yi 是公用块存贮器名称，每个 Ai 是变量名、数组名或数组说明符的序列，序列内用逗序分隔。Ai 中的元素组成了用名称 Yi 说明的公用块存储区。如果省略任何的 Y 只留下两个相邻的斜道符号（//）那末这样的存贮块叫做无名公用区。如果第一个公用块名称（Y1）省略掉了，那末两条斜杠也可以省略。

例如：

```
      COMMON /AREA/A,B,C/BDATA/X,Y,Z,
     X       FL, ZAP(30)
```

在本例中，分配了两个公用存贮块：AREA，其空间给三个变量；和 BDATA，其空间给四个变量和一个 30 个元素的数组 ZAP。

例如：

```
      COMMON //A1,B1/CDATA/ZOT(3,3)
```

X //T_2,Z_3

在这个例中，A1，B1，T_2 和 Z_3 按此顺序赋给了无名公用区。A1 前边的一对斜杠是可以省略的。名叫 CDATA 的公用存贮块是给 9 个元素的数组 ZOT 的，因此，ZOT(3,3) 是一个数组说明符，ZOT 不必预先说明（见第 2 小节“数组说明符”）。

补充注意事项：

(1) 公用块名称可以在同一个 COMMON 语句中，或在几个 COMMON 语句中多次出现。

(2) 公用块名称由 1 至 6 个字母数字符号组成，它的第一个字符必须是字母。

(3) 公用块名称必须和整个程序中使用的任何子程序名称不同。

(4) 公用区的大小可以用 EQUIVALENCE 语句增加，请看第 7 小节“EQUIVALENCE 语句”

(5) 相同名称的公用块长度在所有出现该名称的程序单元内不必相等。但是，如果长度不同，那末规定最大长度的程序单元必须首先装入（请参阅用户指南中有关 LINK-80 的讨论）。公用区的长度是包含 COMMON 语句中说明的变量和数组所需要的存贮单元的数目，除非使用等价（EQUIVALENCE）语句扩充。

7. 等价（EQUIVALENCE）语句

EQUIVALENCE 语句的使用允许两个或多个实体共用相同存储单位。语句的一般形式如下：

EQUIVALENCE (u1), (u2), …, (un)

其中，每个 ui 代表两个以上用逗号隔开的变量或数组元素的序列。序列中的每个元素都由处理器赋给了相同的存储单位（或部分存储单位）。序列中各元素出现的次序是没有意义的。

例如：

EQUIVALENCE (A, B, C)

在目标程序过程中，变量 A，B 和 C 将共用同一个存储单位。

如果在 EQUIVALENCE 语句中使用数组元素，那末，下标的数目必须和数组说明符建立的维数的数目相同，或者必须是 1 个，其中 1 个下标规定了与数组的第一元素有关的数组元素的数目。

例如：

假如数组 Z 的维数已说明为 Z(3,3)，那末在 EQUIVALENCE 语句中 Z(6)和 Z(3,2) 有相同的意义。

补充注意事项：

(1) 数组元素的下标必须是整型常数。

(2) 多维数组的元素在需要的时候可以用单个下标引用。

(3) 变量可以通过 EQUIVALENCE 语句赋给公用块。

例如：

COMMON /X/A,B,C

```
EQUIVALENCE (A, D)
```

在此情况中，变量 A 和 D 共用公用块 X 中的第一个存储单位。

(4) EQUIVALENCE 语句能够增加用 COMMON 语句指出的公用块的大小，它在公用块的末尾添加更多的元素。

例如：

```
DIMENSION R(2,2)
COMMON /Z/W,X,Y
EQUIVALENCE (Y,R(3))
```

得到的公用块将具有以下形状：

变　量	存储单位
W=R(1,1)	0
X=R(2,1)	1
Y=R(1,2)	2
R(2,2)	3

由 COMMON 语句建立的公用块含有 3 个存储单位，通过 EQUIVALENCE 语句它已扩充到 4 个存储单位。

公用块的大小只能从 COMMON 语句建立的最后一个元素向前增加；而不能从它的第一个元素向后增加。

注意，在此例子中，EQUIVALENCE (X, R(3))是不成立的。COMMON 语句确定 W 为公用块中的第一个元素，而要使 X 和 R(3) 等价就是要使 R(1) 为第一个元素。

(5)要使同一数组的两个元素或属于同一个或不同的公用块的两个元素等价是不成立的。

例如：

```
      DIMENSION XTABLE(20), D(5)
      COMMON A, B(4) /ZAP/C, X
       ⋮
      EQUIVALENCE (XTABLE(16), A(7),
     X            B(3), XTABLE(15)),
     Y            (B(3), D(5))
       ⋮
```

这条 EQUIVLENCE 语句有下列错误：

① 它企图使同一数组的两个元素 XTABLE(16) 和 XTABLE(15) 等价。

② 它企图使同一公用块的两个元素 A(7) 和 B(3) 等价。

③ 由于 A 不是数组，因此 A(7) 是非法引用。

④ B(3) 与 D(5) 等价使公用块从它定义的起始点向后扩充。

8. 数据（DATA）初始语句

DATA 初始语句是非执行语句，它提供了将数据值编译成目标程序的方法，并将这些数据赋给变量和数组元素，供其他语句引用。

DATA 语句的形式如下：

DATA list/u1, u2, …, un/, list…/uk, uk+1, …, uk+n/

其中，"list"代表变量、数组或数组元素名称的清单，ui 是数目与清单中元素相应的常 数。这种清单项目与常数一一对应的例外情况是数组名称（无下标）可以出现在清单中，要充填数组同样需要许多常数出现在斜杠之间的相应位置上，为了说明相同 的 常 数 ui 连续重复 k 次，允许写成 k*ui 来代替 ui，k 必须是正整数。哑宗量不能出现在清单中。

例如：

```
      DIMENSION C(7)
      DATA A, B, C(1), C(3)/14.73,
     X              -8.1, 2*7.5/
```

这意味着 A=14.73, B=-8.1, C(1)=7.5,C(3)=7.5

每个常数 ui 的类型必须和清单中相应的项目的类型相符，只有 Hollerith 或文字常数可以和任何类型的项相配对。

在使用 Hollerith 或文字常数时，那末，其字符串中字符的数目不能大于相应项目需 要的存储单位数目的 4 倍，即，1 个字符对应一个逻辑变量，2 个字符对应一个整型变量，4 个或 4 个以下字符对应一个实型变量。

如果只指定少数 Hollerith 或文字符号，那末在后尾加空格来充填剩余的存贮器。

十六进制数据也以相似的方式存储，如果只使用少数十六进制符号，那末在前面加足够的零来充填剩余的存贮单位。

下面的几个例子说明了 DATA 语句的许多特性：

```
      DIMENSION HARY(2)
      DATA HARY, B/4HTHIS, 4H OK., 7.86/
      REAL LIT(2)
      LOGICAL LT, LF
      DIMENSION H4(2,2), PI3(3)
      DATA A1, B1, K1, LT, LF, H4(1, 1), H4(2, 1),
     1      H4(1, 2), H4(2, 2), PI3/5.9, 2.5E-4,
     2      64, .FALSE., .TRUE., 1.75E-3,
     3      0.85E-1, 2*75.0, 1., 2., 3.14159/
     4      LIT(1)/'NOGO'/
```

七. FORTRAN 控制语句

FORTRAN 控制语句是可执行语句，它影响和引导 FORTRAN 程序的逻辑流 程。本节讲述的这些语句是：

1. 转移（GO TO）语句:
   1) 无条件 GO TO
   2) 计算 GO TO
   3) 赋值 GO TO
2. 标号赋值（ASSIGN）语句

3. 条件（IF）语句：

1) 算术 IF

2) 逻辑 IF

4. 循环（DO）语句

5. 继续（CONTINUE）语句

6. 停（STOP）语句

7. 暂停（PAUSE）语句

8. 调用（CALL）语句

9. 返回（RETURN）语句

当其他语句的语句标号是控制语句的一部分时，这种语句标号必须是控制语句出现的同一程序单元中的和可执行语句有关的语句标号。

1. 转移（GO TO）语句

1) 无条件 GO TO

无条件 GO TO 语句用于使控制无条件地转移到程序单元中某个其它语句的任何时候。此语句为以下形式：

```
    GO TO k
```

其中，k 是同一程序单元中一个可执行语句的语句标号。

例如：

```
        GO TO 376
310 A(7)=V1-A(3)
        ⋮
376 A(2)=VECT
        GO TO 310
```

在这些语句中，语句 376 在程序的逻辑流程中超前语句 310，它们都是程序的一部分。

2) 计算 GO TO

计算 GO TO 语句的形式是：

```
    GO TO(K1, K2, ···, Kn), j
```

其中，Ki 是语句标号，j 是一个整变量，$1 \leqslant j \leqslant n$。

这个语句使控制转到语句标号 Kj，如果 $j<1$ 或 $j>n$，那么控制就转到计算 GO TO 语句的下面一条语句。

例如：

```
      J=3
      ⋮
      GO TO (7, 70, 700, 7000, 70000), J
310   J=5
      GO TO 325
```

当 J=3 的时候，计算 GO TO 将控制转移到语句 700，改变 J 使它等于 5 就改变转移到语句 70000，令 J=0 或 J=6 将使控制转移到语句 310

3） 赋值 GO TO

赋值 GO TO 语句具有以下形式：

GO TO J, (K1,K2,…, Kn)

或

GO TO J

其中，J 是一整变量名称，Ki 是可执行语句的语句标号。这语句使控制转移到语句标号等于当前 J 值的语句。

限制条件：

(1) ASSIGN 语句必须逻辑超前赋值 GO TO。

(2) ASSIGN 语句必须赋值给 J，如果指定了标号清单，那末 J 必须是标号K 清单中包含的语句标号。

例如：

GO TO LABEL, (80, 90, 100)

只有语句标号 80，90，或 100 可以赋给 LABEL。

2. 标号赋值（ASSIGN）语句

这个语句为以下形式：

ASSIGN j TO i

其中，j 是可执行语句的语句标号，i 是整变量。

这语句用于连接含有整变量i的每个赋值 GO TO 语句，当执行赋值 GO TO 语句时，控制就转移到语句标号 j。

例如：

```
ASSIGN 100 TO LABEL
⋮
ASSIGN 90 TO LABEL
GO TO LABEL, (80, 90, 100)
```

3. 条件（IF）语句

IF 语句依据条件将控制转移到一系列语句之一。有两种 IF 语句：算术 IF 和逻辑 IF

1） 算术 IF

算术 IF 语句的形式是：

IF (e)m1, m2, m3

其中，e 是算术表达式；m1，m2 和 m3是语句标号。

表达式 e 的计算决定了三种转移可能性之一：

如果 e 为：	转移到：
<0	m1
=0	m2
>0	m3

例如：

语　句	表达式值	转移到
IF (A)3, 4, 5	15	5
IF (N-1)50,73,9	0	73
IF (AMTX(2,1,2))7,2,1	−256	7

2) 逻辑 IF

逻辑 IF 语句的形式是：

IF (u)S

其中，u 是逻辑表达式；S 是 DO 语句或其他逻辑 IF 语句以外的任何可执行语句。逻辑表达式 u 求得的值为 .TRUF. 或 .FALSE.，*第四节曾对逻辑表达式进行过讨论。

控制条件：

如果 u 是假，则忽略语句 S，控制就转到逻辑 IF 语句下面的一条语句；但是，如果表达式是真，那末控制转到语句 S，然后程序控制按正常条件进行。

如果 S 是置换语句（V=e，请看第五节），那末变量和等号（=）必须在同一行上，并且都要紧跟在 IF(u) 后边，或者在 IF(u) 下边的单独的继续行上，继续行左边为空白的行空间，见下边例 4 和例 5 。

例如：

```
(1) IF(I. GT. 20) GO TO 115
(2) IF(Q. AND. R) ASSIGN 10 TO J
(3) IF(Z) CALL DECL(A,B,C)
(4) IF(A. OR. B. LE. PI/2) I=J
(5) lF(A. OR. B. LE. PI/2)
   X  I=J
```

4. 循环（DO）语句

DO 语句在 FORTRAN 中提供了一个反复执行一段语句序列的方法。这语句取下列两种形式之一：

1) DO k i=m1, m2, m3

或 2) DO k i=m1, m2

其中，k 是一语句标号，i 是整变量或逻辑变量，m1，m2 和 m3 是整常数或整变量或逻辑变量。如果 m3=1，则可以省略，如上面之 2)。

在使用 DO 语句中有下列条件和限制：

(1) DO 和第一个逗号必须出现在初始行。

(2) 语句标号 k，称作终端语句，必须是可执行语句。

(3) 终端语句必须实际跟在与它有关的 DO 之后。可执行语句在 DO 的后面，在终端语句的前面，并且包括终端语句组成了 DO 语句的区域。

(4) 终端语句不可以是算术 IF，GO TO，RETURN,STOP，PAUSE 或其他 DO 语句。

(5) 如果终端语句是逻辑 IF 语句，并且它的表达式为 .FALSE.，那末 DO 区域内的语句就被反复执行。

如果表达式是 .TRUE.，那末执行此逻辑 IF 语句，然后反复执行 DO 区域内的语句。此逻辑 IF 语句不能是 GO TO，算术 IF，RETURN，STOP 或者 PAUSE 等语句。

(6) 控制整变量 i 称作 DO 区域的索引，它必须是正数并且不能被区域内的任何语句所改变。

(7) 如果 m1，m2 和 m3 是 INTEGER*1 型变量或常数，那末 DO 循环就更短也执行得更快，但是，区域被限于 127 次叠代。例如，依据索引变量的类型，一个有常数限制和增量为 1 的 DO 循环的总开销为：

索引变量类型	总开销	
	微秒	字节
INTEGER*2	35.5	19
INTEGER*1	24	14

(8) 在第一次执行 DO 区域中的语句的过程中，i 等于m1；第二次执行，i=m1+m3；第三次，i=m1+2*m3……等等，一直到 i 等于这个序列中小于等于 m2 的最大值，于是就称 DO 已经满足。甚至在 m1≤m2 时，也总是最后执行一次 DO 区域中的语句。

在 DO 已经满足以后，控制就转到终端语句下面的语句，否则控制转回到 DO 语句下面的第一条可执行语。

例如：计算 $\sum_{i=1}^{100} Ai$，其中，A 是一维数组。

```
100       DIMENSION A(100)
   ⋮
          SUM=A(1)
          DO 31 I=2, 100
31        SUM=SUM+A(I)
          END
```

(9) DO 语句的区域可以扩充到包括所有逻辑上能在 DO 和终端语句之间执行的语句，因此，DO 区域的一部分可以位于那些实际上不在 DO 语句和终端语句之间而逻辑上在 DO 区域执行的地方，这称为扩充区域。

例如：

```
          DIMENSION A(500), B(500)
   ⋮
          DO 50 I=10,327,3
   ⋮
          IF(V7-C*C)20,15,31
30
   ⋮
50        A(I)=B(I)+C
   ⋮
20        C=C-.05
          GO TO 50
31        C=C+.0125
          GO TO 30
```

(10) 把控制转移到区域本身或同一 DO 语句的扩充区域以外的 DO 语句区域是不成立

的。

(11) 在 DO 语句区域内可以存在其他 DO 语句，在这种情况下，循环必须是嵌套的，即，如果一个循环区域含有其他的循环，那末内循环必须整个都包含在外循环区域内。

内循环的终端语句也可以是外循环的终端语句。

例如，有一个 15 行 15 列的二维数组 A，和一个 15 个元素的一维数组 B，下列语句计算数组 C 的 15 个元素，公式如下：

$$C_k=\sum_{j=1}^{15} A_{kj}B_{jn} \quad , \qquad k=1,\ 2,\cdots\cdots,15$$

```
      DIMENSION A(15, 15), B(15), C(15)
      ⋮
      DO 80 K=1, 15
      C(K)=0.0
      DO 80 J=1, 15
   80 C(K)=C(K) +A(K, J) *B(J)
      ⋮
```

5. 继续（CONTINUE）语句

CONTINUE 语句也归可执行语句一类，但是它的执行什么事都不做。CONTINUE语句的形式如下：

```
CONTINUE
```

当用那些不允许的终端语句或仅仅是条件执行语句作为正常的终端语句时,CONTINUE常常用作循环语句区域中的终端语句。

例如：

```
      DO 5 K=1, 10
      ⋮
      IF(C2)5, 6, 6
   6  CONTINUE
      ⋮
      C2=C2+.005
   5  CONTINUE
```

6. 停（STOP）语句

STOP 语句为以下形式之一：

```
      STOP
或    STOP c
```

其中，c 是一到六个字符的任何字符串。

当在目标程序执行过程中遇到 STOP 时，字符串 c （如果有的话）就显示在操作员的控制台上，并且结束程序的执行。

因此，STOP 语句构成了程序的逻辑结尾。

7. 暂停（PAUSE）语句

PAUSE 语句为以下形式之一：

```
PAUSE
或      PAUSE C
```

其中，c 是最多可达六个字符的任何字符串。

当目标程序执行过程中遇到 PAUSE 时，字符串 c（如果有的话）就显示在操作员控制台上，并且中止程序的执行。

程序继续执行的判定不在程序的控制之下。如果通过操作员的干与（对处理器的状态不做其它改变）恢复执行，那末按照 PASUE 下面的正常执行次序继续执行。

在操作员控制台上打入“T”就可以结束执行，打入其他字符将恢复执行。

8. 调用（CALL）语句

CALL 语句使控制转移到 SUBROUTINE 子程序中，并提供了子程序使用的参量。CALL 语句的一般形式和详细讨论将在第九节 FUNCTIONS 和 SUBROUTINES 中出现。

9. 返回（RETURN）语句

RETURN 语句的形式、使用和解释将在第九节中讲述。

10. 结束（END）语句

END 语句必须是任何 FORTRAN 程序的实际上的最后一条语句。它具有以下形式：

```
END
```

END 语句是可执行语句并可以有语句标号，它将控制转移去执行系统出口例行程序 $EX，使控制返回到操作系统。

八、输　入/输　出

FORTRAN 有一系列语句用以规定计算机内存与外部数据处理设备或大容量存储设备，如磁带、磁盘、行式打印机、穿孔卡片处理器、键盘打印机等之间数据传输的条件和控制。

这些语句分组如下：

(1) 格式读（READ）和写（WRITE）语句，它在计算机和 I/O 设备之间传输格式信息。

(2) 无格式 READ 和 WRITE 语句，它以与内部存储格式相同的格式传输无格式二进制数据。

(3) 辅助 I/O 语句，用于文件的定位和分界。

(4) 在内存单元之间传输数据的编码（ENCODE）语句和译码（DECODE）语句。

(5) 格式（FORMAT）语句，连同格式数据传输一起使用，在内部数据表示和外部字符串形式之间提供了数据转换和编辑信息。

1. 格式READ/WRITE语句

1) 格式读（READ）语句

格式READ语句用于从输入设备向计算机传送信息。

READ语句有两种可用的形式：

READ (u, f, ERR=L1, END=L2) k

或 READ (u, f, ERR=L1, END=L2)

其中，

u——规定一个实际的和逻辑的设备号码，它既可以是无符号的整数也可以是一个整型变量，其范围是1至255。如果使用整变量，则必须在READ语句执行之前赋给它整数值。

设备号1，2，3，4，5已预先规定为控制台电传打印机，2为行式打印机（如果有的话），6—10规定为磁盘文件（请见附录5），这些设备以及11—255可以由用户重新规定（见附录2）。

f——是一个FORMAT语句的语句标号，这个语句描述了输入传输中使用的数据转换的类型。f也可以是一个数组名称，在这种情况下，格式信息可以在执行时输入给程序。

L1——是一个FORTRAN语句标号，如果遇到I/O错误，I/O处理器就将控制转移到此语句。

L2——是一个FORTRAN语句标号，如果遇到文件结果标志，I/O就将制控转移到此语句。

k——是规定输入数据的变量名清单，用逗号分隔。

READ (u, f) k用来输入若干项目，它按照清单k中的名称从逻辑设备u的文件上，使用FORMAT语句f规定的这些项目的外部表示法输入这些项目（FORMAT语句请见第7小节）。ERR=和END=两项是选择项，如果不指定的话，I/O错误和文件结束会引起致命性运行错误。

下列注意事项进一步规定了READ (u, f) k语句的功能：

(1) 每次执行READ语句时，就从输入文件上读入新的记录。

(2) 由一条READ语句输入的记录数目取决于清单k和格式规定。

(3) 清单k规定了由输入文件读取的项目数和存储它们的内存单元。

(4) 在一条清单中可以出现任何数目的项目，项目的数据类型可以不同。

(5) 如果输入记录中的量多于清单中的项目，那末，只有数目和清单中项目数相等的量被传输，剩余的量都被忽略。

(6) 关于清单k的精确规定在第6小节中讲述。

例如：

(1) 假如有4个数据被穿成了卡片，每个之间有3个空格隔开，数据由卡片的第1列开始，其区段宽度分别为3，4，2，5个字符。语句

```
      READ (5, 20)K, L, M, N
20    FORMAT (I3, 3X, I4, 3X, I2, 3X, I5)
```

将读取卡片（假设逻辑设备5号已规定为卡片阅读机），并将输入数据赋给变量K，L，M和N。FORMAT语句也可以是

```
20  FORMAT (I3, I7, I5, I8)
```

有关FORMAT语句的完整描述请见第7小节

(2) 输入数组（ARRY）的量：

```
READ (6, 21)ARRY
```

在清单中只需出现数组名称（见第6小节），那末数组ARRY的全部元素就用标号为21的FORMAT语句规定的相应格式读入和存储。

READ (u, k) 和FORMAT语句一起可以用于将H型字母数字数据读入现有的H型区段（请见第7小节之3，Hollerith转换）。

例如，语句

```
READ (I, 25)
⋮
25      FORMAT(10HABCDEFGHIJ)
```

使得输入设备I上文件的下10个字符被读入并代替FORMAT语句中的字符ABCDEFGHIJ。

2) 格式写（WRITE）语句

格式WRITE语句用于将信息从计算机传送到输出设备。

有两种可用的语句形式：

```
WRITE (u, f, ERR=L1, END=L2) k
```

或 `WRITE (u, f, ERR=L1, END=L2)`

其中，

u—规定逻辑设备号，

f—是一个FORMAT语句的语句标号，此格式语句说明了输出传输使用的数据转换类型，

L1—规定一个I/O错误分支，

L2—规定文件结束（EOF）的分支，

k—规定输出数据的变量名单，用逗号隔开。

WRITE (u, f) k用于将清单k中规定的数据输出给逻辑设备u上的文件，数据的外部表示是用FORMAT语句f规定的，（见第7小节FORMAT语句）。

下列注意事项进一步规定了WRITE语句的功能：

(1) 可以用一条WRITE语句输出几个记录，其数目取决于清单和FORMAT的规定。

(2) 数据是连续地输出的，直到清单中规定的数据用完为止。

(3) 如果数据输出给规定了固定记录长度的设备，而清单中规定的数据填不满记录，那末剩余的记录是填上空格。

例如：

```
WRITE (2, 10)A, B, C, D
```

将赋给变量A，B，C，D的数据按标号为10的FORMAT语句的格式输出给2号逻辑设备。

当FORMAT语句中规定了要输出的字符时，WRITE (u, f) 可以用于输出字母数字

的信息，在这种情况下，变量清单是不需要的。

例如，要在设备1上输出字符“H CONVERSION”

```
      WRITE (1, 26)
      ⋮
26  FORMAT (12HH CONVERSION)
```

2. 无格式READ/WRITE

无格式的输入/输出（I/O）（即，没有数据转换）用以下语句达到：

```
READ (u, ERR=L1, END=L2) k
WRITE (u, ERR=L1, END=L2) k
```

其中：

u—规定逻辑设备号，

L1—规定I/O错误分支，

L2—规定EOF分支，

k—是规定I/O数据的、用逗号分隔的变量名清单。

下列注意事项进一步确定了无格式I/O语句的功能：

(1) 无格式READ/WRITE语句执行无须转换或编辑的数据之内存映象传输。

(2) 传输的数据与清单k中的变量数目符合。

(3) 在无格式READ中，变量名清单的总长度必须不长于记录长度。如果逻辑记录长度和清单长度相同，那末读入了整个记录，如果清单长度比逻辑记录长度短，则跳过记录中的非阅读项。

(4) WRITE (a) k语句输出一个逻辑记录。

(5) 逻辑记录可以跨越一个以上的物理记录来延长。

3. 磁盘文件I/O

在READ或WRITF磁盘文件（LUN 6—10）时，自动打开(OPEN)此文件以输入/输出(I/O)，在用命令ENDFILE（见下一小节）关闭文件或程序正常结束以前，文件是一直开着的。

注：对磁盘文件进行顺序输出时要当心，如果对现有的文件进行输出，那末现有的文件就被删除并被相同名称的新文件代替。

1) 随机磁盘I/O

请参阅FORTRAN使用手册第三节。

某些版型的FORTRAN-80也有随机磁盘I/O，对于随机磁盘存取，在READ或WRITE语句中使用选择项REC=n规定记录号。例如：

```
I=10
WRITE (6, 20, REC=I, ERR=50) X, Y, Z
⋮
```

这个程序段在LUN6上书写记录10。如果上述记录10存在，那末重复地写它；如果记录10不存在，那末文件延长，增加了一个。要想读一个不存在的记录将产生I/O错误。

在随机存取文件中，记录长度随不同的FORTRAN版型而变化，请见FORTRAN使用手册第三节。用户希望随机读取的任何文件应是由FORTRAN（或BASIC）随机存取语句建立的。这种方法（使用二进制或格式WRITE语句）建立的文件，在数据充填不满记录时，将用0充填各记录以达到适当长度。

用READ或WRITE语句打开的任何磁盘文件都赋给了一个空缺文件名，它是给操作系统规定的。也请参阅FORTRAN使用手册。

2) OPEN子程序

另一方面，文件可以用OPEN子程序打开。LUN 1 — 5也可以用OPEN赋给磁盘文件。OPEN子程序允许程序规定文件名和与LUN有关的设备。

非存在文件的OPEN将以适当的名称建立一个空文件。跟在顺序输出后面的现有文件的OPEN将删除现有文件。跟在输入后面的现有文件的OPEN允许存取文件的当前内容。

OPEN调用的形式根据不同的操作系统而变化。请见FORTRAN使用手册第三节。

4. 辅助的I/O语句

有三条辅助的I/O语句：

```
BACKSPACE u
REWIND u
ENDFILE u
```

所有这三条语句的作用取决于它们使用的LUN（见附录2）。在LUN是终端或行式打印机时，这三条语句定义为无操作。

在LUN是磁盘驱动器时，ENDFILE和REWIND命令允许程序进一步控制磁盘文件。ENDFILE u关闭与LUN u有关的文件，REWIND u关闭与LUN u有关的文件，然后又打开它。这时候，不执行BACKSPACE，因此如果使用它的话就产生错误。

5. 编码/译码（ENCODE/DECOE）

ENCODE语句和DECODE语句按格式规定将数据从内存的一个区段传送到另一个区段。DECODE将数据由ASCII格式变成规定格式，ENCODE将数据由规定格式变成ASCII格式。这两条语句的形式是：

```
ENCODE (A, F) K
DECODE (A, F) K
```

其中：A是数组名称，

F是FORMAT语句号，

K是I/O清单，

DECODE语句与READ语句相似，因为它引起ASCII到内部格式的转换。ENCODE是与WRITE语句相似的，它引起内部格式到ASCII的转换。

注：应该注意使数组A总是大到足够可以包括全部的要处理的数据。不存在检查溢出的方法。溢出的ENCODE操作中，数组将可能擦去其后面的重要数据，溢出DECODE操作将企图处理数组后面的数据。

6. 输入/输出清单说明

大多数 READ/WRITE 语句的形式可以含有一个数据名称的序列清单，它标识着要传输的数据。清单项目的序列必须和对应数据（输入）序列相同，或者存在（输出）于外部 I/O 介质中。

清单具有下述格式：

m1，m2，…，mn

其中，mi 是用逗号隔开的清单项目。

1） 清单项目类型

清单项目可以是单一的数据标识符或者是复合的数据标识符。

（1）单一数据标识符项目是变量名或数组元素。可以用括号将这些项目的一个或多个括起来，这不改变它们的含义。

例如：

```
A
C(26,1), R,K, D, (I,J)
B, I(10, 10), S, (R,K), F(1,25)
```

注：(I,J) 项定义为清单中的两个项目，而 (26,1) 是下标。

（2）复合的数据标识符项目有两种形式：

① 在清单中不带下标的数组名称，它等效于数组的每个连续元素的清单。

例如：如果 B 是一个二维数组，那末清单项目 B 等效于 B(1,1)，B(2,1)，B(3,1)，…，B(1,2)，B(2,2)，…，B(j,k)，其中，j 和 k 是 B 的下标极限。

② 隐循环项目，是一个或多个单一数据标识符，或其他的隐循环项目，它们用逗号隔开，并有表达式：

i=m1，m2，m3 或 i=m1，m2

隐循环项目要用括号括起来。

元素 i，m1，m2，m3 的含义和 DO 语句所定义的相同。隐循环适用于括号内有隐含式的所有清单项目。

例如：

隐循环清单	等效清单
(X(I), I=1,4)	X(1), X(2), X(3), X(4)
(Q(J), R(J), J=1,2)	Q(1), R(1), Q(2), R(2)
(G(K), K=1,7,3)	G(1), G(4), G(7)
((A(I,J), I=3,5), J=1,9,4)	A(3,1), A(4,1), A(5,1), A(3,5), A(4,5), A(5,5), A(3,9), A(4,9), A(5,9)
(R(M), M=1,2),I,ZAP(3)	R(1), R(2), I, ZAP(3)
(R(3), T(I), I=1,3)	R(3), T(1), R(3), T(2), R(3), T(3)

因此，矩阵的元素可用和它们在存储器中的顺序不同的顺序传输。例如，数组 A(3,3)，

以 A(1,1)，A(2,1)，A(3,1)，A(1,2)，A(2,2)，A(3,2)，A(1,3)，A(2,3)，A(3,3) 这样的顺序占有存储器，若用隐循环清单项目 ((A(I,J)，J=1,3)，I=1,3) 规定此数组的传输，则传输的顺序为：A(1,1)，A(1,2)，A(1,3)，A(2,1)，A(2,2)，A(2,3)，A(3,1)，A(3,2)，A(3,3)

2. 清单说明中的特殊注意事项

(1) 清单的顺序是从左至右，当有控制隐循环的索引参量时，重复项目要括在括号内(不同于下标)。

(2) 数组是用输入/输出清单中出现的数组名称（无下标）传输的。

(3) 常数只能作下标或索引参量出现于输入/输出清单中。

(4) 对于输入清单，隐循环元素 i，m1，m2 和 m3 不能作为清单项目出现于括号内。

例如：

(1) READ(1,20) (I,J,A(I)，I=1，J，2)　　是不允许的

(2) READ(1,20)I，J，(A(I)，I=1,J,2,)　　是允许的

(3) WRITE(1,20)(I,J,A(I)，I=1,J,2)　　是允许的

考虑以下例子：

```
DIMENSION  A(25)
A(1)=2.1
A(3)=2.2
A(5)=2.3
J=5
WRITE (1,20)J，(I,A(I)，I=1，J，2)
⋮
```

这个 WRITE 语句的输出是

5，1，2.1，3，2.2，5，2.3

(1) 在一个清单中可以有任何数目的项目。

(2) 在格式传输 (READ (u，f)k，WRITE(u，f)k)中，每个项目的类型必须同 FORMAT 语句指定的相同。

7. 格式 (FORMAT) 语句

FORMAT 语句是非执行、生成语句，它要连同格式 READ 语句和 WRITE 语句一起引用，它们规定了计算机存贮器和外部介质表示之间传输数据的转换方法和数据编辑信息。

FORMAT 语句需有语句标号(f)，供 READ(u，f) k 或 WRITE(u，f) k 语句引用。

FORMAT 语句的一般形式如下：

n FORMAT(S_1，S_2，…，S_n/S_1'，S_2'，…，S_n'/…)

其中，n 是语句标号，每个 Si 是字段说明符。词汇 FORMAT 和括号必须象上面那样表示。斜杠 (/) 和逗号 (，) 是字段分隔符，这将在分隔符小节讲述。字段被定义为由一个被传输项目占有的外部记录。

1） 字段说明符

字段说明符说明了数据字段的大小，并根据每个被传输数据规定了转换类型。FORTRAN 字段说明符可以有下列任何形式：

说明符	类别	说明符	类别
rFw. d	数值转换	rAw	Hollerith 转换
rGw. d		nHh1h2…hn	
rEw. d		'l1l2…ln'	
rDw. d		nX	空格说明
rLw		mP	比例因子
rLw	逻辑转换		

其中：

(1) w 和 n 是正的整常数，它规定了外部数据表示中的字段宽度 （包括数字、小数点、代数符号）。

(2) d 是整数，它说明了外部数据表示中的小数的位数。

(3) 符号 F，G，E，D，I，A，L 表示加在输入/输出清单项目上的转换类型。

(4) r 是一个任选的非零整数，表示说明符将重复 r 次。

(5) hi 和 li 是 FORTRAN 字符组中的字符。

(6) m 是一个表示比例的整常数（正的，负的，或零）。

2） 数值转换

有任何数值转换的输入操作允许数据用“自由格式”表示，即在外部表示中可以用逗号分隔字段。

F 型转换　　形式：Fw.d

使用这个转换来处理实型或双精度型数据，处理 w 个字符，其中有 d 位小数。

F 输出：

将数值转换并这样输出：负号（如果为负数），接着是数值的整数部分，随后是小数点和数值的 d 位小数部分。如果数值充不满字段，那末在字段中它向右对齐，左边插入足够的空格来充填字段。如果数值需要的字段位置多于 w，那末输出数值的前 w—1 位，前边有一个星号。

F 输出的例子：

FORMAT 说明符	内部的数值	输出（其中 b 为空格）
F10.4	368.42	bb368.4200
F7.1	-4786.361	-4786.4
F8.4	8.7E-2	bb0.0870
F6.4	4739.76	*.7600
F7.3	-5.6	b-5.600

在上面第四行中的*号表示丢失了前面的位。

F 输入：（见下面的 E 输入的描述）

E 型转换　　形式：Ew．d

这个转换用于处理实型或双精度型数据，处理 w 个字符，其中有 d 位小数。

E 输出：

转换数值，舍入成 d 位并按以下顺序输出：

(1) 负号（如果为负数）
(2) 零和小数点
(3) d 位小数
(4) 字母 E
(5) 指数的符号（负号或空格）
(6) 两位指数

如上面已讲过那样，数值是右齐的，如果需要的话就在字段 w 中的左边位置上填 以 空格。字段宽度 w 应满足关系：

$$w > d + 7$$

否则会丢失有效数字。E 输出的一些例子如下：

FORMAT 说明符	内部的数值	输出 (b＝空格)
E12.5	76.573	bb.76573Eb02
E14.7	－32672.354	－b.3267235Eb05
E13.4	－0.0012321	bb－b.1232E－02
E18.2	76321.73	b.76Eb05

E 输入：

在 E，F 或 G 转换下处理的数据值在外部输入介质中可以用相当自由的格式。两 者 的格式是相同的。格式如下：

(1) 开头的空格（忽略）
(2) ＋或－号（无符号输入假设为正）
(3) 数字串
(4) 小数点
(5) 第二串数字
(6) 字母 E
(7) ＋或－号
(8) 十进制指数

上述清单中的每一项都是任选的，但要遵守以下条件：

(1) 如果选用（上述）FORMAT 项 3 和 5，则需要 4 。

(2) 如果选用 FORMAT 项 8，则 6 或 7 或两者都需要。

(3) 所有的非开头空格都认为是 0 。

输入数据的长度可以是任何位数，正确的大小将被导出，但是只保留第三节中规定的实数范围的精度。

E 输入、F 输入和 G 输入的例子：

FORMAT 说明符	输入 (b＝空格)	内部的数值
E10.3	＋0.23756E＋4	＋2375.6
E10.3	bbbbb17631	＋17.631
G 8.3	b1628911	＋1628.911
F12.4	bbbb－6321132	－632.1132

注意，在以上例子中，如果输入字符中没有给出小数点，那末 FORMAT 说明中的 d 就随同指数 （如果有的话）一起建立小数点。 如果在输入字符中包含有小数点， 就忽略 d 说明。

在输入格式说明中，字母 E，F 和 G 是可以互换的，其最终结果相同。

D 型转换

D 输入和 D 输出与 E 输入和 E 输出是相同的，不同处在于它用“D”代替“E”来说明指数。

G 型转换　　形式：　Gw．d

使用这个方法处理实型或双精度型数据，它处理 w 个字符，其中 d 是有效位数。

G 输入：　　（参阅 E 输入中的描述）

G 输出：

输出转换的方法是输出数值大小的函数，令 n 为数值的幅度，下表显示怎样输出数据：

幅度	等效的转换
$.1<=n<1$	F(w－4).d，4X
$1<=n<10$	F(w－4).(d－1)，4X
⋮	⋮
$10^{d-2}<=n<10^{d-1}$	F(w－4).1，4X
$10^{d-1}<=n<10^{d}$	F(w－4).0，4X
其他	Ew.d

I 型转换　　形式：Iw

这种转换形式只转换整型数据，w 说明字段宽度。

I 输出：

将数值转换成整常数，负数前边加－号。如果数值填不满字段，那末它在字段中向右对齐并在左边插入足够空格来填满字段。如果数值超过字段宽度，那末只输出最后有效位W-1个字符，其前边有一个星号。

例如：

FORMAT 说明符	内部数值	输出（b＝空格）
I6	＋281	bbb281
I6	－23261	－23261
I3	126	126
I4	－226	－226

I 输入：

输入 w 个字符的字段并转换成内部的整型格式，可以在整数前边加一负号，如果不加符号，此数值就认为是正的。

容许的整数数值范围是－32768 到 32767。凡是非开头空格都作 0 处理。

例如：

FORMAT 说明符	输入（b＝空格）	内部数值
I4	b 124	124
I4	－124	－124
I7	bb6732b	67320
I4	1b2b	1020

3） Hollerith 转换

A 型转换

A 转换的形式如下：　　Aw

这个说明符使计算机根据说明的清单项目读入或输出非修改的 Hollerith 字符。

用 Aw 在内部和外部表示之间传输的实际字符的最大数目是相应清单项目的存贮单位数目的 4 倍（即，1 个字符对应逻辑型项目，2 个字符对应整型项目，4 个字符对应实型项目，8 个字符对应双精度项目）。

A 输出：

如果 w 大于 4n（其中 n 是清单项目需要的存贮单位数目），那末外部输出字段将由 w-4n 个空格和随后的 4n 个内部表示的字符组成。如果 w 小于 4n，那末外部输出字段将由内部表示的最左 w 个字符组成。

例如：

FORMAT 说明符	内部	类型	输出（b＝空格）
A1	A1	整型	A
A2	AB	整型	AB
A3	ABCD	实型	ABC
A4	ABCD	实型	ABCD
A7	ABCD	实型	bbbABCD

A 输入：

如果 w 大于 4n（其中 n 是相应清单项目所需要的存贮单位数目），那末取外部输入字段的最右 4n 个字符。如果 w 小于 4n，那末在内部表示中就出现 w 个字符，它是向左对齐的，后面有 4n－w 个空格。

例如：

FORMAT 说明符	输入字符	类型	内部（b＝空格）
A 1	A	整型	Ab
A 3	ABC	整型	AB
A 4	ABCD	整型	AB
A 1	A	实型	Abbb
A 7	ABCDEFG	实型	DEFG

H 转换

H 转换的形式如下：　　nHh 1 h2…hn 或 'h1h2…hn'

这些说明符在说明符与外部字段之间处理 Hollerith 字符串，其中每个 h 代表 ASCⅡ字符组中的任何字符。

注：需特别考虑到如果在第二种形式中，文字串里要用一个单引号（'），那末，在字符串里这个单引号要用两个连续的单引号表示，请看下面的举例。

H 输出：

将 n 个字符 h 放到外部字段中。在 nHh1h2…hn 形式中，字符串里的字符数目必须精确地与 n 说明相同。否则，就把别的说明符中的字符作为字符串的一部分取来。在两种形式中，空格都是当作字符的。

例如：

FORMAT 说明符　　输出（b＝空格）

1HA	或'A'	A
8HbSTRINGb	或'bSTRINGb'	bSTRINGb
11HX(2, 3)=12.0	或'X(2, 3)=12.0'	X(2, 3)=12.0
12HIbSHOULDN'T	或'IbSHOULDN''T'	IbSHOULDN T

H 输入：

用输入记录中的 n 个字符代替字符串 hi 的 n 个字符，结果按字段说明符得到了新的字符串。例如：

FORMAT 说明符		输入	结果说明符
4H1234	或'1234'	ABCD	4HABCD 或'ABCD'
7HbbFALSE	或'bbFALSE'	bFALSEb	7HbFALSEb或'bFALSEb'
6Hbbbbbb	或'bbbbbb'	MATRIX	6HMATRIX 或'MATRIX'

4) 逻辑转换

逻辑转换的形式如下： Lw

L 输出：

如果对应此说明符的输出清单项目的值是 0，则输出 F；否则输出 T。如果 w 大于 1，那末在字母前有 w-1 个空格。

例如：

FORMAT 说明符	内部数值	输出 (b=空格)
L1	=0	F
L1	<>0	T
L5	<>0	bbbbT
L7	=0	bbbbbbF

L 输入：

外部表示占有 w 个位置，它由任意的空格，接着字母 "T" 或 "F"，随后是任意的字符等组成。

5) X 说明符

X 转换的形式如下： nX

这一说明符不产生转换，也不处理输入/输出清单中的对应项目。在用于输出时，它在输出记录中插入 n 个空格。在输入情况下，这个说明符使得输入记录中的下 n 个字符被跳过。

输出举例：

	FORMAT 语句	输出 (b=空格)
3	FORMAT(1HA,4X,2HBC)	AbbbbBC
7	FORMAT(3X,4HABCD, 1X)	bbbABCDb

输入举例：

	FORMAT 语句	输入字符串	得到的输入
10	FORMAT (F4.1,3X,F3.0)	12.5ABC120	12.5,120
5	FORMAT (7X,I3)	1234567012	012

6) P 说明符

P 说明符用于规定实型转换 (F、E、D、G)的比例因子，其形式是 nP,其中n是整常

数（正、负或零）

在开始调用每个格式I/O（每个READ或WRITE语句）时，比例因子自动置零。如果在扫描FORMAT时遇到了P说明符，那末比因子改变为n，在遇到另一个P说明符或I/O结束以前，比例因子保持不变。

比例因子对输入的影响：

在E、F或G输入过程中，比例因子只对没有指数的外部表示起作用，在这种情况下，内部值将比外部值小10**n倍（在存贮前将数值除以10**n）。

比例因子对输出的影响：

E输出，D输出：

系数相对小数点向左移n位，同时指数减去n（数值保持相同）。

F输出：

外部数值为10**n乘内部数值。

G输出：

如果内部数值小到能用F转换输出，则忽略比例因子，否则其作用与E输出相同。

7) FORMAT语句的特殊控制性

(1) 重复说明

① E、F、D、G、I、L和A字段说明符都可以用形式rEw.d，rFw.d，rCw.d，rIw，rLw，rAw中的重复计数r表示成重复说明符。

以下FORMAT语句对是等效的：

```
      66 FORMAT(3F8.3, F9.2)
```

等效于：

```
      66 FORMAT (F8.3, F8.3, F8.3, F9.2)
      14 FORMAT (2I3, 2A5, 2E10.5)
```

等效于：

```
      14 FORMAT (I3, I3, A5, A5, E10.5, E10.5)
```

② 一组字段说明符的重复是这样达到的：将这组字段说明符用括号括起来，重复计数放括号前边。没有重复计数，则表示r=1。包括FORMAT语句需要的括号在内，最多允许两层括号。

注意，下列等效语句：

```
   22 FORMAT (I3, 4(F6.1, 2X))
```

等效于：

```
   22 FORMAT (I3, F6.1, 2X, F6.1, 2X, F6.1, 2X, F6.1, 2X)
```

③ FORMAT说明符的重复也发生在这种时候：当FORMAT语句中的所有说明符都已经用了，而输入/输出清单中还有项目没有处理过。在这种情况下，由FORMAT语中第一个打开的括号开始重复使用FORMAT说明符。在重复使用中，说明符前边的重复计数仍然有效。这种类型的重复使用FORMAT说明符，在每次开始重复使用时，结束当前记录的处理，并开始处理新的记录。在这些情况下，记录分界和下一小节中一样。

输入举例：

```
      DIMENSION A(100)
      READ (3,13) A
      ⋮
```

```
    13  FORMAT (5F7.3)
```

在这个例子中，输入20个记录的每个记录中的前5个量，并赋给数组A的数组元素

输出举例：

```
        ⋮
        WRITE (6,12) E, F, K, L, M, KK, LL, MM, K3, L3, M3
        ⋮
    12  FORMAT (2F9.4, (3I7))
```

在这个例子中，输出3个记录，记录1含有E，F，K，L和M，因为说明符317用了两次，所以记录2含有KK，LL，MM，记录3含有K3，L3，M3。

(2) 字段分隔符

在FORMAT语句中，两个相邻的说明符必须用逗号或一个斜杠或多个斜杠分隔。

例如：　2HOK/F6.3 或 2HOK，F6.3

斜杠不仅分隔字段说明符，而且也规定了格式记录的分界。

每个斜杠结束一个记录的处理并建立一个记录的处理。剩余的输入记录就被忽略，而输出记录的剩余部分就用空格充填。连续的斜杠（///…/）在输入时使连续的记录被忽略，在输出时就输出连续的空记录。

输出举例：

```
        DIMENSION A(100), J(20)
        ⋮
        WRITE (7, 8)J, A
    8   FORMAT (10I7/10I7/50F7.3/50F7.3)
```

在此例子中，由WRITE语句的清单规定的数据，按照FORMAT语句的说明，输出到设备7上。输出的四个记录如下：

记录1	记录2	记录3	记录4
J(1)	J(11)	A(1)	A(51)
J(2)	J(12)	A(2)	A(52)
⋮	⋮	⋮	⋮
J(10)	J(20)	A(50)	A(100)

输入举例：

```
        DIMENSION B (10)
        ⋮
        READ (4, 17) B
    17  FORMAT (F10.2/F10.2///8F10.2)
```

在此例子中，两个数组元素B(1)和B(2)接受连续的记录的第一个数据作它们的值（这两个记录的剩余部分被忽略）。第三个记录和第4个记录被忽略，数组的剩余元素由第5个记录来充填。

8) 格式控制，清单说明和记录分界

应注意FORMAT制控、输入/输出清单和记录分界之间的下列关系和相互作用：

(1) 格式READ语句或格式WRITE语句的执行开始了FORMAT控制。

(2) 数据的转换取决于输入/输出清单中的元素和FORMAT语句中的字段说明符提供的有关信息。

(3) 如果有输入/输出清单，那末在FORMAT语句中必须至少有一个类型说明符E、F、D、G、I、L或A。

(4) 格式READ语句的每次执行都输入新的记录。

(5) 输入清单中的每个项目对应着记录中的字符串和FORMAT语句中的类型说明符E、F、G、I、L或A。

(6) H和X说明符在外部记录和字段说明符之间直接传达信息，而不引用清单项目。

(7) 在输入中，只要遇到了FORMAT语句中的斜杠或者FORMAT说明符已用完并开始重复使用说明符，那末结束当前记录的处理，并引起：

① 忽略记录中的任何非处理字符。

② 如果必须有更多的输入以满足清单的需要，那末读入下个记录。

(8) READ语句的结束要在输入清单中的全部项目已经满足时并且：

① 下一FORMAT说明符是E、F、G、I、L或A。

② FORMAT控制已经到达FORMAT语句的最后一个外层右括号。

如果输入清单已经满足，但是下个FORMAT说明符是H或X，那末处理更多数据（可能输入新的记录），直到上述条件之一存在为止。

(9) 如果FORMAT控制到达FORMAT语句的最后一个右括号，但是还有很多清单项目要处理，则使用全部或部分说明符。（请参阅前一小节，重复说明的描述之3）。

(10) 在执行格式WRITE语句时，每次遇到FORMAT语句中的斜杠或FORMAT控制已经到达最右边的右括号就输出一个记录。FORMAT控制的结束按以上(8)中所描述的READ结束的两种方法之一。未完成的记录是用空格充填的，以保持记录长度。

9) FORMAT托架控制

每个格式输出记录的第一个字符是用于将托架控制信息传达给输出设备的，因此永不印出第一个字符。托架控制字符决定了在打印这行以前执行的动作。选择项如下：

控制字符	打印前执行的动作
0	跳2行
1	换页
+	不换行
其他	跳1行

10) 数组中格式说明

格式READ或WRITE语句的格式引用f（见第1小节）可以是数组名称，它代替语句标号。如果实行这种引用，那末在执行READ/WRITE语句时，数组中含有信息的第一部分——以自然顺序，必须构成一个有效的FORMAT说明。数组可以在结束FORMAT说明的右括号后面含有非FORMAT信息。

插在数组中的FORMAT说明具有FORMAT语句定义的相同形式（即，以左括号开始，以右括号结束）。

可以用DATA初始语句，或用READ语句随同带有Aw的FORMAT一起将FORMAT说明插入数组中。例如：

假设 FORMAT 说明 (3F10.3, 4I6)

或相似的 12 个字符的说明已存入一数组。这数组最少要 3 个存贮单位。

以下的 FORTRAN 程序显示了建立 FORMAT 说明，然后引用数组的格式 READ 或 WRITE 的各种方法。

```
C  DECLARE A REAL ARRAY
      DIMENSION A(3), B(3), M(4)
C  INITIALIZE FORMAT WITH DATA STATEMENT
      DATA A/'(3F1', '0.3, ', '416)'/
       ⋮
C  READ DATA USING FORMAT SPECIFICATIONS
C      IN ARRAY A
      READ(6, A)B, M
C  DECLARE AN INTEGER ARRAY
     DIMENSION IA(4), B(3), M(4)
       ⋮
C  READ FORMAT SPECIFICATIONS
      READ(7, 15)IA
C  FORMAT FOR INPUT OF FORMAT SPECIFICATIONS
   15 FORMAT(4A2)
       ⋮
C  READ DATA USING PREVIOUSLY INPUT
C  FORMAT SPECIFICATION
      READ(7, IA)B, M
       ⋮
```

九、函 数 和 子 程 序

FORTRAN 语言提供了经常需要的编制过程的定义和使用方法，这样，此过程的语句或语句段只需在程序中出现一次，但是在任何时候和每次需要的时候，可以引用它们和将它们引入程序的逻辑执行序列中。

这些过程是：

(1) 语句函数

(2) 库函数

(3) 函数 (FUNCTION) 子程序

(4) 子例行 (SUBROUTINE) 子程序

这些过程的每一个都有它们各自的引用和定义的独特要求。这些要求在本节的以下各小节中讨论，但是，某些特点对于全部或两个或几个过程是共同的。这些共同的特点如下：

(1) 这些过程的每一个都是用它们的名称引用的，在所有情况下，名称是一至六个字母数字符号，其第一个是字母。

(2) 前三个过程称为“函数”，并且有以下几点是相同的：

① 它们总是单值的（即，它们给引用它们的程序单元带回一个值）。

② 它们是用含有函数名的表达式引用的。

③ 如果单值解的数据类型和预先约定习惯（即隐含规则——译注）所表示的类型不同，那末它们必须用类型说明来规定类型。

(3) FUNCTION 子程序和 SUBROUTINE 子程序都是作为程序单元考虑的。

在下面的这些过程的描述中，调用程序这一术语的意思是引用某一过程的程序单元或过程。术语“被调用程序”的意思是被引用的过程。

1. 程序（PROGRAM）语句

PROGRAM 语句提供了指定主程序单元名称的方法。此语句的形式为：

PROGRAM name

其中 name 是主程序单元的名称，它由 1 — 6 个字母数字组成，其第一个符号为字母。如果使用此语句，则 PROGRAM 语句必须出现在主程序单元中其他任何语句的前边，如果在主程序中不出现 PROGRAM 语句，则编译器将名称 $MAIN 赋给此程序。

2. 语句函数

语句函数用一个算术或逻辑赋值语句来定义，并且只和其中出现语句函数的程序单元相关。语句函数的一般形式如下：

f (a1, a2, …, an)=e

其中，f 是函数名称，ai 是哑宗量，e 是算术或逻辑表达式。

语句函数的次序、结构和使用的规则如下：

1) 如果程序单元中存在语句函数，那末它们的定义必须放在程序单元中所有可执行语句之前和所有说明语句的后边。

2) ai 是不同的变量名称或数组元素，但是是哑变量，它们在程序单元中可以具有和别处出现的相同类型的变量一样的名称。

3) 表达式 e 的构成遵循第四节中的规则。表达式只可以含有对哑宗量和非文字常数、变量和数组元素、实用函数和数学函数、以及预先定义的语句函数等的引用。

4) 任何语句函数的名称或宗量的类型，要是和预先约定习惯规定的类型不同，则必须用类型说明语句进行规定。

5) f 和 e 之间的关系必须遵循第五节中的置换规则。

6) 语句函数是用它的名称及后面跟有带括号的宗量清单来调用的。用调用中指定的宗量对表达式求解，并用求得的结果代表引用的语句函数。

7) 在类型上，每个宗量清单中的第 i 项参量必须和语句函数中的第 i 项哑元相一致。

下例展示了语句函数和语句函数调用：

```
C STATEMENT FUNCTION DEFINITION
C
   FUNC1 (A, B, C, D) = ((A+B)**C) /D
C STATEMENT FUNCTION CALL
```

C

A12=A1-FUNC1 (X, Y, 27, C7)

3. 库函数

库函数是一组实用函数和数学函数，它们是FORTRAN系统“内部固有的”。它们的名称是预先给处理器规定的并自动地规定类型。这些函数列出在表9—1和表9—2中。在表中，如果需要两个以上的宗量，则宗量表示成a1，a2，…，an；否则只需要一个宗量。

当库函数的名称用在算术表达式中的时候，库函数就被引用了。这种引用取以下形式：

f (a1, a2, …, an)

其中，f是函数名称，ai是实际宗量。宗量的类型、数目和顺序必须和表9—1和表9—2中指出的说明一致。

除表9—1和9—2中列出的函数以外，还有四个额外的库子程序，允许对8080（或Z80）硬件直接存取。它们是：

PEEK, POKE, INP, OUT

PEEK和INP是逻辑函数；POKE和OUT是子程序。PEEK和POKE允许对任何内存单元直接存取。PEEK(a) 得到由a指定的内存单元的内容。CALL POKE(a1,a2)使得由a1指定的内存单元的内容被a2代替。INP和OUT允许对I/O口直接存取。INP(a)的作用是从入口a输入并得到一个8位的输入值。CALL OUT(a1, a2) 将a2值输出到由a1指定的出口。

例如：

```
A1=B+FLOAT(17)
MAGNI=ABS(KBAR)
FDIF=DIM(C, D)
S3=SIN(T12)
ROOT=(-B+SQRT(B**2-4.*A*C)) / (2.*A)
```

表9—1 内 部 函 数

函数名称	定 义	宗量类型	函数类型
ABS	|a|	实	实
IABS	（绝对值）	整	整
DABS		双精	双精
AINT	a的符号乘以<=|a|的	实	实
INT	最大整数	实	整
IDINT		双精	整
AMOD	a1 (mod a2)	实	实
MOD	（求余数）	整	整
AMAX0	Max(a1, a2,…)	整	实
AMAX1	（取大数）	实	实
MAX0		整	整

MAX_1		实	整
$DMAX_1$		双精	双精
FLOAT	整型转换成实型	整	实
IFIX	实型转换成整型	实	整
SIGN	a_2 的符号乘以 $\|a_1\|$	实	实
ISIGN		整	整
DSIGN		双精	双精
$AMIN_0$	Min(a1, a2,…)	整	实
$AMIN_1$	(取小数)	实	实
MIN_0		整	整
MIN_1		实	整
$DMIN_1$		双精	双精
DIM	a1—Min(a1, a2)	实	实
IDIM	(求正差)	整	整
SNGL	(双精度型转换为实型)	双精	实
DBLE	(实型转换为双精度型)	实	双精

表9—2 基本的外部函数

名 称	自变量数目	定 义	自变量类型	函数类型
EXP	1	e**a	实	实
DEXP	1	(指数)	双精	双精
ALOG	1	ln(a)	实	实
DLOG	1	(自然对数)	双精	双精
ALOG10	1	$\log_{10}(a)$	实	实
DLOG10	1	(普通对数)	双精	双精
SIN	1	Sin(a)	实	实
DSIN	1	(正弦)	双精	双精
COS	1	cos(a)	实	实
DCOS	1	(余弦)	双精	双精
TANH	1	tanh(a) (双曲正切)	实	实
SQRT	1	(a)**(1/2)	实	实
DSQRT	1	(平方根)	双精	双精
ATAN	1	arctan(a)	实	实
DATAN	1	(反正切)	双精	双精
$ATAN_2$	2	arctan(a1/a2)	实	实

DATAN2	2	$\left(\frac{a1}{a2}\text{的反正切}\right)$	双精	双精
DMOD	2	a1(mod a2) (求余数)	双精	双精

4. FUNCTION 子程序

以 FUNCTION 语句开始的程序单元叫做函数子程序。FUNCTION 语句具有下列形式之一：

```
t FUNCTION f (a1, a2,…,an)
```

或 `FUNCTION f (a1, a2,…,an)`

其中：

(1) t 是 INTEGER、REAL、DOUBLE PRECISION 或 LOGICAL，或者如第二种形式所示那样没有 t。

(2) f 是函数子程序的名称。

(3) ai 是哑宗量，至少必须有一个哑宗量，它代表变量名称、数组名称或子例行子程序或其他函数子程序的哑名称。

5. 函数子程序的结构

函数子程序的结构必须遵守以下限制：

(1) FUNCTION 语句必须是程序单元的第一条语句。

(2) 在函数子程序中，函数名称必须在赋值语句的等号的左边，或作为输入语句的输入清单中的一项至少出现一次。这确定了函数的值，于是它将被带回到调用程序。

附加的值可以通过给哑宗量赋值返回到调用程序。

例如：

```
      FUNCTION Z7(A, B, C)
      ⋮
      Z7=5.* (A - B) + SQRT(C)
      ⋮
C     REDEFINE ARGUMENT
      B=B+Z7
      ⋮
      RETURN
      ⋮
      END
```

(3) 哑宗量清单中的名称不可以出现在函数子程序中的 EQUIVALENCE, COMMON 或 DATA 语句中。

(4) 如果哑宗量是数组名称，则在子程序中必须有数组说明符，其维数信息和调用程序中的一致。

(5) 函数子程序不可以含有 BLOCK DATA 语句、SUBROUTINE 语句、别的 FUNC-

TION 语句、或任何引用 FUNCTION 定义的语句或别的引用 FUNCTION 定义的子程序，除此以外可以含有任何 FORTRAN 语句。

(6) 函数子程序的逻辑结束是 RETURN 语句，必须至少有一个 RETURN 语句。

(7) 子程序必须用 NED 语句作实际结束。

例如：

```
      FUNCTION SUM(BARY, I, J)
      DIMENSION BARY(10, 20)
      SUM=0.0
      DO 8 K=1, I
      DO 8 M=1, J
    8 SUM=SUM+BARY(K, M)
      RETURN
      END
```

6. 引用函数子程序

只要函数名称及伴随的自变量清单作为运算量用于表达式中，就调用了函数子程序。这种调用取以下形式：

f (a1, a2,…,an)

其中，f 是函数名称，ai 是实际宗量，形式中必须有括号。

宗量 ai 在类型，顺序和数目上必须与被调用函数子程序的 FUNCTION 语句中的哑 宗量一致。它们可以为下列任何一个：

(1) 变量名称

(2) 数组元素名称

(3) 数组名称

(4) 表达式

(5) 子例行或函数子程序名称

(6) Hollerith 或文字常数

如果 ai 是子程序名称，则其名称必须预先用出现在 EXTERNAL 语句中的方法使它和普通变量有所区别，并且被调用函数子程序中的相应哑宗量必须用于程序引用中。

如果 ai 是 Holleith 或文字常数，则相应的虚拟变量应有足够的存贮单位，它完 全 对应常数所需要的存贮器数量。

在调用函数子程序时，程序控制就转到 FUNCTION 语句下面的第一条可执行 语 句。

下面的举例显示了函数子程序的引用：

```
      Z10=FT1+Z7(D, T3, RHO)
      DIMENSION DAT (5.5)
      ⋮
      S1=TOT1+SUM (DAT, 5, 5)
```

7. 子例行（SUBROUTINE）子程序

以 SUBROUTINE 语句开头的程序单元称为子例行子程序。SUBROUTINE 语句具有以下形式之一：

SUBROUTINE s(a_1, a_2, …, an)

或 SUBROUTINE s

其中，s 是子例行子程序的名称，每个 ai 是代表变量或数组名称、或者其他子例行或函数名称的哑宗量。

8. 子例行子程序的结构

(1) SUBROUTINE 语句必须是子程序的第一条语句。

(2) SUBROUTINE 子程序名称只能出现在初始的 SUBROUTINE 语句中而不可出现在其他语句中。

(3) 哑宗量的名称一定不能出现在子程序中的 EQUIVALENCE,COMMON 或DATA 语句中。

(4) 如果哑宗量是数组名称，则数组说明符必须出现在维数信息和调用程序中一致的子程序中。

(5) 如果哑宗量代表一数值，此数值是由子例行子程序确定并带回到调用程序的，那末，这些哑宗量必须出现在程序中置换语句的等号左边、输入语句的输入清单中、或者作为子程序引用中的参量。

(6) 子例行子程序中不可以含有 BLOCK DATA 语句、FUNCTION 语句、别的SUBROUTINE 语句、PROGRAM 语句、或者引用子例行子程序定义的任何语句、或别的引用子例行子程序定义的子程序，除此以外可以含有任何 FORTRAN 语句。

(7) 子例行子程序可以含有任何数目的 RETURN 语句，至少必须有一个。

(8) RETURN 语句是子程序的逻辑终点。

(9) 子行例子程序的实际结尾是 END 语句。

(10) 如果由调用程序传送到子例行子程序的实际宗量是子例行或函数子程序的名称，那末，相应的哑宗量必须用于被调用子例行子程序中，作为子程序引用。

例如：

```
C  SUBROUTINE TO COUNT POSITIVE ELEMENTS
C     IN AN ARRAY
      SUBROUTINE COTNT P(ARRY, I, CNT)
      DIMENSION ARRY(7)
      CNT=0
      DO 9 J=1, I
      IF(ARRY(J))9, 5, 5
9     CONTINUE
      RETURN
5     CNT=CNT+1.0
```

```
      GOTO 9
      END
```

9. 引用子例行子程序

可以使用 CALL 语句调用子例行子程序。CALL 语句具有下列形式之一：

```
CALL s(a1, a2,…, an)
```

或 `CALL s`

其中，s 是子例行子程序名称，ai 是子程序使用的实际宗量。ai 在类型、顺序和数目上必须和定义程序的 SUBROUTINE 语句中的哑宗量相对应。

CALL 语句中的自变量必须遵循下列规则：

(1) 出现在自变量清单中的函数和子例行名称必须预先出现在 EXTERNAL 语句中 。

(2) 如果被调用的子例行子程序含有数组变量说明符，则 CALL 语句必须含有数组 的实际名称和实际的维数说明作为自变量。

(3) 如果子例行子程序的哑自变量清单中的项目是一数组，则 CALL 语句自变量清 单中相应的项目也必须是数组。

在调用子例行子程序时，程序控制就传送到 SUBROUTINE 语句下面的第一条可执 行语句。

例如：

```
      DIMENSION DATA(10)
      ⋮
C     THE STATEMENT BELOW CALLS THE
C     SUBROUTINE IN THE PREVIOUS PARAGRAPH
C
      CALL COUNTP(DATA, 10, CPOS)
```

10. 由函数和子例行子程序返回

函数或子例行子程序的逻辑结尾是 RETURN 语句，它将控制传送回调 用 程 序。RETURN 语句的一般形式只是单词：

```
RETURN
```

下面给出了 RETURN 语句的使用规则：

(1) 在每个子例行或函数程序中至少必须有一个 RETURN 语句。

(2) 函数子程序的 RETURN，是返回到调用程序中函数引用下面的指令序列。

(3) 子例行子程序的 RETURN，是返回到调用程序中的下一条可执行语句，它在逻辑上应在 CALL 语句之后。

(4) 由函数子程序返回的子程序单值结果，可用于表达式计算，此表达式中进行了函数引用。

(5) 由子例行子程序返回的值，赋给了子例行中的自变量，它们可由调用程序使用。

例如：

调用程序单元

```
      ⋮
      CALL SUBR(Z9, B7, R1)
      ⋮
```

被调用程序单元

```
      SUBROUTINE SUBR(A, B, C)
      READ (3, 7) B
      A=B**C
      RETURN
7     FORMAT (F9.2)
      END
```

在这个例子中，在执行 RETURN 以后，对调用程序而言，Z9 和 B7 就成为可用的了。

11. 在子程序中处理数组

如果调用程序处理一个属于子程序的数组名称，则子程序必须含有数组有关的维数信息。如果子程序的任何哑宗量代表数组或数组元素，那末，子程序必须含有数组说明符。

例如，一个表示计算任何一维数组元素平均值的函数子程序可以是下面这样的：

调用程序单元

```
      DIMENSION Z1(50), Z2(25)
      ⋮
      A1=AVG(Z1, 50)
      ⋮
      A2=A1-AVG(Z2, 25)
      ⋮
```

被调用程序单元

```
      FUNCTION AVG(ARG, I)
      DIMENSION ARG(50)
      SUM=0.0
      DO 20 J=1, I
20    SUM=SUM+ARG(J)
      AVG=SUM/FLOAT(I)
      RETURN
      END
```

注意，由函数子程序处理的实际数组是在调用程序中规定维数的，并且将数组名称和它们的实际维数通过引用函数子程序传送给函数子程序。函数子程序本身包含有哑数组并指定了数组说明符。

维数信息也可以在参量清单中传送函数子程序，例如：

调用程序单元

```
      DIMENSION A(3, 4, 5)
      ⋮
```

```
CALL SUBR(A, 3, 4, 5)
⋮
END
```

被调使程序单元

```
SUBROUTINE SUBR(X, I, J, K)
DIMENSION X(I, J, K)
⋮
RETURN
END
```

只有在数组名称和所有的可变维数是哑宗量的时候，使用可变维数才是有效的。可变维数必须是整型。在被调用程序中改变任何可变维数的值是无效的。

12. 数据块（BLOCK DATA）子程序

数据块（BLOCK DATA）子程序的唯一用途是在装入 FORTRAN 目标程序过程中将公用（COMMON）块中的数据置初值。数据块子程序以下列形式的 BLOCK DATA 语句开头：

BLOCK DATA〔子程序名称〕

并以 END 语句结束。这子程序只能含有类型、EQUIVALENCE、DATA、COMMON 和 DIMENSION 等语句，并遵循以下考虑：

(1) 如果 COMMON 块中的任何元素要置初值，那末块中的全部元素必须都列在 COMMON 语句中，那怕它们不是全部置初值。

(2) 用一个数据块子程序可以完成多个 COMMON 块中数据的置初值。

(3) 在任何给定时刻，可以装入多个数据块子程序。

(4) 任何个别的 COMMON 块项目只能用一个程序单元置初值。

例如：

```
BLOCK DATA
LOGICAL A1
COMMON/BETA/B(3, 3) /GAM/C(4)
COMMON/ ALPHA/A1, C, E, D
DATA B/1.1, 2.5, 3.8, 3*4.96,
    12*0.52, 1.1/, C/1.2E0, 3*4.0/
    DATA A1/.TRUE./, E/-5.6/
```

附录 1. 语言的扩充和限制

FORTRAN-80 语言包括了以下的对 ANSI 标准 FORTRAN（X3.9—1966）的扩充：

1. 如果 c 用于'STOP c'或'PAUSE c'语句中，则 c 可以是任何六个 ASCII 字符。

2. 在 READ 和 WRITE 语句中使用 ERR＝和 END＝可以规定错误分支和文件结尾分支。

3. 在 FORTRAN 库中增加了标准子程序 PEEK，POKE，INP，以及 OUT。

4. 语句函数可以使用下标变量。

5. 十六进制常数可以用于整型常数允许正常使用的任何地方。

6. 容许用 Hollerith 数据的文字形式（在单引号之间的字符串）代替标准的 nH 形式。

7. 在表达式中容许用 Hollerith 和文字代替整型常数。

8. 对连续行的数目没有限制。

9. 混合型的表达式和赋值是允许的，并自动地进行转换。

FORTRAN-80 对标准 FORTRAN 作了如下限制；

1. 没有复数（COMPLEX）数据类型，它将包含在以后的版型里。

2. 说明语句必须按以下顺序出现；

   (1) PROGRAM，SUBROUTINE，FUNCTION，BLOCK DATA

   (2) 类型，EXTERNAL，DIMENSION

   (3) COMMON

   (4) EQUIVALENCE

   (5) DATA

   (6) 语句函数

3. 给每个数据类型：整、实、双精、逻辑，分配了不同的计算机内存数量。

4. 置换语句的等号和循环（DO）语句的第一个逗号必须出现在初始语句行上。

有关这些语言的扩充和限制的详细叙述包含在本手册文本的相应章节中。

附录 2. I/O 接 口

输入/输出操作是适当逻辑设备号（LUN）的驱动器程序的调度表。$LUNTB 是调度表，对每个可能的 LUN，它包括一个 2 个字节的驱动程序地址。在其开头还有 1 字节的入口项目，它含有的内容是最大的 LUN 加 1。初始运行时的程序包提供了 10 个 LUN（1—10），它们中的任何一个都可以由用户重新定义或增添，只要简单地改变 $LUNTB 中的相应入口项目和添加更多的驱动程序。运行时的系统使用 LUN 3 作为错误和其他用户的通信。因此，LUN 3 应对应于操作员控制台。$LUNTB 的初始结构显示在此附录后面的清单中。

设备驱动程序也含有局部调度表。注意，$LUNTB 含有每个设备的一个地址，每个设备还真正具有七种可能的操作：

1) 格式读

2) 格式写

3) 二进制读

4) 二进制写

5) 倒带

6) 退格

7） 文件结束

每个设备驱动程序最多含有七个程序，这七个程序的每一个的起始地址按上列要求的顺序放在驱动程序的开头。$LUNTB 中的入口项指示着这个局部表，运行时的系统就检查此表得到相应程序的地址以处理所需要的 I/O 操作。

以下的约定适用于各 I/O 程序：

1)存贮单元 $BF 含有 READ 和 WRITE 的数据缓冲器地址。

2)对于 WRITE，要写的字节数目在存贮单元 $BL 中。

3)对于 READ，读的字节数目应返回 $BL中。

4)所有的 I/O 操作在出口以前都置位条件代码以表明错误条件、文件结束或正常返回：

(1) CY=1，Z=不论什么——I/O 错误

(2) CY=0，Z=0 ——遇到文件结束

(3) CY=0，Z=1 ——正常返回

运行时的系统在调用驱动程序以后检查条件代码，如果它们表明为非正常返回，则控制就传送到由“ERR=”或“END=”规定的标号，要是没有规定标号，就得到致命错误。

5) $IOERR 是打印“ILLEGAL I/O OPERATION（非法 I/O 操作）”信息（非致命的）的总程序，如果某些操作在特定的设备上不允许（如，在 TTY 上作二进制 I/O），就会使用这个程序。

注：I/O 缓冲器具有固定的 132 字节的最大长度，除非在安装时将它改变。如果驱动程序允许输入操作超过缓冲器的末尾写的话，那末基本的运行时的变量会受影响，后果是不可预测的。

本附录后面的清单含有 TTY 驱动程序的例子。REWIND、BACKSPACE 和 ENDFILE是作为非操作执行的，二进制 I/O 为错误。与运行时的程序包一起提供的 TTY 驱动程序：

```
MAC80  1.0  PAGE  1

                    00100  ;            TTY I/O DRIVER
                    00200
0000                00300                EXT       $IOERR, $BL, $BF, $ERR,
                                                      $TTYIN, $TTYOT
0012                00400  IRECER         EQU       022       ; INPUT RECORD
                                                        TOO LONG
0000                00500                ENTRY     $DRV3
0000  0013'       00600  $DRV3:          DW        DRV3FR    ; FORMATTED READ
0002  0042'       00700                DW        DRV3FW    ; FORMATTED
                                                        WRTTE
0004  0010'       00800                DW        DRV3BR    ; BINARY READ
0006  0010'       00900                DW        DRV3BW    ; BINARY WRITE
0008  000E'       01000                DW        DRV3RE    ; REWIND
000A  000E'       01100                DW        DRV3BA    ; BACKSPACE
```

```
000C  000E'       01200                DW      DRV3EN  ; ENDFILF
000E  AF          01300  DRV3EN:       XRA     A       ; THESE OPERAT-
                                                         IONS ARE NO-OPS
                  01400                                  FOR TTY
000E              01500  DRV3RE        EQU     DRV3EN
000E              01600  DRV3BA        EQU     DRV3EN
000F  C9          01700                RET
0010  C3 0000*    01800  DRV3BW:       JMP     $IOERR  ; ILLEGAL OPERAT-
                                                         IONS
                  01900                                  ; (PRINT ERROR
                                                         AND RETURN)
0010              02000  DRV3BR        EQU     DRV3BW
0013  AF          02100  DRV3FR:       XRA     A       ; READ
0014  32 0000 *   02200                STA     $BL     ; ZERO BUFFER
                                                         LENGTH
0017  CD 0000 *   02300  DRV31:        CALL    $TTYIN  ; INPUT A CHAR
001A  E6 7F       02400                ANI     0177    ; AND OFF PARITT
001C  FE 0A       02500                CPI     10      ; IGNORE LINE
                                                         FEEDS
001E  CA 0017'    02600                JZ      DRV31
0021  F5          02700                PUSH    PSW     ; SAVE IT
0022  2A 0015 *   02800                LHLD    $BL     ; GET CHAR POSIT
                                                         IN BUFFER
0025  26 00       02900                MVI     H,0     ; ONLY 1 BYTE
0027  EB          03000                XCHG
0028  2A 0000*    03100                LHLD    $BF     ; GET BUFFER ADDR
002B  19          03200                DAD     D       ; ADD OFFSET
002C  F1          03300                POP     PSW     ; GET CHAR
002D  77          03400                MOV     M,A     ; PUT IT IN BUFFER
002E  13          03500                INX     D       ; INCREMENT $BL
002F  EB          03600                XCHG
0030  22 0023 *   03700                SHLD    $BL     ; SAVE IT
0033  FE 0D       03800                CPI     015     ; CR?
0035  C8          03900                RZ              YES—DONE
0036  7D          04000                MOV     A,L     ; $BL
0037  FE80        04100                CPI     128     ; MAX IS DECIMAL
                                                         128
0039  AD 0017'    04200                JC      DRV31   ; GET NEXT CHAR
003C  CD 0000*    04300                CALL    $ERR
003F  12          04400                DB      IRECER  ; INPUT RECORD
                                                         TOO LONG
0040  AF          04500                XRA     A       ; CLEAR FLAGS
0041  C9          04600                RET
```

```
0042  3A 0031 *  04700 DRV3FW:   LDA     $BL       ; BUFFER LENGTH
0045  B7          04800           ORA     A
```

MAC 80 1.0 PAGE 1

```
                   00100 ; COMMENT
                   00200 ;              DRIVER ADDRESSES FOR LUN'S
                                        1 THROUGH 10
                   00210 ;
0001               00220 LPT            EQU     1          ; UNIT 2 IS
                                                            LPT
0001               00230 DSK            EQU     1          ; UNITS 6-10
                                                            ARE DSK
0000               00235 DTC            EQU     0          ; DTC COMM-
                                                            UNICATIONS
                                                            UNIT 4
                   00240 ;
                   00300
0000               00400                ENTRY   $LUNTB
0000               00500                EXT     $DRV3
0000  0B           00600 $LUNTB:        DB      013        ; MAXLUN+1
0001  0000 *       00700                DW      $DRV3      ; THEY ALL POINT
                                                            TO $DRV3 FOR
                                                            NOW
0003               00800                IFF     LPT
                   00900                DW      $DRV3
0003               01000                ENDIF
0003               01100                IFT     LPT
0003               01200                EXT     LPTDRV
0003  0000 *       01300                DW      LPTDRV
0005               01400                ENDIF
0005  0001 *       01500                DW      $DRV3
0007               01510                IFF     DTC
0007  0005 *       01600                DW      $DRV3
0009               01602                ENDIF
0009               01604                IFT     DTC
                   01605                EXT     $CMDRV
                   01606                DW      $CMDRV
0009               01608                ENDIF
0009  007 *        01700                DW      $DRV3
000B               01800                IFF     DSK
                   01900                DW      $DRV3
                   02000                DW      $DRV3
                   02100                DW      $DRV3
                   02299                DW      $DRV3
```

```
                    02300                   DW      $DRV3
000B                02400                   ENDIF
000B                02500                   IFT     DSK
000B                02600                   EXT     DSKDRV
000B    0000 *      02700                   DW      DSKDRV
000D    000B *      02800                   DW      DSKDRV
000F    000D *      02900                   DW      DSKDRV
0011    000F *      03000                   DW      DSKDRV
0013    0011 *      03100                   DW      DSKDRV
0015                03200                   ENDIF
0015                03300                   END
```

MAC80 1.0 PAGE 2

```
LPT     0001    DSK     0001    DTC     0000    $LUNTB  0000
$DRV3   0009 *  LPTDRV  0003 *  DSKDRV  0013 *
```

MAC80 1.0 PAGE 2

```
0046    C8          04900           RZ              ; EMPTY BUFFER
0047    2A 0029 *   05000           LHLD    $BF     ; BUFFER ADDRESS
004A    3D          05100           DCR     A       ; DECREMENT LENGTH
004B    F5          05200           PUSH    PSW     ; SAVE IT
004C    3E 0D       05300           MVI     A,13    ; CR
004E    CD 0000 *   05400           CALL    $TTYOT  ; OUTPUT IT
0051    7E          05500           MOV     A,M     ; GET FIRST CHAR IN BUFFER
0052    FE 2B       05600           CPI     '+'
0054    CA 0079'    05700           JZ      DR3FW2  ; NO LTNE FEEDS
0057    FE 31       05800           CPI     '1'
0059    C2 0064'    05900           JNZ     DR3FW1  ; NOT FORM FEEDS
005C    3E 0C       06000           MVI     A,12    ; FORM FEED
005E    CD 004F *   06100           CALL    $TTYOT  ; OUTPUT IT
0061    C3 0079 '   06200           JMP     DR3FW2
0064    3E 0A       06300   DR3FW1: MVI     A,10    ; LF
0066    CD 005F *   06400           CALL    $TTYOT
0069    7E          06500           MOV     A,M     ; GET CHAR BACK
006A    FE 20       06600           CPI     ' '
```

```
006C  CA 0079 '     06700                JZ      DR3FW2          ; NO MORE LINE
                                                                     FEEDS
006F  FE 30         06800                CPI     '0'
0071  C2 0079 '     06900                JNZ     DR3FW2          ; NO MORE LINE
                                                                     FEEDS
0074  3E 0A         07000                MVI     A,10            ; LF
0076  CD 0067 *     07100                CALL    $TTYOT
0079  F1            07200  DR3FW2:       POP     PSW             ; GET    LENGTH
                                                                     BACK
007A  23            07300                INX     H               ; INCREMENT
                                                                     PTR
007B  C8            07400  DRV32:        RZ
007C  F5            07500                PUSH    PSW             ; SAVE CHAR
                                                                     COUNT
007D  7E            07600                MOV     A,M             ; GET NEXT CH-
                                                                     AR
007E  23            07700                INX     H               ; INCREMENT
                                                                     PTR
007F  CD 0077 *     07800                CALL    $TTYOT          ; OUTPUT CHAR
0082  F1            07900                POP     PSW             ; GET COUNT
0083  3D            08000                DCR     A               ; DECREMENT
                                                                     IT
0084  C3 007B'      08100                JMP     DRV32           ; ONE MORE TI-
                                                                     ME
0087                08200                END

MAC80 1.0      PAGE  3

$IOERR  0011*   $BL     0043*   $BF     0048*   $ERR    003D*
$TTYIN  0018*   $TTYOT  0080*   IRECER  0012    $DRV3   0000'
DRV3FE  0013'   DRV3FW  0042'   DRV3BR  0010'   DRV3BW  0010'
DRV3RE  000E'   DRV3BA  000E'   DRV3EN  000E'   DRV31   0071'
DR3FW2  0079'   DR3FW1  0064'   DRV32   007B'
```

附录 3. 子程序连接法

本附录定义了正常的子程序调用，像 FORTRAN 编译器产生的一样。它包括了简化 FORTRAN 程序与那些用其他语言（如 8080 汇编）书写的程序之间的连接。

没有参量的子程序引用产生了单纯的“CALL”指令。对应的子程序应通过单纯的“RET”返回（CALL 和 RET 是 8080操作码——有关说明请见汇编手册或 8080 参考手册）。

带有参量的子程序引用得到了稍微复杂的调用序列，参量总是由引用传送（即，传送的内容实际上是实际参量的低位字节的地住）。因此，不管什么类型，每个参量总占 2 个字节。

传送参量的方法取决于传递的参量数目：

1. 如果参量数目小于或等于3，则它们用寄存器传递。参量1将HL在中，参量2（若有的话）在DE中，参量3（若有的话）在BC中。

2. 如果参量数目大于3，则它们如下这样传递：

(1) 参量1在HL中，

(2) 参量2在DE中，

(3) 参量3至n在连续的数据块中，BC将指出此数据块的低位字节（即，参量3的低位字节）。

注意：用此方案，子程序必须知道有多少参量以便寻找它们。相反，调用程序决定了传递参量的正确数目，既不是编译器也不是运行时的系统来检查正确的参量数目。

如果子程序期待3个以上参量，并需要将它们传送到局部的数据区，则有系统子程序来完成这一传送。这个传递自变量的子程序取名为 $AT，其调用方法是用HL指示局部数据区，用BC指示第三个参量，A含有要传递的自变量数目（即，自变量总数减2）。子程序在调用 $AT 以前，存贮了前两个参量。例如，如果子程序想有5个参量：它应该是这样的：

```
SUBR:   SHLD    P1      ; SAVE   PARAMETER1
        XCHG
        SHLD    P2      ; SAVE   PARAMETER2
        MVI     A,3     ; NO. OF PARAMETERS LEFT
        LXI     H,P3    ; POINTER TO LOCAL AREA
        CALL    $AT     ;TRANSFER THE OTHER 3PARAMET -
        ⋮               ERS
        〔子程序实体〕
        ⋮
        RET             ; RETURN TO CALLER
P1:     DS      2       ; SPACE FOR PARAMTER 1
P2:     DS      2       ; SPACE FOR PARAMETER 2
P3:     DS      6       ; SPACE FOR PARAMETER 3 — 5
```

当在子程序中存取参量时，不要忘记它们是指示被传送的实际自变量的指针。

注意：使调用程序中的自变量在数目、类型和长度上与子程序所期待的参量一致（完全是程序员的责任），这适用于FORTRAN子程序，也适用于那些用汇编语言书写的程序。

FORTRAN函数（第九节）根据类型用寄存器或内存返回它们的值。逻辑结果用（A）返回，整数用(HL)，实数在 $AC 内存中，双精度数在 $DAC 内存中。$AC 和 $DAC 是尾数的低位字节的地址。

附录 4. ACSII字符的代码

十进制代码	字 符	十进制代码	字 符	十进制代码	字 符	
000	NUL	043	+	086	V	
001	SOH	044	′	087	W	
002	STX	045	—	088	X	
003	ETX	046	•	089	Y	
004	EOT	047	/	090	Z	
005	ENQ	048	0	091	[	
006	ACK	049	1	092	\	
007	BEI	050	2	093	]	
008	BS	051	3	094	∧	(or↑)
009	HT	052	4	095	<	(or←)
010	LF	053	5	096	′	
011	VT	054	6	097	a	
012	FF	055	7	098	b	
013	CR	056	8	099	c	
014	SO	057	9	100	d	
015	SI	058	:	101	e	
016	DLE	059	;	102	f	
017	DC_1	060	<	103	g	
018	DC_2	061	=	104	h	
019	DC_3	062	>	105	i	
020	DC_4	063	?	106	j	
021	NAK	064	@	107	k	
022	SYN	065	A	108	l	
023	ETB	066	B	109	m	
024	CAN	067	C	110	n	
025	EM	068	D	111	o	
026	SUB	069	E	112	p	
027	ESCAPE	070	F	113	q	
028	FS	071	G	114	r	
029	GS	072	H	115	s	
030	RS	073	I	116	t	
031	US	074	J	117	u	
032	SPACE	075	K	118	v	
033	!	076	L	119	w	
034	″	077	M	120	x	
035	#	078	N	121	y	
036	$	079	O	122	z	

续 表

十进制代码	字 符	十进制代码	字 符	十进制代码	字 符
037	%	080	P	123	}
038	&	081	Q	124	|
039	'	082	R	125	{
040	)	083	S	126	~
041	(	084	T	127	DEL
042	*	085	U		

LF＝换行　FF＝换页　CR＝回车　DEL＝删除

附录 5. FORTRAN-80 子程序库

FORTRAN-80 程序库含有许多子程序，供用户在 FORTRAN 或汇编程序中引用。在以下叙述中，$AC 涉及浮动累加器；$AC 是尾数的低位字节的地址。$AC+3 是指数的地址。$DAC 涉及双精度累加器；$DAC 是尾数低位字节的 地址。$DAC+7 是双精 度指数的地址。

所有的算术子程序（加、减、乘、除、乘方）都附有以下的调用约定习惯。

1) 自变量 1 用寄存器传递：

整数用〔HL〕

实数用 $AC

双精度数用 $DAC

2) 自变量 2 根据类型或者用寄存器或者用内存来传递：

(1) 整数用〔HL〕传递；如果〔HL〕含有自变量 1 则用〔DE〕。

(2) 实数或双精度数的值是用由〔HL〕指示内存传递的（〔HL〕指示尾数的低位字节）。

在库中含有以下算术子程序：

功能	名称	自变量 1 类型	自变量 2 类型
加	$AA	实	整
	$AB	实	实
	$AQ	双精	整
	$AR	双精	实
	$AU	双精	双精
除	$D9	整	整
	$DA	实	整
	$DB	实	实
	$DQ	双精	整
	$DR	双精	实
	$DU	双精	双精
乘方	$E9	整	整
	$EA	实	整

	$EB	实	实
	$EQ	双精	整
	$ER	双精	实
	$EU	双精	双精
乘	$M9	整	整
	$MA	实	整
	$MB	实	实
	$MQ	双精	整
	$MR	双精	实
	$MU	双精	双精
减	$SA	实	整
	$SB	实	实
	$SQ	双精	整
	$SR	双精	实
	$SU	双精	双精

还有数值类型之间转换用的附加库程序，自变量总是由这些转换程序用相应的寄存器传递和返回：

逻辑型用[A]

整数用[HL]

实用数 $AC

双精度数用 $DAC

名称	功能
$CA	整型转换成实型
$CC	整型转换成双精度型
$CH	实型转换成整型
$CJ	实型转换成逻辑型
$CK	实型转换成双精度型
$CX	双精度型转换成整型
$CY	双精度型转换成实型
$CZ	双精度型转换成逻辑型

（周宝兴 译　陈品璞 校）

第四篇　LINK-80 连接装配程序参考手册

LINK-80 连接装配程序把 FORTRAN 编译器和 MACRO-80 汇编器产生的浮动目标文件取来，并以能执行的形式把它们装入内存。另外，LINK-80 自动地搜索系统库(FORLIB)并把需要的程序装入，以满足任何不定的整体引用（即在系统库里由被编译程序产生转子）。

LINK-80 给用户提供了几种装载选择。程序可以装在用户指定的位置，并且，程序区与数据区可以在内存中分开。由 LINK-80 产生的可执行文件的内存映象能写到盘上。可执行文件名称的空缺后缀是/CMD。(要和 TRS-80 的 TRSDOS 一起使用)

1. 运行 LINK—80

当你给 TRSDOS 一个命令 L80 时（#2 号盘必须在盘驱动器上），你就把 LINK-80 连接装配程序运行起来了。当装配程序准备接受命令时，它用一个星号提醒用户，在含有 E 或 G 开关（见第 2 小节）的命令之后，或者在命令级按 BREAK 键之后，装配程序就返回到 TRSDOS。

命令行也由 LINK-80 支承。

1) LINK-80 命令

一串文件名称和（或）开关构成了 LINK-80 的命令。命令格式是

〔filename 1〕〔-switch 1〕〔, filename 2〕〔-switch 2 …〕

所有的文件名称必须以 TRSDOS 文件名格式表示。

在 LINK-80 接受命令以后，它会装载或搜索（见 S 开关）指定的文件。然后列出其余不定的全部符号，并在每个后边跟一个星号。

*=TEST　　检验文件 TEST/CRF 并产生一交叉引用的列表文件 TEST/LST

*T=TEST　检验文件 TEST/CRF 并产生一交叉引用的列表文件 T/LST.

交叉引用的列表文件和普通的列表文件之区别在于：

(1) 每一源语句用一交叉引用数来编号的。

(2) 在列表清单末端，变量名以字母次序以及引用或定义它的行号次序出现。有确定符号的行号用'#'来标志。

例如：

```
*MAIN
DATA        5200        5300
SUBR 1 *          (SUBR 1 是不定的)
DATA        5200        5300
*SUBR 1
*-G               (开始执行——见下)
```

特别，为了执行某一FORTAN程序和子程序，则用户敲入文件名清单，后面跟一个-G（开始执行）。在执行开始以前，LINK—80总是搜索系统库（FORLIB／REL）以满足任何未定的外部引用。如果你希望首先搜索你自己的库，则在装配命令串的末尾添加一个后边跟-S的文件名称。

2) LINK-80 开关

在LINK-80命令串中可以给若干开关，用以指定影响装配过程的动作。每一个开关前必须有一划线(-)，这些开关是：

开 关	作 用
R	复位。使装配程序回到它的初始状态。如果由于错误你已装载了一个不正确的文件并希望重新开始，则使用-R。在命令串中，只要一碰到-R就会立即发生影响。
E或E：Name	退出LINK-80并返回到操作系统。将在当前盘上搜索系统库以满足任何存在的不定整体。选择形式E：Name（其中Name是以前在一个模块中定义过的整体符号），使用Name作为程序的开始地址。使用-E是为装入程序并返回到监控程序。
G或G：Name	只要当前命令行一解释完，程序就开始执行。将在当前盘上搜索系统库以满足任何存在的不定整体。在执行实际开始以前，LINK-80打印两个数及BEGIN EXECUTION信息。这两个数是开始地址和下一个可用字节的地址。选择形式G：Name（其中，Name是以前在模块之一中定义的一个整体符号）。使用Name作为程序的开始地址。
N	当作了-E或-G后，如果指定一个⟨filename⟩-N，则程序就以选择的名称（带有空缺后缀CMD）存到盘上。
P和D	-P和-D允许为下一个被装载程序设置一个起始点。-P和-D即时产生影响（不延期），但不影响已经装入的程序。形式是-P：⟨address⟩或者-D：⟨address⟩，其中⟨address⟩是当前打印输出基数中所希望的起始点（空缺基数是十六进的，-O设置八进的基数，-H设置十六进基数）。LINK-80作一空缺的-P：⟨Link Origin⟩，（即5200）。如果不给出-D，则对于每一模块，数据区装载在程序区之前。如果给出-D，则所有的数据区和公共区都从数据起始点开始装载，程序区从程序起始点开始装载。例如：

```
*-P：200，F00
Data        200        300
*-R
*-P：200    -D：400，F00
Data        400        480
Program     200        280
```

U	只要当前命令行一被解释，就立即列出程序的起始点和末尾点、数据区以及所有的不定整体。如果作了-D，则仅打印出程序信息。否则程序存在数据区。

M　　列出程序的起始点和末尾点以及数据区、所有的确定整体和它们的值、所有后边跟一星号的不定整体。如果已作了-D，则只打印出程序信息。否则，程序存在数据区。

S　　立即搜索在命令串前有-S 的文件名以满足任何不定整体。

例如：

*-M　　列出全部整体

*MYPROG, SUBROT, MYLIB-S

装载 MYPROG.REL 和 SUBROT.REL，然后搜索 MYLIB.REL 以满足任何仍然不定的整体。

* -G　　主程序开始执行

2. 连接例子

```
DOS READY
L80
*EXAMPL, EXMPL 1 -G
DATA        5200        52AC
〔5200        52AC〕
〔BEGIN  EXECUTION〕
      1792          14336
      14336         - 16383
      - 16383          14
          14          112
          112         896
DOS  READY
```

3. LINK 兼容的目标文件的格式

注意：本节是为希望知道 LINK-80 浮动目标文件的装配格式的用户提供的参考材料。大多数用户可以跳过这一节，因为它不包含操作这个包所必须的材料。

LINK 兼容的目标文件由 bit（位）流组成。bit 流内的各个区除了如下面所注释的以外，都不调整在字节边界上，为浮动目标文件所使用的 bit 流保持目标文件的尺寸是最小的，从而减少了盘的读／写次数。

装载项有两个基本类型：绝对和浮动。项的第一 bit 表示这种类型之一。如果第 1 bit 是 0，则跟着的 8 bit 作绝对字节装载。若第 1 bit 是 1，则下两个 bit 用来表示四类浮动项之一：

0 0　　专门的 LINK 项（见下面）

0 1　　程序相对的。在加进当前程序基数以后装载跟着的 16 bit

1 0　　数据相对的。在加进当前数据基数以后装载跟着的 16 bit

1 1　　公用相对的。在加进当前公用基数以后装载跟着的 16 bit

专门的 LINK 项的构成由 bit 流 100 后面跟着：

控制区 4 —bit。

任选的 A 区，它由两 bit 地址类型构成。这两 bit 地址除了 00 说明绝对地址外，其他均和上述的两 bit 区相同。

任选的 B 区，它由给定符号长度的 3 个 bit 和符号构成，符号的每个字符最多可达 8 个 bit。

专门的 LINK 项的一般表示为：

```
100        xxxx     YY   nn      ZZZ+Characters Of Symbol name
                    A 区                B      区
```

其中 xxxx　4 —bit 控制区（0 —15 以下）

yy　2 —bit 地址类型区

nn　16—bit 值

zzz　3 —bit 符号长度区

以下专门类型仅有 B 区：

0	入口符号（作为搜索的名称）
1	选择 COMMON 块
2	程序名称
3	请求搜索库
4	为进一步扩展而保留

以下专门的 LINK 项均有 A 区和 B 区：

5	定义 COMMON 尺寸
6	链接外部（A 是地址链的头，B 是外部符号的名称）
7	定义入口点（A 是地址，B 是名称）
8	为进一步扩展而保留

以下专门的 LINK 项仅有 A 区：

9	外部加位移，在执行以前 A 值立即加上当前指令计数器所开始的两个字节
10	定义数据区尺寸（A 是尺寸）
11	给 A 设置装载指令计数器
12	链接地址，A 是链接的头，用当前指令计数器来替换链中的所有入口，链中的最后一个入口是绝对 0 地址区
13	定义程序尺寸（A 是尺寸）
14	结束程序（给字节边界的强制）

以下特定的 LINK 项既没有 A 区也没有 B 区：

15	结束文件

4. LINK—80 错误信息

LINK—80 有如下错误信息：

? No Start Address	-G 开关已发出，但没有装载主程序
? Loading Error	输入的最后一个文件不是 LINK—80 目标文件的合

适格式

? Out of Memory　　没有足够的内存去装载程序

? Command Error　　不能识别的 LINK—80 命令

? ⟨file⟩ Not Found　　命令串中所给的 ⟨file⟩ 不存在

%2nd COMMON Larger／xxxxxx／

COMMON 块／xxxxxx／的第一个定义不是最大一个定义。重新排列装载顺序模块或改变 COMMON 块定义

%Mult.Def.Global yyyyyy

在装载过程中碰到整体（内部的）符号 yyyyyy 的定义多于 1 个

%Overlaying [Progam / Data] Area [, Start=xxxx / , Public=⟨symbol name⟩(xxxx) / , External=⟨symbolname⟩(xxxx)]

-D 或 -P 使得已装入的数据被破坏

? Intersecting [Program / Data] Area

程序区与数据区相交叉，地址或外部链接入口在这个交点中。因为在这个区域交叉点中，所以最终值不能转换为当前值

? Start Symbol——⟨name⟩——Undefined

在给出-E：或-G：之后，指定的符号没有定义

Origin [Above / Below] Loader Memory, More Anyway(YorN)?

在给出-E 或-G 之后，或者数据区或者程序区有处于装配程序外边的起始点或顶端（即装配程序起始点到了内存的顶部）。如果给一个 Y [CR] 则 LINK-80 就移动这个区并继续，如果什么也不给，则 LINK-80 就出口。在另一种情况下，如给出了-N，则内存映像就已经被贮存完毕

? Can't Save Object File

在存贮文件时出现盘错误

5. 程序中断信息

如果整体符号已经被装载的程序所确定，则 LINK-80 把第一个自由的存贮单元的地址存在这个被称为 $MEMRY 的整体符号中，$MEMRY 被放到数据区 + 1 的顶端。

注意，如果给了-D 而数据起始点又小于程序区，则用户必须确信有足够的空间使程序免遭破坏。这对具有 FORTRAN-80 盘驱动器的用户特别实际。因为 FORTRAN-80 是用 $MEMRY 去分配盘缓冲器和 FCB 的。

（陈品璨 译　周宝兴 校）

第五篇　EDIT-80 用户指南

一、EDIT-80 操　作

1. 引　言

EDIT-80 是一种行定位与字符定位的文本编辑器（要和 TRS-80 的 TRSDOS 一起使用）。EDIT-80 命令是简单易懂的，而且是强有力的，足以适应用户的大多数要求。对于初学者或者对于只需要粗略使用 EDIT-80 的人来说，这篇资料的前四节就已包含了完成相当广泛的编辑范围的所有必要信息，其余几节描述 EDIT-80 的提高方面，借以提供用户更复杂的技术。

2. 运行 EDIT-80

为了运行 EDIT-80，在 TRSDOS 命令级键入

EDIT

则 EDIT—80 会响应

FILE:

要求给出文件名，输入你的文件名称。文件名称应使用 TRSDOS 文件名格式：

文件名〔/extension〕〔•password〕〔•drive#〕

如果文件名属于已经存在的某一文件，则在敲入文件名后跟着敲 ENTER 键，EDIT-80 就会读入这个文件。如果这个文件没有行号，EDIT-80 会给加上：从行号 100 开始，增量为 100。在 EDIT-80 打印出

```
Version x.x
Copyright 1977, 78 (c) by Microsoft
Created: xxxx
xxxxBytes free
*
```

后，它就处在命令级了，用指示符 * 号表示。EDIT-80 的所有命令都在 * 号后输入。

如果文件名属于一个新建立的文件，则在键入文件名后跟着敲入 BREAK 键。EDIT—80 将回答信息

```
Creating
Version x.x
Copyright 1977, 78 (c) by Microsoft
Created: xxxx
xxxxBytes free
*
```

接下来输入命令 I（见进一步描述 I 命令的第二、1 节）。EDIT-80 将印出第一行的行号 00100，后跟一个表格符（tab）。

```
*I
00100
```

现在，你就可以键入你文件的第一行。一行最多由 255 个字符组成，并用 ENTER 键终止。在每一行输入后，EDIT-80 会印出下一行的行号，增量是 100。这是个常设增量（有各种命令可以改变这个常设增量——见第二节），在你的 EDIT—80 文件中可用的行号是 00000 至 99999。

注意：微软件产生诸如 TRS-80 FORTRAN 和 MACRO—80 所有包含 EDIT—80 行号的支援输入文件。

如果在打入或编辑一行时，印出一个错误，则用删除键(←)去删除不正确的字符。如果在敲一行时，你希望擦去此整个一行并重新开始，则敲 SHIFT ←键。

当你希望停止键入程序行并返回到命令级时，则在下一个可用行号后按 BREAK 键。

3. 结束编辑

为了由 EDIT-80 出口，打入出口命令：

```
*E
```

这个出口命令把被编辑文件以建立文件时所用的文件名写到盘上。自这个文件建立后的编辑期间，需要用出口命令指定一个文件名。见第六、1 节。

如果不要将被编辑文件写到盘上而从 EDIT—80 出口，则打入退出命令：

```
*Q
```

在执行这个退出命令以后，在编辑期间所打入的所有变更全部丢失。

4. 行号和范围

EDIT—80 的大多数命令需要引进行号或者行号范围。某一行号由数目本身来指定（前空可不必敲）或者由 EDIT—80 认作行号的三个特殊字符之一来指定。这些特殊字符是：

.（句点）	相当于当前行
↑（上箭头）	相当于第一行
*（星号）	相当于最后一行

范围可以用两个方法之一来指定：

1） 用一个冒号。如下表示

```
200 : 1000
```

系指行号 200 至行号 1000 之间（包括两者）的所有行。如果行号 200 和 1000 不存在，则范围将从大于 200 的第一行号开始，由小于 1000 的最后一行行号结束。

2） 用一个警叹号。如下表示

```
200! 3
```

系指行号 200 开始的三行范围。如果行号 200 不存在，则 200! 3 系指 200 以后第一行开始

的三行范围。

这里有一些说明行号和范围的例子（和打印命令一起表明）：

P.：2000	打印范围从当前行开始，第 2000 行结束
P500	打印第 500 行
P.	打印当前行
P.！15	打印范围从当前行开始至下 15 行以后结束
P↑：1500	打印范围从第一行开始至第 1500 行结束
P↑：*	打印整个文件

有关范围说明的更广泛的例子见附录 3 。

5. 格式表示

通过本资料，给使用者以 EDIT-80 命令一般格式的指导。这些格式遵循如下惯例：

1) 方括号内的项是任选的。

2) 大写字母的项必须按所示格式输入。

3) 括在尖括号内的小写字母项要由用户提供：

〈position〉	提供任意行号（至多五个数字）或者“.”，“↑”或“*”
〈range〉	提供任意〈position〉或任意的〈range〉 〈range〉=〈position〉：〈position〉或〈position〉！〈number〉
〈inc〉	提供一非零整数以用作行号之间的增量
〈filename〉	提供如一、2节中所描述的任意合法的 TRSDOS 文件名称

4) 必须按所示格式那样包含标点符号。

5) 由垂直线所分隔的项是互相排斥的，任选其一。

6) BREAK 属于中断键，用 $ 来响应。如果在格式表示中你看到一个 $，那么，这就是中断键。

7) 在任何命令格式中，空格和表格符都是无意义的。除了在行号内或文件名称内以外。

8) 在下面划线的项由 EDIT-80 打印出。

二、行间编辑

通过打印、插入、删除或替换整个的行或一组行来编辑某一文件称作行间编辑。本节描述用于这些功能的命令。

1. 插入命令

插入命令用于将文本行插入到文件中去。在插入方式期间，EDIT-80 为你印出每一个行号。插入命令的格式是：

I[〈position〉[，〈inc〉|；〈inc〉]]

在〈position〉处开始插入行，继续到敲 BREAK 键止或直到该文件中的可用空间用完为止（在这种情况下，EDIT—80 都返回到命令级）。

如果命令中不包含〈inc〉，则空缺增量就是常设增量，〈inc〉指定一个新的增量，然后将它建立为常设增量。;〈inc〉指定一个用于当前命令的暂时增量，但不改变常设增量。

如果插入命令中不提供宗量（I ENTER），那么，在最后一个插入命令被终止的地方，用最后一个暂时增量恢复插入。如果仅提供〈position〉即（I〈position〉 ENTER ），则用常设增量。

EDIT—80 不允许插入已经存在的某一行。如果〈position〉是已经存在的行号，则命令I〈position〉会加一常设增量(或者是暂时增量——如果已经指定了的话)到〈position〉上，而允许在行号〈position〉+〈inc〉插入。如果行〈position〉+〈inc〉已经存在，或者在〈position〉和〈position〉+〈inc〉之间存在该行号，则会打印出错误信息。

如果没有开始一个新的逻辑行，则换行键（↓）可用于开始一个新的自然行，这是与微软件BASIC 源文件兼容的。

有一个使用插入命令的例子：

```
*I7740, 10
07740                K=K+1
07750                GO TO 400
07760     $
*
```

注意，插入用 BREAK 键终止。BREAK 键可以在插入的最后一行的末尾敲入（代替 ENTER ）或者在下一行的开始敲入。如果 BREAK 键是那一行上打入的第一个键，则不存贮该行。

2. 删除命令

删除命令会删去文件中的一行或一些行。命令的格式是

D〈range〉

在删除命令执行后，当前行（“.”）放到被规定为删除范围的第一行。

删除命令的例子：

D7000	删除第 7000 行
D.	删除当前行
D200：900	删除第 200 行至 900 行
D2000：*	删除第 2000 行至最后一行的所有行

3. 替换命令

替换命令兼有删除命令和插入命令的效能。命令格式是

R〈range〉[，〈inc〉|；〈inc〉]

替换命令删除在〈range〉内的所有行，允许用户去打入新的文本，宛如已发出了插入命令一样（EDIT—80 打印出行号）。

行号之间增量的挑选和插入命令的那些规定相同（见第 1 小节）。

有一个使用替换命令的例子：

```
*R500, 600, 50
00500                DO 80 I=1,7
00550                Y(I)=ALOG (Y(I))
00600    80          CONTINUE
*
```

在上述例子中，在 500 至 600 范围内的所有行都被删除，并用暂时增量为 50 的三个新行(500，550 和 600)来替换。因为 EDIT—80 已没有足够的空间来建立行 650，因此插入自动停止。

4. 打印命令

打印命令在终端打印出程序行，其命令格式是：

P〈range〉

打印命令的例子：

P.,700	打印当前行到第 700 行间的所有行
P800,*	打印第 800 行至文件末尾间的所有行

在命令级按 [line feed] 键(↓)，会把当前行后面的一行印出。在命令级按 [BREAK] 键，会把当前行前面的一行印出，打入 P [ENTER] 会把下 20 行打印出来。

5. 列清单命令

列清单命令

L〈range〉

除了输出到行式打印机外其他均和打印命令相同。

6. 重编行号命令

重编行号命令是把文本行重新编号。为了插入，或者只是为了编制文件的行号，你可以去重新编行号“形成空位”。重编行号命令的格式为

N〔〈start〉〕〔,〈inc〉|;〈inc〉〕〔=〈range〉〕

式中

(1) 〈start〉是新的行序列的第一个行号。如果省去〈start〉而包含〈range〉，则将〈start〉置成〈range〉的第一行。如果〈start〉和〈range〉都省略，而包含〈inc〉，则〈start〉置〈inc〉的值。如果〈start〉省略而包含〈inc〉，且〈range〉仅指定一个页数（即=/2），则〈start〉也置成该页上的〈inc〉值。如果〈start〉、〈range〉和〈inc〉都省略，则〈start〉置常设增量值。

(2) 〈inc〉是新行序列的行号之间的增量。增量的选择和插入命令中所描述的那些相同（第 1 小节）。

(3) 〈range〉是重新编号的行号的范围。如果〈range〉省去，则整个的文件被重新编号。

如果当前行被重新编号，则把“.”重新放到相同的物理行上去。

如果重编行号命令引起行号超出行序列界限，或者如果 EDIT—80 使用给定增量不能适应所有行，则将得到“out of order”的错误信息。

由于 EDIT—80 执行重编行号命令须要有内部存储量，所以，想把一个很大的文件重编行号会引起“Insufficient memory”错误。如果发生这种情况，则只有把文件的小部分重

编行号，然后把它写到磁盘上，再把另一部分重编行号，写到磁盘上，以此类推。（见写命令，六、3 节）。

重编行号命令的例子：

N7000；100＝200：1000	行 200 至 1000 被重编行号，在 7000开始增量为100
N，10＝400：*	行 400 行至末尾被重编行号，以 400 开始增量为 10
N9000＝10000：*	在 9000 开始以常设增量把 10000 至末尾的所有行 重编行号
N，100	用增量 100 把整个文件重编行号
N，5＝2350！10	这个命令可用于产生空位以便插入，它通过压缩由 2350 行开始的 10 行而实现

三、行内编辑——变更方式

迄今已讨论的行间编辑命令是让你通过插入、删除或替换来编辑整行。然而，许多编辑情况需要改变某一存在的行而不必重新打入这一行。不重新敲入而编辑一行，称为行内编辑，它在变更方式中进行。

1. 变更命令

变更命令用于进入变更方式。命令格式为：

A〈range〉

在变更方式，EDIT—80 印出被变更行的行号并等待变更方式子命令。

2. 变更方式子命令

变更方式子命令用于在行内移动光标；搜索文本；或者插入、删除或替换文本。这些子命令在终端上没有响应。

许多变更方式子命令都可在其前放一整数，使命令执行数次（当不指定整数时，空缺数总是 1）。在许多情况下，整个命令也可冠以一个负号，借以改变命令动作的正常方向。例如：

D	删除下一个字符
6D	删除下六个字符
-D	删除最后一个字符
-12D	删除最后的十二个字符

每一个变更方式子命令描述如下。子命令的摘要在附录 2 中给出。

注意，在下面的描述中，$ 表示 BREAK，〈ch〉表示任一字符，〈text〉表示任意长度的字符串，而 i 表示任一整数。

3. 光标位置

下列命令或终端键用于改变行中光标的位置。光标的所在位置称为“当前位置”。

i [space] 空过字符。i [space] 把光标向右移动 i 个字符。-i [space] 把光标向左移动 i 个字符。这些空过的字符就被打印出来。

→ 移动光标到行的末尾。如果前面有一负号，则光标移动到行的开头。

L 打印出行的剩余部分，而光标的位置又被放到行的开头。继续处理下一个变更方式子命令。

P 打印行的剩余部分并将光标收回到当前位置。继续处理下一个变更方式子命令。

W 将光标移动到下一个字的开始。字定义为文字、数字、“.”、“$”或“%”的连续集合。iw 使光标向前推进 i 个字、-iw 使光标向左移回 i 个字。

4. 插入文本

I 插入文本。I⟨text⟩ $ 在当前位置开始插入给定的文本。注意，文本后必须跟 [BREAK] 或 [ENTER]。

B 在当前位置插入空格(blank)，B 命令可以在前面冠以整数以便插入许多空格。空格仅能插入到光标的右面。

G 插入字符，iG ⟨ch⟩插入 i 个⟨ch⟩。

X 行的延伸，X 子命令打印出行的剩余部分，进到插入方式，并让你在行的末尾插入文本。-X 子命令移到行的开头并进到插入方式。（不要忘记，你插入结束时应该用 [BREAK] 或 [ENTER]）

5. 删除文本

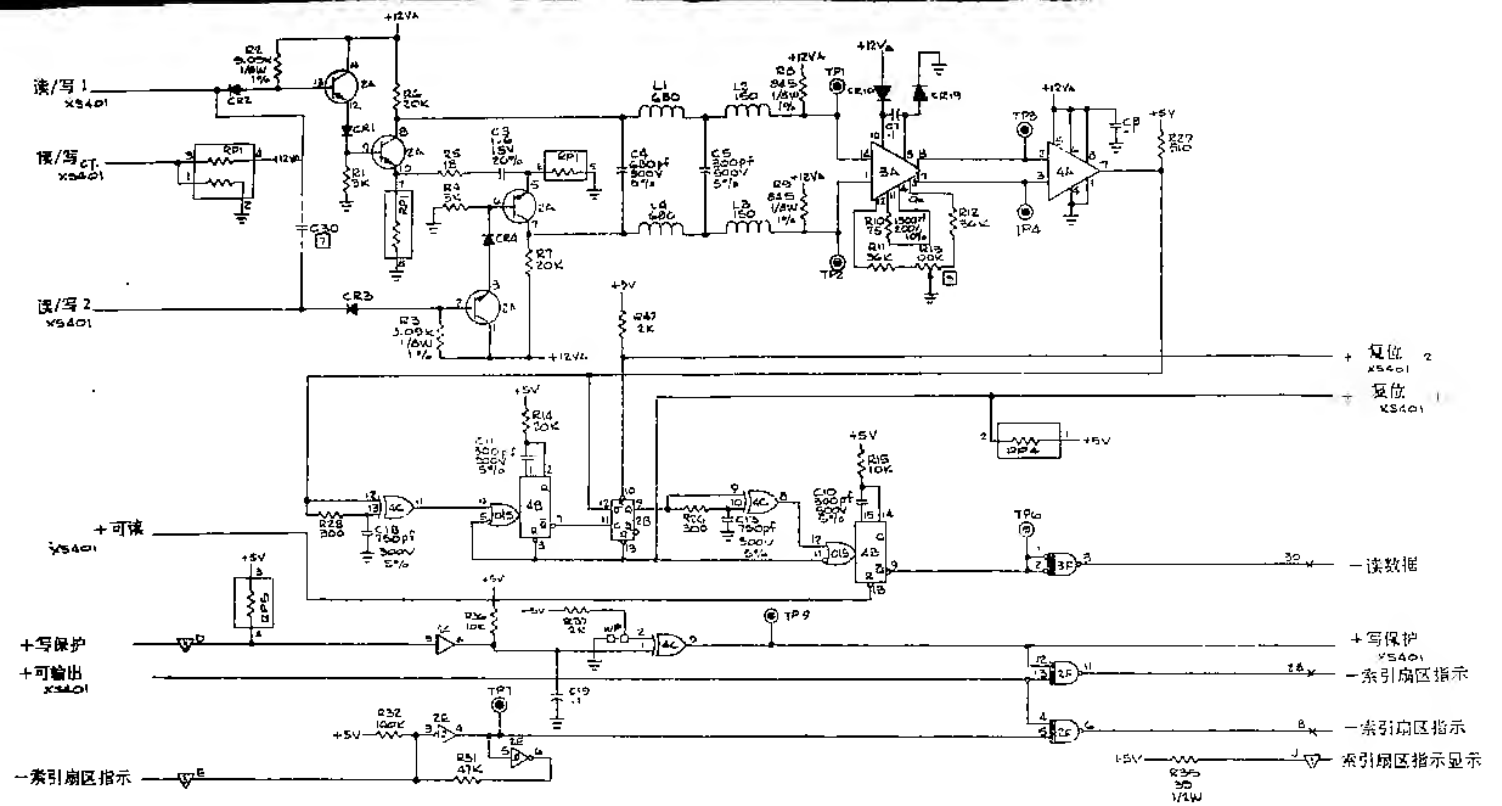
D 在当前位置删除字符，iD 删除在当前位置开始的 i 个字符，-iD 从当前位置向左删除 i 个字符。被删除的字符用两个惊叹号围起来。

← 后退箭头键也可以用于删除字符。当前位置左边的那个字符立即被删除。i [back-arrow] 等价于-iD。

H 删掉（砍掉）光标右边的（如果敲入-H，则为光标左边的）行的剩余部分，并进入插入方式。下一个插入宛如已敲入 I 命令那样进行。

K 删除（除掉）字符，K⟨ch⟩删除⟨ch⟩以前但不包括⟨ch⟩的所有字符。iK⟨ch⟩删除⟨ch⟩第 i 次出现以前的所有字符。-iK⟨ch⟩删除包括⟨ch⟩在内的、⟨ch⟩第 i 次出现以前的所有字符。如果⟨ch⟩没有找到，则命令不执行。

O 删除（清除）文本。O⟨text⟩ $ 删除⟨text⟩下一次出现以前但不包括⟨text⟩的所有文本。iO⟨text⟩ $ 删除⟨text⟩第 i 次出现



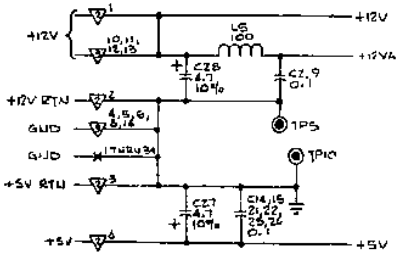
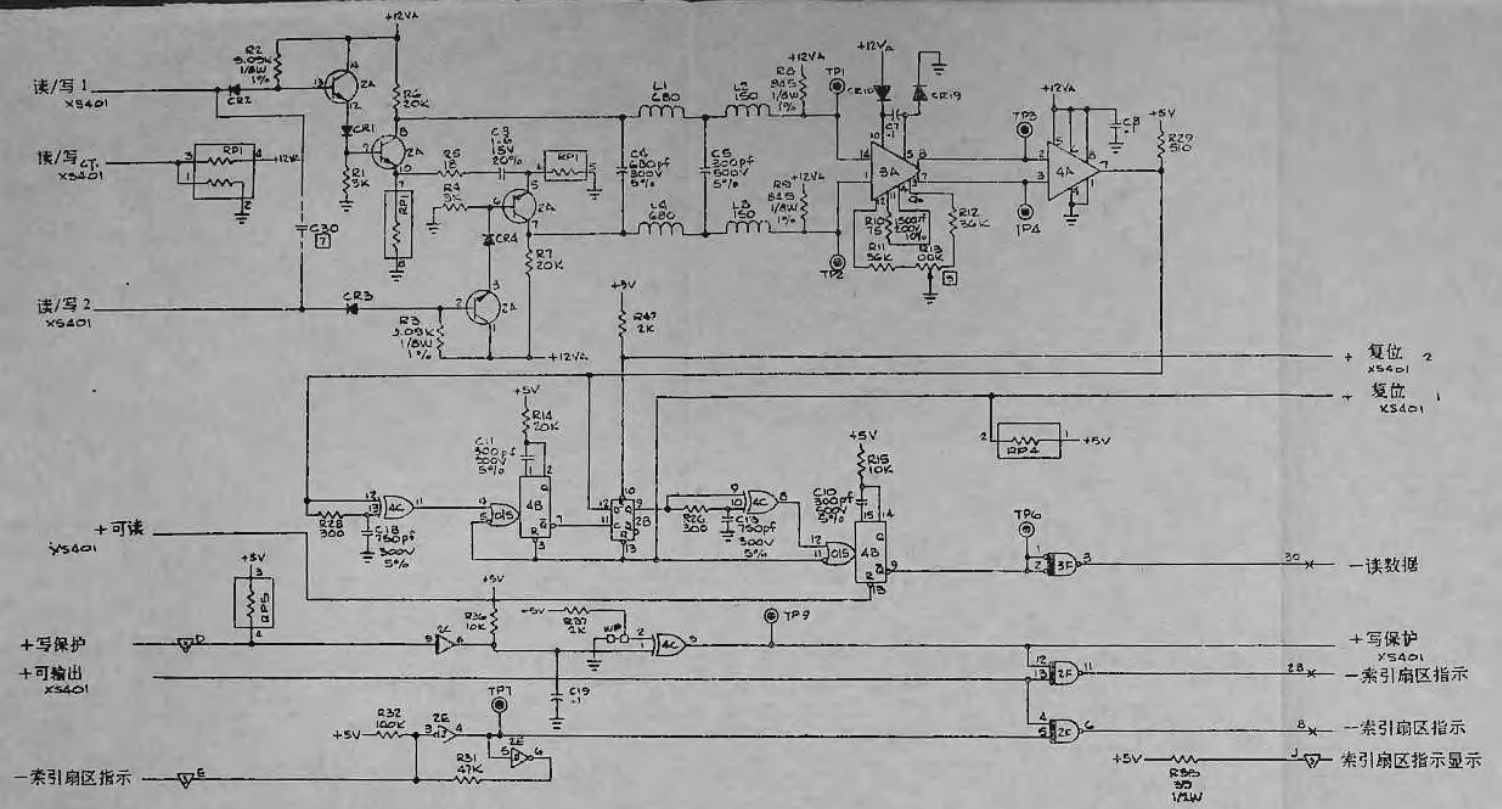


图9 控 制 逻 辑 图

控制逻辑图说明（除指定外，均按此例）

1. 全部电容：微法拉，50V，+80，−20%
2. 全部二极管都是1N4148
3. 全部电感：微亨，10%
4. 全部电阻：欧姆，1/4瓦，5%
5. □——□表示可以任选的分流器。
6. 关指J1，——2——指J2，——3——指J3，——4——指J4
7. 原件没有装上。
8. ID的第4脚接地。
9. R13的值可以是50K



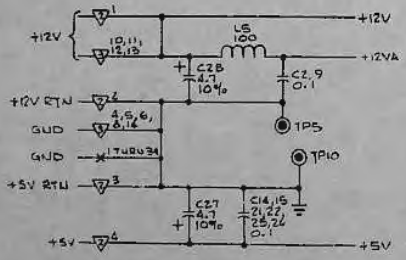


图9 控 制 逻 辑 图

控制逻辑图说明（除指定外，均按此例）

1. 全部电容：微法拉，50V，+80，−20%
2. 全部二极管都是1N4748
3. 全部电感：微亨，10%
4. 全部电阻：欧姆，1/4瓦，5%
5. □——□表示可以任选的分流器。
6. ※指J1，——2——指J2，——3——指J3，——4——指J4
7. 原件没有装上。
8. ID的第4脚接地。
9. R13的值可以是50K

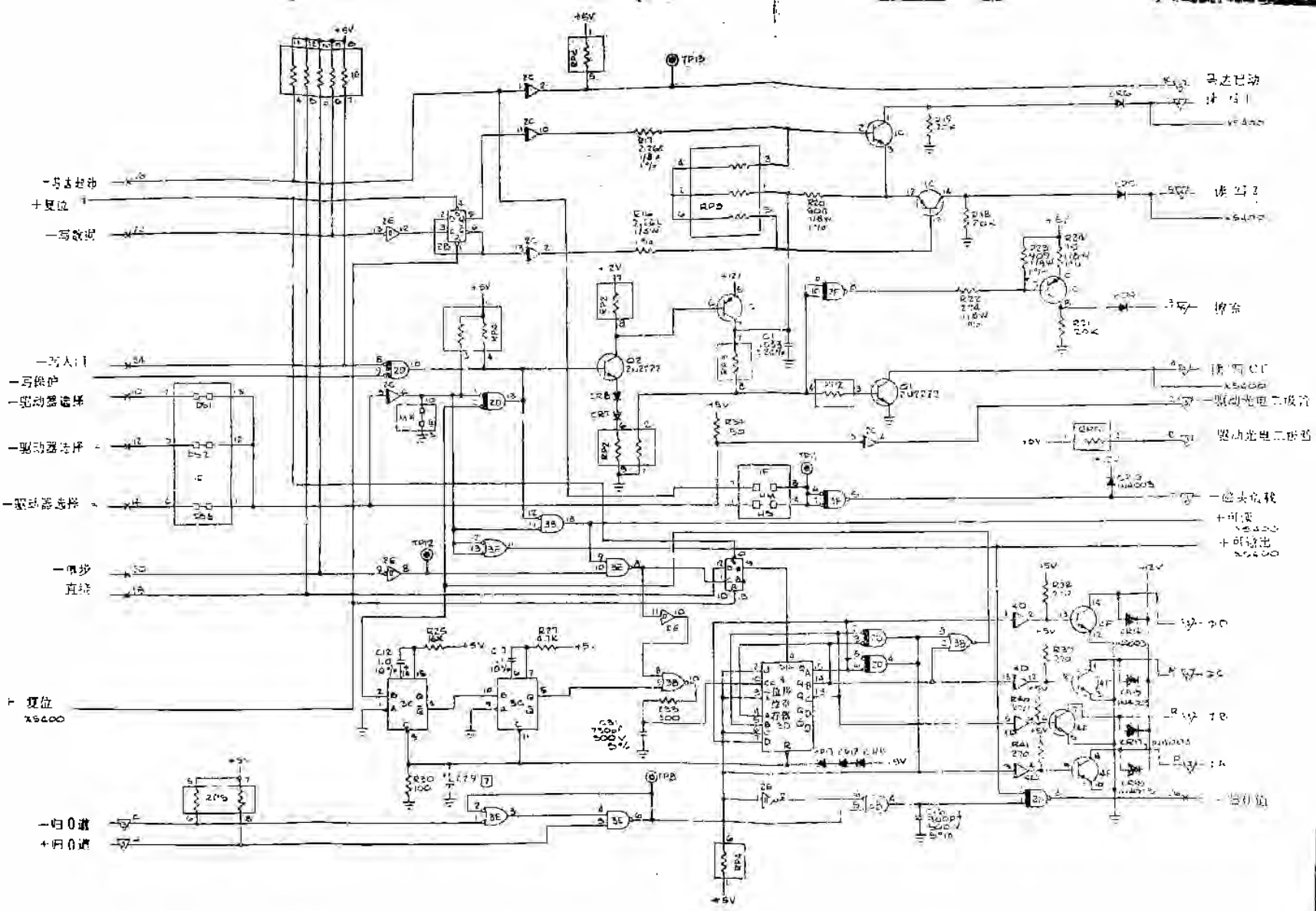


图10 读/写逻辑图

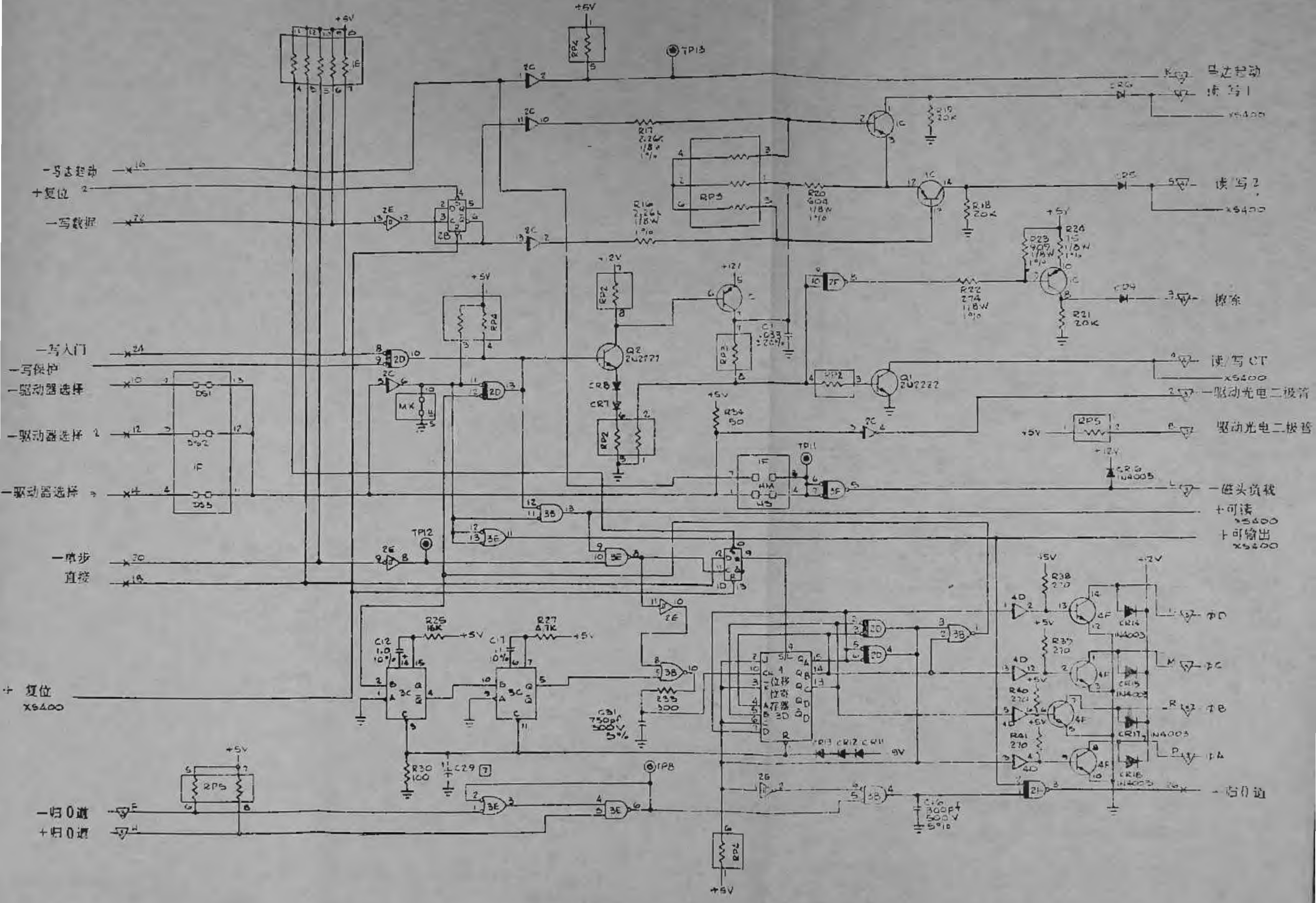

马达起动
读写 1
读写 2
擦除
读/写 CT
一驱动光电二极管
驱动光电二极管
一磁头负载
十可读
十可输出
一归0道
十复位
一写数据
一写入门
一写保护
一驱动器选择
一单步
直接
十 复位
X5400
十归0道

图 10 读／写逻辑图

以前的所有文本。-iO⟨text⟩$ 删除包括⟨text⟩在内的、⟨text⟩第 i 次出现以前的所有字符。

T　删掉（截去）光标右边的（或者，如果是敲入 -T，则光标左边的）行的剩余部分，并退出变更方式。

Z　删除字，iZ 删除下面 i 个字。-iZ 删除光标左边的 i 个字。

6. 替换文本

R　替换文本。iR⟨text⟩$ 删除下面 i 个字符并用 ⟨text⟩ 替换它们。-iR⟨texs⟩$ 替换光标左面的文本。被删除的字符夹在两个惊叹号之间来响应。

C　每次改变一个字符。C⟨ch⟩ 改变下一个字符为 ⟨ch⟩，仅有新的字符被响应。iC 可以跟 i 个字符以改变许多个字符；或者也可以跟少于 i 个字符而用 [BREAK] 终止，在这种情况下，剩余的字符将不改变。-iC 相当于作 i [back arrow]，然后再作 iC。i [back arrow] 夹在两个惊叹号间响应。

7. 寻找文本

S　检索字符，S⟨ch⟩在当前位置以后搜索下一次出现的⟨ch⟩，并将光标定位在该字符前。iS⟨ch⟩搜索第i次出现的 ⟨ch⟩，-S⟨ch⟩和-iS⟨ch⟩搜索（第 i 次）出现前的⟨ch⟩，而光标立即定位在其前面。在光标位置处的字符不在搜索之列。如果 ⟨ch⟩ 没找到，则命令被忽略。

F　寻找文本，F⟨text⟩$ 找寻下一次出现的⟨text⟩并把光标定位在串的开头，iF⟨text⟩$ 寻找第 i 次出现的⟨text⟩，-F⟨text⟩$ 和 -iF⟨text⟩$ 寻找（第 i 次）出现以前的⟨text⟩并把光标定位在其前。

8. 结束和重新开始变更方式

[CR]　回车。打印行的剩余部分，输入改变和终止那一行的变更。

A　和回车相同。

E　输入改变和终止那一行的变更，但不打印该行的剩余部分。

N　恢复原来的行（但不贮存变更内容），或者移到下一行（如果 A⟨range⟩命令仍在进行的话），或者返回到命令级。

Q　恢复原来的行（不存变更内容），出口（退出）变更方式，并返回到命令级。

[Shift]←　恢复原来的行，停留在变更方式并把光标重新放到行的开头。用 ↑Y 作响应。

9. 扩展命令

扩展命令在命令级发出并用来扩展行。命令格式为

X〈range〉

X 命令的作用等价于打入 A 命令，后面跟一个 X 子命令。在进入 X 命令后，可在行的末尾继续敲入被插入文本。不要忘记，你现在是在变更方式，因此可以使用任何的变更方式子命令，直至按 BREAK 键为止。

扩展命令对于在汇编语言程序里安放注解特别有用。

四、寻找和替换命令

当需要改变文本的某一部分时，往往不是立即知道这部分文本位于文件的何处。即使文件的清单在手边，但要去扫描清单以便找到必须改变的特定项的行号，这也是一件很讨厌的事。

EDIT—80 寻找和替换命令允许用户去很快地找出文本并作必要的修改。

1. 寻找命令

寻找命令找出文件中某一给定的文本串并打印出包含该串的行。命令格式是：

F[〈range〉][,〈limit〉][ENTER]|$〈string〉$

式中 $ 为换码键，〈limit〉是包含要寻找的〈string〉的行数。零极限会寻找出出现〈string〉的所有行。下列规则应用于寻找命令格式：

(1) 如果省略 $〈string〉$ 则使用寻找命令中给出的最后一串。

(2) 如果省略〈limit〉而包括 $〈string〉$，则假设〈limit〉为 1 。

(3) 如果〈limit〉与 $〈string〉$ 都省略，则假设为以前的〈limit〉。

(4) 如果省略〈range〉而包含 $〈string〉$，则使用以前的寻找命令的整个范围。

(5) 如果〈range〉和 $〈string〉$ 都省略，则从最后一次找到的那一行开始继续搜索以前的串。

如果搜索不成功，则打印出错误信息。

有一个使用寻找命令进行编辑的例子：

```
*F↑:*$WHI (I) $          寻找包含 WHI (I) 的第一行
01100  WHI (I) =0         打印出行 1100
*F [ENTER]               寻找下一行
01400 IF (P.GT.WHI(I))WHI(I)=2P  打印行 1400
*A.                      捕捉该行中的错误，变更它
01400
      ⋮
*F, 2$WLO (I) $          寻找文件中包含 WLO (I)
01200 WLO (I) =9999      的头两行（范围仍是.:*）
01500 IF(P.LT.WLO(I))WLO(I)=P  打印出行 1200 和 1500
*A.                      变更行 1500
01500
```

```
      ⋮
*F.: *$AVG$                          寻找文件中包含 AVG 的第一行
Search fails                         没有包含这样串的行
*F$MEAN$                             尝试改寻 MEAN 串
03700 MEAN=SUM/40                    打印出行 3700
*F, 0                                寻找包含 MEAN 的所有其他行
04200 IF (P.GT.MEAN) M=M+1           (搜索从 3700 行以后开始)
06700 WRITE (6,170) MEAN,M           找到两行 (4200 和 6700)
*A 4200                              变更 4200，等等
04200
      ⋮
```

2. 替换命令

替换命令找出一给定串，用一新的串替换它，并打印出新的行。此命令格式为：

S〔〈range〉〕〔〈limit〉〕[ENTER] | $〈old string〉$〈new string〉$

式中$就是[BREAK]，〈limit〉是用〈new string〉替换〈old string〉的行数，零极限将把所有出现的〈old string〉均用〈new string〉来替换。〈new string〉可以是空串。

下列规则用于替换命令格式：

(1) 如果省略$〈old string〉$〈new string〉$，则使用在最后一次替换命令中所给出的串。

(2) 如果省略〈limit〉而包含$〈old string〉$〈new string〉$，则假设〈limit〉为零。

(3) 如果〈limit〉和$〈old string〉$〈new string〉$都省略，则假设为以前的〈limit〉。

(4) 如果省略〈range〉而包含$〈old string〉$〈new string〉$，则使用以前的替换命令中的整个范围。

(5) 如果〈range〉和$〈old string〉$〈new string〉$都省略，则替换从最后一次不使用替换的地方继续进行。

如果〈old string〉没有找到，则会打印出错误信息。

例如：

```
*S↑: 5000$ALPHA$BETA$                从第 1 行至第 5000
00950  BETA(A)=ABS(1.-LST(K))         行用 BETA 替换
01750  WRITE(6,400)BETA(K)           所有出现的 ALPHA
04100  IF(BETA(K).LT.0)GOTO 9000
```

五、分 页

有可能需要把 EDIT-80 文件分成几段，这个段称为页，它是用页标记来分开的。文件的第一页总是 Page 1，EDIT-80 总是从多页文件的 Page 1 进入命令级。每一个顺序页都以页标记开始并顺序编号。在任一给定页上，可以使用行号的整个范围（00000 至 99999

或任一部分）。

当读入某一文件时，如果碰到一换页符，则在文件的该点进入一页标记。如果 EDIT-80 碰到一行号，它小于先前的行号，则会自动插入一页标记以便适当的行号顺序仍可以继续使用。当 EDIT-80 写一文件到磁盘时，就会输出带有每页标记的换页符。从而，当开列文件清单时，文件的每一新页都在新的物理页开始。

1. 指定页号

在单页文件中，只需要行号来表示〈position〉，在多页文件中，为决定〈position〉，EDIT—80 必须知道页号以及行号，也就是说，〈position〉由

〈line〉[/〈page〉]

来表示，式中

〈line〉是“.”，“↑”，“*”或至多 5 个数字的数。

〈page〉是“.”，“↑”，“*”或至多 5 个数字的数。

当指定某一页时，字符“.”，“↑”和“*”分别指当前页、第一页和最后一页。如果省略〈page〉，则假设是当前页。

其次，在多页文件中，〈range〉可以由

〈position〉:〈position〉

或

〈position〉!〈number〉

来表示，也可以包含页号。如果从范围的第一行号中省略页号，则假设为当前页。如果在范围的第二行号中省略页号，则假设在和范围的第一行号相同的页上。

在此有一些包括页号说明的行号与范围的例子：

100/2:*/*	从第 2 页的第 100 行至最后一页的最后一行
100/2:*	第二页的第 100 行至该页末尾
100:*/5	当前页的第 100 行至第 5 页的最后一行
100/*	最后一页的第 100 行
100/.:*/3	当前页的第 100 行至第 3 页的最后一行

范围说明的更多例子见附录 3。

2. 插入页标记

页标记可以由用户任意地插入到文件中。为了插入页标记，使用标记命令，格式是：

M〈position〉

页标记立即插在〈position〉之后，〈position〉必须存在，否则会打印出错误信息。

在标记命令执行后，仍保留当前行基准（“.”）。就是说，如果〈position〉在“.”之前，则“.”会移到下一页并仍指向相同的物理行。

3. 删除页标记

页标记用 K（kill）命令删除，命令格式为：

K/〈page〉

K 命令删除在〈page〉后的页标记。例如，在 4 页文件中，K/2 会删除第 2 个页标记（第 3 页开始的页标记），并且将页编号成1，2，3。在 K/〈page〉命令执行以前，在〈page〉上的最后一行的行号必须低于在〈page〉+1 上的第一行行号。

4. 开始命令

用开始命令可返回到页的开头。开始命令格式是：

B〔/〈page〉〕

如果省略〈page〉，则 B 命令会返回到第一页的开头。

5. 其他命令和页标记

(1) 页边界交叉的删除命令会删除范围内的所有行，但不删除页标记。

(2) 不是打印当前页的打印命令会在打印命令中指定的第一行之前先打印出新的页号。

(3) 当输出是作列表命令时，换页符会和每一个页标记一起打印出，而页号会在每一页上打印出来。

(4)用惊叹号指定的范围可以跨页。

(5) 如果在重编行号命令中指定的范围跨页，则在每一新页开始重编行号，第一行号等于增量。其次，在重编行号命令中，〈start〉和〈range〉的第一行必须在同页上。

六、由 EDIT—80 出口

第一节第 3 小节介绍了从 EDIT—80 出口的出口命令和退出命令。这两个命令在本节中将更全面地描述。也将叙述写命令等其他命令。

1. 出口命令

出口命令用于写一个文件到磁盘并返回到 TRSDOS，命令的格式是：

E〔〈filename〉〕〔-〈switch〉〕

被编辑的文件以〈filename〉为名称存在磁盘上。当第一次出口新文件时，〈filename〉可以省略（在这种情况下，赋予打开的文件名）。否则，每一出口需要一新的文件名。以前文件当作备用。

选择项〈switch〉控制输出的格式（见第 5 小节）。

2. 退出命令

退出命令用于返回到 TRSDOS 而不将被编辑文件写到磁盘上。为了退出编辑，简单地打入：

Q

在退出命令以后，在编辑期间所进入的全部改变均丧失。

3. 写命令

写命令把被编辑的文本写到磁盘上，然后返回到 EDIT—80 命令级。它不从编辑器出

口，而且文件的当前位置也不变。命令格式为：

W〔⟨filename⟩〕〔-⟨switch⟩〕

在第一次写一个新文件时，不需要文件名称。然而，在以后所有的写和出口命令中，是需要文件名称的。

选择项⟨switch⟩控制输出的格式（见第 5 小节）。

4. 索引文件

当为了编辑而读入一文件时，EDIT-80 产生它所需要的每一块磁盘文件的信息。对于一个小的文件，这个信息只需要很少时间就能产生，并可以随时读入文件。然而，对于一个较大的文件（5K 或者更大），读入文件所需要的时间延迟就变得很重要了。因此，当 EDIT-80 存贮一个有 42 个记录或更多一些记录的文件到磁盘上时，它也存贮一个小文件到磁盘上，这个小文件是从文本文件中分离出来的，它包含有文本文件所需要的信息。

这个小文件称为索引文件，而它能比文本文件更快地读入。EDIT—80 存贮到磁盘上的索引文件的文件名和文本文件名相同（不包含口令—passwords），仅在其后缀的头两个字母前面有一 Z。例如，如果文件称为 F00/MAC.SAM，则索引文件称为 F00/ZMA。

当请求 EDIT-80 编辑某一文件时，它首先检验索引文件。如果索引文件存在，则 EDIT-80 读入代替文本的索引文件。如果文本文件已被另一个编辑者修改或改变并已存入 BASIC 中，则必须注意，在用 EDIT-80 再次编辑文本文件之前用户必须删除索引文件。如果不删除索引文件，则 EDIT-80 将得到关于文本文件的无意义的信息。

5. 参数

当读入某一文件时，EDIT—80 期待它是以它自己的表示法表示的。如果文件以另一个表示法出现，则 EDIT-80 会加上行号并试图将这个文件转换成 EDIT-80 标准格式。然而，如果有适当的开关附加在输入文件名上，则有几种 EDIT—80 可以按受的其他表示法。开关前面总有一条短划线（—）：

filename〔/ext〕〔.password〕〔：drive#〕〔-switch〕

例如：FOO/BAS.SAM—BASIC

1) BASIC 开关

如果将 BASIC 开关加到输入文件名中，则 EDIT-80 会用如下算法读入文件：

(1) 所有的引导空格和表格符都从每一行中除去。

(2) 第一个非空格字符必须是一数字。

(3) 开头 1 至 5 个数字被转换为行号。开头的数字大于 5 个就构成一个致命性错误。

(4) 如果第一个非数字不是空格或表格符，则插入一个表格符。如果第一个非数字是一空格，则把它换成表格符。如果第一个非数字是表格符，则单把它留下。

(5) 关于输出，如果已经选择 UNSEQ 开关（见下一小节），则行号中的引导零被抑制，而表格符被转换为空格。

因为 BASIC 利用行号去控制程序执行的顺序，因此 BASIC 的用户在用 N 命令去重新编行号时应当小心。微软件 BASIC 会忽略 EDIT-80 文件的页标记，所以 BASIC 文件可以有多页。然而可以保证，在程序中不会有重复的行号出现。

2) SEQ 和 UNSEQ 开关

如果 SEQ 开关被加到输入文件名上，则 EDIT—80 会用和 BASIC 开关相同的算法去解释文本文件。然而，当文件输出时，则会以标准的 EDIT-80 格式输出，除非 UNSEQ 开关被加到输出文件名上。

在输入上的 UNSEQ 开关告诉 EDIT-80 将它自己的行号加到输入的文件上，而不管它像什么。如果输入文件在行的开始有数字，而其高位 (bit) 又不解释为行号，则必须使用这个开关。

关于输出，为输出非标准文件，必须指定 UNSEQ 开关（如果还没有的话）。这就是说，如果在输入中指定 BASIC 而在输出中指定 UNSEQ，则输出文件没有行号也不尾随有表格符。如果在输入中指定 UNSEQ 开关而用户希望输出标准文件，则在输出中的 SEQ 开关会取代 UNSEQ 开关。

附录 1. 命令摘要

命　令	格式和描述	
Alter	A⟨range⟩	进入变更方式
Begin	B[⟨page⟩]	移动到⟨page⟩的开始，空缺是 page 1
Delete	D⟨range⟩	删除行
Exit	E[⟨filename⟩][-⟨switch⟩]	将被编辑文本写入磁盘并返回编辑器
Find	F[⟨range⟩][,⟨limit⟩][ENTER] | $⟨string⟩$	寻找⟨string⟩的出现
Insert	I[⟨position⟩][,⟨inc⟩|;⟨inc⟩]	从⟨position⟩处开始插入行，增量用⟨inc⟩，如没有宗量，则继续先前的插入命令
Kill	K/⟨page⟩	删除在⟨page⟩末的页标记
List	L⟨range⟩	在行印机上打印行
Mark	M⟨position⟩	在⟨position⟩后插入一个页标记
Number	N[⟨start⟩][,⟨inc⟩|;⟨inc⟩][=⟨range⟩]	重编在⟨range⟩内的行号，从⟨start⟩处开始，增量为⟨inc⟩
Print	P[⟨range⟩]	在终端上打印行。如无宗量，则打印下 20 行
Quit	Q	退出编辑器而不把被编辑文本写到磁盘上
Replace	R⟨range⟩[,⟨inc⟩|;⟨inc⟩]	替换一些行，增量用⟨inc⟩
Substitute	S[⟨range⟩][,⟨limit⟩][ENTER] | $⟨old string⟩$⟨new string⟩$	用⟨new string⟩代替⟨old string⟩
Write	W[⟨filename⟩][—⟨switch⟩]	

		将被编辑文本写入磁盘而不从编辑器出口
eXtend	X〈range〉	允许在行末插入文本

附录 2. 变更方式子命令摘要

命　令	格　式	作　用
A	A	打印行的剩余部分，进入改变并终止该行的变更
B	〔i〕B	插入空格
C	〔-〕〔i〕C〈ch〉〔…〈ch〉〕	替换字符
D	〔-〕〔i〕D	删除字符
E	E	进入改变并终止该行变更
F	〔-〕〔i〕F $〈text〉$	寻找〈text〉
G	〔i〕G〈ch〉	插入 i 个〈ch〉
H	〔-〕H〈text〉$	删除行的剩余部分并进入插入方式
I	I〈text〉$	插入〈text〉
K	〔-〕〔i〕K〈ch〉	删除〈ch〉前的所有字符
L	L	把光标定位在行的开始
N	N	恢复原行并移动到下一行（如果 A〈range〉命令仍在进行）或者返回到命令级
O	〔-〕〔i〕O〈text〉$	删除〈text〉$ 前的所有字符
P	P	光标又回到当前位置
Q	Q	从变更方式出口并恢复原行
R	〔-〕〔i〕R〈text〉$	用〈text〉替换 i 个字符
S	〔-〕〔i〕S〈ch〉	寻找〈ch〉
T	〔-〕T	删除行的剩余部分并终止行的变更
W	〔-〕〔i〕W	把光标移过 i 个字
X	〔-〕X	扩展行
Z	〔-〕〔i〕Z	删除字
〔-〕→		移动光标到行的末端
←		删除字符
〔-〕〔i〕 space		把光标移过 i 个字符
ENTER		打印行的剩余部分，输入改变并终止该行的变更
Shift ←		恢复原行，停留在变更方式并把光标回复到行的开始处。用↑Y作响应

附录 3. 记号摘要

用在本资料中的记号定义如下：

〈line〉　　=〈number〉 | • | ↑ | *

〈page〉　　=〈number〉 | • | ↑ | *

〈position〉　　=〈line〉[/〈page〉]

〈range〉　　=〈position〉[: 〈position〉 | ! 〈number〉]

式中

〈number〉　　=〈digit〉 | 〈number〉〈digit〉

〈digit〉　　= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

范围的简化记号

下列范围说明的简化形式可和 EDIT—80 命令一起使用。

简化记号	等价于	范围说明
/〈page〉	↑/〈page〉 : */〈page〉	〈page〉的全部
/〈page 1〉 : /〈page 2〉	↑/〈page 1〉 : */〈page 2〉	〈page 1〉的第 1 行到〈page 2〉的最后一行
:	↑/1 : */*	整个文件
〈position〉:	〈position〉: */*	〈position〉至文件末，例如：. : 和 • /. : */*相同
: 〈position〉	↑/1 : 〈position〉	文件的第一行至〈position〉例如：.和↑/1 : ./.相同

附录 4. EDIT—80 专用字符

break	中断进行中的命令并返回到 EDIT—80 命令级
→	打入一个表格符
shift ←	擦去正在敲入的行并让你从头开始。当在变更方式使用时，shift ←恢复原行，停留在变更方式并把光标回复到行的开始处

通过把 shift 键、向下箭头键（↓）和适当的字母键同时按下可敲出控制字符。

Control	O	由 EDIT—80 命令终止／恢复输出（在终端或行式打印机上）
Control	S	停止／恢复 EDIT—80 命令的执行

附录 5. 错误信息

致命性错误 (Fatal Errors)

盘 I/O 错误是致命性的。会打印出相应的 TRSDOS 错误信息。

任何的 TRSDOS 系统错误信息都是致命的。

非法行格式 (Illegal line format)

当 EDIT-80 找到一奇怪内容的行或一奇怪的行号时，出现非法行格式错误。当编辑一个由 EDIT-80 产生的文件时，通常是不应当出现的。这通常是由阅读没有编辑意义的文件引起的，例如二进制文件。

编辑错误信息 (Edit Error Messages)

非法命令 (Illegal command)

告诉用户敲入的是不存在的命令或者是格式有错的命令。

可用的内存不足 (Insufficient memory available)

当用户把文件作了足够多的改变而耗尽了 EDIT-80 的存贮区时出现这种错误。这仅仅发生在当一个大文件有许多改变时或者当大部分码在被插入时或重新编号时。应当用 W 命令去压缩存贮。

没有给定的串 (No string given)

告诉用户，在发出了 F 和 S 命令后没有搜索到串。这通常在下列情况下发生：一是在发出有宗量的 F 或 S 命令之前先用了不带宗量的 F 或 S 命令。二是在范围之后敲入了不搜索串的〈escape〉键。

没有这样的行 (No such line(s))

如果某一命令引用了不存在的一行或一范围，则会发出这个信息。通常，当适当的页号从行或范围中省略时，出现这个信息。

行太长 (Line too long)

当用户企图输入多于 255 个字符的一行时，发出这个信息。当读入一行或者由于变更一行的命令产生同样的结果时，也会发生这种情况。

丧失次序 (Out of order)

这表明，如果命令已经执行了，而文件的行号不是上升次序。当你试图在没有足够的空档去插入或者试图去删除页标记时，常会发生这种情况。

搜索失败 (Search fails)

提供的这个信息告诉用户搜索没有成功。

卷绕 (Wrap around)

产生大于 99 999 的行时打印出这个信息。

文件错误 (File Error)

文件已经存在 (File already exists)

如果用户试图把已存在的文件名称去赋给某一新文件，或者在 E 或 W 命令下，试图用已存在的文件名称去重新命名一文件，就会发出这一信息。

文件没找到 (File not found)

如果在命令中指定的文件没有找到，则发出此信息。

非法文件说明 (Illegal filespecification)

通知用户命令串包含有某类非法字符。

附录 6. 输出文件格式

编译器和汇编器应当忽略包含在 EDIT-80 输出文件中的行号和页标记（除了在清单文件中的外），微软件 TRS-80 FORTRAN 和 MACRO-80 两者均是这样作的。

行号由 5 个十进数字后跟一个表格符组成。全部 6 个字节，其高位 bit（第 7 位）都等

于1。这里不介绍用 TRSDOS LIST 命令来列表 EDIT-80 文件。图示字符可以出现在行号中。用 EDIT-80 的打印命令代替。

当写一个具有 -BASIC 集的文件时，行号的高位 bit 等于 0。每一行号后跟一个空格，其高位 bit 等于 0。

页标记是一高位 bit 等于 1 的换页字符。

（陈品瓒 译　周宝兴 校）

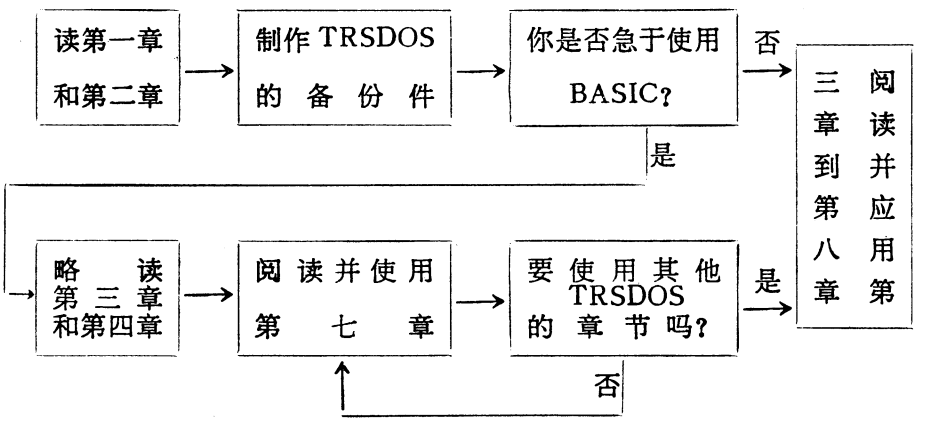
第六篇　TRS-80磁盘操作系统和磁盘BASIC参考手册

前　　言

本手册反映了TRS磁盘操作系统和磁盘BASIC之间的关系。先叙述磁盘操作系统，因为它是基本软件。磁盘BASIC是由TRSDOS（磁盘操作系统）支持的语言，因此放在后面叙述（如果以后提供别的语言，就和磁盘BASIC一起收入本手册）。

不要以为必须严格地按顺序阅读本手册。如果你是一位LEVEL Ⅱ BASIC的老手，又希望从磁盘BASIC开始，可以先阅读第七章，等以后需要的时候或者想要阅读的时候再回过头来阅读TRSDOS各章节。

怎样使用本手册？



一、概　　述

1. 序　　言

本书是TRS-80磁盘操作系统的操作手册和参考手册的综合。它将告诉你怎样操作硬件和怎样使用软件。

对许多读者来说，本书的内容太多了（有人会说："我只希望使用计算机，并不想了解计算机本身"），请不用担心，这本书的编写方法允许你直接开始使用磁盘BASIC编写程序（如果你希望这样做的话）。那末，你只须阅读小型磁盘操作这一章，浏览一下磁盘操作系统概述和磁盘操作系统命令两章，然后阅读磁盘BASIC。

磁盘BASIC只是TRS磁盘操作系统的一个侧面。它不是TRSDOS本身的一部份，而是TRSDOS运行的一个程序。使用磁盘BASIC时，可以对TRS磁盘操作系统的能力一无

所知，就好象软席卧铺乘客没有关于机车、货车、餐车以及列车的其他部份的知识一样。诚然，TRSDOS 所必须做的一切都是使你能够自由地使用 BASIC。但是，总有一天你会希望自己对火车开往什么地方、火车的时刻表怎样，以及列车的敞蓬车上装着什么有发言权，那时你就需要弄懂磁盘操作系统。

图 1 说明计算机、扩展接口和小型磁盘驱动器之间的关系。

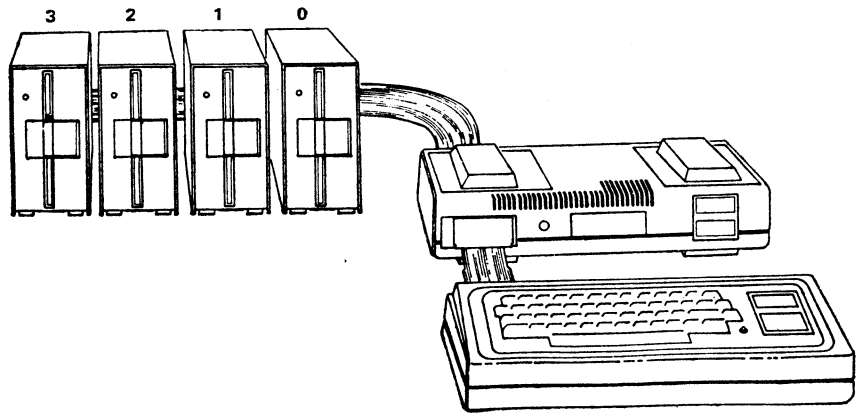


图 1　计算机、扩展接口、驱动器

第一个驱动器（0 号驱动器）总是要插预先录制了操作系统软件——管理程序和包括磁盘BASIC在内的几个辅助程序的 TRSDOS 软塑料磁盘。管理程序装入RAM的前4K字节。只要处在TRS磁盘操作系统控制之下，管理程序就留驻在那里。　只有在需要时才把辅助程序装入内存。

第二个、第三个和第四个驱动器可以插入数据软盘，用来储存用户的程序和数据。

扩展接口内装有实时时钟、磁盘控制器集成电路和可选择添加的RAM（地址在32767以上）。

键盘/计算机内装有一个驻机(在ROM内)程序，它在电源接通时开始工作，从系统软盘（0 号驱动器内）上调入TRS磁盘操作系统的管理程序。若小型磁盘驱动器一个也没有联机，这个 ROM 程序就转去控制 LEVEL Ⅱ BASIC 系统。

本书中你一定要精通的一部份是词汇表，那里给出本书中使用的全部“计算机词汇”和有特殊意义的常用词汇的定义。即使你已经熟悉这些术语，这个词汇表对你也将大有裨益，因为它是专门为 TRS-80 编写的。

首先要复制一个“备份”

随同小型磁盘驱动器 26-1160，可以得到一个 TRSDOS 软磁盘。这个软磁盘上存放有操作系统软件。没有这个软盘，就不可能使用磁盘操作系统。

所以，揭去 TRSDOS 软盘的写保护带以前，磁盘操作的第一步应该是在空磁盘上复制 TRSDOS。关于怎样复制 TRSDOS 软盘的备份，可以在小型磁盘操作这一章的末尾找到简要说明。

2. 符号规定

在叙述命令、语句和人机对话时，为了明确和方便起见，要使用下列符号。

␣　　这个专用符号表示必须要留出的空格。除了这个符号指定的空格外，语法结构中可以出现任意多个空格。例如：

DIR␣:1

R的后面必须留空格。

[ENTER]　　表示按下[ENTER]键。

〈SPACE〉　　表示按下空格键。

大写字母和标点符号　　指必须严格地逐个输入的字母和符号。只有不必打入的标点符号（括号和三连句点…）是特例，下面另作说明。例如：

LOAD "*filespec*"

只有命令LOAD和双撇号要逐字打入；*filespec*（文件标识符）由用户指定。

用虚线括的大写字母　　表示用户按照计算机的提示信息输入的字符。用这种表示方法来区分是计算机送出的符号还是用户输入的符号。例如：

HOW MANY FILES? 5 [ENTER]

计算机询问，用户用"5"回答。

斜体小写字母　　表示在一定情况下机器可以接受而由用户提供的字母、文字或数值。例如：

var=*exp*

左边是变量名称，右边是表达式。

[　　]　　括号把可供选择的内容括起来。例如：

CLOSE〔*file num*〕

CLOSE 后面的 *file num*（文件编号）是由用户选择的，实际上括号并不打入计算机。

…　　括号内的三连句点表示本括号中前边的内容可以重复。例如：

INPUT〔"提示信息"；〕*var*〔,*var*…〕输入的变量表中可以包含多个变量。实际上并不打入这个三连句点。

var([,…])　　表示数组。括号中无逗号表示一维数组；一个逗号表示二维数组；等等。例如：

A$(,)　指二维字符串数组。

B1( )　指一维数组。

exp　　字符串表达式或数值表达式。

var　　字符串变量或数值变量名称。

nmexp　　数值表达式，包括常数、变量和函数。

nmvar　　数值变量名称。

exp$　　字符串表达式，包括常数（放在双撇号内）、变量、函数和字符串算子。

var$　　字符串变量名称。

con　　字符串常数或数值常数。

nmcon　　数值常数。

数值后缀　　加上数值后缀，用来区分同一类型的不同参数和自变量。

例如：

COPY ␣ *filespec* 1 ␣ TO ␣ *filespec* 2

3. 型（Version）和版（Release）的说明

有些用户对术语“Version X．Y”有点迷惑不解。其中“X”和“Y”是随 TRSDOS 不断更新而变化的，所以在这里解释一下。

一个新的型表示对以前的型做了重大扩充。例如，新的型可以包括新的实用程序、高级语言等等。型用整数 1，2，3……编号。

一个新的版是所给定型的前一版的简单改进。一般说来，后一版和前一版相比，或者增强了某些命令的功能，或者应用范围更广了，或者解决了前一版中的某些问题。版用十进小数.1.2.3,……编号。

因此，我们看到“Version 2.1”就知道这是 2 型 1 版的缩写。

注：本手册阐述的是 TRSDOS Version 2.1（TRS 磁盘操作系统 2 型 1 版）和磁盘 BASIC Version 1.1（磁盘BASIC 1 型 1 版）。随以后型和版的需要，这本手册也要更新。

二、小型磁盘操作

1. 序　言

TRS-80 小型磁盘驱动器是为使用 TRS-80 微型计算机的顾客特制的海量存储设备。它把高速盒式磁带机的紧凑性和大型磁盘驱动单元的数据存取的可靠性结合起来了。依靠磁性把信息写入磁盒，也靠磁性从磁盘中读出信息。

简单说来，小型磁盘是由读写磁头（和磁带机相同）、步进马达（在软盘上移动磁头）、驱动马达和使软盘旋转的轮毂装置、以及必要的控制读写过程和马达速度的逻辑电路组成。参看图 2。

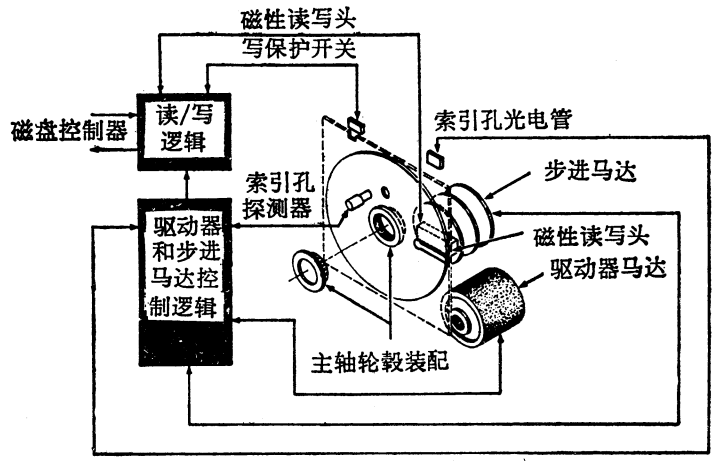


图 2　小型磁盘驱动器的功能部件

驱动器有两种型号，Radio Shack 产品编号分别为 26-1160和26-1161。磁盘系统中必须有一个（仅要一个）26-1160，最多可以有三个26-1161。

随同 26-1160 有：

驱动单元：有一个 26-1161 所没有的专用终端电阻器，

连接电缆：把 26-1160 和多到三个 26-1161 驱动器连到扩展接口上去，

一个 TRSDO S软盘：盘上有操作系统软件、实用程序软件、磁盘 BASIC 等等。

随同 26-1161 有：

驱动单元：没有终端电阻，

空磁盘：可以格式化或用做 TRSDOS 的备件。

2. 连　　接

连接时要断开 TRS-80 系统中各部份的电源。

查看一下随同 26-1160 买来的带状连接电缆，其一端有一单个插座，另一端有四个印刷板插座贯串带状电缆。把单个插座插到扩展接口的左后面的插口上，如图 3 所示。注意插的方向一定要使电缆由下边引出。

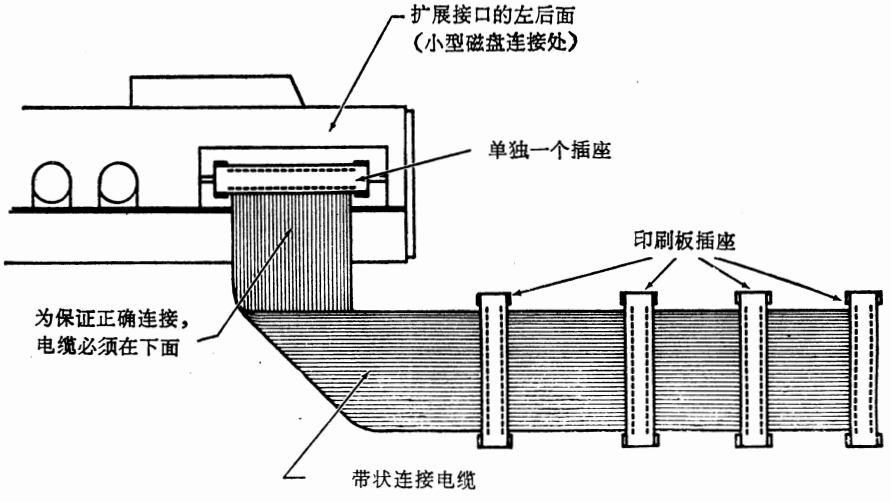


图 3　带状电缆和扩展接口的连接

把驱动器连到电缆上以前，要注意下列规则：

（1）26-1160 必须是电缆上最后一个驱动器（或称终端驱动器），即全部驱动器中，它必须离扩展接口最远。这是因有它有上面说过的终端电阻。

（2）最靠近扩展接口的插座上总要插一个驱动器。其他插座上可以空着。

把每个小型磁盘驱动单元联到电缆上时要注意插座的定位以保证正确连接，如图 4 所示。每个插座内是一个小塑料接插件。若插不进去时，请检查插座的方向，让定位销和导向缝对准。

若只有一个驱动器（必须是 26-1160），要把它插到第一个插座上，驱动器和扩展接口间不要留下空的插座，让后面三个插座空着。

若有两个驱动器，就把 26-1161 联到第一个插座上，把 26-1160 联到第二个插座上，后面两个插座空着。

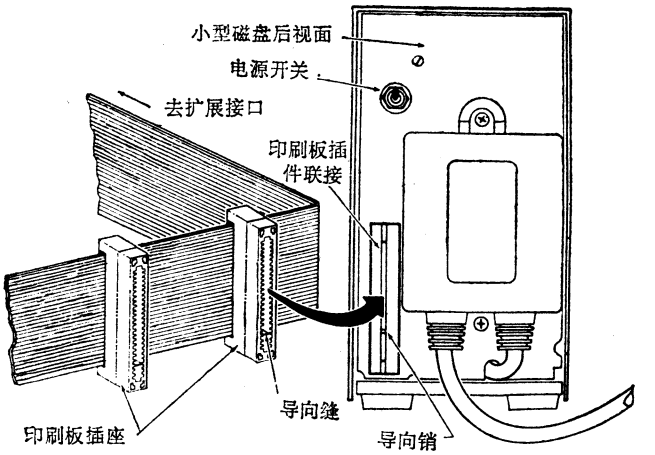


图 4　电缆和小型磁盘驱动器的联接举例：

图 5（略）是联有四个驱动器的小型磁盘系统。用电源线把每个盘驱动器都接到 120V[注] 交流电源上。

驱动器编号

TRSDOS 至少要有一个小型磁盘驱动器，最多可以管理四个驱动器。在 TRSDOS管理下，这些驱动器叫做 0，1，2，3 号驱动器（0 号驱动器离扩展接口最近，3 号驱动器离得最远）。这样的编号不能改变，必须顺序把它们插到带状电缆插座上去。

计算机在执行引导操作时（开电源或复位时）自动地把 TRSDOS 从 0 号驱动器装入内存。因此，接通电源或者计算机复位时，要先把 TRSDOS 软盘插进 0 号驱动器。实际上，除非特殊情况，使用 TRSDOS 的时候 TRSDOS 软盘应一直保留在 0 号驱动器内。

3. 操　作

磁盘系统接通电源以前，必须了解驱动器工作的一些情况。

在电源开关为“ON”（开）的状态下，磁盘驱动器并不是在连续不断地转动着。只有计算机送来 Motor-on（马达开）信号时才旋转。若接上的驱动器不止一个，即使只有其中的一个驱动器和计算机交换信息，Motor-on 信号也同时把它们都接通或关闭。计算机在对磁盘进行存取前约一秒钟送出这个信号，以使驱动器到达工作速度。

计算机对一个磁盘进行存取的时候，这个小型磁盘面板上的红灯（LED）总是亮着的。

注意：在驱动器还运转着的时候（即某一个红灯亮着的时候），不要打开驱动器的闩锁，插入或取出软盘。

软盘怎样工作

简单地说，软盘是个圆形塑料薄片，其一面涂有一层非常光滑的铁磁物质。和 45 转/分唱片相似，软盘上有一个容纳驱动器轮毂的大的中心轴孔，还有一个小孔，用以在磁盘旋转时检索软盘。

一个空盘（空白的新盘或者消过磁的盘），上面没有信息。TRSDOS 有一个专门的实用程序（叫做FORMAT），把空盘划分成同心的磁道(Track)和叫做扇面（Sector）的子道。

注　1）有的是接120V，有的产品要接220V，请按规定电压连接——译者

参看图 6 。这样的划分类似书本的分页。（FORMAT 也把少量系统信息和簿记信息写到每个软盘上去。要了解更多资料的话，请参看扩展实用程序 FORMAT）。

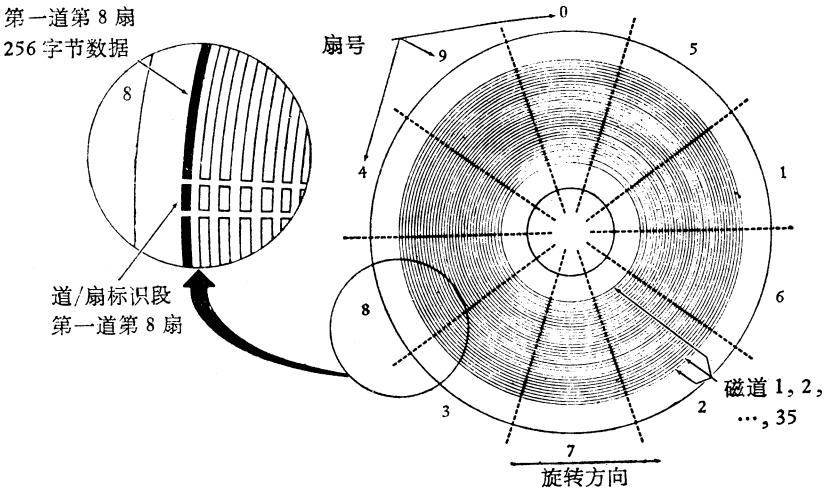


图 6　格式化后磁盘上磁道/扇面的划分

每一片磁盘永久地封装在一个保护套里，以防止磁盘表面被弄弯、折皱、划伤或弄赃。磁盘装进驱动器以后，轮毂装置就紧紧抓住软盘。驱动器的马达电源接通以后，软盘就在保护套里面旋转，专门处理过的保护套的衬里在软盘旋转时就把软盘擦拭干净。

TRSDOS软盘的上部（标牌向上）有一片纸带，这片带子盖住了软盘的写入保护缺口。当这个缺口被盖住后，软盘就被保护起来、不能再写入信息（写操作将变更盘上的信息。相反，读操作不变更盘上的信息，仅仅取出信息）。若打算在盘上写入信息，就把这片带子从盘上揭掉。而将这片带子贴在任何一个软盘的写保护缺口上，就不用想在这个盘上写入数据。请看图 7 。

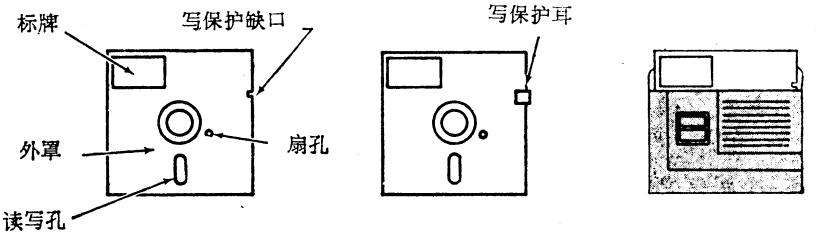


图 7　软盘、有写入保护的软盘、插在保护袋中软盘

把软盘插进驱动器

（1）必须在磁盘驱动器停止转动时才可以插入和取出软盘。

（2）打开驱动器的面板，轻轻地把软盘插进竖直的狭缝，写保护缺口要在上面，标牌向右（图 8 略）。在没有把软盘整个都插进去使之完全就位之前，万万不能关上闩锁。否则会弄坏软盘。

（3）关上小型磁盘驱动器的闩锁，这样，心轴的轮毂装置就和软盘紧紧配合上了。若不能很容易地关闭，就不要勉强。重新插软盘，再试试看。

接通电源的顺序

要先接通外围设备的电源（磁盘驱器动、打印机、扩展接口等），最后接通 TRS-80 的中央处理机/键盘的电源。注意：计算机还在工作的时候，开关外围设备会引起系统混乱，这是错误的操作。这样做可能会丢失目前打开的文件。

小型磁盘驱动器的电源开关在它的后面板上。乒乓开关向上表示电源接通，向下表示电源断开。电源接通的顺序应该是：

（1）接通扩展接口。

（2）接通磁盘驱动器：先接通最后面的驱动器 26-1160，然后接通别的驱动器。

（3）接通 TRS-80 中央处理机／健盘的一瞬间，计算机就从 0 号驱动器把 TRSDOS 装入内存。所以，接通 CPU 的电源以前，要按上述"插入磁盘"的方法小心谨慎地把TRSDOS 软盘插进 0 号驱动器。然后，就可以把格式化了的磁盘插进别的驱动器。然而，这都要在驱动器停止转动时进行。

另一种办法是把各个驱动器接在一条电源线上，用一个电源开关把它们一起接通。

4. 软盘的保养

软盘是精密记录设备，精心管理能够延长软盘的寿命。一般来说，下述保养措施对盒式磁带机和高保真度唱盘也适用。

（1）只要不在驱动器上使用，就要随时把软盘保存在它的保护盒里。没有必要就不要把软盘留在驱动器上。例如，系统关闭后，就要把软盘取出。

（2）要远离磁场（变压器、交流马达、强磁体等等）。强磁场会破坏盘上的信息。

（3）只能把磁盘存放在它的封套中。不要触摸任何暴露在外的盘面，也不要擦拭盘表面。否则可能划伤软盘以至破坏数据。

（4）要远离热源，避免阳光的直接照射。关于储存的温度范围请参看后面的"技术指标"一节。

（5）不要让烟灰、灰尘或别的尘埃玷污软盘。

（6）不要用硬东西，如圆珠笔、铅笔等在软盘的保护套上直接写字，那样会损坏磁盘表面，只能用毛笔。

（7）软盘插入驱动器前，驱动器马达必须是关着的（红灯不亮，马达也没有声音）。

（8）磁盘要垂直地存放在文件夹或架子上。避免磁盘边缘受压（和唱片的存放相同）。

如果遇到问题……

磁盘存取中如果经常出现磁盘 I/O错误，可能是磁盘有损伤，或者驱动器有些问题，或者别的硬件有问题。更换现有的驱动器或磁盘，或许能解决这个问题。

如果有的磁盘反复出问题，就把可能读出的文件复制到另一个盘上，然后把这个有问题的磁盘用大型消磁器（Radio Shack 产品号：44-210）消磁，再把它格式化（参看扩展实用程序 FORMAT）。

格式化的过程中也检查磁盘片的缺陷，封锁有缺陷的磁道，把剩下可用的软盘提供给用户。

如果发现小型磁盘驱动器有问题（几个磁盘片都发生存取错误），请把这个驱动器送到当地的 Radio Shack 门市部检修。

5. 技术指标

驱动器和磁盘的技术指标：

存储容量（用户可用字节数）	
格式化的磁盘	83,060
TRSDOS软盘	58,880
磁盘的结构	
每个磁盘的磁道数	35
每磁道字节数	2,560
每个磁道的扇面数	10
每扇面字节数	256
数据传输速率	12.5k字节/秒
平均存取时间	750ms
驱动器马达起动时间	1 秒
所需物品	Radio Shack软磁盘，产品号26-305或26-0405（每包三片）
软盘寿命[1]	2.5×10^6次读写/每磁道（110小时），估计可用 5 年。
软盘上数据的存储寿命	20年
软盘保存温度	50-120°F(12-52°C)
尺寸	
驱动单元	$6\frac{3}{8}\times3\frac{1}{2}\times13\frac{1}{4}$英寸 (16.2×8.4×33.7cm)
磁盘（带保护套）	$5\frac{1}{4}\times5\frac{1}{4}\times\frac{1}{32}$英寸 (13.3×13.3×0.08cm)
电源	120V 60Hz 35W (28瓦) (或220V 50Hz 35W)

注 1) 软盘寿命常常受不合理的保养所限制。按上节的建议进行保养，能够延长磁盘寿命。

6. 示意图

控制逻辑图和读/写逻辑图，电源线路图。

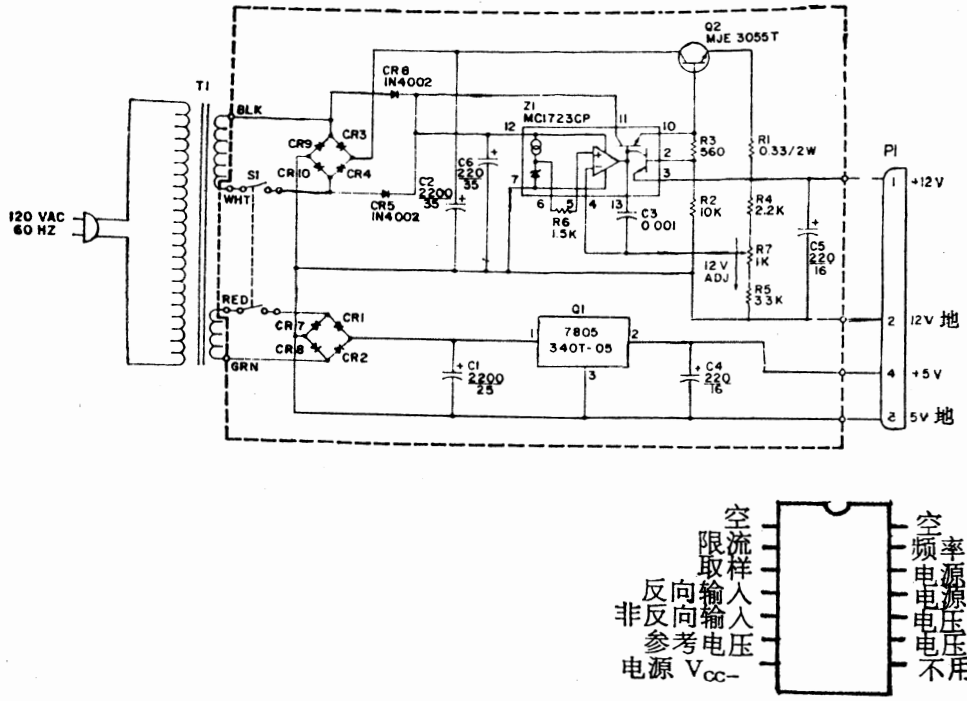


图11 电 源 线 路

注：除指定外，均按此例

1. 全部电阻都是 1/4W、5%、以欧姆为单位、K=1000
2. 电容的标称数值是微法拉、工作电压
3. →指顺时针旋转

7. 复制 TRSDOS 的备份(BACKUP)

在使用TRSDOS软盘进行工作以前，先按以下指令制做一份系统软件的“备份”。这样，一旦原件出了问题，也不至于坐等买来另一个 TRSDOS 软盘才能工作。

按前面磁盘操作一章所讲方法，把小型磁盘系统连接好，接通电源。在接通 CPU 之前，一定要先把 TRSDOS 软盘放进 0 号驱动器(为安全起见，在复制完成前，不要揭掉 TRSDOS 软盘的写保护带）。

如果你连接了几个驱动器，就把空盘放到 1 号驱动器上。若只有一个驱动器，那末，BACKUP 将会告诉你何时把空盘放进 0 号驱动器。在空盘上不要贴写保护带。

CPU 接通电源以后，显示器上将出现：

```
TRSDOS-DISK OPERATING SYSTEM-VER2.1
DOS READY
-
```

请打入：

BACKUP [ENTER]

系统就显示出：

TRSDOS DISK BACKUP UTILITY VER 2.1

如果你只联接了一个驱动器，就打入：

SOURCE DRIVE NUMBER? 0 [ENTER]

DESTINATION DRIVE NUMBER? 0 [ENTER]

如果你有两个以上的驱动器，则打入：

SOURCE DRIVE NUMBER? 0 [ENTER]

DESTINATION DRIVE NUMBER? 1 [ENTER]

然后按月/日/年形式打入日期。例如，若是1978年 8 月 3 日，就打入：

BACKUP DATE (MM/DD/YY)? 08/03/78 [ENTER]

于是，TRSDOS 起动 BACKUP 程序。首先，把空盘格式化，封锁有缺陷的磁道，再把 TRSDOS 软盘的内容复制到格式化了的空盘上。

如果你只有一个驱动器，BACKUP 将告诉你什么时候插入目的盘（空盘）什么时候重新插入源盘(TRSDOS)。复制过程中，要数次交替插入两个软盘。

工作完成后，显示出信息：

BACKUP COMPLETE-PRESS ENTER TO CONTINUE

如果显示的信息是：

BACKUP REJECTED DUE TO (…)

那么，就要用消磁器把软盘消磁，再重新进行 BACKUP 过程。如果仍不能通过，就要换用另一个空盘试试。

三、 TRS磁盘操作系统概述

1. 序 言

和整个 TRS-80 微计算机系统一样，TRS 磁盘操作系统的设计也考虑到不同用户的需要：

（1） 初学计算机者。他希望开头简单些，再逐步深入学习。

（2） 有经验的程序员。他希望写复杂的程序，也想采用某些机器语言的系统程序，以完成交付给他的各种各样复杂的任务。

（3） 纯用户。其兴趣只在使用程序，而不编写程序（例如,在办公室的 TRS-80 上使用盘点存货程序的职员）。

什么是操作系统?

读完这本书，才能有完整的概念。现在只介绍一点概况。

操作系统是一个主程序。它把一个复杂的计算机系统，包括各种各样的输入/输出(I/O)设备、存储设备和程序，有效而又简单明了地组织起来。操作系统保证每种资源执行它必须执行以及命令它执行的任务。因此，用户就没有必要知道它必须执行的任务是什么。

下面对操作系统的工作稍行剖析（对于不熟悉的术语，请参考词汇表）：

（1） 把中央处理单元(CPU)和各种输入/输出设备、存储设备联系起来。

（2） 接受并解释操作命令。

（3） 利用安排CPU时间表，分配输入/输出通道、存储器和其他系统资源，“指导”用户程序（以及用户需要的系统实用程序）有次序的进入和退出运行。

（4） 管理中断。管理前台和后台任务的执行。

（5） 提供一些以不同方式出现在每个程序中的基本程序。这样可以节约存储器和程序设计的时间。

并不是必须十分了解操作系统才能使用操作系统。例如，使用磁盘 BASIC 时，就根本看不见 TRSDOS 的存在，但是，这个系统始终存在并运行调动 BASIC 的程序，BASIC 程序又在运行你自己的程序命令。

其他情况下，对用户来说，操作系统的作用就十分明显了。这时允许你直接打入系统命令。这就是TRSDOS的DOS READY 状态。

什么是 TRSDOS？

TRS-80 磁盘操作系统（TRSDOS）是一组综合的系统程序和文件管理实用程序。它如此之复杂（功能强大）就是因为它以磁盘为基地。

由软盘把这个系统装入内存，并用磁盘存储内部簿记信息以及用户建立的数据和程序。TRSDOS 完全使用动态的磁盘空间分配，所以，可以自由地打开文件、管理文件，而不必顾及它们在盘上实际存放位置。若一个文件的指定空间已经填满，TRSDOS 会自动寻找并给与更多的空间来存储多余的数据（假定盘上还有可用的空间）。

（磁盘上的所有信息——程序、数据以及 TRSDOS 本身——都以文件的形式存在。关于文件的详细内容，可参阅词汇表和技术资料等章节）

除了执行上面“什么是操作系统”中所讲的系统程序的功能以外，TRSDOS 还包括几个文件管理实用程序，可以管理和修改软盘上的文件：复制、添加、重新命名、改变保护状态等等。

TRSDOS 怎样使用RAM

TRSDOS 包括：

（1） 管理程序文件

（2） 辅助系统程序文件

（3） 库命令文件

（4） 扩展实用程序文件(BACKUP 和 FORMAT)

（5） 磁盘 BASIC 文件。

接通电源后管理程序就被装入RAM，并在TRSDOS 运行期间一直驻留在那里，因此，称之为“常驻”TRSDOS 程序。管理程序包括一些系统程序、表、指针和输入/输出驱动单元。

辅助系统文件包括运行用户程序和命令所必须装入的程序和命令。这些程序装入内存的“覆盖区”。在TRSDOS 把一个程序运行完后，可以把另一程序装入同一区域(或叫覆盖)。覆盖使用说明系统程序的运行不影响用户内存区（地址51FFH以上）。

库命令文件包括运行大多数操作命令所需的程序。这些程序装入内存的地址是从5200H

到6FFFH。因此，用户的机器语言程序一般应使用 6FFFH 以上的区域，才不致受库命令运行的影响。

TRSDOS扩展实用程序只有在你打入它们的文件名称(BACKUP 和 FORMAT)时才装进内存。这两个程序可能用到整个可用的内存区。因此，在输入这两个程序时，甚至会将常驻的 TRSDOS 程序冲掉。

磁盘 BASIC 大大加强了 LEVEL Ⅱ BASIC 的功能。打入它的文件名BASIC时，就装进从5200H开始的内存区，并开始运行这个程序。

2. 打入命令

无论什么时候，只要显示出提示信息：

DOS READY

就可以打入操作命令了。形式最简单的操作命令只是一个单词。系统命令或库命令，扩展实用程序名或用户命令程序都是这样的命令。这些将在后面仔细讲述。

例如：

DIR [ENTER]

告诉 TRSDOS 显示 0 号驱动器的用户文件目录。

通常，操作命令需要几个词。例如，要删除某个文件，必须指定文件名。

KILL XYZ [ENTER]

告诉 TRSDOS 寻找文件名XYZ，把它从存放该文件的软盘目录中删去，并开放这个文件占有的空间。

一般来说，一条操作命令由命令和一个或多个带有特定参数的文件标识符 组成：

命令[␣*filespec*] [␣(*param*)] [␣TO] [␣*filespec*] [␣(*param*)]

其中："*filespec*" 是一个有效的 TRSDOS 文件标识符（下面仔细讲）

"*param*" 是一个参数，用来说明这个命令怎样影响指定的文件。

这个命令的格式看起来很复杂，因为它是普遍形式。从每条命令的实例中将看到，实际的命令可能十分简单。

无论什么时候，打完一个命令后，按下 [ENTER] 健，TRSDOS 就按以下顺序处 理 这个命令：

（1） 检查打入的命令是否系统命令或库命令；是的话，立即执行……，否则

（2） 检查打入的内容是否实用程序名称；是的话，就执行这个扩展实用程序……，否则

（3） 检查每个驱动器上的磁盘的目录，看打入的命令是否列为用户命令文件，若是，就装入并执行这个文件。

3. 文件标识符 (File Specification)

文件标识符是访问指定文件的手段。无论在 TRSDOS、磁盘 BASIC中还是在其他命令程序（例如，TAPEDISK）中都是这样。

磁盘文件标识符的格式是：

文件名[/*ext*][. *pw*][: *d*]

其中：文件名是由1到8个字符组成的文件的名称。第一个字符必须是字母。

ext 是文件名的扩展，由1到3个字符组成，第一个必须是字母。文件名扩展是可有可无的选择项，如果使用的话，它的前面要加上斜线符号“/”。

pw 是一个可以选择的通行字。由1到8个字符组成。如果使用通行字，在它的前面必须加上一个句点符号“.”。

d 是驱动器标识符，也是选择项，*d* 取值0，1，2或3，这取决于你打算指定哪个驱动器。驱动器标识符前必须加冒号“:”。

文件标识符的中间不要插入任何空格，否则TRSDOS就认为这个文件标识符在第一个空格处结束。如果被截断了的标识符是有效的，用户就得不到出错的信息。

有效的文件标识符举例：

A	DRIVECHK:1
GAMES/BAS	AUG1578
PAYROLL/BAS.SESAME	INVNTORY
SECRETS.MYNAME	SORTER/VR1
TAXES/TXT.TEAPARTY:1	DRIVECHK:2
POETRY/TXT:1	DATA11
AUG3078/DAT.JQD	SORTER/VR2
CHKWRITR/BAS.VERSION2	

取一个完全“填满”的文件标识符来研究一下：

TAXES/TXT . TEAPARTY : 1

TAXES是文件名，TXT是文件名扩展，TEAPARTY是通行字，应到1号驱动器访问这个文件。如果使用这个文件标识符建立一个文件，这个文件就被放到1号驱动器中去。如果读或写这个文件，TRSDOS就去访问1号驱动器。

什么使文件标识符唯一化？

文件名、文件名扩展和驱动器标识符都可以使特定的文件标识符唯一化。通行字不能做到这一点。

例如，下列文件标识符代表不同的文件：

A

DRIVECHK:0　DRIVECHK:1　DRIVECHK:2　DRIVECHK:3

然而，下列文件标识符就不能用来访问不同的文件：

RECEIPTS　RECEIPTS.AUG3078

RECEIPTS.AUG3178

（有些情况下，两个不同的通行字可以存取同一个文件。参看TRSDOS库命令ATTRIB）

关于扩展

可以完全随个人习惯使用文件名扩展。这样一来，用外加三个字符形成的文件名扩展，就可能构造更加适用的文件名。例如：

PAYROLL/AUG　PAYROLL/SEP　PAYROLL/OCT

然而，按约定习俗将扩展作为类型标识符使用，就更有意义了。建议采用下列扩展：

/BAS　以压缩格式存储的BASIC程序文件。

/TXT　ASCII 文本：以 ASCII格式存储的 BASIC 程序、源文件等等。
/CMD　机器语言命令文件
/CIM　不一定能执行的内存 (RAM) 映象文件
/REL　浮动的机器语言程序文件
/SYS　系统程序或文件。是 TRSDOS 的一部分。用户文件不要使用。
/OV_n　覆盖数n
/DVR　输入/输出驱动器模块。

使用这种扩展的一个好处是只要查看软盘的目录表就可以知道程序的类型。另一个好处是给 TRSDOS 配备了识别某些扩展的能力。

例如，假设一个文件具有扩展 /CMD，当你打入：

文件名　[ENTER]

而省略扩展部 /CMD 时，TRSDOS 也装入并执行这个文件。这就是为什么只要打入

BASIC　[ENTER]

就可以执行文件BASIC/CMD的原因。同样，用户自己的程序也可以写上标识性名扩展。

关于驱动器标识符

若用户指定驱动器标识符，TRSDOS 在执行一个命令时，就使用此指定的驱动器。若省略驱动器标识符，TRSDOS 就从 0 号驱动器开始查找全部驱动器的目录，然后启动有指定的文件名/扩展的第一个驱动器。但是，如果命令是建立文件，TRSDOS 就跳到一个没有写保护的盘上去。

例如，假定有四个名为DRIVECHK 的文件，分别放在 0 到了 3 符驱动器中，那末，每次访问 DRIVECHK （没有驱动器标识符），将都是去访问 0 号驱动 器，使用文件标识符 DRIVECHK：0，DRIVECHK：1，DRIVECHK：2，DRIVECHK：3，四个文件就都能存取。

关于通行字

通行字是建立文件时赋给的。可以用 ATTRIB 和PROT 命令更改通行字。具有通行字的文件只有在指明通行字或磁盘的主通行字时才能进行存取。所以，一旦给文 件 赋了通行字，就不要忘记这个通行字。

重要的是必须认识到每个文件都是有通行字的。甚至在建立文件时并没有明确指定通行字的文件，也有通行字。这时，通行字就是八个空格的字符组。

例如，假设已经存在一个文件 SAMPLE（没有明显通行字的文件），又要建立一个新的文件 SAMPLE．WOTERBOY，那末，TRSDOS 就回答你一个信息：FILE ACCESS DENIED（拒绝存取文件）。这是因为你打算用错误的通行字去存取一个现已存在的文件。正确的通行字是八个空格的字符串——它可以在文件标识符中略去，因为八个空格是一个空缺通行字。

四、 TRS 磁盘操作系统命令

1. 系统命令

这三个命令(BASIC2、DEBUG、TRACE)是和用户的RAM（地址 5200H——内存终地址）“不接触”的。这些命令所需要的代码装进常驻程序和十六进制 5200 之间的覆盖区。别的命令叫做“库命令”，用的地址在 5200H到 6FFFH之间。所以，要把你的机器语言程序的地址分配在7000H以上，以保护它们避免被实用程序冲掉。

BASIC2（转入LEVEL Ⅰ BASIC）

```
BASIC2
```

这个命令没有自变量和参数，仅仅是将控制转入 LEVEL Ⅰ BA SIC。这个命令一旦执行完毕，RAM 中就不再驻有 TRSDOS。于是 TRS-80 的功能就和LEVEL Ⅰ BASIC 机器一样了。

使用这个命令可给不需使用磁盘的程序增加内存。另一种可能的用途是从磁盘中把机器语言程序装入内存高区，再用 BASIC2 命令转入 LEVEL Ⅰ BASIC。这样就可以通过 USR 函数在 LEVEL Ⅰ 控制下对此程序进行存取操作。例如：

```
BASIC2    ENTER
```

```
MEMORY SISE? ENTER
RADIO SHACK LEVEL Ⅰ BASIC
READY
>-
```

要重新装入 TRSDOS 时，就压 RESET 键或打入：

```
SYSTEM    ENTER
* ? 0     ENTER
```

DEBUG（实时查错程序）

DEBUG [␣ (*param*)]
其中*param*=ON或 OFF。如果是ON，可以不写

DEBUG 是一个实时调试程序包，它用于机器语言程序、包括前台任务和后台程序（参考词汇表）的调试。可以使用 DEBUG 检查、修改 Z80 的各寄存器和存储单元的内容，转移到指定的地址开始运行，直到所选择好的断点为止，在程序中一次移动一条指令（或一次 CALL）或多条指令。

DEBUG中的字节值和地址都要用十六进制形式给出，这是 DEBUG 所要求的数制。

DEBUG装进覆盖区。51FF以上的地址不受影响。

打入

DEBUG [ENTER]

就具备了查错的能力。正常的 TRSDOS 命令仍可以继续解释。下列条件之一成立时，就可以装入并执行查错程序。

(1) 按下 BREAK 键

(2) 装入程序以后，执行这个程序的第一条指令以前

(3) 发现和磁盘有关的错误

注意：不能调试 TRSDOS 系统程序和只能执行的用户程序(execute—only user routines)。当这些保护性程序作为 DEBUG 的“目标”时，可以用 DEBUG 检查/修改寄存器和 RAM 的内容，但不能进行单步、转移等操作。再者，由于 DEBUG 装入 RAM 的覆盖区，不能与别的覆盖程序一起使用。

DEBUG 提供两种显示格式：

(1) 寄存器显示，同时显示间接地址上 RAM 的内容以及任意 64 字节一“页” RAM的内容。

(2) 全荧光屏显示，显示 256 字节一页RAM的内容。

在寄存器显示格式中，DEBUG 显示出全部 Z80 寄存器的内容，以两个八位寄存器或一个十六位寄存器对来解释。因为多数程序用几组寄存器对作间接指针或变址寄存器。因此，随每个寄存器对一起出现的是 16 字节的间接地址上的数据。每个标志寄存器用其标志位的 ASCII 代码形式表示。

另外 64 字节的内存内容在显示器的底部显示成四行。

下面是一个典型的 DEBUG 显示次序。

```
DEBUG [ENTER]
DOS RERDY
[BREAK]
```

```
AF =3E 28 --1-1---
BC =0A 3E ⇒ 09 BA C2 60 09 7D 93 C2 60 09 C9 21 27 41 CD D3
DE =01 04 ⇒ 1A 4D 45 4D 4F 52 59 20 53 49 5R 45 00 52 41 44
HL =00 54 ⇒ 01 01 5B 1B 0A 1R 08 18 09 19 20 20 0B 78 B1 20
AF'=FF FF SZ1H1PNC
BC'=4D BE ⇒ 51 51 CD FC 51 7E 23 18 EC 02 02 00 4E 03 32 E7
DE'=01 07 ⇒ 4D 4F 52 59 20 53 49 5A 45 00 52 41 44 49 4F 20
HL'=4D 00 ⇒ F2 51 06 10 CD 65 51 3A 5D 40 EF 41 20 13 CD F2
IX =40 15 ⇒ 01 E3 03 00 00 00 4B 49 07 58 04 31 3E 20 44 4F
IY =FF FF ⇒ FF | F3| AF C3 74 06 C3 00 40 C3 00 40 E1 E9 D3 9F
SP =41 E8 ⇒ 52 04 DD 03 15 40 15 40 18 43 3F 3F 40 00 E3 05
PC =00 60 ⇒ 0B 78 B1 20 FB C9 31 00 06 3A EC 37 3D FE 02 D2
      1010⇒ 28 10 FE 44 28 0C FE 30 28 F0 FE 20 28 ED FE 2E
      1020⇒ 20 03 2B 36 30 7B E6 10 28 03 2B 36 24 7B E6 04
      1020⇒ D0 2B 70 D9 32 D8 40 21 30 41 36 20 C9 FE 05 E5
      1040⇒ DE 00 17 57 14 CD 01 12 01 00 03 82 FA 57 10 14_
```

这个显示画面中，寄存器B的内容是十六进制值0A，寄存器C是3E。把BC寄存器对作为指针，指向的地址是0A3E。因此，在标记BC＝0A3E⇒右边的是内存0A3E到0A4D单元的内容。在这个例子中0A3E是09，0A3F内容是BA等等。

标志寄存器F和F′作了不同的处理。这两个标志寄存器的16个位上，每位用一个字母代码来表示，这样容易解释标志的状态。例如，第7位是符号位（最左边的位）。所以，只要这一位是“置位”状态，就显示出字母S。

下面是各标志位的代码表

位	状 态	置位代码	非置位代码
7	符号	S	—
6	零	Z	—
5	未用	1	—
4	半进位	H	—
3	未用	1	—
2	奇偶/溢出	P	—
1	负	N	—
0	进位	C	—

上面显示的画面上，F的各标志位都是非置位(不算未用位5和3)状态。F′的各标志位都是置位状态。

PC寄存器下面的四个附加行中，每行显示的是以箭头左边地址为起点的16个字节的内

容。这四行总是显示内存中相邻的64个字节的内容。这四行显示的起始地址或者是0000，或者是最近用D命令指定的地址（后边详细讲述）。

显示器左下方的空白区域是显示输入命令的地方。

DEBUG 命令

注意：一些命令只要按下面规定的命令键就立即执行。另一些命令要按〈SPACE〉键或 ENTER 键后才执行。下面分别予以说明。

命令	需要按的键	完成的操作
A	无	显示出和每一显示的值对应的ASCⅡ符号或图形符号。当该数值不能用ASCⅡ或图形符号显示时，就显示一个句点“.”
C	无	单步进入下一指令，与执行CALL完全一样（下一指令由PC寄存器确定）。目标程序不能是系统程序和仅能执行的程序。
D*aaaa*	〈SPACE〉	全荧光屏显示方式。显示从*aaaa*地址开始的内存内容。
G*aaaa* [,*bbbb*[,*cccc*]]	〈ENTER〉	把*aaaa*放入PC寄存器。在*bbbb*和*cccc*处设置断点并运行。
H	无	用十六进制形式显示整个内存和寄存器。
I	无	单步执行下一指令（由PC寄存器确定）。目标程序不能是读保护程序(Read—protected)
M[*aaaa*]	〈SPACE〉	令目前的修改地址为*aaaa*，修改的回答显示在荧光屏的左下角。如果省略*aaaa*，就把上次修改的地址作为*aaaa*，如果*aaaa*已在显示器上，它的内容就用一对竖线(|…|)括起来。
R*rp*␣*dddd*	〈SPACE〉	把值*dddd*送入寄存器对*rp*，*rp*可以是下列任何一对寄存器：AF，BC，AF'，BC'，IX，IY，PC等。
S	无	用全荧光屏显示方式，显示256个连续的内存字节。按X就返回寄存器显示方式。
U	无	动态更新显示方式。这样可以观察前台任务的执行。按下任何一个键两秒钟就可退出这个方式。
X	无	令显示为寄存器方式，同时撤消除R以外的正在打入的命令。
;	无	显示的内存增加一页（寄存器显示方式中，一页是64个字节，全荧光屏方式一页是256个字节）。
—	无	显示的内容退回一页。

注意：打入命令期间不能用退格键“←”删除错误。要按X键才能撤消刚打入的命令。如果打入地址或数值时发生错误，只要在错误的地址后紧接着打入正确的地址就可以。DEBUG只看最后打入的四位数字。例如

D 474080 〈SPACE〉

告诉DEBUG显示出包括地址4080在内的一页内存的内容。

关于M命令（修改内存）

要改变内存单元内容，就打入 M*aaaa*，再按〈SPACE〉键。此命令修改内存地址为 *aaaa* 的内容，并在显示器的左下角显示出内存修改提示信息。例如，打入

M7F00 〈SPACE〉

得到

```
    7F00 => |20| 00  00 00 00 00  92 B2 20 B3 B3 B3 B3 B3 B3 B3
7F00_7F10 => B3 B3 B3 B3 B3 B3 B3 B3 BB B3 BB B3 BB B3 3B BB
20-_ 7F20 => FB BB BB BB BB FB FB BB 00 FB FB FB FB FB FB FB
    7F30 => 00 FB FF FB FF FB FF FF 00 FF FF FF FF FF FF FF_
```

注意，7F00 的值用“| |”括起来了，两条竖线总是出现在荧光屏上被修改地址出现的地方。

要修改 7F00 的内容，就打入两位新的内容并按下空格键，于是就修正了已显示出的内容，然后DEBUG 把被修改地址加 1 。不输入新的内容，只按下空格键，就不改变这个地址的内容。只将被修改地址加 1 ，保持这一地址的内容不变。

要退出修改内存状态，就按下X或者 ENTER 键。

如果只打入：

M 〈SPACE〉

DEBUG 就按上一个指定的被修改地址进行，如果以前没有指定过被修改地址就用0000。

显示器上常常有两个值被“| |”括起来：一个在64字节的内存显示区，另一个在和寄存器对有关的间接内存区。这是因为被修改地址的内容显示了两次，一次是直接的，另一次是间接的。

关于G命令

打入

G402D ENTER

不用重新初始化就能从查错程序返回 TRSDOS。当处于前面说过的三个条件之一时又可以重新进入 DEBUG。

在使用这种出口后，为取消 DEBUG，可打入

DEBUG (OFF) ENTER

DIR ENTER

处在查错状态下，打入

G [ENTER]

就从 PC 寄存器内的地址开始执行。

打入

G 0000 [ENTER]

就重新初始化 TRSDOS。

关于U命令（更新显示）

在更新状态下，只执行前台任务。为了看到工作情况，需要查看前台任务使用的寄存器或内存单元。

最好的例子是实时时钟。

打入

D 4040 〈SPACE〉

就显示出地址 4040 到 4046 的值，这些地址存放的时间和日期如下：

地址	内容
4040	25毫秒的实时记时（Scheduling）计数器
4041	秒
4042	分
4043	小时
4044	年
4045	日
4046	月

现在打入U，就将看到由这个时钟前台任务更新的值。

DEBUG 的其它用途

可以通过磁盘 BASIC 存取 DEBUG。它可以帮助查找栈指针、表地址等等，参看DISK BASIC。

取消DEBUG程序

只要DEBUG在复盖区内，TRSDOS就可能突然进入 DEBUG，例如，由于存在错误。如果不希望发生这种情况，打入下列命令就可把 DEBUG 程序消去。

G 402D [ENTER] （返回TRSDOS）

DEBUG (OFF) [ENTER]

DIR [ENTER]

TRACE（PC寄存器的动态显示）

TRACE [␣ (*param*)]

其中*param*=ON 或 OFF。如果是ON，可以略去

TRACE 命令起动一个前台任务。这个前台任务将在电视显示器的左上角显示出用户程序指令计数器（PC 寄存器）的内容。每 8 毫秒以当前后台程序的运行地址更新该四位十六进制数，例如

TRACE [ENTER]

因为 TRACE 是前台任务，因此在磁盘操作系统的准备（DOS READY）状态，磁盘 BASIC 状态或者任何其它的程序中，都可以使用 TRACE。要 TRACE 暂时停止下来则禁止全部中断（磁盘 BASIC 的 CMD"T"命令），执行磁盘 BASIC的 CMT"R"命令，中断恢复，TRACE 也重新启动。

和DEBUG程序一起使用时，TRACE对调试机器语言程序有很大帮助。在运行 BASIC 程序期间，这条命令不经常使用。

要停止 TRACE，请执行命令：

TRACE (OFF) [ENTER]

2. 库 命 令

这些命令在 RAM 中占用的区域是 5200H到 6FFFH。它们是按需要成块地装进来的。例如，DATE和TIME，若需要其中一个，两者就一起装进内存。如果 RAM 中已经有了这个命令的代码，TRSDOS就不再耗费时间去装进这个命令。

AUTO（接通电源时自动键盘输入）

AUTO [␣*dos—command*]

其中 *dos—command* 是操作命令或可执行命令文件的文件标识符

注意，使用AUTO命令，必须从系统软盘上揭去写保护带。

AUTO命令给用户变更接通电源的次序，接通电源后立刻执行指定的命令。

打入

AUTO *dos—command* [ENTER]

TRSDOS就把这个 *dos—command*（磁盘操作系统命令）作为“自动键盘输入”（Automatic .key-in）写在 0 号驱动器内的软盘上，取代原来的自动键盘输入。每当接通电源使用这个软盘时，TRSDOS 的初始化就总是自动键入这个磁盘操作系统命令。自动键入取代了键盘输入。

要把接通电源的次序恢复正常，就打入：

AUTO [ENTER]

这样就把任何自动键盘输入清除。

例如

AUTO CLOCK 在下次接通电源后，自动装入并执行显示时钟的命令。

AUTO BASIC 在下次接通电源后，TRSDOS 就自动装入磁盘 BASIC，并开始初始对话。

注意：在接通电源时，一直按下 ENTER 键就可以取消自动键盘输入。如果磁盘操作命令不是一个工作命令程序。这或许是回到系统控制的唯一办法。

ATTRIB（设置保护特征）

ATTRIB ␣ *filespec* ␣ (*param*[, *param*…])

其中 *param* 可以是下列参数之一：

param（参数）	意义
I	使该文件在正常的开列目录命令中看不见
ACC=通行字 1	把“通行字 1”作为新的存取通行字赋与这个文件
UPD=通行字 2	把“通行字 2”作为新的更新通行字赋与这个文件
PROT=级	把这个“级”赋与该文件，变为新的存取保护级：（KILL, RENAME, WRITE, READ, EXEC）

这个命令中的文件标识符必须确实已经存储在一个驱动器中了。

这个命令通过改变通行字，也改变用通行字设置的存取自由度的方式；或者只改变通行字，也许只改变通行字设置的存取自由度的方式，变更一个文件的保护状态（参见TRSDOS概述那一章的文件标识符一节）。

用指定 I 参数的方法已经给了文件不可见的特征，又要显示目录中看不见的文件，就必须在DIR命令中指定参数I。没有其他办法消去不可见特征，除非把这个文件复制成没有 I特征的文件。

例如：

```
ATTRIB VIDSCAN/CMD : 1 (I) ENTER
DOS READY
DIR:1  ENTER
```

```
FILE DIRECTORY——DRIVE1  MANUAL--09/01/78
CHESS/CMD P    MENU/TXT    TEST/BAS P
DOS READY
DIR:1 (I) ENTER
```

```
FILE DIRECTORY——DRIVE 1  MANUAL--09/01/78
CHESS/CMD P    VIDSCAN/CMD I    MENU/TXT
TEST/BAS P
DOS READY
—
```

全部文件都被两个通行字保护：一个存取通行字，一个更新通行字，存取通行字和更新通行字可以相同，也可以都由空格组成。使用更新通行字可以把全部功能都赋予文件，用户可以撤消、重新命名、写入等等。但是使用存取通行字只给文件有限的功能，和在ATTRIB命令中用 PROT 参数指定一样。

保护级是一套等级，每低一级都暗示可以执行较低的功能。

级　别	功　能
KILL	全部功能
RENAME	重新命名、写、读、执行
WRITE	写、读、执行
READ	读、执行
EXEC	只能执行

建立文件时指定的通行字既是存取通行字又是更新通行字。（如果不指定通行字，就用八个空格的字符串作为存取和更新兼用的空缺通行字）。

一旦建立了文件以后，就可以把不同的字符串值赋予存取通行字和更新通行字。实际使用中，有两个不同的通行字是很有用的。例如，假定你有数据文件PAYROLL，要求雇员使用这个文件编制工资卡。如果这个文件是用空缺通行字（空格）建立的，使用命令：

```
ATTRIB  PAYROLL(ACC=EMPLOYEE, UPD=MANAGER,
PROT=READ)
```

这就只允许雇员读文件，而只有经理才可以修改文件。

要删除通行字（置为空格），就在通行字定义符的等号后面省略不写通行字。例如，

```
ATTIB  PAYROLL.  MANAGER(ACC=)
```

就把存取通行字置为空格。而更新通行字保持不变。

注意：要从磁盘 BASIC 存取一个文件，需要 READ 或更高级的功能。

CLOCK（显示实时时钟）

CLOCK [␣ (*param*)]
其中 *param* = ON 或 OFF，如果不指定参数，就认为是 ON

打入

CLOCK　[ENTER]

就把内部的实时时钟强行显示在电视显示器的最上面的一行上（显示的位置是53到60）。显示在这些位置上的其它字符都要被实时时钟取代。

时钟的显示通过“前台任务”一秒钟更新一次。换句话说，只要允许中断，TRSDOS就不管正在执行的“后台任务”(DISK BASIC, TAPEDISK 等等)是什么，都周期地实施中断，自动更新时钟的显示。

接通电源时，TRSDOS 处于时钟关闭状态。

要停止显示时钟的功能，就执行命令：

CLOCK　(OFF)　[ENTER]

关于实时时钟的详细情况请参看 TIME 命令。

COPY（复制文件）

COPY ␣ 文件名1 ␣ TO ␣ 文件名2

以新的文件名（文件名2）建立文件（文件名1）的付本。如果已有一个文件的标识符是文件名2，那么这个文件原有内容就要被冲掉。这个命令不改变文件1的内容。

必须至少有两个磁盘驱动器，才能把文件从一个磁盘复制到另一个磁盘上去。

例如：

COPY PAGE7/TXT:0 ␣ TO ␣ PAGE7/TXT:1

把0号驱动器上的文件PAGE7/TXT以相同的文件名/名扩展复制到1号驱动器上。

COPY OLDFILE/BAS.PDQ TO DEADFILE

把文件OLDFILE复制成名为DEADFILE的文件。注意文件OLDFILE有一个通行字保护着，而文件DEADFILE没有通行字保护。文件DEADFILE将建立在0—3号驱动器中的头一个没有写保护的驱动器上。

DATE（设置日期）

DATE ␣ *mm/dd/yy*

其中 *m m*是一个2位数字的月份标识符，*mm*=01到12

*dd*是一个2位数字的日期标识符，*dd*=01到31

*yy*是一个2位数字的年份标识符，*yy*=00到99

例如，假设日期是1978年8月3日，打入：

DATE 08/03/78 ENTER

这个命令重新设置实时日期数据。电源接通时，日期被置为00/00/00。每过24小时的时钟周期，日期就修正一次；实时时钟的日历还有计算28，29，30，31天的月份的逻辑。

DEVICE（询问设备情况）

DEVICE

这个命令无自变量和参数，它只列出当前已有的输入/输出设备：KI代表键盘，DO代表电视显示器，PR代表行式打印机。例如：

DEVICE ENTER

DIR（显示目录）

DIR [␣ : *d*] [␣ (*param*[, *param*…])]

其中：*d*是驱动器标识符，*d*=0,1,2,3,*d*取0时可以忽略不写

*param*可以是下列参数之一：

参数	意义
S	显示全部系统文件和非不可见文件
I	显示全部不可见文件和非系统文件
A	显示出全部被显示文件的磁盘空间分配

这个命令读出并显示出指定驱动器或所用驱动器上的文件目录。如果不指定参数，只显示出非不可见用户文件。

磁盘空间的分配是用下列符号表示的：LRL（逻辑记录长度），EOF（文件终端，即使用的最高记录编号），SIZE（用区(granule)度量，1区等于1/2磁道，即1.25K字节）。

例如：

DIR [ENTER]

显示出0号驱动器上的全部文件，这个命令的典型输出可能是：

```
FILF DIRECTORY…DRIVE 0      TRSDOS--10/03/78

VIDSCAN2/CMD   CLKAXFSS/BAS    SFLFCTRC/DVR
TBUG/CMD       EDTASM/CMD      GLOSSARY/BAS
LISTTER/BAS    TAPFDISK/CMD    KBFIX/CIM
DISKDUMP/BAS   GLOSSACC/BAS    VIDSCAN/CMD

DOS READY
-
```

DIR :1 (I, S) [ENTER]

显示出包括系统文件和不可见文件在内的全部文件。这个命令的典型输出可能是：

```
FILF DIRECTORY…DRIVE 1    MRNUAL  …09/01/78

BOOT/SYS SIP     DIR/SYS SIP    CHESS/CMD  P
MFNU/TXT         TEST/BAS   P

DOS READY
-
```

注意：某些文件旁边的P，表示这些文件的通行字不是空格字符串。

DIR (A) [ENTER]

给出0号驱动器上用户文件的磁盘空间分配。下面是一个典型的例子：

```
FILE DIRECTORY - - - DRIVE 0      TRSDOS - - 11/18/78

EDTASM/CMD          LRL=256/EOF=27/SIZE=6GRANS
RSM/CMD             LRL=256/EOF=18/SIZE=4GRANS
VHMTBUG/CMD         LRL=256/EOF= 0 /SIZE=2GRANS
SEQCHECK/TXT        LRL=256/EOF= 2 /SIZE=1GRANS
TBUG/CMD            LRL=256/EOF= 5 /SIZE=2GRANS
TAPEDISK/CMD        LRL=256/EOF= 2 /SIZE=1GRANS
CPRINT/BAS          LRL=256/EOF= 1 /SIZE=1GRANS
HMRSM/CMD           LRL=256/EOF=18/SIZE=4GRANS

DOS READY
-
```

如果荧光屏上不能显示出全部目录，就只显示出前12行。再按随便哪个键就能看到剩下的部份，每次增量 16 行。

DUMP（把内存的内容转贮到磁盘上）

DUMP ␣ *filespec*␣ (START=X'*aaaa*', END=X'*bbbb*', [, TRA=X'*cccc*'])

其中*aaaa*，*bbbb*，*cccc*都是 4 位十六进制地址

aaaa 是要转贮到磁盘上的机器语言程序或数据块在内存中的首址，*aaaa* 必须大于6FFFH

*bbbb*是数据块在内存中的终址，必须大于*aaaa*

*cccc*是转移地址，TRSDOS执行这个文件时，就从*cccc*开始。如果省略*cccc*，就使用 420DH，这是正常的重新进入 TRSDOS 的地址（即不重新初始化，在显示出DOS READY时重新进入TRSDOS）

如果已有这个文件标识符，这个文件标识符标识的文件原有的内容就被冲掉。

如果标识符中没有名扩展，TRSDOS 就自动把名扩展 CIM（内存映象）赋与该文件。一旦将机器语言程序转贮到磁盘上以后，有两种执行这个程序的方法：

（1） 只要打入：文件标识符 ENTER，TRSDOS 就把这个文件装入内存并从转移地址开始运行这个文件。

（2） 只要打入：DEBUG ENTER 和文件标识符 ENTER。TRSDOS把文件装入内存，然后进入DEBUG。这时，PC中是转移地址。此后，就可以单步执行（I 命令），也可以调用——单步执行（C命令），或者打入G ENTER 全部运行。

注意：带有名扩展/CMD的文件，只要打入文件名（不用带扩展），按 ENTER 键就可以装入内存并运行这个文件。TRSDOS把/CMD 作为空缺名扩展处理。

例如：

```
DUMP  GRAPHICS (START=X'7000',END=X'70A0',TRA=X'7000')
DUMP  DATA/CIM:1 (START=X'8000', END=X'B050')
```

KILL（删除文件）

KILL ␣ 文件标识符

这个命令删除指定的文件并开放这个文件所占用的空间供系统使用。如果这个文件标识符中没有驱动器标识符，TRSDOS 就去搜索第一个可能含有该文件的驱动器并企图删除 这个文件。若文件所在软盘处于写保护状态，TRSDOS就不能删除这个文件。

例如：

```
KILL  OLDFILE/BAS. PASSWORD
```

FREE（显示出各个驱动器上还没有被使用的空间）

FREE

这个命令没有自变量和参数。它显示所使用的各个盘上剩余的还没有被使用的空间的数量。用还能够建立多少个文件和还有多少个区未被使用来表示。（每个软盘上最多能容纳48

个用户文件，数据软盘有67个区可供用户文件使用。TRSDOS软盘只有44个区可供用户使用)。

例如：

```
FREE  ENTER
DRIVE  0 --  TRSDOS    37 FILES,    25 GRANS
DRIVE  1 --  TRSDOS    33 FILES,    27 GRANS
DOS READY
-
```

LIB（显示库命令）

```
LIB
```

这个命令不需要自变量和参数。这个命令显示出可用的 TRSDOS 系统的全部库命令，这些命令是装入十六进制 5200 到 6FFF 之间的。

例如：

LIB　ENTER

LIST（在显示器上列出文本文件内容）

```
LIST␣ 文件标识符
```

读出指定文件并把它的内容列在电视显示器上。因为 LIST 命令给出的是文件中数据的 ASCII表示。所以本命令中的文件必须是文本文件 (text file)。如果把 LIST命令用于非文本本件，那末显示器上充满的是 ASCII 和图形符号的毫无意义的序列。

文本文件包括：

(1) 用 ASCII 方式存储的 BASIC 程序，

(2) 由 BASIC 的顺序写入语句 (PRINT#n) 建立的数据文件，

(3) 汇编语言源码，等等。

在 LIST 命令执行期间，同时按下 SHIFT 和@键，直到列表暂停，显示器成暂时“冻结”。按下任何一个键都可继续列表。TRSDOS 只在列出一个完整的物理记录后才接受这种暂停信号。这就是为什么要保持按下SHJFT和@键的状态直到 TRSDOS “注意” 到暂停命令为止的原因。

例如：

LIST　PROG1/TXT

LOAD（装入机器语言文件）

```
LOAD ␣ 文件标识符
```

把指定的文件装进 RAM 并返回到TRSDOS控制状态。指定的文件所包含的必须是Z80目的码，一般是用 DUMP 或 TAPEDISK 命令建立的目的码文件。

LOAD通常用于把几个程序装入内存，所以它们都能被一个主程序调用。这个主程序是另一个机器语言程序或一个 BASIC 程序（当然，这些不同的文件都必须装入 RAM 的非复盖区）。

要装入辅助的目标代码程序并通过主目标代码程序来运行，可先装入辅助程序，然后打入程序名并按下 ENTER

例如：

```
LOAD  GRAPHICS
LOAD  DATA/CIM:1
```

PRINT（在行式打印机上列出文本文件）

```
PRINT ␣ 文件标识符
```

和 LIST 命令的功能相似，只是把输出送往行式打印机。指定文件必须是文本形式（ASCII形式）。

例如：

```
PRINT    SEQCHEK/TXT
PRINT    PAGE7/TXT:0
```

PROT（使用软盘的主通行字）

```
PROT [ ␣ :d] [ ␣ (param [, param…])]
  其中：:d 是驱动器标识符，d=0，1，2，3。如果没有给出驱动器
        标识符，就使用第一个驱动器
  param可以是下列参数之一：
```

参数	意义
PW	改变主通行字
UNLOCK	取消全部用户文件中的通行字
LOCK	把一个主通行字赋予各个用户文件

UNLOCK 和 LOCK 是互相排斥的参数。仅能使用其中之一。

这个命令改变指定驱动器内非系统文件的保护状态。要使用本命令，必须预先知道在FORMAT 或BACKUP 过程中赋给这个软盘的主通行字。要访问的盘必须没有写保护。

注意：TRSDOS 软盘的通行字是PASSWORD.

要改变主通行字，就指定PW为参数，要从全部用户文件中取消主通行字，就指定UNLOCK 作为参数。要把一个软盘的主通行字赋给全部用户文件，就指定LOCK作为参数。（此后主通行字就变成这些文件的更新通行字和存取通行字）

例如：

```
PROT  :1   (UNLOCK)   ENTER
```

打入这个命令以后，TRSDOS 就询问 1 号驱动器上软盘的主通行字。如果输入的通行字是正确的，TRSDOS就从这个软盘的文件中取消各用户赋予的通行字。

PROT (PW, LOCK)

在正确指出主通行字以后，TRSDOS 就请求用户输入新的主通行字。由于这个命令包含LOCK选择项，就把这个新的主通行字赋给了全部用户文件。

使用 PROT 命令的一个典型显示次序如下：

```
DOS READY
PROT(LOCK) [ENTER]
MASTER PASSWORD?      PASSWORD [ENTER]
DOS READY
DIR [ENTER]
```

```
FILE DIRECTORY --- DRIYE 0  TRSDOS--10/21/78
EDTASM/CMD P     RSM/CMD P     VIDSCAN/CMD P
VHMTBUG/CMD P    SEQCHECK/TXT P TBUG/CMD P
TAPEDISK/CMD  P  HMRSM/CMD  P
DOS READY
—
```

从这里看到，全部用户文件现在都用主通行字保护起来了。

RENAME（文件改名）

```
RENAME ␣ filename 1 [/ ext 1 ] [. PSW] [ : d] ␣ TO ␣
       filename  2 [/ext 2 ]
  其中 filename 1 ,filename 2 是TRSDOS文件名
       ext 1 ,ext 2 是名扩展
       :d 是驱动器标识符(d=0, 1, 2, 3)
       PSW 是一个通行字
```

这个命令改变一个文件的名称，由第一个文件名/名扩展改变为第二个文件名/扩展。注意，第二个文件名/扩展不要包含通行字或驱动器标识符。由于需要识别第一个文件，第一个文件标识符就要包含通行字和驱动器标识符。

这个命令不能更改文件的保护特征，也不能把文件移到另一个驱动器上去。从前的通行字、保护级和目录特性（不可见或非不可见）都赋给了重新命名的文件，并且这个文件仍留在同一个软盘上。

这个命令先检查文件名，看看预定的新名在同一个盘上是否与现有的文件名重复。如果重复，就取消这个命令并显示出错误信息。

例如：

RENAME MATHPAK TO MATHPAK/BAS

给文件名加了一个名扩展。

RENAME ABCDE/DAT TO ABCDEF/DAT

只改变文件名。

RENAME PAYROLL 1/TXT. GSR TO PAYROLL 2/TXT

改变了文件名，但自动保留通行字。

RENAME FILE 1:3 TO FILE 2

只改变了驱动器 3 上的这个文件名。

TIME（设置实时时钟）

TIME ␣ *hh* : *mm* : *ss*

其中 *hh*是 2 位数的时标识符

*mm*是 2 位数的分标识符

ss 是 2 位数的秒标识符

这个命令设置时钟。按通电源时，这个时钟置为00:00:00。

注意：TRSDOS采用一天24小时的时钟，23:59:59 以后，时钟又从 00:00:00 开始，日期增加 1 。

当前的时间存贮在十六进制4040—4046单元。只要允许中断，它的内容就通过实时时钟自动调正。

例如：

TIME 08:24:00

参看 DATE 和 CLOCK 命令。

VARIFY（写后自动读）

VARIFY [␣ (*param*)]

其中 *param*=ON或OFF；为ON时可以略去不写

VARIFY ENTER

命令 TRSDOS 核对用户的写磁盘操作（例如磁盘BASIC的文件写入）。要保证在磁盘写入过程中不丢失或不改变数据，可使用这个命令。例如复制（COPY）文件前，就可以使用这个命令。

然而，使用这个命令时，磁盘存取速度只有原来的一半。

打入：

VARIFY (OFF) ENTER

就取消了写后自动读出进行核对的命令。

注意：接通电源时，TRSDOS处在VARIFY (OFF) 状态。

这个命令不影响系统表和目录的写入，写入目录和系统表时随时都在进行核对。

五、 扩展实用程序

1. 磁盘操作系统实用程序

这是一些专用程序，严格地说，它们不是磁盘操作系统的一部份。用户可以调用这些程序进行某些很有用的工作。与系统程序和库命令相反，这些扩展程序可能使用十六进制地址6FFF以上的内存单元。因此，装入实用程序时，RAM中的任何程序都可能丢失。

BACKUP（复制软盘）

```
BACKUP [ ␣: d1 ␣ TO ␣: d2]
  其中：:d1  是源驱动器标识符
        d2  是目的驱动器标识符
        d1, d2=0, 1, 2, 3
```

如果省略驱动器标识符，BACKUP就提请用户同时打入源驱动器和目的驱动器的号码。

这个实用程序可以复制整个 TRSDOS 软盘，或数据软盘。用户可以使用任何两个驱动器来进行复制工作，也可以只使用 0 号驱动器，但要在 BACKUP 程序告诉你的时候更换源盘和目的盘。

如果目的软盘未经格式化，BACKUP 程序先把这个软盘格式化，封锁有缺陷的磁道，然后把源盘文件复制到这个软盘上去（如果因为封锁了一些磁道，目的软盘不能够容纳源盘上所有的数据，BACKUP 就拒绝复制）。

只有源盘的主通行字、软盘名称和预先已经格式化了的软盘的主通行字、软盘名称相符合时，BACKUP程序才接受已格式化的软盘，这时，BACKUP程序跳过格式化步骤，开始复制和核对过程。如果因为某些原因，BACKUP 拒绝接受一个软盘的话，就要用消磁器将这一软盘消磁，再重新复制。

例如：

```
BACKUP
BACKUP  :0  TO  :0
BACKUP  :0  TO  :1
```

下面是只使用 0 号驱动器的典型的复制次序：

```
DOS RERDY
BRCKUP :0 TO :0 ENTER
```

```
TRSDOS BACKUP UTILITY VER 1.2
BACKUP DATE (MM/DD/YY)? 10/08/78
〈INSERT SOURCE DISK〉
```

以后，BACKUP程序就按照需要提醒你插进源盘（原件）和目的盘（付本）。

在使用两个驱动器进行复制时，就不必交替地插取软盘了。

FORMAT（准备数据软盘）

```
FORMAT
```

这个实用程序用于准备数据软盘，使软盘包含的系统信息最少，而给用户存放程序和数据文件的空间最大。TRSDOS软盘有 44 个区（55K 字节）可给用户文件使用，数据软盘有67个区（83.75K 字节）可给用户使用。

注意：除了复制备份或进行格式化以外，只能在 1，2，3 号驱动器上使用数据软盘。

FORMAT给一个空白软盘（新的或消过磁的）记录上磁道和扇面的边界，然后初始化，加上目录和引导文件。在格式化过程中，TRSDOS 允许用户指定任何想要封锁的磁道，这样，就可以把这些磁道用于非 TRSDOS 文件。除非你有另外的（非 TRSDOS 的）方法存取这个软盘，否则不要封锁任何磁道。

FORMAT 程序将封锁所有有缺陷的磁道，以免在这些区域丢失数据。

如果在磁盘存取的过程中，发生 READ 错误，就要把这个软盘消磁并重新格式化。如果有损坏的磁道，FORMAT 程序就把这些损坏的磁道封锁，将其余部份仍然可用的软盘提供给用户。

若要封锁磁道，可以分别指定，或指定一个范围。例如：

1，3—5　　封锁1，3，4，5磁道

TRSDOS 是永远不会往被封锁的磁道上写入信息的。

下面是一个使用 1 号驱动器的典型的 FORMAT 顺序：

```
DOS READY
FORMAT [ENTER]
```

```
DISK FORMATTER UTILITY 2.1
WHICH DRIVE IS TO BE USED? 1 [ENTER]
DISKETTE NAME? MRNUAL [ENTER]
CREATION DATE (MM/DD/YY) ? 10/08/78 [ENTER]
MASTER PASSWORD? TRS80 [ENTER]
DO YOU WANT TO LOCK OUT ANY TRACKS? Y [ENTER]
WHICH TRACKS(0-34)? 1,6-8 [ENTER]
FORMAT THE LOCKED-OUT TRACKS? N [ENTER]
```

2. 辅助实用程序

TAPEDISK（把磁带文件复制成磁盘文件）

这个实用程序把SYSTEM（系统）磁带装入内存，然后把它转贮到磁盘上的指定文件中去（SYSTEM磁带是用编辑/汇编、查错程序建立的，或者是Radio Shack提供的SYSTEM磁带）。

因为TAPEDISK程序使用地址低于54F4H（十进制21748）的内存区，所以，不要打算用TAPEDISK将低于十六进制地址54F4的磁带文件装入内存。

注意：大多数的Radio Shack公司的SYSTEM磁带是为LEVEL Ⅱ使用而设计的，在磁盘BASIC程序中不能工作，因为两者使用的RAM区域不相同。

打入：

TAPEDISK [ENTER]

就可以装入并执行TAPEDISK程序，一出现提示符

?

就表示已经进入TAPEDICK程序了。任何时候，只要在当前的一行上出现提示符，就可以打入下列三个命令之一。

（1） 由磁带输入

```
C
```

这是启动录音机的命令（使用TAPEDISK程序时，应把录音机直接联接到TRS-80的磁带机插座上，不能联接到扩展接口的插座上去）。

打入：

? C [ENTER]

在文件输入以后，可以输入另外的系统磁带或者打入另外的命令。

（2） 转贮入磁盘

```
F ␣ filename [/ext] [ • Password]:d ␣ aaaa ␣ bbbb ␣ cccc
   其中：
      filename是一个TRSDOS文件名
      /ext 是一个任选的名扩展
      • Password 是一个任选的通行字
      :d 是所需要的驱动器标识符，d=0，1，2，3
      aaaa 是RAM中的起始地址
      bbbb 是RAM中的终址
      cccc 是执行此文件的入口地址
      所有地址都是十六进制的四位数
```

在打算把一个程序从RAM中转贮到磁盘上去时，就打入F命令。例如，假设一个程序在RAM中的地址是7000到70FF，入口是700A，

打入：

? F USRCODE/CMD:1 7000 70FF 700A [ENTER]

当这个程序转贮完以后，又显示出提示符。

（3） 返回 TRSDOS

```
E
```

用这个命令可以经过正常的再入口（不重新初始化）返回 TRSDOS。

下面是一个典型的 TAPEDISK 顺序：

```
DOS READY
TRPEDISK [ENTER]
? C
? F GRAPHICS/DAT:0 6A00 6FFF 402D [ENTER]
? E [ENTER]
DOS READY
_
```

DISKDUMP/BAS（检查磁盘文件）

这是一个磁盘 BASIC 程序。必须先装入DISK BASIC，再装入 DISKDUMP/BAS 才能执行这个程序：

```
BASIC [ENTER]
```

```
HOW MANY FILS? [ENTER]
MEMORY SIZE? [ENTER]
RADIO SHACK DISK BASIC VERSION 1.1
READY
RUN "DISKDUMP/BAS [ENTER]
```

使用DISKDUMP 程序可查看任何用户磁盘文件的内容。此程序有助于你使用各种随机的或顺序的磁盘输出语句进行试验，也有助于你对磁盘 I/O 程序进行调试查错。

这个程序将把被检查的内容送到行式打印机上打印出来。如果没有连接行式打印机，就把全部的 LPRINT 语句（第 170，240，250 行）改为 PRINT，然后把 160 行改为

160 GET 1, SN

这个程序提请你打入文件名，然后打入你需要检查的扇面号。也可以不打入扇面号，只简单地按 [ENTER] 键，那末，就从扇面 1 （该文件的第一个物理记录）开始逐扇顺序检查。

如果指定的扇面号大于 EOF 号（文件结束），也不给出错误信息，而认为你指定的扇

面上各字节都是0。

扇面内容的显示是一次显示16个字节，先以十六进制代码显示出这16个字节，然后是对应的ASCII代码。ASCII代码用“¦”符号括起来，其中的句点“.”代表该字节内容不是字母数字的ASCII代码。

下面是一个典型的DISKDUMP过程：

```
                        SECTOR DUMP UTILITY 1.1

FILESPEC : SEQCHECK/TXT

SECTOR NUMBER (OR'ENTER' FOR NEXT SECTOR) : ENTER
```

```
FILESPEC: SEQCHECK/TXT          SECTOR: 1
 0 35  20 43  4C 53 3A 20 43  4C 45  41  52  20  31 30  30 ¦5 CLS:CLEAR 100¦
16 30  0D 31  30  20 41  24 3D 49 4E 4B 45  59  24 3A 49 ¦0.10 A$=INKEY$:I¦
32 46  41 24  3D 22 22  54 48  45 4E 31  30  0D 31 35  20 ¦FA$=""THEN10.15¦
48 49  46 20  41  24 3D 22 40  22 54  48  45  4E 20 32  35 ¦IF A$="@"THEN 25¦
64 0D 32  30  20  50 52  49 4E 54 41  24  3B 3A 42 24  3D ¦.20 PRINTA$;:B$=¦
80 42  24 2B 41  24 3A 47  4F 54 4F 31  30  0D 32 35  20 ¦B$+A$:GOTO 10.25¦
96 50  52 49  4E 54 3A 50 52  49 4E 54  22  44  41 54  41 ¦PRINT:PRINT"DATA¦
112 20 49 4D 41  47 45  20 57  49 4C 4C 20  41  53 20  4F ¦ IMAGE WILL AS O¦
128 4E 20 4E 45  58 54  20 4C 49 4E 45  2E 20 28 22  43 ¦N NEXT LINE.("C¦
144 48 52 24  28  39 31  29 22  3D 42  59  54  45  20 44  45 ¦HR$(91)"=BYTE DE¦
160 4C 49 4D 49  54 45  52 29  22 0D 33  30  20  46 4F 52 ¦LIMITER)".30 FOR¦
176 49 25 3D 31  20 54  4F 20  4C 45  4E 28  42  24 29  3A ¦I%=1 TO LEN(B$):¦
192 20 50 52 49  4E 54  20 41  53 43  28  4D 49  44 24  28 ¦PRINT ASC(MID$(¦
208 42 24 2C 49  25 29  29 43  48 52  24  28  39  31 29  3B ¦B$, I%)) CHR$(91);¦
224 3A 4E 45 58  54 0D 33  35  20 50  52  49  4E 54 0D 35 ¦:NEXT. 35 PRINT.5¦
240 30 20 4F 50  45 4E 22  4F 22 2C 31  2C  22  54 45  53 ¦0 OPEN"O", 1, "TES¦
```

六、 TRSDOS的技术资料

1. 内存的组织

TRS-80磁盘操作系统由1K ROM的常驻CIO(Character-orientedI/O)驱动程序和4K RAM的驱动程序、调度程序、表、指针等组成。LEVEL Ⅱ BASIG也使用这个ROM常驻CIO驱动程序，因此，它是LEVEL Ⅱ BASIC的12K ROM的一部份。

由于LEVEL Ⅱ和DISK BASIC向上兼容，两种BASIC都需要加上0.5K RAM.所以用户RAM从5200（十六进制）开始，结果，在16K机器中用户RAM只有11.5K。

注：从7000（十六进制）开始的内存中就完全遇不到TRSDOS和磁盘BASIC的代码了。

TRSDOS由常驻系统程序和随需要而从磁盘上装入的几个互相覆盖的程序组成（例如，打开文件或关闭文件）。

系统采用模块设计。4K RAM 的最低处是系统入口矢量，紧接着是中断管理、磁盘文件管理、任务调度、通用常驻系统程序。系统缓冲器和覆盖区在 4K RAM 的最后部份。

因为所有主要的系统命令都是按需要从磁盘中以实用程序（库命令和扩展实用程序）的形式装入内存的，所以，很容易扩充TRSDOS系统的能力而不影响RAM存储器的需要量。

2. 磁盘的组织

每一个 TRSDOS 系统软盘上都有 TRSDOS 系统程序、实用命令库、文件目录和系统表。

最小的系统软盘的开头有目录信息（占整个一个磁道）和占半个磁道的 TRSDOS 引导程序和其它信息。这就是说，每个TRSDOS软盘都可以自我装入，(Self-loading)，虽然这个软盘上不一定有TRSDOS 系统。这样做就可以阻止计算机引导只有用户数据文件的磁盘。

实用命令库在磁盘上是任意选用的。由于实用命令程序并不经常需要，所以，对于多驱动器用户来说，把驱动器 1 到驱动器 3 中使用的磁盘格式化就非常方便。这种“数据软盘”含有的系统代码极少，而给用户文件留下了大量的空间。文件长短的最大限度受软盘的物理尺寸所限，因为一个文件必须完整地存放在一个软盘里面。

软盘是单面软盘，有 35 个信息磁道，每个磁道有 10 个扇面，每个扇面 256 个字节。参看第二章。

通常，数据的读/写操作只能从扇面的边界开始，并且必须严格组成256个字节。TRSDOS 使得用户只要做少量的工作就可得到最大的灵活性：这就是自动地把全部文件组合成块或分解成块，以用户指定的逻辑记录长度进行存取。如果需要还可以跨越两个扇面。

这种系统磁盘文件结构，在整个磁盘上把一个文件自动地分成几个小段，使磁盘空间得到充分利用。系统又把这些小段联结成逻辑上相邻的文件。而用户没有必要知道文件的物理位置。这种结构省去了耗费时间的磁盘装填操作(Disk-packing)。

3. 文件的结构

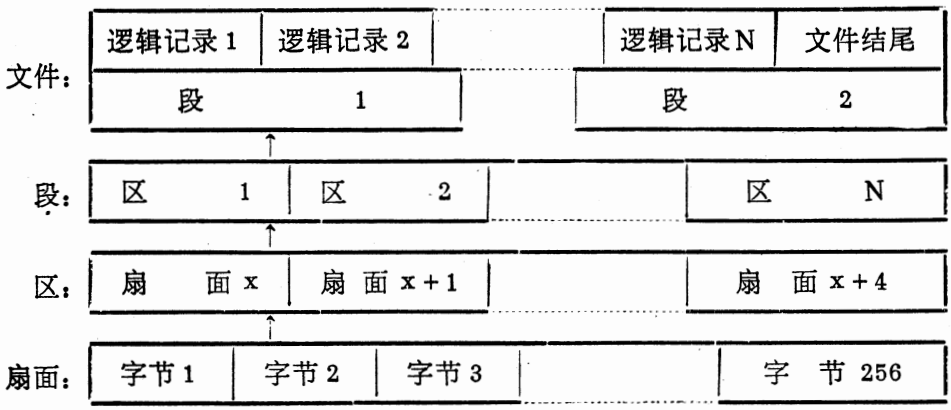
TRSDOS 文件由一段(Segment)或多段存储空间组成，每段由一个到 32 个物理上相邻的存储区（Granale）组成。区是最小的可分配的存储单元，由 5 个扇面组成（1.25 K字节），参见下图。

文件是以区为单位加长的，因此，每个文件末尾总还有一些未被使用的空间。可以把小文件放到这些空间内，和原有的文件相邻。

其作用是减少文件（多次加长的文件）中出现“系统颠簸”的次数。一个全扇面配置的系统不具备这种优点。

每次扩展磁盘文件（初始建立文件或加长文件）都可先多分配给该文件若干个区。这取决于文件总长度、磁盘空间数量、饱和度等因素。在关闭文件时，就把这些额外的区以及含有文件结束标志的区以后的所有区都退回给系统。

TRSDOS 文件的结构



LRN：逻辑记录编号，用来指定一个单独的、用户定义的逻辑记录。这种逻辑记录是磁盘输入/输出过程中可以寻址的最小单位（物理记录是实际读出和写入磁盘的单位）。

文件：一组逻辑记录，可以用 TRSDOS 命令寻址的最大信息单位。

扇面：由相邻的 256 个字节构成的一个物理记录。

区：可分配给文件的最小存储单位。

4. 汇编语言输入/输出(I/O)的系统程序

这些资料是为编写汇编 I/O程序的用户准备的。其中对调用次序和每个必需的 I/O程序的参数都作了解释。我们假定用户已熟悉 Z-80 的机器代码。

下列符号在本节作为标准符号使用：

HL⇒*xxxx* 寄存器 HL 中存有机器码格式的地址 *xxxx*（如果地址 *xxxx*＝34B2H，则寄存器中的值是H＝34，L＝B2）

DE⇒*xxxx* 寄存器 DE 中存有机器码格式的地址 *xxxx*（如果 *xxxx*＝5AF1H，则寄存器的值是D＝5A，E＝F1）

B＝*xx* 寄存器B中存有二进制形式的数值 *xx*。如果 *xx*＝64（十进制），则B＝40H

A＝*xx* 寄存器A中存有二进制形式的数值*xx*。如果*xx*＝127（十进制），则A＝7FH。寄存器 A 用来存放 I/O 调用的TRSDOS的错误代码。本篇末尾有错误代码及其含意表。

Z＝OK 如果从系统程序成功地返回，就把零标志位Z置位(OK)

X'*nnn*' 以十六进制表示的硬件RAM地址（例如，402D是X'402D'）

LRL 逻辑记录长度，只能是 1-255 字节。可以任意定义一个记录的长度，但最大不超过 255 个字节，长度为 0 是物理记录的特殊情况，专指LRL＝256 字节。

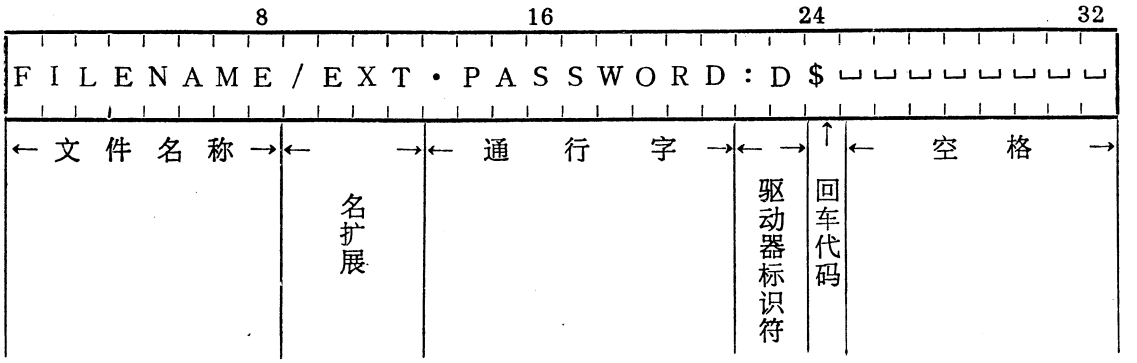
BUFFER RAM中用户指定的 256 个字节，为TRSDOS 整扇面读出或整扇面写入而设置。如果 LRL＝0，在 I/O以前和以后，这个区域归用户管理。如果 LRL 在 1 到 255 之间，由 TRSDOS 管理这个区域。使用逻辑记录过程中，这个区域不能变动。

UREC 用户记录：由用户指定的用户逻辑记录区，它是一串相邻的 RAM 字节，它的长度必须等于 LRL。占用的区域和 BUFFER 占用的区域不能重迭。

数据/设备控制块

DCB (Date/device Control Block) —数据/设备控制块定义为用户赋给的 RAM 中相邻的 32 个字节。在命令OPEN 执行以前和 CLOSE 执行以后，DCB是“左满”的、压缩了的（没有空格）ASCII字符串，和标准的 TRSDOS 文件标识符相同。

32字节 DCB 的内容是：



注：/EXT，.PASSWORD，:D是选择项

$ 代表回车 (X'0D')　␣代表空格 (X'20')

执行OPEN命令时，DCB的解释

Z-80 RAM 中的格式为低位/高位，即最高有效字节跟在最低有效字节后面。（即，地址=7CC8在RAM中是C8 C7）

地址	长度	解释
DCB+0	3	备用
+3	3	物理缓冲器地址(低位/高位)
+5	1	在当前记录的末尾设置间隔符
+6	1	文件驱动器编号驻在地
+7	1	备用
+8	1	在最后的物理记录上设置最后的间隔符为EOF
+9	1	LRL (逻辑记录长度)
+10	2	NRN (下一记录编号——打开文件时置成 X'0000'—低位/高位)
+12	2	ERN (结束记录编号——文件最后一个记录的编号—低位/高位)
+14	18	备用

NRN：下一记录编号，定义为由下一个系统调用 READ/WRITE 来读/写的记录的编号。它在每一次系统调用后就自动加 1 。为了处理随机文件，使用 POSN调用命令把 TRSDOS 指向你希望指向的下一记录。

ERN：结束记录编号，是当前文件中的最后一个记录的编号。在执行 CLOSE命令期间把它放入目录，所以，如果期望它是正确无误的，用户必须在把记录加进文件以后，关闭这个文件。此值也可以用来确定文件末尾的位置，于是能把新的记录加到这一文件的末尾。要确定一个文件的末尾，可以使用带有记录号 NRN+1 的

POSN。POSN 将在后面讲述。

TRSDOS 中的物理记录和逻辑记录

定义一个物理记录为磁盘的一个扇面。磁盘的一个扇面容纳 256 个用户数据字节。术语“区”（Granule）的定义是 5 扇磁盘空间。磁盘的 35 个磁道的每一磁道有 2 个区。区是 TRSDOS 可以分配的最小空间量。为了编程方便，文件中的物理记录编成 0 号到 N 号。于是，文件中最大记录号（N）是分配给文件的区数的 5 倍减 1（(5*G)－1）。所有的TRSDOS区分配都是在写入文件时按需要进行分配，不是在建立文件时进行分配。

下面是每个 $5\frac{1}{4}$ 英寸磁盘的空间配置表。

字 节	扇 面	区	磁 道	磁 盘
256	1	—	—	—
1280	5	1	—	—
2560	10	2	1	—
89600	350	70	35	1

逻辑记录由TRSDOS用户来定义，长度可以为 1 个字节到255 个字节之间的任意数值。一旦文件用指定的 LRL （逻辑长度）打开，直到关闭为止，长度不变。要改变一个文件的 LRL 就必须关闭这个文件，再用新的LRL 打开这个文件。

每次打开文件都要设置一个固定的逻辑记录长度。为了最大限度利用磁盘的空间，TRSDOS 将把逻辑记录“组合”成一个物理记录。

“组合”（Blocking）是把多个逻辑记录放进一个物理记录内。例如，四个 64 字节的逻辑记录正好组成一个 256 字节的物理记录。为了在使用下一个物理记录之前把上一物理记录填满，TRS磁盘操作系统会把一个逻辑记录分成两部份（即，跨记录）。在物理记录的长度恰好不是逻辑记录长度的倍数时，就会发生这种情况。

如果用户希望自己装填，在调用 INIT/OPEN 时，可以把逻辑记录的长度指定为 0 字节，然后，用户就必须自己管理 256 字节的物理记录缓冲器的内容。如果LRL＝0，TRSDOS就不去管用户的逻辑记录，这种特殊情况下，TRSDOS 只把物理记录读入缓冲器或把整个缓冲器内容写入物理记录。

基本TRSDOS I/O程序的调用

与处理文件 I/O 有关的基本的 TRSDOS 程序有八种：

INIT	在目录中建立新文件并打开这个文件，但不进行区的分配
OPEN	打开一个已经存在的磁盘文件
POSN	为读/写特定的逻辑记录定位
READ	从磁盘上或从物理缓冲器中把一个逻辑记录读入RAM
WRITE	把一个逻辑记录从RAM 中写入磁盘或物理缓冲器
VERF	写入然后校验，方法是读回写入的数据并和RAM 中的原始数据进行比较。只适用于LRL＝0的物理记录。
CLOSE	关闭一个打开的文件
KILL	关闭一个文件并从目录中删去这个文件

下面讨论上面列出的每个程序并详述调用方法，注意，调用这些系统程序都要用到寄存

器F，在返回以前不要恢复它原来的数值。为了正确使用此资料，要把下面叙述全部读完，对于不明白的地方再参考其他资料把它们都弄明白。如果你成功地做到这一点就会发现，TRSDOS是程序设计的良好工具。这里提供的转移矢量和说明专门用于TRSDOS VERSION 2.1（即，版型2.1的TRS磁盘操作系统），再版的TRSDOS可能改变一些地址和说明。

（1） INIT（转移矢量=X'4420'）

INIT为TRSDOS提供即将建立的新文件进入目录的入口点，并为此文件打开DCB。INIT扫描目录，寻找DCB中给定的文件标识符名称，如果找到这个文件标识符名称，INIT就简单地打开这个文件付诸使用；如果找不到这个文件标识符名称，就用这个文件标识符为名建立一个新文件。

入口：　HL⇒BUFFER
　　DE⇒DCB
　　B⇒LRL
　　CALL 4420H

出口：　Z=OK
　　C　如果新文件建立了，进位标志C为ON（置位）
　　A=TRSDOS 错误代码（错误代码列在本章末尾）

（2） OPEN（转移矢量=X'4424'）

OPEN是打开目录中已有文件的DCB的方法。在进入OPEN以前，DCB中必须含有被打开文件的文件标识符。

入口：　HL⇒BUFFER
　　DE⇒DCB
　　B⇒LRL
　　CALL 4424H

出口：　Z=OK
　　文件不存在，则Z=0
　　A=TRSDOS 错误代码

（3） POSN（转移矢量=X'4442'）

POSN确定一个文件的位置以便读/写一个随机选择的逻辑记录。因为POSN与逻辑记录有关，因此，要进行适当的计算以确定存有该数据的物理记录。后面有READ的POSN把一个记录传送给RAM；后面有WRITE的POSN从RAM中传送出一个记录。

注意，要读一个文件的第一个逻辑记录，在定位这个文件时令逻辑记录为0。为了把一个新记录加到一个文件的末尾，就使用记录号ERN+1定位在这个文件末尾。

入口：　DE⇒DCB（必须预先打开）
　　BC=欲定位的逻辑记录编号
　　CALL 4442H

出口：　Z=OK
　　A=TRSDOS 错误代码

（4） READ（转移矢量=X'4436'）

如果LRL>0，READ就把逻辑记录传送到地址为UREC的RAM中去，这个逻辑

记录的编号在DCB中是NRN，长度LRL在打开文件时确定。被传送的这个记录来自打开时定义的RAM缓冲器。如果TRSDOS必须读入新的物理记录以满足需要，它就自动读入新的物理记录。只要需要就安排“跨”逻辑记录。传送完成以后，READ将DCB中的NRN自动加1。为了第一次READ（读）第一个记录，INIT/OPEN把NRN置0，即令NRN=X‘0000’。

如果LRL=0，READ就把一个物理记录从磁盘文件送入RAM缓冲器。这个缓冲器是在打开时定义的。HL寄存器没有使用。和上面一样NRN自动加1。

入口：　HL⇒UREC(LRL≠0时)，若LRL=0，则不用。
　　DE⇒DCB
　　CALL 4436H

出口：　Z=OK
　　A=TRSDOS错误代码(EOF=X‘1C’或X‘1D’)（对EOF或NRF，参看错误代码28、29）

（5） WRITE（转移矢量=X‘4439’）

如果LRL＞0，WRITE从地址为UREC长度为LEL（打开文件时定义的）的RAM区域传送出一个逻辑记录，把这个记录送入打开文件时定义的RAM缓冲器。如果为满足需要，TRSDOS必须写入一个物理记录，它就自动去进行。TRSDOS将根据需要处理“跨”记录。在INIT/OPEN时，DCB中NRN值置为X‘0000’，所以就写第一个记录。每一逻辑记录传送完以后，DCB中的NRN值就加1。

如果LRL=0，WRITE就把一个物理记录从RAM缓冲器传送入一个磁盘文件，安放的位置是DCB中的NRN。RAM缓冲器只在INIT/OPEN时定义。WRIT E后，DCB中的NRN值和上面一样自动调正。

入口：　HL⇒UREC（LRL≠0时）
　　若LRL=0，则不用
　　DE⇒DCB
　　CALL 4439H

出口：　Z=OK
　　A=TRSDOS错误代码

（6） VERF（转移矢量=X‘443C’）

VERF和WRITE之间的差别只是VERF先把一个物理记录写入磁盘，然后读回，放入一个不由用户确定的专用的TRSDOS的RAM区，此后，逐字节比较这个专用区和原来写入缓冲器的内容，以保证写入记录是正确的。

入口：　HL⇒和上述WRITE相同
　　DE⇒DCB
　　CALL 443CH

出口：　Z=OK
　　A=TRSDOS错误代码

（7） CLOSE（转移矢量=X‘4428’）

CLOSE在TRSDOS做完最后的处理后关闭文件。程序结束前关闭每一个打开着的文

件是非常重要的。如果不关闭文件的话，已写入新记录的文件的目录入口是不正确的。这里不再介绍别的情况。但是，为了管理好文件，完成全部的“内务处理”对TRSDOS是非常重要的。

入口：　　DE⇒DCB
　　CALL 4428H

出口：　　Z=OK
　　A=TRSDOS错误代码

（8） KILL（转移矢量=X‘442C’）

KILL删除已打开文件的目录入口，然后关闭DCB。原有文件占用的磁盘空间现在可重新用于别的目的了。其他方面，KILL和CLOSE相同。

入口：　　DE⇒DCB
　　CALL 442CH

出口：　　Z=OK
　　A=TRSDOS错误代码

补充资料

这里再给出几个可能令人感兴趣的程序和地址。要特别注意错误处理程序。它并不进行错误的校正，只是在电视显示器上显示出TRSDOS错误信息。

（1） CALL 402DH 在程序末尾正常返回TRSDOS

（2） X‘4318’ 存放最后打入的TRSDOS命令的64字节缓冲器的地址。要在程序执行（运行）时，把打入的专用参数译码，就用到这个地址。

（3） 若HL⇒8字节缓冲器，那末
　　CALL 446DH 将时间变成8个字节ASCⅡ格式：HH:MM:SS返回
　　CALL 4470H 把日期变成8个字节ASCⅡ格式：MM/DD/YY返回

时间和日期的二进制形式在TRSDOS RAM中被分配在下列单元内：

X‘4040’　　时钟，实时时钟脉动节拍，25ms
X‘4041’　　时刻，二进制，三字节，秒、分、时
X‘4044’　　日期，二进制，三字节，年、日、月

（4） 在电视显示器上显示TRSDOS错误代码

```
CALL 4420H  系统I/O调用的例子。任何调用都如此。Z标志不置位表示I/O出错
JR Z, OKGO  若没有错误，跳过错误信息显示
OR 80H      某种（任意的）详细说明的错误信息。寄存器A已存有一行信息显示的恰当代码
CALL 4409H  在电视荧光屏上显示错误信息
……          这里是用户对某种错误进行纠正的代码（程序）
……
OKGO        程序由此继续执行
```

5. TRSDOS的错误代码（用寄存器A返回）

编　号	错误原因	错　误　说　明
00	—	无错
01	MD	读出标题时奇偶错误
02	D	读出时查找错误
03	XK	读出时丢失数据
04	MD	读出时奇偶出错
05	FMD	读出时找不到数据记录
06	P	企图读系统数据记录
07	P	同上
08	UP	设备不适用
09	MD	写入标题时奇偶错
10	D	写入时查找错误
11	XC	写入时丢失数据
12	MD	写入时奇偶出错
13	FMD	写入时找不到数据记录
14	XD	在磁盘驱动器上写入失败
15	UPX	写入被保护的软盘
16	PS	非法的逻辑文件编号（DCB不好）
17	MPDS	目录读出错误
18	MPDS	目录写入错误
19	UP	非法的文件名（DCB不好）
20	MPDS	GAT（区分配表）读错
21	MPDS	GAT写错
22	MPDS	HIT（无用信息索引表）读错
23	MPDS	HIT 写错
24	UP	目录中无此文件
25	UP	拒绝文件存取（违反保护）
26	UP	目录已满（最多48个文件）
27	UP	磁盘已满（最多70个区）
28	P	遇到EOF（文件末尾）
29	P	NRF（找不到文件）超出文件范围
30	UP	目录已满，不能延长文件
31	UP	找不到程序
32	UP	指定的驱动器号为非法
33	UP	无设备空间可用于新设备
34	MPUS	装入文件格式错误，不是程序
35	XCS	内存故障
36	PUXC	企图装入ROM存储器
37	P	试图非法存取被保护的文件
38	UP	文件还未打开
39—62		还未定义，保留
63	P	未知错误代码

可能的错误原因代码的解释：（上表第2列）

C	TRS-80 CPU 故障	P	用户程序错误
D	磁盘驱动器故障	S	TRSDOS故障，再引导(Reboot)
F	磁盘未格式化	U	用户过程错误
M	磁盘介质故障	X	扩展接口故障

七、 磁　盘 BASIC

1. 引　言

磁盘 BASIC 是对LEVEL Ⅱ BASLC的扩充，使得 BASIC 程序和数据可以从磁盘上输入/输出。磁盘 BASIC 是存在 TRSDOS 软盘上的内部信息文件，名字叫 BASIC，名称的扩展是/CMD。

DISK BASIC程序装入RAM后，就自动地控制了ROM中的 LEVEL Ⅱ BASIC 程序，可使用大部份LEVEL Ⅱ 的子程序和附加的程序。因为原设计LEVEL Ⅱ 就与 DISK BASIC 向上兼容（查看 LEVEL Ⅱ 的内存图，特别是 37DE—37EC 一段，就可看到这一点）。

装入后，磁盘 BASIC从地址 5200（十进制 20992）开始大约占用 5.8k 字节的 RAM。

为了装入并运行磁盘 BASIC 程序，先要接通磁盘操作系统（参看“系统操作”）。于是，显示出

DOS READY

现在，打入

BASIC [ENTEN]

TRSDOS 就把 BASIC 程序调入 RAM 并开始“初始对话”，即告诉 BASIC 如何按用户的要求，组织内存的一组问答。

首先问：

HOW MANY FILES? —

按照要在同一时间中打开（使用）磁盘文件的最大数目来回答，回答可以是 0 —15之间的任何数值。

（存在磁盘上的每个程序和每组数据都看做“文件”。实际上，包括系统软件在内，磁盘上的每件信息都以文件的形式存放）。

输入的数字告诉 BASIC 程序建立多少个 “I/O 缓冲器” 和 “数据控制块”。如果同时使用几个文件，就需 要有几个缓 冲器。 每个缓冲器在用户 RAM 中占用 290 字节（缓冲器 256 个字节加上 34 个字节作数据控制块〔DCB〕）。因此，尽量不要输入太大的不必要的数字。

如果不输入数字，简单地压下 [ENTER] 键，BASIS 就建立三个文件缓冲器，给用户同时使用。

磁盘 BASIC 自动建立一个缓冲区用于BASIC程序的输入、存储和归并，这个缓冲区在用户要求建立的任意数据文件缓冲器之后。不管用户如何回答 FILE? 的问题，为了程序的 I/O，这个缓冲区总是要有的。

假设你要使用两个文件，一个用于输入数据，一个用于输出数据，因而必须用“2”回答问题 FILE? 如果一次只打开一个文件， 你实际上只需要保留一个文件缓冲器/控制块就够了。

例如：

HOW MANY FILES? [0] [ENTER]

BASIC就不建立磁盘文件用的I/O缓冲器，所以你不能打开文件，然而有最大的RAM供用户程序使用。

HOW MANY FILES? 15 ENTER

告诉BASIC建立15个I/O缓冲器和控制块，可以同时打开15个文件，然而，这就把可用内存减少 15 * 290＝4350 字节。

HOW MAMY FILES? ENTER

告诉BASIC同时可打开3个文件。

回答问题FILES? 以后，BASIC就问

MEMORY SIZE? —

你要把使用和运行BASIC程序的最高内存地址（十进制形式）打入进去。在回答地址以上的内存，将要被BASIC系统保护起来。

为什么要保护内存

可以把机器语言程序或子程序装入内存高区。然后经专门定义的USRn函数用磁盘BASIC系统存取这些程序，或经SYSTEM命令存取这些程序。可以用SYSTEM命令从磁带上装入机器语言的程序，也可在DOS READY状态用LOAD命令装入，或者使用DEBUG或BASIC系统的POKE命令一次置入内存的一个字节。如果不保护内存，这种程序在运行BASIC语句时就被冲掉。

例如：

MEMORY SIZE? 32000 ENTER

BASIC系统就保护32000以上的内存。如果内存是16KRAM，就表示有32767—32000＝767个字节被保护起来用于存储机器语言程序。

如果不需要保护内存，就只按 ENTER 键，不用输入任何数字。

MEMORY SIZE? ENTER

就有最大的RAM给BASIC使用。

参看内存分配图，以熟悉TRS-80的各种内存结构(16K，32K，48K)中的十进制地址。

回答问题MEMORY SIZE? 以后，就显示出：

```
RADIO SHACK DISK BASIC VERSION 1.1
READY
>—
```

你现在可在磁盘BASIC控制下进行工作了。

要跳出BASIC系统，返回DOS READY状态，就打入

CMD"S" ENTER

这是不经系统初始化，正规返回DOS状态的方法。如果RAM中有BASIC程序，这样返回就会使程序丢失。所以在使用CMD"S"以前要把程序存到磁盘或磁带上去。

2.对LEVEL Ⅱ BASIC的扩充

磁盘 BASIC 相对 LEVEL Ⅱ 而言，增加了许多和磁盘无关的特色。先简列如下，以后按字母次序详细叙述。

&H	十六进制常数前缀记号
&O	八进制常数前缀记号
CMD"D"	启动并调入实时查错程序
CMD"R"	开中断（起动实时时钟）
CMD"S"	正常返回 TRSDOS（转移到 EXIT 程序）
CMD"T"	关闭中断（关闭实时时钟）
DEF FN	定义一个 BASIC 语句函数
DEF USR	定义一个外部机器语言程序入口
INSTR	字符串内函数，在目标串内查找子字符串。
LINE PRINT	由键盘输入一行
MID$=	替换目标字符串的一部份（用在等号左边）
TIME$	由实时时钟取出时刻和日期数据
USRn	调用外部程序(n=0,1,2,…9)

1） 盒式录音机的使用

BASIC使用盒式录音机输入/输出以前，必须用CMD"T"命令关闭中断。这是因为盒式录音机是时敏的(timing—sensitive)，如果每 25毫秒中断一次就不能工作。盒式录音机工作完成之后，执行语句 CMD"R" 就可以重新打开中断。

磁盘 BASIC 控制下，CLOAD命令不需要文件名，故不能用文件名识别几个磁带文件。CLOAD 总是把遇到的磁带上的第一个文件输入。然而，CSAVE 仍需要文件名。在磁盘 BASIC 控制下用 CSAVE 命令存储的程序，可由 LEVEL Ⅱ 的 CLOAD "文件名" 命令识别装入。

在 LEVEL Ⅱ 控制下使用的CLOAD? 命令(CLOAD核实) 是把RAM中的一个BASIC 程序和磁带上的程序进行比较。在磁盘BASIC 下用CLOAD? 的命令，不能和使用LEVEL Ⅱ 存在带中的程序进行比较，它只能与在磁盘 BASIC 下存在带中的程序进行比较。

2） 错误信息

出现错误时，磁盘 BASIC 不只是显示缩写，可以 "拼写出" 完整的错误信息，这样就省得用户再去查找了。

例如：

```
ERROR (14)    ENTER
```

磁盘 BASIC 的响应是：

```
OUT OF STRING SPACE
```

注意：用来模拟出错条件的ERROR 函数，只能用非磁盘错误代码才能工作。

3） &H和&O（十六进制和八进制常数）

使用十六进制或八进制常数往往比用十进制更方便。例如，用十六进制形式更容易由键

盘输入内存地址和字节数值。&H和&O 就可以让用户把这样的常数引用到程序中去。

&H和&O要作为数的前缀，数应紧接着写在后面，格式是：

&H *dddd*

这里的 *dddd* 是十六进制常数 0，1，…9，A，B，…F 组成的一位至四位数

&O *ddddd*

这里的 *ddddd* 是八进制数0，1，…7的序列，&O *ddddd* 总要小于/等于177777（十进制）

注：前缀&O中可以省略"O"，因此，&O *ddddd*＝&*ddddd*。

常数经常表示为有符号的整数。

任何十六进制大于&H7FFF 的数或八进制大于 &O 77777 的数都认为是负数，以下表说明。

八进制数	十六进制数	十进制数
&1	&H1	1
&2	&H2	2
&77777	&H7FFF	32767
&100000	&H8000	－32768
&100001	&H8001	－32767
&100002	&H8002	－32766
&177776	&HFFFE	－2
&177777	&HFFFF	－1

十六进制和八进制常数不能用来响应INPUT提示符，也不能放在DATA语句中。通常，必须把十六进制或八进制常数放在括号内，防止发生语法错误。

例如：

```
PRINT  &H5200, &O51000
```

显示出等于上述两个常数的十进制数（都等于20992）。

```
POKE  &H3C00, 42
```

把十进制数42（星号的ABCII代码）放入地址为 3C00的显示内存单元中去。

```
100  FOR  I=(&H3C00)  TO (&H3FFF)  STEP  (&H40)
200  IF  A=(&H37E8)  THEN  A=A+1
300  POKE  A%, (X%  AND  &HFF)
```

这段程序把 X% 的高位有效字节取出来，把结果 POKE 到单元 A% 中去。

4) CMD"D"（执行查错程序）

```
CMD"D"
```

执行这个语句使TRSDOS调入并执行查错程序（请看TRSDOS的DEBUG命令）。查错程序在内存中安排在磁盘 BASIC 之下，所以 BASIC 程序不受影响。

如果想不经过重新初始化就返回 BASIC，可按

G　[ENTER]

当 READY 信息出现以后，就可以继续运行 BASIC 程序。

执行CMD"D"命令之后，一旦按 BREAK 键查错程序就暂停下来。按G [ENTER]就再一次退回 BASIC。在按 BREAK 键时暂时中断的任何程序，打入 CONT 后就可以继续执行下去。

要从查错程序返回 BASIC 并进行初始化，打入

G 5200　[ENTER]

这时会丢失 BASIC 程序的文本和变量的数值。

例如：

```
100'                    PROGRAM : DEBUG
110'EXAMPLE OF EXECUTION WITH DEBUG WITHIN A PROGRAM
120'
130 CLS : PRINT TAB(15); 'DEBUC EXAMPLE  : PRINT
140 PRINT"ENTERING DEBUG"
150 FOR I=1 TO 500: NEXT I  'DELAY A WHILE
160'
170'***ENTER DEBUGGINC PAGKAGE ***
180'
190 CMD"D"
200'
210'*** RETURN HERE WHEN "G" ENTER TYPED IN DEBUG ***
220'
230 CLS : PRINT : PRINT"YOU HAVE RETURNED FROM DEBUG"
240 END
```

```
RUN [ENTER]
```

```
                DEBUG  EXAMPLE
              ENTERING  DEBUG
```

```
AF =44 42  -Z- - - -N-
BD =69 01 => 57  49  54  48 49  4E 2O 41  20  50 52  4F 47  52  41  4D
DE =69 B8=> 44  22  00  C3 69  C8 00 3A 93  FB 00  FD 69  D2 00  3A
HL =40 B7=> 69  55 FF FF FF  FF FF FF FF 00  00  00  00  00  00  54
AF' =FF FF SZ1H1PNC
BC' =4D EB=> 51 51  CD FC 51  7E 23  18  EC 02  02  00 4E O3 32  E7
```

```
DE'=01 07 =>4D 4F 52 59 20 53 49 5A 45 00 52 41 44 49 4F 20
HL =4D 00 =>F2 51 06 10 CD 65 51 3A 5D 40 FE 41 20 13 CD F2
IX =40 15 =>01 E3 03 00 00 00 4B 49 07 58 04 31 3E 00 44 4F
IY =FF FF=>FF F3 AF C3 74 06 C3 00 40 C3 00 40 E1 E9 C3 9F
SP =BD 6C=>BA 69 1E 1D 00 00 04 04 20 00 00 00 00 00 00 00
PC =57 08 =>E1 C9 3A 29 5B F6 C0 CD 09 44 E1 C9 D7 E5 3E 11
      1010=>28 10 FE 44 28 0C FE 30 28 F0 FE 2C 28 EC FE 2E
      1020=>20 03 2B 36 30 7B E6 10 28 03 2B 36 24 7B E6 04
      1030=>C0 2B 70 C9 32 D8 40 21 30 41 36 20 C9 FE 05 E5
G ENTER 1040=>DE 00 17 57 14 CD 01 12 01 00 03 82 FA 57 10 14
```

```
YOU HAVE RETURNED FROM DEBUG
READY
>—
```

5) CMD"R"（起动时钟〔开中断〕）

```
CMD"R"
```

使用盒式录音机，完成输入/输出操作以后，执行这个命令就立刻重新启动实时时钟。参看CMD"T"。

6) CMD"S"（返回TRSDOS）

```
CMD"S"
```

执行这个命令就返回操作系统命令状态。不把系统重新初始化，只跳出BASIC 系统。

使用CMD"S"以前，要把 BASIC 程序存到磁盘或磁带上去，防止驻留在内存的BASIC 程序丢失。

7) CMD"T"（关闭时钟〔关中断〕）

```
CMD"T"
```

进行BASIC 的磁带输入/输出操作以前必须执行这个命令。磁带机操作是时敏的，不允许引起中断的操作（如实时时钟、跟踪、时钟显示等）窃取时间。

必须先执行 CMD"T"，然后才能执行的命令有：

CLOAD	CLOAD?	CSAVE
INPVT#-1	INPUT#-2	PRINT#-1
PRINT#-2	SYSTEM	

上列各操作完成以后，可以执行 CMD"R" 命令重新打开中断。例如：

```
10 OPEN"I", 1, "TEXT/BAS"
```

```
20  CMD"T" : INPUT#1, A, B, C
30  CMD"R"
```

注意：CMD"D"（进入查错）以后，可以用CMD"T"来防止按下BREAK键时使BASIC转移去执行DEBUG程序。

8) DEF FN（定义函数）

DEF FN *var1* (*var2* [,*var*…])=*exp*

其中：*var1* 是函数名，它是LEVEL Ⅱ任何正确的变量名称

var2 及以后的*var*都是用来确定函数值的自变量

exp 是表达式，包含等号左边出现的那些变量

这个语句让用户定义自己的隐含函数。也就是只用名字就调用这个函数，隐含的功能自动完成。一旦函数被DEF FN语句定义以后，可以用带有前缀FN的函数名来调用这个函数，就和使用内部函数（如SIN、ABS、STRING$）一样。

给函数命名的变量的类型确定了函数将取数值的类型。例如，函数名有单精度特征，则不管自变量的精度如何，这个函数总取单精度值。例如：

```
DEFFNA$(TITLE$, GRAPHICS%)=STRING$(LEN
        (TITLE$), GRAPHICS%)
```

函数A$需要两个自变量，一个整型，一个字符串；函数A$取值为字符串。

```
DEFFNRC!(A)=1/(A*A)
```

函数RC!只需要一个自变量，不管自变量精度如何，RC!都取单精度值。

在DEF FN语句中用作自变量的特殊变量名是不给函数赋值的。以后调用这个函数时，可用任何一个类型相同的变量名来代替。一个变量，在DEF FN语句中用作自变量时，它的值不受影响。定义一个函数至少需要一个自变量，即使这个自变量实际上并不把数值传送给该函数，也要这样做。例如：

```
DEF FNR(A)=RND(0)
```

又例如：

```
(1)  10  DEF FNMLT(A, B)=A*B
     20  INPUT"ENTER ARGUMENTS"; X, Y
     30  PRINT"PRODUCT IS"; FNMLT(X, Y)
```

注意，这里的FNMLT是用自变量A，B定义的，在30行中调用这个函数时，用变量X，Y。任何两个有效的变量名都可用来把其值传送给这个函数。

```
(2)  DEF FNRR(A, B)=A+INT(B*RND(0))   取A，B间的一个随机数
     DEF FNL8$(A$)=LEFT$(A$, 8)       取字符串变量A$的前8个字符
     DEF FNX#(A#, B#)=(A#-B#)*(A#-B#)
                                      取两个双精度值的差值平方
100  'PROGRAM : STRING
110  'EXAMPLE OF STRING DEFFN FUNCTION
120  '
130  ' *******FUNCTION TO CONCATENATE STRINGS*******
135  '
```

```
140 DEF FNADD$(A$, B$)=A$+" "+B$
150 CLS : PRINT TAB(15); "STRING DEFFN EXAMPLE"
160 PRINT : F$=""  : INPUT "ENTER FIRST NAME"; F$
165 IF F$=""THEN END
170 INPUT "ENTER LAST NAME"; L$
180 '
190 ' ******* ADD F$ TO L$ WITH 1 BLANK IN BETWEEN *******
200 '
210 Z$=FNADD$ (F$, L$)
220 PRINT TAB(6); "FULL NAME:"; Z$
230 GOTO 160
```

```
RUN [ENTER]
```

```
                STRING DEFFN EXAMPLE

ENTER   FIRST   NAME?   JOHN [ENTER]
ENTER   LAST    NAME?   DOE  [ENTER]
        FULL    NAME:   JOHN DOE
```

```
100 'PROGRAM: MINMAX
110 'EXAMPLE OF DEFFN FEATURE
120 '
130 '******* DEFINE MIN AND MAX FUNCTIONS *******
135 '
140 DEF FNMIN(A, B)=(A+B-ABS(A-B))/2
150 DEF FNMAX(A, B)=(A+B+ABS(A-B))/2
160 '
170 '*******READ 1ST VALUE-CALL IT THE MIN AND MAX *****
180 '         MN IS CURRENT MINIMUM VALUE
190 '         MX IS CURRENT MAXIMUM VALUE
200 '
210 READ MN : MX=MN
220 '
230 '*******GET NEXT VALUE AND FIND NEW MIN/MAX*******
240 '
250 READ V : IF V=99999 THEN 320 'V=99999 MEANS ALL DONE
260 MN=FNMIN(MN, V)  '  GET NEW MINIMUM
270 MX=FNMAX(MX, V) 'GET NEW MAXIMUM
```

```
280  GOTO 250
290  '
300  '******** PRINT RESULTS *******
310  '
320  PRINT"MINIMUM VALUE=", MN
330  PRINT"MAXIMUM VALUE=", MX
340  '
350  '*******DATA FOLLOWS—LAST VALUE MUST BE 99999*******
360  '
370  DATA 1.2, 2, 3, 4.7, 5.332, 0.314, 6, 7, 8.3, 9.57, 99999
```

```
)RUN  ENTER
MINIMUM VALUE=.314
MAXIMUM VALUE=9.57
READY
)370 DATA -1, 9, 0, 8, 1, 7, 2, 6, 3, 5, 4, 99999  ENTER
)RUN  ENTER
MINIMUM VALUE=-1
MAXIMUM VALUE=9
READY
)-
```

9) DEFUSR（定义USR子程序的入口地址）

DEFUSR*n*=*nmexp*

其中 *n* 是数字0，1，…9之中的一个，如果略而不写，就认为取值为 0

*nmexp*是数值表达式，它的值用来规定机器语言子程序的入口地址

用这个语句定义机器语言子程序的入口，最多可以定义 10 个机器语言子程序的入口（LEVEL Ⅱ只可使用一个 USR 程序，入口地址用 POKE 语句送入 RAM）。

例如：

```
100    DEFUSR3=&H7D00
```

把 7D00（十进制 32000）作为调用 USR3 的入口。用户程序调用 USR3 时，控制就转移到从 7D00 开始的用户子程序中去。

有三种方法把机器语言程序送入 RAM，以供使用 USR 调用该机器语言程序。

（1） 用 TRS-80 编辑/汇编程序（Radio Shack 产品号26—2002）把源码转换成磁带上的目的文件，然后用 SYSTEM 命令把这个磁带程序装入内存（要回答 MEMORY SIZE 来保护这些代码，防止被 BASIC 系统冲掉）。

（2） 使用 TRSDOS 系统查错程序调入这机器代码子程序（把它转储到磁盘中保存起

来）。调用磁盘BASIC，回答MEMORY SIZE把这个子程序保护起来。

（3） 用磁盘 BASIC 程序，把子程序（每个字节的十进制数值）POKE 到 RAM 的高区。在初始化的时候，要适当地回答 MEMORY SIZE？ 把要 POKE 进子程序的区域保护起来。

参看USR*n*。

10) INSTR（字符串查找功能）

INSTR([*n*,] *exp*1 $, *exp*2 $)

其中 *n*是一个数字，指在 *exp*1 $ 中开始查找的位置。*n* 不写就认为是 1 （位置 1 定义为字符串中第一个字符）

*exp*1 $ 是被查找的字符串

*exp*2 $ 是希望查找到的子字符串

这个函数的功能是在一个字符串内进行查找，看这个字符串内是否包含有另一个字符串。如果有，这个函数就把子字符串在目标字符串中的起点位置取作函数的值。否则函数取值是 0 。注意，整个子字符串必须包含在被查找的字符串内，否则就取 0 值。此外，INSTR 由用户指定位置开始查找，只给出这个子字符串第一次出现的位置。

例如，令(A $ ="ABCDEFG")：

表 达 式	结果
INSTR(A $,"BCD")	2
INSTR(A $,"12")	0
INSTR(A $,"ABCDEFGH")	0
INSTR(3,"1232123", "12")	5

关于使用 INSTR 的例题，请看在 MID $ =一节中的 EDIT 程序。

11) LINE INPUT（从键盘上输入一行）

LINE INPUT["*prompt*"]; *var* $

其中 *prompt* 是给用户的提示信息

var $ 是字符串变量名称，把用户输入的内容送入用这个名称命名的字符串变量

除了以下几点差别， LINE INPUT （或 LINEINPUT，可不用中间的空格） 语句和 INPUT 语句的功能一样：

(1) 当执行这个语句时，计算机等待键盘输入，不显示出问号"？"。

(2) 每一个 LINE INPUT 语句只能把输入的内容送给一个变量。

(3) 逗点(，)和双撇号（ ）也作为字符串中的一部份被接收。

(4) 不滤去前导空格，前导空格也将成为字符串变量的一部份。

(5) 按下 ENTER 键是结束字符串输入的唯一的方法。

LINE INPUT 是一种输入字符串数据的简便方法，不必去担心分隔符的输入。因

为 ENTER 键是唯一的一种分隔符。如果你希望任何人都能向程序输入信息，而又不用特殊指令，就使用语句 LINE INPUT，然后分析输入的字符串。

有时需要把逗点(，)、双撇号(")和前导空格作为输入数据的一部份。这种情况下使用 LINE INPUT 语句的效果最好。例如：

```
LINE INPUT  A$
```

不显示任何提示信息，输入A$的内容。

```
LINE INPUT "LAST NAME, FIRST NAME? "; N$
```

就显示出提示信息然后输入数据。不能把逗点(，)用来结束字符串的输入。

试行下列程序，从中可体会到 LINE INPUT 的功能。

```
100 'PROGRAM : LNINPUT
110 'EXAMPLE OF LINEINPUT STATEMENT
120 '
130 CLEAR 300 : CLS
140 PRINT TAB(15); "LINE INPUT STATEMENT": PRINT
150 PRINT: PRINT"***ENTER TEXT***"
151 '
152 '*** GET STRING, THEN PRINT IT ***
153 '
155 A$ =""     'SET A$ TO NULL STRING
160 LINEINPUT"==>"; A$
165 IF A$ ="" THEN END  'IF STILL NULL STRING, STOP!
170 PRINT A$
180 GOTO 155
```

```
RUN  ENTER
```

```
                LINE INPUT STATEMENT
*** ENTER TEXT ***
==>        EXAMPLE TEXT
THIS TEXT HAS EMBEDDED LINE FEEDS AND TABS
IN IT. LINEINPUT ALSO ALLOWS DELIMITER(,:;"ETC). ENTER
     EXAMPLE TEXT
THIS TEXT HAS EMBEDDED LINE FEEDS AND TABS
IN IT. LINEINPUT ALSO ALLOWS DELIMITER (,:;" ETC).
==>ENTER
READY
>_
```

12) MID$ =（取代字符串的一部份）

MID$(*var*$,*n*1[,*n*2])=*exp*$

其中 *var*$ 是被修改的字符串变量的名字

*n*1 用来指定被取代的起始位置

*n*2 指有多少个字符被取代，如果 *n*2 略去不写，则*n*2就取 *exp*$ 的长度和 LEN(*var*$)－*n*1＋1 二者中较小的一个

这个语句的功能是用指定的子字符串来取代一个字符串的一部份，提供了强有力的字符串编辑能力。

注意：被修改字符串（*var*$）的长度不因 MID$ = 语句而改变。如果用作取代的字符串（*exp*$）太长，超过被修改字符串（*var*$）指定部份的长度，则把 *exp*$ 右端超出的字符舍去。

但是如果你指定被取代的字符数（*n*2）大于用作取代的子字符串（*exp*$）长度，那就不考虑指定的长度(*n*2)。

下列各例的 A$ 都是“ABCDEFG”。

例 号	表 达 式	结 果
1	MID$(A$,3,4)="12345"	"AB1234G"
2	MID$(A$,1,2)=""	"ABCDEFG"
3	MID$(A$,5)="12345"	"ABCD123"
4	MID$(A$,5)="01"	"ABCD01G"
5	MID$(A$,1,3)="＊＊＊"	"＊＊＊DEFG"

上面例2中，指定的被取代长度超过用作取代的子字符串的长度（这里是零），因此使用取代的子字符串的长度。实际上不取代任何字符。

程序例子：EDIT

这个程序接收一个初始字符串，询问取代位置和替代字符串，然后用MID$ =进行字符串的替代运算，并打印新的字符串。当输入取代位置为零时，程序就停止。

```
100 '          PROGRAM : EDIT
110 'EXAMPLE OF INSTR FUNCTION FOR TEXT EDITTING
115 '
120 CLEAR 800 : CLS
130 PRINT TAB(15); " STRING-FUNCTION EDITOR"
135 '
140 '******* GET INITIAL TEXT *******
145 '
150 PRINT : PRINT"ENTER INITIAL TEXT STRING"
160 S$="" : LINE INPUT S$ : IF S$=""THEN END
165 '
170 '******* GET TARGET & REPLACEMENT STRINGS *******
```

```
175 '
180 T$=""：PRINT：LINE INPUT"    TARGET STRING"; T$
185 IF T$=""THEN END
190 LINE INPUT "REPLACEMENT STRING "; R$
195 IF LEN(T$)<>LEN(R$)THEN PRINT "CAN'T CHANGE STRING
    LENGTH"：GOTO 180
200 '********MAKE REPLACEMENT(S) AND PRINT NEW STRING***
210 I=1  'VARIABLE I POSITIONS TO BEGINNING POINT OF SEARCH
220 I=INSTR(I,S$,T$)：IF I=0 THEN 150  'I=0 IF NOT FOUND
230 MID$(S$,I)=R$  'MAKE REPLACEMENT
240 PRINT "POSITION—"; I:  PRINT S$
250 I=I+LEN(R$): GOTO 220  'ADVANCE POSITION
```

```
RUN ENTER
```

```
                STRING-FUNCTION EDITOR

ENTER INITIAL TEXT STRING
CHANGE "DISC" TO "DISK" EACH TIME IT OCCURS...(DISC=>DISK)
        TARGET STRING  DISC  ENTER
REPLACEMENT STRING  DISK  ENTER
POSITION—9
CHANGE "DISK" TO "DISK" EACH TIME IT OCCURS...(DISC=>DISK)
POSITION—48
CHANGE "DISK" TO "DISK" EACH TIME IT OCCURS...(DISK=>DISK)
ENTER INITIAL TEXT STRING  ENTER
READY
>_
```

13) TIME$（取实时时钟的值）

```
TIME$
```

TIME$是一个无自变量的函数，执行时，把存储在实时时钟存储块内的当前日期和时间取出，组成一个字符串作为该函数的值。这个字符串长达 17 个字符，其格式如下：

MM/DD/YY ␣ HH:MM：SS（月/日/年　　时：分：秒）

小时采用 24 时制，例如下午 1:30 显示为13:30。

要调整时刻和日期，先进入 DOS READY状态，再用 TRSDOS 命令 TIME 和 DATE 置入，如下所示（例如，假定时日是 1979 年 1 月 1 日下午三点半）

```
TIME    15 : 30 : 00   ENTER
DATE    01/01/79       ENTER
```

也可以在磁盘BASIC控制下把日期和时刻数据POKE到相应的地址，以调整日期和时刻（参看TRSDOS库命令CLOCK）

可以打印TIME$函数值，或在用户程序内部使用。例如：

```
1000 IF LEFT$(TIME$,14)="07/04/79  20:00"THEN  2000
1010 GOTO 1000
2000 REM...IT'S 8PM ON JULY 4TH,1979
2010 REM...START FIREWORKS DISPLAY
 .
 .
 .
```

下面是显示时刻和日期的程序CLOCK，按下@键后程序停止执行。

```
100  '          PROGRAM: CLOCK
110  'EXAMPLE OF TIME$
120  '
130  CLS : PRINT CHR$(23)  'GET INTO 32 CHARACTER MODE
140  '
150  '******* PRINT TIME AND DATE *******
160  '
170  PRINT @ 264,"THE TRS-80 TIME IS ;
180  PRINT @ 458,"DATE:  "; LEFT$(TIME$,8);
190  PRINT @ 586,"TIME:  "; RIGHT$(TIME$,8);
200  '
210  '******* STOP IF "@" KEY IS DEPRESSED *******
220  '
230  A$=INKEY$: IF A$="@"THEN END ELSE 180
```

14) USRn（调用用户的外部子程序）

USR[*n*](*nmexp*)

其中：*n*指10个可能调用的USR之一，*n*=0,1,…9。如果*n*略而不写，就认为*n*=0

nmexp 是－32768和＋32767之间的常数，被作为整自变量传送给USR子程序

这些函数USR0到USR9把控制转向由DEFUSRn语句定义的机器语言子程序。在BASIC程序语句中遇到调用USR时，控制就转向用DEFUSRn语句定义的地址。这个地址是一个机器语言子程序的入口。机器语言子程序中遇到RET或JP 0A9A指令时，控制就返回到BASIC程序中调用USR的地方。

注：如果在使用 DETUSRn 定义子程序入口以前调用 USRn 子程序，就会出现非法函数调用 (ILLEGAL FUNCTION CALL) 的错误。

可以经 USR 语句的一个自变量直接把一个数值送入机器语言程序或者把一个数值从机器语言程序取回。这种数据的传送也可以间接经POKE语句和PEEK函数来完成。

例如：

```
10  DEFUSR1=&H7D00
20  REM…MORE PROGRAM LINES HERE
100 A=USR1(X)
```

上面这个程序的功能是：

（1） 定义子程序 USR1 的入口为十六进制 7D00（10 行）。

（2） 把控制转向这个子程序；当调用子程序时就可以把X 的值传送给子程序(100行)。

（3） 从子程序返回 BASIC 时，变量 A 可以等于由子程序送回的值（子程序一定要执行下述的转移 (JUMP) 指令）；否则，将把X的值赋于A（100 行）。

向USR子程序传送自变量和回送数据

在 BASIC 主程序和USR子程序之间来回传送数据的方法有好几种。下面列出两个主要方法：

1）用 POKE 语句把自变量值送入 RAM 中固定的存储单元，机器语言程序可以读取这些数值并把结果放进另一些 RAM 单元中去。由子程序返回 BASIC 后，可以把这些地址作为 PEEK 函数的自变量而取得子程序的输出值。这是来回传送两个或多个自变量的唯一方法。

2）把一变量值作为USRn调用的自变量就可以传送给子程序。然后使用专用的ROM调用程序来读取这个自变量，或把某个数值送回给 BASIC 程序。这个方法限定送入一个自变量，和回送一个数值（两者都是整型的）。

ROM调用

CALL 0A7FH	把 USR 的自变量送入 HL 寄存器对，H 存高位字节，L 存低位字节。这条 CALL 指令必须是 USR 子程序的第一条指令。
JP 0A9AH	用这条转移指令返回 BASIC，HL 中的整型数成为 USR 调用的输出。如果不需要把HL的内容送回，就简单地执行一条 RET指令，代替这条转移指令。

举例：下面列出一个汇编后的机器语言程序，它通过 BASIC 程序中的USR调用接收自变量，左移一位，把结果送回给 BASIC。

```
00100;
00110; SHIFT FUNCTION (左移功能)
00120;
00130;      MACHINE CODE PROGRAM TO LEFT SHIFT
00140;      AN ARGUMENT SENT FROM BASIC AND RETURN
```

```
                00150;      THE RESULT BACK TO BASIC (把 BASIC 送来的自
                            变量左移，把结果送回BASIC)
                00160;
7D00            00170       ORG     7D00H
                00180;
                00190; EQUATES AND ENTRY POINTS (等值语句，入口)
                00200;
0A7F            00210  GETARG EQU     0A7FH    ;从 BASIC 取自变量
0A9A            00220  PUTANS EQU     0A9AH    ;把结果送回BASIC
                00230;
7D00 CD7F0A     00240  SHIFT  CALL    GETARG   ;从 BASIC 取数
7D03 CB15       00250         RL      L        ;左移寄存器L
7D05 CB14       00260         RL      H        ;左移 H，结果在 HL 中
7D07 C39A0A     00270         JP      PUTANS   ;结果送回BASIC
                00280;
7D00            00290         END     SHIFT
```

下面程序中含有 SHIFT 程序（即上面的左移程序——译注）的十进制代码。用POKE语句把这些代码送进 RAM，作为 USR 程序的存入。运行该程序，当输入的数值是零时，程序就停止。

注意：下面两个 BASIC 程序都要求保护地址在 31999 以上的内存，以便给 USR代码使用（用 31999 回答 MEMORY SIZE？）

```
100 '        PROGRAM: SHIFT
110 'MACHINE LANGUAGE USER FUCTION TO LEFT SHIFT
120 '
130 ' ******* MACHINE CODE AT 7D00 HEX *******
140 '
150 DEFUSR5=&H7D00
160 '
170 '******* POKE USER PROGRAM INTO MEMORY *******
180 '
190 FORX=32000 TO 32009  '7D00 HEX EQUALS 32000 DECIMAL
210       READ A
220       POKE X, A
230 NEXT X
240 '******* GET VALUE FROM USER *******
250 '
260 CLS : PRINT TAB(15); "USR5 LEFT-SHIFT FUNCTION"
270 PRINT: INPUT "ENTER INTEGER VALUE", V
```

```
280 IF V=0 THEN END
290 PRINT "LEFT SHIFTED VALUE="; TAB(32); USR5(V)
300 GOTO 270
310 '
320 '******* DATA IS DEMICAL CODE FOR HEX PROGRAM *****
330 '
340 DATA 205, 127, 10, 203, 21,203, 20, 195, 154, 10
```

RUN [ENTER]

```
                USR5 LEFT-SHIFT FUNCTION

ENTER INTEGER VALUE? 7 [ENTER]
LEFT SHIFTED VALUE=          14
ENTER INTEGER VALUE? -7 [ENTER]
LEFT SHIFTED VALUE=         -13
ENTER INTEGER VALUE? 32767 [ENTER]
LEFT SHIFTED VALUE=          -2
ENTER INTEGER VALUE? 0 [ENTER]
READY
)_
```

下面列出的是在显示器上输出“全白”的汇编程序（与 CLEAR 键“相反”）。

```
                00100;
                00110; ZAP OUT SCREEN USR FUNCTION
                00120;
7D00            00130       ORG    7D00H
                00140;
                00150; EQUATES
                00160;
3C00            00170  VIDEO EQU    3C00H        ;显示用RAM的起点
00BF            00180  WHITE EQU    0BFH         ;全白图形符号字节
03FF            00190  COUNT EQU    3FFH         ;欲移入的字节数
                00200;
```

```
                00210; PROGRAM CHAIN MOVES X'BF' INTO ALL OF
                       VIDEO RAM
                00220;
7D00 21003C     00230  ZAP    LD      HL, VIDEO     ;源地址
7D03 36BF       00240         LD      (HL), WHITE   ;置入第一个字节
7D05 11013C     00250         LD      DE, VIDEO+1   ;目的地址
7D08 01FF03     00260         LD      BC, COUNT     ;迭代数目
7D0B EDB0       00270         LDIR                  ;执行!!!
                00280;
7D0D C9         00290         RET                   ;返回BASIC
7D00            00300         END     ZAP
```

下面用 POKE 语句把这个程序送入 RAM，即把 USR 程序存入。

```
100 '        PROGRAM : USR1
110 'EXAMPLE OF A USER MACHINE LANGUAGE FUNCTION
115 'DEPRESS THE '@' KEY WHILE NUMBERS ARE PRINTING TO STOP
120 '
130 ' ******* POKE MACHINE PROGRAM INTO MEMORY *******
140 '
150 DEFUSR1=&H7D00
160 FOR  X=32000 TO 32013  '7D00 HEX EQUAL 32000 DECIMAL
170      READ A
180      POKE X, A
190 NEXT X
192 '
194 '******* CLEAR SCREEN & PRINT NUMBERS 1 THRU 100 ****
196 '
200 CLS
205 PRINT TAB(15); "WHITE-OUT USER ROUTINE":PRINT
210 FOR  X=1  TO  100
220      PRINT X;
225      A$=INKEY$ : IFA$="@"THEN END
230 NEXT X
240 '
250 '******* JUMP TO WHITE-OUT SUBROUTINE *******
260 '
270 X=USR1 (0)
280 FOR X=1 TO 1000: NEXT X    'DELAY LOOP
290 GOTO 200
```

```
300 '
310 '******* DATA IS DEMICAL CODE FOR HEX PROGRAM *****
320 '
330 DATA 33,0,60,54,255,17,1,60,1,255,3,237,176,201
```

执行这个程序，与同样效果的 BASIC “全白”输出程序比较，后者将耗用较长时间。

3. 和磁盘有关的特点

在 TRS 磁盘操作系统中，程序和数据都是以“文件”的形式存储的。每个程序，每组数据都有它自己的独特的文件标识符，文件标识符由文件名和标识信息所组成。

在进行磁盘输入/输出（包括调出和装入BASIC程序）以前，请参看第三章“TRSDOS概述”。也要浏览一下第一章“概述”中的“符号规定”一节，这是了解语句语法结构的唯一方法。

磁盘 BASIC 程序提供了一组在 TRSDOS 控制下进行磁盘输入/输出操作功能很强的命令，语句和函数。它们可以分为两大类：

(1) 文件操作：把文件作为一个单位来处理，而不以文件中的个别记录来处理。

(2) 文件存取：为数据文件的读写进行准备，对文件进行读和写。

在“文件操作”中有以下的命令：

KILL	从磁盘上删除程序或者数据文件
LOAD	把磁盘上的 BASIC 程序装入内存
MERGE	把磁盘上的一个 ASCII 格式的 BASIC 程序和目前在 RAM 中的同样形式的程序合并
RUN “*program*”	装入磁盘上的 BASIC 程序并执行
SAVE	把内存中驻留的 BASIC 程序存入磁盘

在“文件存取”中有以下语句和函数：

语句：

OPEN	为存取打开文件（如果需要就建立一个文件）
CLOSE	关闭存取的文件
INPUT#	按顺序存取方式读取磁盘
LINE INPUT#	按顺序存取方式读一行数据
PRINT#	按顺序存取方式写入磁盘
GET	按随机存取方式读取磁盘
PUT	按随机存取方式写入磁盘
FIELD	给随机存取文件缓冲器规定区段 (field) 大小和区段名
LSET	把值放入指定的缓冲器区段，区段右部填入空格
RSET	把值放入指定的缓冲器区段，区段左部填入空格

函数：

CVD	从磁盘中以 GET 语句取出以后，恢复双精度数的数值形式
CVS	从磁盘中以 GET 语句取出以后，恢复单精度数的数值形式
CVI	从磁盘中以 GET 语句取出之后，恢复整型数的数值形式
EOF	在读文件过程中，检查一下是否读到了文件的结束标记
LOF	获取文件中最后一个记录的编号。
MKD$	把双精度数变换成字符串的格式，这样才能够用 PUT 语句将它写入磁盘
MKI$	把整型数变换成字符串的格式，这样才能够用 PUT 语句将它写入磁盘
MKS$	把单精度数变换成字符串的格式，这样才能够用 PUT 语句将它写入磁盘

文件操作

1) **KILL**（从磁盘上删除一个文件）

KILL *exp*$

其中 *exp*$ 是字符串表达式，用来表示一个已存在文件的文件标识符

这个命令和 TRSDOS 的 KILL 命令一样（参见 TRSDOS 库命令）。

例如：

```
KILL "OLDFILE/BAS.PSW1"
```

从存有这个文件的第一个磁盘上删去这个指定的文件。

不要删去已打开的文件，否则会破坏软盘上的内容（先要关闭打开的文件）。

2) **LOAD**（从磁盘上调取 BASIC 程序文件）

LOAD *exp*$ [, *R*]

其中 *exp*$ 是字符串表达式，用来表示磁盘上一个BASIC程序文件的文件标识符

R 告诉 BASIC 系统，调进这个程序以后就执行这个程序

这个命令把一个 BASIC 程序文件调进 RAM。如果选用了 R，BASIC 系统就自动执行这个程序。否则，返回BASIC 的命令状态。

如果 LOAD 命令没有选用 R，就删去内存中原驻有的BASIC程序，清除所有变量，关闭所有已打开的文件。LOAD 命令选用了 R，虽删除BASIC 程序，清除所有变量，但不关闭已打开的文件。

LOAD 命令选用 R 和命令 RUN *exp*$，R 等效，无论两个命令中的哪一个都可以在一个程序中用来链接程序：一个程序调用另一个程序。

如果调用非 BASIC 文件，就出现 DIRECT STATEMENT IN FILE(直接陈述的文件) 或 LOAD FORMAT ERROR（装入格式的错误）。

举例：

```
LOAD "PROG1/BAS:2"
```

清除原驻留的 BASIC 程序，从 2 号磁盘中装入 PROG1/BAS，返回 BASIC 的命令状态。

```
10  RAM……INSTRUCTIONS
1000  LOAD "PROG2/BAS" , R
```

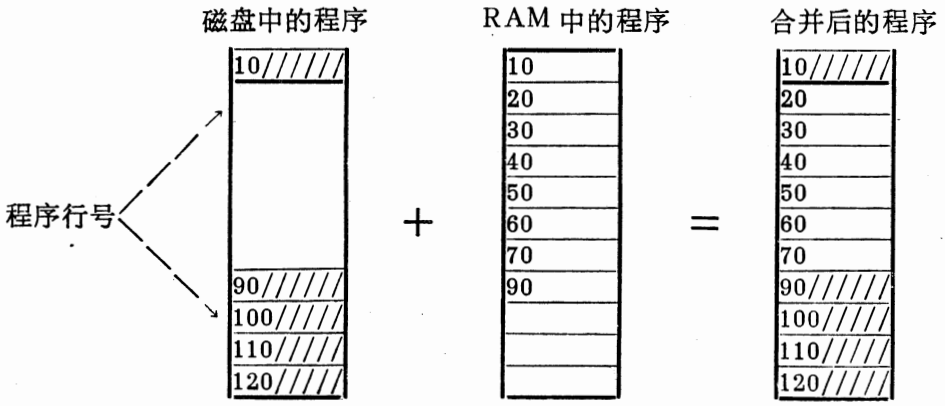
这是链接两个程序的例子，第一个程序用来给出命令，然后装入第二个程序（PROG2/BAS）的“工作”部份。注意，上述的 1000 行和 1000 RUN "PROG2/BAS"等效。

3) **MERGE**（把磁盘程序和驻留程序合并）

MERGE *exp*$

其中 *exp*$ 是字符串表达式，用来表示 ASCII 格式 BASIC 磁盘文件的文件标识符，即该程序是采用 ASCII 格式存储的

除了装入新的程序 *exp*$ 以前不清除原驻留程序以外，该命令和 LOAD 命令完全相同。换言之，MERGE 把 *exp*$ 和原驻留程序合并到一起，按照行号的顺序把 *exp*$ 程序插入到驻留程序中去。如果 *exp*$ 中的行号和驻留程序行号相同，就用 *exp*$ 中的内容代替原驻留行的内容。



MERGE 提供了把程序模块组合起来的简便方法。例如，可以使用这个命令把常用的一组 BASIC 子程序和各种程序组合在一起。例如，RAM 中有下述程序：

```
10    REM…MAIN  PROGRAM
20    GOSUB  1000
30    REM…MORE PROGRAM LINES HERE
```

```
999   END
1000  REM…NEED TO ADD SUBROUTINES HERE
1010  REM…SO USE MERGE COMMAND
1020  PRINT "SUBROUTINE NOT AVAILABLE" : RETURN
```

并假定，磁盘上存有下列 ASCII 格式的程序：

```
1000  REM…BEGINNING OF SUBROUTINE
1010  PRINT "EXECUTING SUBROUTINE…"
1020  REM…MORE PROGRAM LINES HERE
1100  RETURN
```

这个子程序叫做 SUB/TXT，然后用语句

```
MERGE  SUB/TXT
```

把它们合并，在 RAM 中的程序将是：

```
10    REM…MAIN PROGRAM
20    GOSUB  1000
30    REM…MORE PROGRAM LINES HERE
999   END
1000  REM…BEGINNING OF SUBROUTINE
1010  PRINT  EXECUTING SUBROUTINE…"
1020  REM…MORE PROGRAM LINES HERE
1100  RETURN
```

注意：MERGE 把全部文件都关闭，并清除全部的变量。合并完成以后退回命令状态。

4) **RUN** *"program"*（装入并执行磁盘上的程序）

RUN *exp*$[, R]

其中 *exp*$ 是字符串表达式，表示存在磁盘上的 BASIC 程序的文件标识符

R 表示打开的文件将继续打开

如果不使用 R 选择项，就关闭打开的文件。执行这个命令时，原驻留的 BASIC 程序被文件 *exp*$ 的程序代替。例如：

```
RUN" DISKDUMP/BAS"      ENTER
```

装入并运行 BASIC 系统的按扇转贮程序。

假定用名字 "PROG1/BAS" 把下列程序存入磁盘：

```
10    PRINT" PROG1 EXECUTING…"
20    RUN" PROG2/BAS"
```

并且用名字 "PROG2/BAS" 把下面的程序存入磁盘：

```
10    PRINT" PROG1 EXECUTING"
20    RUN" PROG2/BAS"
```

现在打入：

RUN PROG1/BAS″ ENTER

就可看到一个简单的程序链接的例子。

按下 BREAK 键，程序链接就中止。

5) **SAVE**（把程序存入磁盘）

SAVE *exp*$[, A]

其中 *exp*$ 是字符串表达式，用来表示所用的文件名、和选用的名扩展、通行字和驱动器编号。如果这个文件名已存在，就和建立文件一样，把原有的内容删除

A 表示用 ASCII 格式存放，而不是用压缩格式存放

用这个命令把 BASIC 程序存入磁盘，可以采用压缩格式存放，也可以用 ASCII 格式存放。采用压缩格式占用磁盘空间较少，存放速度也较快。这是 RAM 中 BASIC 程序存放的形式。

选用 ASCII 格式，可以做一些用压缩格式的 BASIC 文件难于做到的事情。例如：

(1) 用 MERGE 命令把 ASCII 格式的磁盘文件进行合并。

(2) 可以用 TRSDOS 的 LIST 和 PRINT 命令把 ASCII 格式的文件打印出清单。

(3) 有的程序要作为其它程序的数据而读出，这种数据程序一定要以 ASCII 格式存放。

习惯上给 BASIC 程序加上名扩展：压缩格式的名扩展用 /BAS。ASCII 格式的名扩展用 /TXT。

SAVE 语句举例：

SAVE FILE1/BAS.JOHNQDOE:3″

用文件名字 FILE1，名扩展/BAS，通行字．JOHNQDOE，按压缩格式把驻留在内存的 BASIC 程序存进 3 号磁盘中去。

SAVE ″MATHPAK/TXT″, A

使用名字 MATHPAK/TXT，按 ASCII 格式把驻留在内存的 BASIC 程序存入第一个没有写保护的磁盘中去。

SAVE 命令完成以后，BASIC 返回命令状态。

文件存取

这一部份分为 4 个小部份

(1) 建立文件并设置缓冲器——OPEN 和 CLOSE

(2) 语句和函数

(3) 顺序输入/输出技术

(4) 随机输入/输出技术

如果你是第一次试行磁盘文件存取，要集中精力搞懂第 1，3，4 部份，只需大致阅读第 2 部份，了解函数和语句如何进行工作的一般概念，以后再仔细学习第 2 部份语句和函数

的语法。

建立文件和设置缓冲器

在初始对话时，打入一个数字以响应 HOW MANY FILES? 打入的数字告诉 BASIC 系统要建立多少个缓冲器处理磁盘的存取（读出和写入）。

缓冲器的个数可由 1 至 15 之间的数字来确定，如果打入：

HOW MANY FILES? 4 ENTER

BASIC 系统就设置 4 个缓冲器，编号为 1，2，3，4。可以把缓冲器看作一个等待区，数据进出磁盘文件都要经过这个区。当你希望存取一个指定文件时，必须告诉 BASIC 系统，用几号缓冲器存取这个文件。你也必须告诉 BASIC，存取的类型——顺序输入、顺序输出还是随机输入/输出。这些都是随同 OPEN 语句一起进行的，随同 CLOSE 语句而结束。

1) **OPEN**（给文件设置缓冲器并置以类型）

OPEN *exp*1$，*nmexp*，*exp*2$

其中 *exp*1$ 是一个字符串表达式或常数，只有第一个字符有意义。这个字符指定用什么类型打开文件

*exp*1$	存取类型
I	顺序输入
O	顺序输出
R	随机输入/输出

nmexp 是 1 到 15 之间的一个数值，告诉 BASIC 系统，这个编号的缓冲器是为 *exp*2$ 指定的文件设置的

exp$2 表示一个 TRSDOS 的文件标识符

这个语句使文件存取成为可能。*exp*1$ 确定经由的缓冲区将用什么类型来进行存取；*nmexp* 确定哪一个缓冲器用来存取文件；*exp*2$ 确定了被存取文件的名字。如果 *exp*2$ 先前并不存在，TRSDOS 可能建立该文件，也有可能不建立，取决于存取是以什么类型进行的。

注意，*nmexp*（缓冲器编号）不能超过初始化期间回答 HOW MANY FILES? 输入的数字。如果输入的是：

HOW MANY FILES? 2 ENTER

则 *nmexp* 只能是 1 和 2。

OPEN 语句举例：

```
100  OPEN" O", 1, "CLIENTLS/TXT"
```

打开文件“CLIENTLS/TXT”，采取顺序输出方式，使用 1 号缓冲器。如果这个文件原先不存在，就建立这个文件；如果这个文件已存在，就删除原有内容（这点在“顺序输入/输出技术”那一部份里面作解释）。

```
100  OPEN "I", 1, "PROG 1 /TXT:1"
```

打开 1 号磁盘上的文件 "PROG1/TXT"，采用顺序输入方式，把 1 号缓冲器交给这个文件使用。如果不存在文件 PROG1/TXT，就送出错误信息。因为不可能从不存在的文件中输入信息。

```
100  INPUT  "MODE(I, O, R)"; MODE$
110  INPUT  "BUFFER NUMBER"; BUFFER%
120  INPUT  "FILE SPECIFICATION"; FILESPEC$
130  OPEN MODE$, BUFFER%, FILESPEC$
```

程序执行时，这一组语句为 OPEN 语句提供自变量。MODE$ 的第一个字符确定存取类型，BUFFDR% 确定所用的缓冲器号码，FILESPEC$ 给出文件标识符。

```
OPEN  "R", 2, "DATA/BAS. SPECIAL"
```

把带有通行字 SPECIAL 的文件 DATA/BAS 打开，使用 2 号缓冲器进行随机输入/输出。如果目前不存在 DATA/BAS，就在第一个没有写保护的磁盘上建立这个文件。

文件打开期间，用赋予该文件的缓冲器的编号来访问这个文件，例如：

```
GET      缓冲器编号          PUT      缓冲器编号
PRINT#   缓冲器编号          INPUT#   缓冲器编号
```

这些语句都是用缓冲器编号来访问被打开的文件，类型必须正确。

一旦用 OPEN 语句打开了一个缓冲器，这个缓冲器就不能再在别的 OPEN 语句中使用，必须先关闭这个缓冲器才能再使用。

关于缓冲器的设置

顺序输入（I 方式）的一个文件可以设置两个以上的缓冲器。然而，顺序输出（O 方式）或随机存取（R 方式）的文件只能设置一个缓冲器。例如：

```
10  OPEN"I", 1, "TEXT/TXT:1"
20  OPEN"I", 2, "TEXT/TXT:1"
```

现在可以经 1 号缓冲器和 2 号缓冲器把文件 TEXT/TXT 顺序输入。

2) **CLOSE**（停止文件的存取）

> CLOSE [*nmexp*[, *nmexp*…]]
>
> 其中 *nmexp* 是 1 至 15 之间的数值，即文件缓冲器的编号（在打开文件时设置的）。如果略掉 *nmexp*，就关闭全部文件

这个命令关闭经指定缓冲器存取的文件。如果以前的 OPEN 语句中没有设置这个编号，命令 CLOSE *nmexp* 就不起作用。

CLOSE 语句举例：

```
CLOSE 1, 2, 8
```

关闭和缓冲器 1，2，8 相联的文件。这些缓冲器现在可以用 OPEN 语句赋与其它文件了。

CLOSE FIRST%+COUNT%

关闭由（FIRST%+COUNT%）指定的缓冲器相联的文件。

不要移走上面还有已打开文件的软盘，要先把文件关闭，才能移走这个软盘。这是因为最后的256个字节数据并不一定已写入磁盘。如果尚未写入的话，关闭文件就把这组数据写入磁盘了。

下列情形就自动关闭全部文件：

NEW [ENTER]

RUN [ENTER]

MERGE 文件标识符 [ENTER]

编辑文件

增加或删除程序行

执行CLEARn语句

磁盘错误

3) **INPUT#** （从磁盘中顺序读出）

INPUT# *nmexp*, *var* [, *var*…]

其中 *nmexp* 用来表示顺序输入文件缓冲器的号码，*nmexp*=1, 2, …15

var 是一个变量名，其中将要存放由磁盘取出的数据

这个语句从磁盘文件中输入数据，数据是顺序输入的。也就是说，第一次打开文件时，指针位置在文件的开头，每输入一次数据，指针加1。如果要重新从文件开头读起，必须把这个文件关闭，再重新打开。

INPUT语句不考虑数据是怎样存放到磁盘上去的：无论是用单个PRINT语句还是用10个不同的PRINT语句存放都没有关系。对INPUT语句来说，重要的是结束符位置和EOF标志。

要从磁盘上成功地读入数据，必须预先知道数据是什么格式。下面将叙述INPUT#语句怎样解释读数据时遇到的各种符号。

把读入的数据赋值给一个变量时，BASIC系统将滤掉前导空格，当遇到第一个非空格字符时，BASIC系统就认为这是数据的开头。当遇到结束符号或出现结束条件时，数据就结束。结束符号是各种各样的，这要看BASIC系统把输入的值赋给数据变量还是字符串变量。

注解：阅读下面部份时，必须记住一个重要的例外情况：当〈EN〉（回车）前面的字符是〈LF〉（换行）时，〈EN〉不作为结束符，它或者作为数据项的一部份（字符串变量时），或者被省略掉（数值变量时）。（在键盘上按“↓”键，就输入换行符〈LF〉，按下[ENTER]键，输入回车符〈EN〉）。

凡遇到有〈EN〉作为结符束时，要注意上述注解。

数值输入

假定磁盘上数据的信息形式是：

␣1.234␣—33␣␣27␣〈EN〉

这里的〈EN〉表示回车符（ASCII 十进制代码 13）。

语句

INPUT#1，A，B，C

或者语句组

INPUT#1，A：INPUT#1，B：INPUT#1，C

执行结果，赋值给 A，B，C。

A＝1.234　　B＝－33　　C＝27

这是因为读入数据赋值给变量时，把空格和〈EN〉都认作结束符。1.234 前面的空格是“前导空格”因此被滤去。1.234 后面的空格是结束符。因此 BASIC 系统接着就把符号“—”赋给第二个变量，其数值是－33。再下面两个空格是结束符。第三个输入 2 开始到 7 结束。

字符串输入

把数据读入字符串变量时，INPUT 语句滤去所有的前导空格。把第一个非空格字符认作数据的开始。

如果第一个符号是双撇号(")，INPUT 语句就把数据作为有引号的字符串处理：顺序地读入各个字符，一直遇到下一个双撇号(")时才结束。逗点(,)空格和回车〈EN〉都放到字符串中去，引号本身不在字符串中。

如果字符串项的第一个字符不是双撇号，INPUT 语句就把数据作为不带引号的字符串处理。顺序地读入各个字符直到遇见第一个逗点(,) 或回车符〈EN〉为止。如果遇到双撇号，也放进字符串去。

例如，磁盘上的数据是：

PECOS, ␣TEXAS" GOOD MELONS"

输入语句　INPUT#1，A$，B$，C$"

其赋值情况如下：

A $＝PECOS

B $＝␣TEXAS "GOOD␣MELONS"

C $＝空字符串

如果在第一个双撇号前插入一个逗点，C$ 的值就是 GOOD MELONS。

以上所述是最简单的例子，只是给读者一点 INPUT 语句怎样工作的概念。然而，还有许多别的输入数据的方法：不同的结束符，不同的目标变量的类型……，与其一个一个叙述不如把这些总括起来，先给出 INPUT 语句如何工作，那些是结束符，结束条件是什么等一般性的描述，然后给出几个例题。

当 BASIC 系统遇到结束符时，就往前扫描看看紧跟在第一个结束符后还有多少结束符。以保证 BASIC 系统在正确的地方找到下一个数据项。

下面列出各种 INPUT# 要寻找的结束符，把各种可能的结束符列成一张表。

（1）数值输入的结束符有

遇文件结束

遇第 255 个数据字符

，（逗点）

〈EN〉

〈EN〉〈LF〉

␣[␣…] [〈EN〉]

␣[␣…] [〈EN〉〈LF〉]

（2）带双撇号（″）的字符串结束符有

遇文件结束

遇第 255 个数据字符

″（双撇号）

″[␣…] [，]

″[␣…] [〈EN〉]

″[␣…] [〈EN〉〈LF〉]

（3）不带双撇号（″）的结束符有

遇文件结束

遇第 255 个字符

，（逗点）

〈EN〉[〈LF〉]

图 12 的流程图说明 INPUT# 语句是怎样把数据赋与变量的。

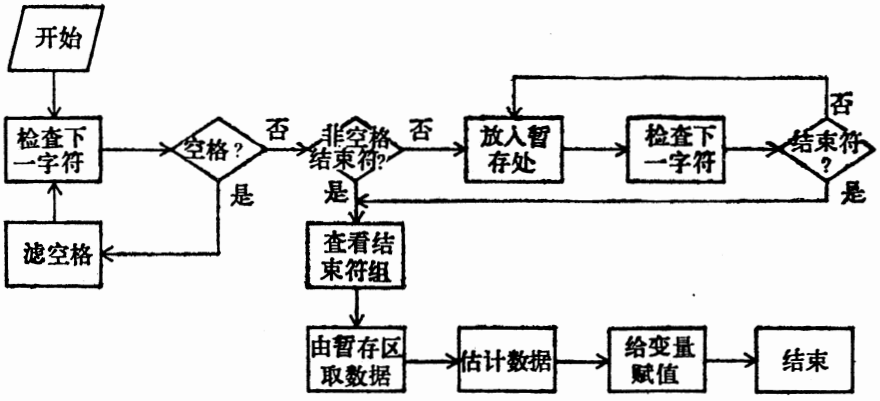


图 12 流程图

下表说明输入语句如何读不同的数据的信息：

INPUT#1，A，B，C

例号	磁盘上的数据信息	赋值情况
1	␣123.45␣〈EN〉〈LF〉␣8.2E4␣␣7000〈EN〉	A＝123.45 B＝82000 C＝7000
2	␣␣␣3〈LF〉〈EN〉4〈EN〉5〈EN〉A12eof	A＝34 B＝5

		C=0
3	1,,2, 3, 4	A=1
		B=0
		C=2
4	1, 3, eof	A=1
		B=3[1]
		C=0 文件结束错误

上述例2中，为什么变量C取0值呢？遇到文件结束（eof）时，就结束了最后的一个数据项。这个最后的数据项是“A12”，要用BASIC中和VAL函数类似的子程序来计算它的数值，由于“A12”的第一个字符不是数字，所以这个变量取值为零。例3中，INPUT#语句寻找第二个数据项时立刻遇到了结束符（逗点），故变量B值取零。

下面的表中说明输入语句如何读磁盘上的不同的数据的信息。

INPUT#1，A$，B$

例号	磁盘上的数据信息	赋值情况
1	␣␣␣ "ROBERTS, J."ROBERTS, M.N eof	A$=ROBERTS,J. B$=ROBERTS,M.N.
2	␣␣␣ROBERTS,J.␣␣␣ROBERTS,M.N〈EN〉	A$=ROBERTS B$=J.
3	THE WORD"QUO", 12345.789〈EN〉	A$=THE WORD"QUO" B$=12345.789
4	BYTE〈LF〉〈EN〉UNIT OF MEMORY eof	A$=BYTE〈LF〉〈EN〉 UNIT OF MEMORY B$=空（文件结束错误）

上面例3中，第一个数据项是不带双撇号的字符串，因此双撇号(")不是结束符，而变成A$的一部份。例4〈EN〉前面有〈LF〉，因此这个回车符号〈EN〉不结束第一个字符串，〈LF〉和〈EN〉都放在A$字符串中。

技术注解：上面的讨论略去了顺序输入过程中输入缓冲器的作用。磁盘BASIC系统总是把256个字节的记录一次读入缓冲器，然后，把缓冲器中的内容整理一下使之满足INPUT#语句的变量表。这就是为什么下列两个语句不需要两次从磁盘读数的道理。

```
100  INPUT#1, A%
200  INPUT#2, B%
```

缓冲器中的256个字节的记录中有足够的数据赋值给A%和B%以及其它变量。

4) LINE INPUT#（从磁盘中读一行文本）

LINE INPUT# *nmexp*, *var*$

其中 *nmexp* 指顺序输入[2]文件缓冲器的编号，*nmexp*=1, 2, …15

var$ 是存放字符串数据的变量名

注 1) 原文中为2——译者

注 2) 原文为输出 OUTPUT——译者

和键盘输入语句LINE INPUT相同，这个语句读入一“行”(LINE)字符串数据赋于字符串变量*var*$。当你希望把ASCII格式的BASIC程序文件作为数据来读或希望读入的数据不受一般的前导符和结束符限止的时候，这个语句就特别有用。

LINE INPUT（或LINEINPUT—间隔可有可无）语句从第一个字符起读入每一个符号，一直读到下述情况为止：

（1）〈EN〉符，前面没有〈LF〉。

（2）文件结束。

（3）第255个数据符号（这个符号也包括在字符之中）。

遇到这类符号：如引号(")、逗点(,)、前导空格、〈LF〉〈EN〉符号时都纳入字符串。

例如，如果数据是

```
10  CLEAR  500  〈EN〉
20  OPEN"I", 1, "PROG"〈EN〉
 ⋮
```

则语句

```
LINE INPUT#1, A$
```

就分别读每一个程序行，一次读一行。

5) PRINT# (顺序写入磁盘文件)

PRINT# *nmexp*, [USING *format*$;] *exp* [*p* *exp*…]

其中*nmexp*指顺序输出文件缓冲器的编号，*nmexp*=1,2,…15

format$是随USING使用的一串字段（field）说明符

p 输入磁盘时两个表达式之间的分隔符。可以用分号“；”，也可以用逗点(，)。（分号更好些）

exp 是一个表达式。计算这个表达式的值，并把计算结果写入磁盘

这个语句把数据按顺序写入指定的文件。第一次打开一个文件作顺序输出时，指针在文件的开头，因此第一次使用PRINT#语句就把数据放入文件的开头。每次PRINT#结束时，指针就向前推进，因此是顺序写入。

PRINT#语句建立的磁盘信息形式和PRINT语句在荧光屏上显示的形式相同。记住这一点就能够正确地使用PRINT语句写入信息，今后可以用一个或多个INPUT#语句读取这些信息。

PRINT#语句把数据写入磁盘以前并不把数据进行压缩，它写入的数据是ASCII码的信息形式。

例如，如果A=123.45，语句　　PRINT#1, A

就在磁盘上顺序写入9字节符号，它们是

␣123.45␣〈EN〉

在PRINT#语句的变量清单中，标点符号是非常重要的。没有用引号括起来的逗点(，)和分号(；)，和在LEVEL Ⅱ内PRINT语句中的作用相同。例如，A=2300，

B=1.303 则

PRINT#1, A, B

在磁盘上放置的数据是

␣2300␣␣␣␣␣␣␣␣␣␣␣␣␣␣1.303␣⟨EN⟩

PRINT# 语句变量清单中 A 和 B 之间的逗点在磁盘文件中造成 10 个额外的空格。一般不希望这样使用磁盘空间，所以用“；”代替逗点语句。

PRINT#1, A; B

把数据写成

2300 1.303⟨EN⟩

只需记住在各项之间是用分号“；”隔开，PRINT# 语句直接把此数值数据写入磁盘。

有字符串数据的 PRINT# 语句更要小心，因为要恰当地插入分隔符，才能正确无误地把数据读回来。如果希望用 INPUT# 把各个字符串分别读入，就必须用明显的分隔符把各字符串项之间分隔开。例如

A$ ="JOHN Q. DOE"　　B$ ="100—01—001"

语句

PRINT#1, A$; B$

在磁盘上产生的信息形式是：

JOHN Q. DOE 100—01—001⟨EN⟩

这样就不再能用 INPUT# 语句回送给两个变量了。若语句写为：

PRINT#1, A$; ", "; B$

产生的信息形式是

JOHN Q. DOE, 100—01—001

就可以用 INPUT# 语句回送给两个变量。

如果字符串数据中不包含分隔符——逗点（，）或⟨EN⟩符号，可以采用上述的方法。但是，如果数据中包含分隔符或前导空格，而又不希望略掉这些分隔符和空格，就必须加引号，把加进的引号和数据一起写入磁盘。例如

A$ ="DOE, JOHN Q."　　B$ ="100—01—001"

若使用语句

PRINT#1, A$; ", "; B$

磁盘信息形式是

DOE, JOHN Q., 100—01—001⟨EN⟩

使用语句

INPUT#1, A$, B$

输入时，A$ 得到的值是“DOE”，B$ 得到的值是“JOHN Q.”。这是因为磁盘信息形式中 DOE 后面有逗点。

要把这个数据正确读出，必须使用函数 CHR$ 把双撇号(")的信息形式插入磁盘。双撇号（"）的十进制 ASCII 码是 34。用法如下：

PRINT#1, CHR$(34); A$, CHR$(34); B$

这样产生的信息形式是

"DOE, JOHN Q." 100—01—001〈EN〉

只要简单地使用语句

INPUT#2, A$, B$

就能正确地读出。

注：也可以使用函数 CHR$ 把别的分隔符和控制码插入文件，例如

CHR$(10) 〈LF〉 换行

CHR$(13) 〈EN〉 回车

CHR$(11)或 CHR$(12) 行式打印机换页

USING 选择项

这个选择项使用户更容易精细地控制编写文件的格式。用这个方法可建立报表文件，然后用 LIST 或 PRINT（TRSDOS 命令）把它打印出来。

这个选择项使用户可控制写入磁盘的字符个数，例如：

A$="LNDWIG"

B$="VAN"

C$="BEETHOVEN"

使用语句

PRINT#1, USING"!. !.% %"; A$; B$; C$ 这就把数据写成简名(nickname) 的形式

L.V.BEET 〈EN〉

这种情况下，就不需要加任何分隔符。请参看 LEVEL Ⅱ BASIC 参考手册中关于 PRINT USING 的叙述。

技术注解：上述讨论略去顺序写入过程中输出缓冲器的作用。实际上数据先进入缓冲器，填满 256 字节的记录以后，就把数据写入磁盘文件中去。这就是为什么每个PRINT# 语句执行时并不一定进行磁盘存取的原因。

随机存取语句

6) FIELD（按区段（*field*）组织随机文件缓冲器）

FIELD *nmexp*, *nmexp*1 AS *var*1$ [*nmexp*2 AS *var*2$]

其中 *nmexp* 指随机存取文件缓冲器编号 *nmexp*=1,2,…,15

nmexp 1 指定第一个区段的长度

*var*1$ 定义第一个区段的变量名

nmexp 2 指定第二个区段的长度

*var*2$ 定义第二个区段的变量名

… 顺序定义缓冲器中别的区段长度及变量名

给缓冲器划分区段（用 FIELD 语句）以前，必须用 OPEN 语句把这个缓冲器交给指定的磁盘文件使用（必须用随机存取方式），然后用 FIELD 语句组织这个随机文件缓冲

器。这样一来就可以把数据传送给磁盘存储器或者进行反向传送。

每个随机存取缓冲器有255个字节，用来存储磁盘向BASIC系统传送的数据，或者从BASIC向磁盘传送数据。必须使用一种方法在这个缓冲器和BASIC系统之间进行存取：从缓冲器中读出数据，或者把新的数据写入缓冲器，FIELD语句就提供了这种存取手段。

可以多次使用FIELD语句以便重新组织文件缓冲器。把缓冲器划分成区段并不清除缓冲器的内容；仅改变存取缓冲器的区段名。两个或多个区段名能够访问同一个缓冲器的某个区段。

例如：

```
FIELD 1, 255 AS A$
```

这个语句告诉BASIC系统，把整个255字节的缓冲器命名为字符串变量A$，如果现在打印A$，就可以看到这个缓冲器的内容。当然，这样打印出来的数据是没有意义的，除非已经使用GET语句从磁盘中读出255字节的记录。

注意：所有的数据（不管是字符串数据还是数值数据）都必须用字符串的形式放入缓冲器。有三对函数（MKI$/CVI，MKS$/CVS，MKD$/CVD）可把数值转换为字符串和反向转换。请参看后面“函数”(Function)的一节。

例如：

```
FIFLD 3, 16 AS NM$, 25 AS AD$, 10 AS CY$, 2 AS ST$, 7 AS ZP$
```

缓冲器3的前16个字节命名为NM$，后面25个字节命名为AD$，再后面10个字节命名为CY$，再后面2个字节命名为ST$，再后面7个字节命名ZP$。剩下195字节还没有划分区段。

关于区段名

区段名NM$，AD$，CY$，ST$和ZP$没有“字符串变量”的原来意义。不占用BASIC系统的字符串变量空间，仅是FIELD语句设置的缓冲器区段。这就是为什么可以使用语句：

```
100 FIELD 1, 255 AS A$
```

而不用担心字符串空间是否有255个字节可以给字符串变量A$使用。

如果在赋值语句的左边使用缓冲器区段名，这个名字就不再指缓冲器区段，也就不能再使用这个名字进行存取了。例如

```
A$=B$
```

这使上述FIELD语句（100行）失效。

随机输入过程中，GET语句把数据取出并放入255字节的缓冲器中，然后就可以使用规定的缓冲器区段名进行存取。在随机输出过程中，LSET和RSET语句把数据放入缓冲器，然后用PUT语句把缓冲器内容送入磁盘文件。

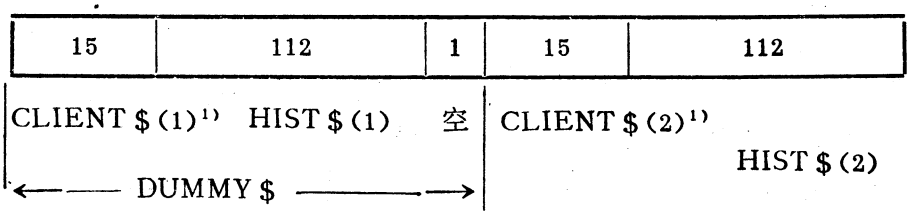
经常要在FIELD语句中使用“哑变量”跳过缓冲器的前一部份，在缓冲器中间某一地点开始划分区段。例如：

```
FIELD 1, 15 AS CLIENT$(1), 112 AS HIST$(1)
FIELD 1, 128 AS DUMMY$, 15 AS CLIENT$(2), 112 AS HIST$(2)
```

上面第二个FIELD语句中，DUMMY$的作用就是把CLIENT$(2)的起点移到第129字

节。用这种方法把两个相同的子记录都写到1号缓冲器内，实际上并不使用DUMMY$来把数据放入缓冲器或者从缓冲器中取出数据。

这个缓冲区内的存储分配如下：



这个划分区段的结构中，只有一个字节（第128字节）没有使用。

7) **GET**（从磁盘上）用随机存取方式读出一个记录）

GET *nmexp*1 [, *nmexp*2]

其中 *nmexp* 1 指定随机存取文件缓冲器的号码，*nmexp*1＝1,2,…,15

*nmexp*2 指定从文件中读取记录的编号，如果省略掉，就读取当前记录

这个语句从磁盘文件中读取一个数据记录，并把这个记录放入指定的缓冲器。从一个文件取出数据以前，必须打开这个文件，并为该文件设置缓冲器。即，在语句

GET *nmexp*1, *nmexp*2

的前面需要使用这样的语句

OPEN "R", *nmexp*1, *filespec*

BASIC系统遇到GET语句，就查找相应缓冲区的控制块，获得：

(1) 存取文件必须的信息，

(2) 给这个缓冲器设置存取方式（必须是R），

(3) 当前记录编号，

(4) EOF（文件结束）记录的编号，即文件的最大记录编号，

(5) 内部使用的其他信息。

然后BASIC系统从以表达式2指定的文件记录中读入数据，并把此记录内容放入缓冲器。如果略去记录编号，就从当前记录中读出。

"当前记录"是指编号比上一次存取的记录编号大1的那个记录。经某个缓冲器第一次存取一个文件时，当前记录编号置1。

举例：

程序语句	功能
1000 OPEN"R", 1, "NAME/BAS"	使用1号缓冲器打开文件NAME/BAS，使用随机存取方式
1010 FIELD 1, …	组织缓冲器
1020 GET 1	把1号记录取入1号缓冲器

注：1) 原文上都写为CL$(1)或CL$(2)的字符串名——译者

```
1025 RAM……ACCESS BUFFER
1030 GET1，30                    把 30 号记录取入 1 号缓冲器
1035 RAM……ACCESS BUFFER
1040 GET1，25                    把 25 号记录取入 1 号缓冲器
1046 RAM……ACCESS BUFFER
1050 GET 1                       把 26 号记录取入 1 号缓冲器
```

如果打算读取的记录编号比文件结束的记录编号还大，BASIC 系统就用 16 进制零把缓冲器填满，并不报告错误。为避免出现这种情况，可以用 LOF 函数确定记录的最大编 号。

8） **PUT**（把一个记录按随机存取方式写入磁盘）

PUT *nmexp* 1，*nmexp* 2

其中 *nmexp* 1　指定随机存取文件缓冲器的编号，*nmexp*1 =1，2，…，15

nmexp 2　指定文件的记录编号，*nmexp*2=1，2，…，335。取决于盘上还留有多少可用空间。如果略去 *nmexp* 2，就假定为当前记录编号

这个语句从文件缓冲器中把数据移入文件在磁盘上的指定空间。把数据移入磁盘以前，必须：

（1）用 OPEN 语句打开文件，并设置缓冲器和确定存取方式（必须是 R），

（2）用FIELD 语句把缓冲器划分区段，

（3）用 LSET 和 RSET语句把数据存入缓冲器。

BASIC 系统遇到语句：

PUT　*nmexp*1，*nmexp*2

就做下列工作：

（1）取得存取磁盘文件的必要信息，

（2）查看缓冲器的类型（必须是 R），

（3）询问是否有存储文件所需的磁盘空间，这必须和用 *nmexp*2 指定的记录相符，

（4）把缓冲器的内容复写到磁盘文件的指定记录中去，

（5）自动调正当前记录编号为 *nmexp*2 + 1。

“当前记录”是编号比上一次存取的记录编号大 1 的那个记录。对某一个缓冲器第一次存取一个文件时，当前记录置 1 。

如果企图写入的记录编号比文件结束的编号还大，则 *nmexp*2 就成为新的文件结束记 录编号。

这一点很重要。当你企图写入一个记录，它的编号超过 EOF 记录编号时，就再分配 给磁盘空间，其数量等于新的记录编号比原有全部记录数多出的那么多。

PUT *nmexp*，336

将会产生磁盘溢出（DISK FULL）的信息，因为 TRSDOS 企图找出存放从 1 到 336 个记

录的空间，而实际上软盘最大记录号是 335 。

举例（假定有一个文件名 SAMPLE/BAS，以前已写了 1 个记录，所以 LOF＝1）：

程序语句	功能
1000 OPEN"R" , 1, "SAMPLE/BAS"	打开文件 SAMPLE/BAS，经 1 号缓冲器随机存取
1010 FIELD1, ……	准备缓冲器
1020 LSET……	把数据放入缓冲器
1030 PUT 1	把缓冲器内容复写到当前记录（1 号）上
1035 LSET……	把数据放入缓冲器
1040 PUT1, 30	获得 2 号记录到 30 号记录的磁盘空间。把缓冲器内容复写到 30 号记录里面去。置 LOF＝30
1045 LSET……	把数据放进缓冲器
1050 PUT1, 25	把缓冲器内容复写到 25 号记录中
1055 LSET……	把数据放入缓冲器
1060 PUT1,	把缓冲器内容复写到当前记录（26 号）中

9) **LSET 和 RSET**（把数据置入随机缓冲器区段）

LSET*var* $ ＝*exp* $　　RSET*var* $ ＝*exp* $

其中 *var* $ 是区段名

exp $ 是数据，将把它放入区段名为 *var* $ 的缓冲区中

这两个语句是把字符——字符串数据存放入由 FIELD 语句设置的区段中去。

例如，NM $ 和 AD $ 已被定义为随机存取缓冲器的区段名。NM $ 的长度是 18 个字符，AD $ 的长度是 25 个字符。现在希望把名字 JIM CRICKET,JR. 和地址 2000 EAST PECAN ST. 放入缓冲器的区段，以便写入磁盘。用以下两个语句来完成：

```
LSET  NM$ ="JIM CRICKET, JR. "
LSET  AD$ ="2000 EAST RECAN ST."
```

缓冲器中放置的数据是：

JIM␣CRICKET,JR.␣␣␣	2000␣EAST␣PECAN␣ST.␣␣␣␣␣␣␣
NM $	AD $

从这里看到，在这二个区段中数据字符串的右边都填满了空格。如果用 RSET 代替 LSET 语句，空格就放在左面。这就是 RSET 和 LSET 的唯一差别。例如：

```
RSET  NM$ ="JIM CRICKET, JR. "
RSET  AD$ ="2000 EAST RECAN ST. "
```

区段中放置的数据是：

␣␣␣JIM␣CRICKET, JR.	␣␣␣␣␣␣␣2000␣EAST␣PECAN␣ST.
NM $	AD $

如果字符串变量的长度超过指定缓冲区段的长度，则总是在右边截断，即右边多余的字符被略去。

10) CVD、CVI 和 CVS（把字符串还原成数值形式）

CVD（*exp*$）

其中 *exp*$ 定义一个有八个字符的字符串，它是存放数值字符串的缓冲器区段名。如果 LEN（*exp*$）＜8，就出现“非法函数调用”的错误。如果 LEN(*exp*$)＞8，只用前八个字符

CVI（*exp*$）

其中 *exp*$ 定义有一个有两个字符的字符串，它是存放数值字符串的缓冲器区段名，如果 LEN(*exp*$)＜2 就出现“非法函数调用”错误，如果LEN(*exp*$)＞2 仅使用前面两个字符

CVS（*exp*$）

其中 *exp*$ 定义一个四字符的字符串，是存放数值字符串的典型的缓冲器区段名，如果LEN(*exp*$)＜4就出现“非法函数调用”错误，如果 LEN(*exp*$)＞4 仅使用前四个字符

这个函数把从磁盘读出的数据恢复为数值形式，用 GET 语句读出数据，存入随机存取文件的缓冲器内。

函数 CVD，CVI，CVS 分别是 MKD$、MKI$、MKS$ 的逆变换。

例如：把名 GROSSPAY$ 看作随机存取文件缓冲器中八字节的区段，读取一个记录以后，GROSSPAY$ 中是数值 13123.38 经 MKD$ 变换后的表示形式。

```
PRINT  CVD (GROSSPAY$) —TAXES
```

打印出差值 13123.38—TAXES．而语句

```
PRINT  GROSSPAY$ —TAXES
```

就产生“类型失配”的错误，因为算术表达式中不能使用字符串值。语句

```
A#=CVD (GROSSPAY$)
```

把数值 13123.38 赋与双精度变量 A#．

11) EOF（文件结束检查）

EOF （*nmexp*）

其中 *nmexp* 指定文件缓冲器的编号，*nmexp*＝1，2，…15

这个函数用来检查是否已经取出文件结束标志之前的所有字符。顺序输入时，利用这个

函数就可以避免“输入越界”(INPUT PAST END) 的错误。

自变量 *nmexp* 指定一个已打开的文件，如果没有读到 EOF 记录，EOF (*nmexp*) 取 0 (假)；读到 EOF 记录，就取值 -1 (真)。例如

```
IF EOF (5)  THEN PRINT "END OF FILE" FILENM$
IF EOF (NM%)  THEN CLOSE NM%
```

下列程序是从文件 DATA/TXT 中读出数据置于数组 A( )中。读出文件最后一个字符时，在 30 行中测试 EOF，如果找到 EOF，程序就从磁盘读数的循环中转移出去，这样就避免发生输入越界的错误。另外，变量 I 中存放输入到数组 A( ) 的下标值。

```
5  DIM A(100)'ASSUMING THIS IS A SAFE VALUE
10 OPEN "I", 1, "DATA/TXT"
20 I%=0.
30 IF EOF(1) THEN 70
40 INPUT# 1, A(I%)
50 I%=I%+1
60 GOTO 30
70 REM PROGRAM CONTINUES HERE AFTER DISK INPUT
```

12) LOF (获取文件结束标记的记录编号)

LOF (*nmexp*)

其中 *nmexp* 指随机存取缓冲器的编号，*nmexp* = 1, 2, …15

这个函数告诉用户该文件中记录的最大编号，也就是最后一个记录编号。这对顺序存取和随机存取都有用。例如，当随机存取一个已经存在的文件时，往往需要知道什么时候读取的是最后一个有效记录，LOF 就提供了这种用途。例如：

```
10 OPEN "R", 1, "UNKNOWN/TXT"
20 FIELD 1, 255 AS A$
30 FOR I%=1 TO LOF(1)
40 GET 1, I%
50 PRINT A$
60 NEXT
```

在第 30 行中 LOF(1) 指可存取的最大记录编号。

注：如果读取的记录编号超过了有文件结束标记的记录，BASIC 系统只简单地用十六进制零值填入缓冲器，并不报出错误。

如果要在文件结束标记处增加一个记录，LOF 就告诉用户加到哪里：

```
100 I%=LOF(1)+1'HIGHEST EXISTING RECORD
110 PUT 1, I%  'ADD NEXT RECORD
```

13) **MKD$、MKI$、MKS$（把数值数据转换成字符串数据）**

MKD$(*nmexp*)
其中 *nmexp* 取双精度数值
MKI$(*nmexp*)
其中 *nmexp* 是整型数，－32767＜＝*nmexp*＜＝32768 如果自变量超出这个范围，就出现“非法函数调用”的错误，自变量的小数部份都自动舍去
MKS$(*nmexp*)
其中 *nmexp* 是单精度数

这些函数把数值变成“字符串”，实际上这个字符串反映的数值与原值一样，只是把内部的数据形式变换成字符了，使得数值数据能够放进字符串变量（参看 LEVEL I 参考手册，VARPTR 函数的内部数值表达式）即：

MKD$ 取值为八字节的字符串
MKI $ 取值为两字节的字符串
MKS $ 取值为四字节的字符串
例如：

```
ASC (MKI$(I%))=I%的低位字节，即 (I%AND 255)
ASC (RIGHT$(MKI$(I%), 1))
    =I%的高位字节，即 INT (I%/256)
LSET  AVG$=MKS$ (0.123)
```

这里的 AVG$一定是四字节随机缓冲器的区段，现在它存有单精度数 0.123 的内部表示形式。

```
LSET  TALLY$=MKI$ (I%)
```

区段名 TALLY$ 现在要存放整数 I%的两字节内部表示形式。

```
A$=MKI$ (8/I)
```

A$ 取作为 8/I 的整数部份以两字节表示，小数部份被略去。这里的 A$是普通字符串变量，不是缓冲器区段名。

假设已打开文件 BASEBALL/BAT（非标准文件名扩展）用缓冲器 2 随机存取，缓冲区已分区段如下：

区段	NM$	YRS$	AVG$	HR$	AB$	ERNING$
长度	16	2	4	2	4	4

NM$ 预定存放字符串；AVG$；AB$和 ERNING$ 存放转换后的单精度值；YR$ 和 HR$ 存放转换后的整型数。

希望写入下列数据记录：

SLOW LEARNER 打棒球历时 38 年，一生平均击球.123，返回本垒 11，处在攻位 32768，……净得—13.75。

使用的产生字符串函数如下：

```
1000  LSET  NM$="SLOW LEARNER"
1010  LSET  YRS$=MKI$ (38)
1020  LSET  AVG$=MKS$ (.123)
1030  LSET  HR$=MKI$ (11)
1043  LSHT  AB$=MKS$ (32768)
1050  LSET  ERNING$=MKS$ (-13.75)
```

执行这个程序以后就可以用 PUT 语句把 SLOW LEARNR 的资料写入磁盘。当用 GET 语句从磁盘读出时，必须用 CVI 和 CVS 函数把字符串的数据恢复成数值形式。

4. 顺序存取技术

顺序输入/输出是把数据存入磁盘文件或者把数据送回给 BASIC 变量的最简单的方法。

为了要写入磁盘，先打开文件规定为顺序输出，用 PRINT#语句送出数据，然后关闭文件。要读回数据，只要简单地打开文件规定为顺序读出，直接用 INPUT# 语句把数据读入 BASIC 变量。数据的次序与写入磁盘的次序相同。

顺序输出的一个例子

假定要存储一个英制——公制常数换算表：

英制单位	公制当量
1英寸 (IN[1])	2.54001 厘米 (CM)
1英里 (MI)	1.60935 千米 (KM)
1英亩 (ACRE)	4046.86 平方米 (SQ.M)
1立方英寸 (CU.IN)	0.01638716 公升 (LTR)
1 (美) 加仑 (GAL)	3.785 公升 (LTR)
1 (液体) 夸脱 (LIQ.QT)	0.9463 公升 (LTR)
1磅 (LB)	0.45359 千克 (KG)

先决定采用什么样的数据信息形式。如果希望的形式是：

英制单位——>公制单位，转换因子〈回车〉

那末存储的第一个数据就是

IN->CM,␣2.540010〈EN〉

(这里的〈EN〉表示回车，十六进制码 0D) 下列程序就是建立这个数据文件的程序

```
10  OPEN"O", 1,"METRIC/TXT"
20  FOR I %=1  TO  7
30  READ  UNIT$, FACTR
40  PRINT#1, UNIT$; ", "; FACTR
50  NEXT
60  CLOSE
```

注：1)此表内括号中的缩写符号就是下面程序中使用的缩写符号——译者

```
70  DATA  IN->CM, 2.54001, MI->KM, 1.60935, ACRE->SQ.M, 4046.86
80  DATA  CU.IN->LTR, 1.638716E-2, GAL->LTR, 3.785
90  DATA  LIQ.QT->LTR, 0.9463, LB->KG, 0.45359
```

第10行建立名叫METRIC/TXT的磁盘文件，为此文件设置1号缓冲器，规定采用顺序输出方式。因为顺序输出总是用ASCII代码的形式来存放数据，故需使用名扩展/TXT。

注：如果已经存有文件METRIC/TXT，10行的作用是使这个文件的全部内容消失。这是因为，不管什么时候打开一个要顺序输出的文件，EOF标志就移至文件的起点。TRSDOS已“忘记”了以前是否写过什么东西了。

第40行把UNIT$和FACTR的当时内容送入文件缓冲器。实际上并不马上写到磁盘上去，只有等缓冲器填满了，或者关闭文件时，（不管那一个先发生）才向磁盘写入。由于字符串项内不包含分隔符，没有必要用双撇号（"）括起来。只需加进逗点（,）就可以了。

第60行关闭文件，EOF标志放在最后一个数据项之后，即在0.45359之后。以后输入时，磁盘BASIC系统就可以知道什么时候读完了全部的数据。

顺序输入的例子

下列程序是从文件METRIC/TXT读出数据，放入两个数组，然后进入转换。

```
 5  CLEAR 500
10  DIM UNITS$(9),FACTR(9)'ALLOWS FOR UP TO 10 DATA PAIRS
20  OPEN"I", 1, "METRIC/TXT"
25  I%=0
30  IF EOF (1) THEN 70
40  INPUT#1, UNIT$(I%), FACTR(I%)
50  I%=I%+1
60  GOTO 30
70  REM…THE CONVERSION FACTORS HAVE BEEN READ IN
100 CLS: PRINT TAB(5)"***ENGLISH TO METRIC CONVERSIONS***"
110 FOR ITEM%=0TOI%-1
120 PRINT USING" (##)    %    %"; ITEM%, UNIT$(ITEM%)
130 NEXT
140 PRINT@704, "WHICH CONVERSION";
150 INPUT CHOICE%
155 PRINT@768, "ENTER ENGLISH QUANTITY";
160 INPUT V
170 PRINT "THE METRIC EQUIVALENT IS"V*FACTR(CHOICE%)
180 INPUT" PRESS ENTER TO CONTINUE"; X
190 PRINT@704, CHR$(31); 'CLEAR TO END OF FRAME
200 GOTO 140
```

第20行打开文件，进行顺序输入，读出指针自动地置位在这个文件的起点上。第30行查看是否读到文件的最后记录。如果确实读到了，程序就从磁盘输入循环转移到使用数据的

程序。第40行是读入一个字符串放入数组UNIT$(　　)中去，又读入一个数值放入单精度数组FACTR(　　)中去。注意,这个INPUT输入语句的变量表和建立数据文件的PRINT#输出语句的变量表完全一样（参看“顺序输出的一个例子”）事实上INPUT#的变量表不一定要和PRINT#的完全一样，下面的语句也可以完成：

```
40 INPUT#1, UNIT$(I%): INPUT#1, FACTR(I%)
```

怎样更新文件

如果要把更多的内容加入英制——公制转换文件，不能简单地打开文件进行顺序输出，用PRINT* 语句送入加进的数据。因为这样做立刻就把文件结束标志移至文件开头，破坏了文件原有内容。要做这项工作，应该：

（1）打开文件进行顺序输入，
（2）读入并储存整个文件（存入一个或多个数组内），
（3）关闭文件，
（4）把新内容放入数组或者修正已有内容，
（5）重新打开该文件，进行顺序输出，
（6）把更新后的数据数组输出到文件中去，
（7）关闭文件。

如果文件太大，内存不允许存储这样大的文件，更新这个文件的方法是：

（1）打开旧文件作顺序输入，
（2）把另一个新的数据文件打开作顺序输出，
（3）输入一组数据并作必要的修改，
（4）把该数据块输出到新文件中去，
（5）重复第3、4步，直到从旧文件中读完全部数据，并作完修改，输出到新文件中去；然后才能着手下一步，
（6）关闭两个文件，
（7）删除旧的数据文件，
（8）用旧文件名称命名新文件（用TRSDOS命令RENAME）。

顺序行输入（LINE INPUT）的例子

以行为单位输入的方法,可以编制一个程序，用这个程序编辑另一个BASIC程序的文件,例如重新编行号,把LPRINT改为PRINT等等,但被编辑的程序必须是用ASCII格式存储的。

对名扩展为/TXT的任何磁盘文件，都可用下列程序来计算程序中共有多少行。

```
10  CLEAR 300
20  INPUT"WHAT IS THE NAME OF THE PROGRAM"; PROG$
30  IF INSTR(PROG$,"/TXT )=0 THEN 110 'REQUIRE /TXT EXTENSION
40  OPEN "I" , 1, PROG$
50  I%=0
60  IF EOF(1)THEN 90
70  I%=I%+1:LINE INPUT#1, TEMP$
```

```
80  GOTO60
90  PRINT "THE PROGRAM IS "I%" LINES LONG"
100 CLOSE: GOTO20
110 PRINT "FILESPEC MUST INCLUDE THE EXTENSION '/TXT'"
120 GOTO20
```

用 ASCII 格式存储的 BASIC 程序，每一个程序行结尾都加上回车符〈EN〉，在〈EN〉之前并不加换行符〈LF〉，所以第 70 行中的 LINE INPUT 每次自动地读一整行，并把这一整行送入变量 TEMP$ 之中。实际上变量I% 在计数。

若打算把一个程序文件 DISKDUMP/BAS 变成文本文件存储，可这样进行：

LOAD "DISKDUMP/BAS" [ENTER]

SAVE "DISKDUMP/TXT", A [ENTER]

这样，就给出 DISKDUMP 程序的第二个以 ASCII 格式的文件。现在可以输入行计数程序，检查 DISKDUMP/TXT 程序有多少行。

顺序存取过程中的磁盘存储器

顺序存取较为简单，一般可以不去注意磁盘存储的细节。只需把数据写入和读出。

下面介绍几个技术细节，给你留下一些印象，这几点有时很重要。

（1）PRINT# 语句并不把数据直接写入磁盘，只是把数据放进 256 字节的缓冲器。这个缓冲器填满后，就把内容自动写入磁盘（关闭文件操作也把缓冲器内容写入磁盘）。

（2）执行 PRINT# 语句，出现磁盘已满的错误时 (DISK FULL ERROR)，输出缓冲器当时的内容还没有写入磁盘文件。已在磁盘文件的数据原封未动，不包括 PRINT语句最后送入的几个值。

如果变量中仍然包含这些数据，就可以直接把数据恢复。

5. 随机存取技术

随机存取有几个顺序存取没有的优点：

(1) 不必从文件的开头开始读起，可以读指定的任何记录。

(2) 修改文件时，不必读出整个文件后再修正数据，然后再写回。可以重写选定的任何记录，也可以对选定的记录增加内容，而其它记录不受影响。

(3) 随机存取更有效：占用空间少，读写速度快。

(4) 把文件打开进行随机存取，可以经同一个缓冲器进行读、写。

(5) 随机存取提供了许多有助于组织数据的语句和函数。一旦数据组织好，随机输入/输出就变得十分简单了。

上述最后一条优点也正是随机存取的“难点”。这里有一些想法。

为了理解随机存取，可以把一个磁盘文件看做一组箱子，类似于挂在墙上的邮政信箱。和邮政信箱一样，文件箱也是编了号码的。文件箱的号码是变化的，但总是 5 的倍数。一个非空文件最少有 5 箱，箱号由 1 到 5。为了存储更多的数据，需要更多的存储空间。TRSDOS 以 5 为增量来增大空间。参看图 13。

这些大小固定的箱子称作“记录”(Record)，每一记录中含有 256 个字节，其中 255 个可

用于存储数据。能使用以下类型的语句，把数据放入任何记录或者从任何记录中读出：

PUT 1，5 把1号缓冲器内容写入记录5

GET 1，5 把5号记录内容读进1号缓冲器。

缓冲器是为数据准备的等待区，把数据写入文件以前，必须先把数据放入为该文件设置的缓冲器中。从文件中读出数据以后，要到缓冲器中取出数据。从上述 PUT 和 GET 语句的例题中看到，以256字节为一块把数据传送给磁盘，或从磁盘中取回数据。

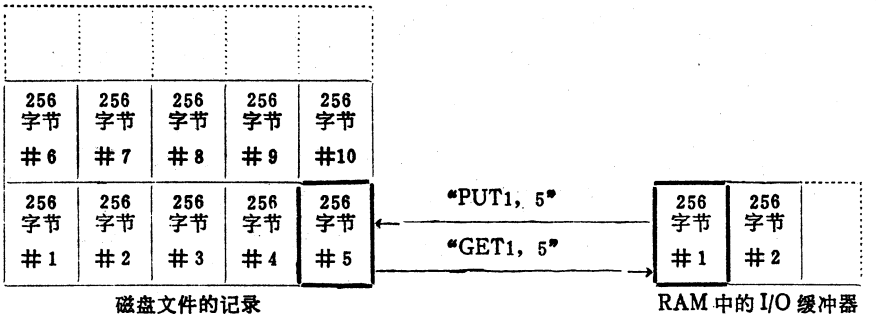


图 13 随机存取

有几种数据类型，但大部份每个数据只占几个字节：

整型数	2 字节
单精度数	4 字节
双精度数	8 字节
字符串	最多 255 字节

因而把缓冲器内容送入磁盘文件以前，应该多把几个数值放入缓冲器中，以免浪费磁盘空间。做法是：

（1）把缓冲器划分成区段（field）并加以命名。

（2）把字符串或数值数据放入区段。

例如，准备在磁盘上存储词汇表，每个记录由一个字和这个字的定义所组成。使用的语句为：

```
100 OPEN "R", 1, " GLOSSARY/BAS"
110 FIELD 1, 15 AS WD$, 240 AS MEANING$
```

第100行打开名为 GLOSSARY/BAS 的文件（如果不存在这个文件，就建立文件），用1号缓冲器随机存取这个文件。第110行在缓冲器1内定义两个区段：WD$由缓冲器的前15个字节组成，后240个字节构成 MEANING$。这里的 WD$ 和 MEANING$ 是区段名。

区段名和字符串名有何差别？

多数字符串变量是表示内存中的某一块区间，称为字符串空间。在这区间中存放字符串的值。而区段名是表示由 FIELD 语句设置的缓冲器区间，所以语句

```
10 PRINT WD$ ":" MEANING$
```

是把上述定义的两个缓冲器区段的内容显示出来。如果不先把数据放入缓冲器，而来读区段中的值是没有任何意义的。LSET、RSET、GET 都可以完成把数据放入缓冲器的功能。下面先用 LSET 和 RSET 做向磁盘输出数据的准备工作。

按 LSET 的定义，我们先放入的是文字 LEFT——JUSTIFY。

```
100  OPEN "R", 1, "GLOSSARY/BAS"
110  FIELD 1, 15 AS WD$, 240 AS MEANING$
120  LSET WD$ = "LEFT—JUSTIFY"
130  LSET MEANING$ ="TO PLACE A VALUE IN A FIELD FROM
     LEFT TO RIGHT; IF THE DATA DOESN'T FILL THE FIELD, BLANKS
     ARE ADDED ON THE RIGHT; IF THE DATA IS TOO LONG, THE EXTRA
     CHARACTERS ON THE RIGHT ARE IGNORED.LSET IS A LEFT—JUS-
     TIFY FUNCTION."
```

第 120 行把引号中的值从左端起存入 1 号缓冲器的第一个区段。130 行以相同的方式存放带引号的字符串，在打入 130 行时，需要插进换行符(LF)（按"↓"键），把整个字符串分成几段，就如上述程序所表示的那样。当把数据读回到字符串变量时，就比较容易地把数据输出了。

注意：RSET 把区段内左边放满空格。需要截断时仍在右边截去多余的字符。

现在，可以用简单的 PUT 语句把缓冲器中的数据写入磁盘：

```
140  PUT  1, 1
150  CLOSE
```

140 行把第一个记录写入文件 GLOSSARY/BAS.

使用下面的程序读出并打印 GLOSSARY/BAS 的第一个记录。

```
160  OPEN R", 1, "GLOSSARY/BAS"
170  FIELD 1, 15 AS WD$, 240 AS MEANING$
180  GET 1, 1
190  PRINT WD$":"MEANING$
200  CLOSE
```

因为 150 行已把文件关闭了，所以还需要用 160、170 行语句。如果不把文件关闭，可以直接执行语句180 。

随机存取：一般方法

上面的例题说明了使用随机存取方式进行读写的必要的顺序，但没表现出这种存取形式的主要优点，特别是没有说明怎样用直接对一记录进行修改的方法更新一个已有文件。下列程序 GJOSSACC/BAS 要把上述例题的词汇表进行扩充，以便说明随机存取方法怎样对文件进行修改的一些技术。在考察这个程序以前，先研究用随机存取方式建立和维护文件的一般方法。

步骤编号		GLOSSACC/BAS 程序的行号
1	打开文件	110
2	缓冲区划分区段	120
3	读取要修改的记录	140
4	显示这个记录的当前内容	145—170

（显示数值数据前先使用CVD、CVI、CVS函数）

5 用LSET或RSET把新内容送入区段 210—230

（送数值数据前要用MKD$、MKI$、MKS$函数）

6 把修改完的内容写入记录中

7 还要修改另一个记录就得从第3步重复起，否则转向第8步 250—260

8 关闭文件

```
10  REM...GLOSSACC/BAS.
100 CLS:CLEAR 300
110 OPEN"R", 1, "GLOSSARY/BAS"
120 FIELD 1, 25 AS WD$, 228 AS MEANING$, 2 AS NX$
130 INPUT"WHAT RECORD DO YOU WANT TO ACCESS", R%
140 GET 1, R%
145 NX%=CVI(NX$) 'SAVE LINK TO NEXT ALPHABETICAL ENTRY
150 PRINT"WORD: "WD$
160 PRINT"DEF'N:":PRINTMEANING$
170 PRINT"NEXT ALPHABETICAL ENTRY: RECORD#"NX%: PRINT
180 W$=""":INPUT "TYPE NEW WORD〈EN〉 OR 〈EN〉 IF OK"; W$
190 D$=""":PRINT" TYPE NEW DEF'N 〈EN〉 OR 〈EN〉 IF OK?"; LINE-
    INPUTD$
200 INPUT "TYPE NEW SEQUENCE NUMBER OR 〈EN〉 IF OK";NX%
210 IF W$〈〉"" THEN LSET WD$=W$
220 IF D$〈〉"" THEN LSET MEANING$=D$
230 LSET NX$=MKI$(NX%)
240 PUT 1, R%
245 R%-NX% 'USE NEXT ALPHA LINK AS DEFAULT FOR NEXT RECORD
250 CLS:INFUT"   TYPE〈EN〉 TO READ NEXT ALPHA ENTRY, OR
RECORD#〈EN〉 FOR SPEC IFIC ENTRY,OR 0 〈EN〉 TO QUIT";R%
260 IF 0<R% THEN 140
270 CLOSE
280 END
```

注意：我们在记录中加进了一个区段NX$(120行)、NX$是用来存放下一个记录的编号，但该编号按字母顺序编排。倘若我们知道哪一些条目要排在前面，这样可对词汇表按字母顺序进行查阅。例如，词汇表内有：

记录号	字(WD$)	定义	指向下一字母条目(NX$)
1	LEFT-JUSTIFY	……	3
2	BYTE	……	4
3	RIGHT-JUSTIFY	……	0
4	HEXADECIMAL	……	1

读记录 2 (BYTE)时，告诉我们下一个是记录 4 (HEXADECIMAL)，记录 4 又告诉我们下一个是记录 1 (LEFT-JUSTITY)。最后一条记录 3 指示 0，它表明词汇表结束。因为 NX$ 只能存放整型数，必须先用 MKI$ (230 行) 把数转换为双字节字符串表达形式。

```
RUN"GLOSSACC/BAS" [ENTER]
```

```
WHAT RECORD DO YOU WANT TO ACCESS? 4 [ENTER]
WORD: HEXIDECIMAL
DEF'N:
CAPABLE OF EXISTING IN ANY OF 16 STATES, E.G., THE
HEXADECIMAL DIGITS 0, 1, 2, ···, 9, A, B, C, D, E, F.
HEXADECIMAL NUMBERS ARE STRINGS OF HEXACECIMAL
DIGITS.

NEXT ALPHABETICAL ENTRY: RECORD# 1

TYPE NEW WORD〈EN〉[OR]〈EN〉 IF OK? HEXADECIMAL
[ENTER]
TYPE NEW DEF'N〈EN〉 OR 〈EN〉 IF OK? [ENTER]
TYPE NEW SEQUENCE NUMBER OR 〈EN〉 IF OK? [ENTER]
```

```
    TYPE〈EN〉 TO READ NEXT ALPHA ENTRY,
    OR RECORD # 〈EN〉 FOR SPECIFIC ENTRY,
    OR 0 〈EN〉 TO QUIT? [ENTER]
WORD: LEFT-JUSTIFY
DEF'N:
TO PLACE DATA IN A FIELD FROM LEFT TO RIGHT, AD-
DING BLANKS AS NECESSARY ON THE RIGHT TO FILL
THE FIELD. ANY EXTRA CHARACTERS ON THE RIGHT
ARE IGNORED.

NEXT ALPHABETICAL ENTRY: RECORD# 3

TYPE NEW WORD〈EN〉 OR 〈EN〉 IF OK? [ENTER]
TYPE NEW DEF'N〈EN〉 OR 〈EN〉 IF OK? [ENTER]

TYPE NEW SEQUENCE NUMBER OR 〈EN〉 IF OK? 2 [ENTER]
```

下列程序按字母次序显示词汇表。

```
300  REM…GLOSSOUT/BAS…
310  CLS: CLEAR 300
320  OPEN"R", 1, "GLOSSARY/BAS"
330  FIDLE 1, 15 AS WD$, 238 AS MEANING$, 2 AS NX$
340  INPUT"WHICH RECORD IS FIRST ALPHABETICALLY"; N%
350  GET 1, N%
360  PRINT: PRINTWD$
370  PRINTMEANING$
380  N%=CVI(NX$)
390  INPUT"PRESS ENTER TO CONTINUE"; X
400  IF N%<>0 THEN 350
410 CLOSE
420  END
```

子记录

上面词汇表的例题中，每条都要把缓冲器的255个字节填满。然而往往不是这种情况，经常每个信息单位仅能填满缓冲器的一部份。在缓冲器中定义几个相同的子记录(Sub-record)，是一个很好的设想。用PUT语句把只有几个字节的有用信息传送给记录，不浪费磁盘空间。例如，我们希望存储一张邮递名单，每个条款包括下列几项：

区段	区段长
姓名	18
地址	25
城市	14
国家	2
购买数量	4

这样每条总长有63字节。

注意：最后一项购买数量是一个单精度的数，需要4字节。因此区段长为4。

如果不考虑磁盘空间的浪费，就使用下面的语句：

```
FIELD 1, 18 AS NM$, 25 AS AD$, 14 AS CTY$, 2 AS ST$,
                    4 AS LP$
```

用PUT语句把这样一个缓冲器内容送入记录，则仅有63字节中有信息，而255－63＝192字节没有用上。更有效的方法是把缓冲器分成几个相同的子记录。在上述情况中，可以建立255/63＝4个子记录，这时只在记录的结尾处浪费了3个字节。

如果不想使用很长的FIELD语句来设置每一区段，则可以对缓冲区一次划分子记录，使用哑字符串STARTHERE$以确定每一子记录在缓冲器中的起始位置。

```
FOR I%=0 TO 3
    FIELD 1, (I%*63) AS STARTHERE$,
```

```
        18  AS  NM$(I%), 25 AS AD$(I%),
        14  AS  CTY$(I%), 2  AS ST$(I%),
        4   AS  LP$(I%)
    NEXT
```

第一次进入循环时，STARTHERE$长度为零，因此NM$（0）从第一个字节开始，AD$(0)在第19字节，等等；LP$(0)在第63个字节结束。

第二次进入循环，STARTHERE$长度为63，因此NM$(1)从第64字节开始，AD$(1)在92字节，LP$(1)在第126字节结束。循环四次后，整个缓冲器就都划分好了。

把数值放入缓冲器子记录的方法：假定邮递名单的各列条款已存储在四个数组内：N$( )，A$( )，C$( )，S$( )，LP$( )。

这样就可以按下列程序把四项条款都置入缓冲器中去：

```
FOR  I%=0  TO  3
LSET  NM$(I%)=N$(I%)
LSET  AD$(I%)=A$(I%)
LSET  CT$(I%)=C$(I%)
LSET  ST$(I%)=S$(I%)
LSET  LP$(I%)=MKS$(LP(I%))
NEXT
```

怎样存取子记录

上述文件中每个记录内部都包含4个子记录，我们需要寻找一种方法，把我们需要的子记录取出来。这就要求每一个子记录只有一个编号，这个编号也只和这一个子记录相关连。例如，要打印邮递名单，从1号记录的第一个子记录开始，一直打印到最后一个记录的最后一个子记录。因此要从1开始按顺序给每一个子记录编号。下面的公式用来编制这个号码（称为键号—Keynumber），严格确定文件中子记录所在位置：

如果子记录的键号记为KEY%，则

$$PR\%=INT((KEY\%-1)/4)+1$$

PR%是存放该子记录的实际记录号码，

$$SR\%=KEY\%-4\cdot(PR\%-1)$$

SR%是子记录在实际记录内的编号。例如，希望存取的条款键号为37（第37条），则它所在的实际记录是：

$$INT((37-1)/4)+1\Longrightarrow 10\text{号记录}$$

10号记录中子记录的位置是：

$$37-4\cdot(10-1)+1\Longrightarrow 1\text{号子记录}$$

下面是建立和控制邮递名单的整个工作程序：

```
100 CLEAR 1000
110 OPEN"R", 1, "MAIL/BAS"
120 CLS: INPUT"TYPE 1〈EN〉 TO WRITE, 2〈EN〉 TO READ, 0 〈EN〉 TO
    QUIT"; N%
```

```
130 IF N%=0 THEN CLOSE:END
140 INPUT"TYPE KEY NUMBER〈EN〉 OR 0〈EN〉"; KEY%
150 IF KEY%=0 THEN 120
160 PR%=INT((KEY%-1)/4)+1
170 SR%=KEY%-4*(PR%-1)
180 FIELD 1, (SR%*63) AS STARTHERE$, 18 AS NM$, 25 AS AD$,
    14 AS[1] CTY$, 2 AS ST$, 4 AS LP$
190 GET 1, PR%
200 IF N%=2THEN300
210 PRINE"WRITING SUBRECORD#"SR%"IN PHYSICAL RECORD#
    "PR%
220 PRINT:PRINT"NAME?"TAB(20);: LINEINPUT N$: LSET NM$=N$
230 PRINT"ADDRESS?"TAB(20);: LINEINPUT A$:LSET AD$=A$
240 PRINT"CITY? TAB(20);:LINEINPUT C$: LSET CTY$=C$
250 PRINT"STATE?"TAB(20);: LINEINPUT S$: LSET ST$=S$
260 PRINT"LAST PURCHASE"TAB(20);:INPUTP!
270 PUT 1, PR%:PRINT:INPUT"PRESS 〈EN〉 TO GO ON";X: GOTO 120
300 PRINT"READING SUBRECORD# "SR%"IN PHYSICAL RECORD#
    PR%
310 PRINT:PRINT"NAME"TAB(20)NM$
320 PRINT"ADDRESS"TAB(20)AD$
330 PRINS"CITY"TAB(20)CTY$
340 PRINT"STATE"TAB(20)ST$ :LSET LP$=
350 PRINT USING"LAST PURCHASE     $#####.##";CVS(LP$)
360 PRINT: INPUT"PRESS 〈EN〉 TO GO ON"; X: GOTO 120
```

这个程序并不需要用四个子记录一下子就把缓冲器填满。当你把一个子记录一旦放进区段的正确位置时，整个缓冲器就写入磁盘。然而没有写入内容的子记录空间并没有浪费，以后可以把这个子记录的内容再加进去。

但这样做并不是最有效的，因为不能一次就建立出一张名单。要想一次建立，最好还是在写磁盘之前用四个子记录把缓冲器填满。上述程序只是说明怎样用随机存取技术修改一个文件。

```
RUN"MAILACC/BAS" [ENTER]
```

注：1) 原文有错，译者已改正，调试通过——译者

```
TYPE 1 〈EN〉TO WRITE, 2〈EN〉 TO READ,
   0〈EN〉 TO QUIT? 1 [ENTER]
TYPE KEY NUMBER〈EN〉OR 0〈EN〉? 3 [ENTER]
WRITING SUBRECORD# 3 IN PHYSICAL RECORD# 1
NAME?                DUNMAN, I.C. [ENTER]
ADDRESS?             2222 SECOND STREET [ENTER]
CITY ?               OLD PORT [ENTER]
STATE?               AZ [ENTER]
LAST PURCHASE?       12.81 [ENTER]
PRESS〈EN〉TO GO ON? [ENTER]
```

```
TYPE 1 〈EN〉 TO WRITE, 2〈EN〉 TO READ,
   0〈EN〉 TO QUIT? 2 [ENTER]
TYPE KEY NUMBEP〈EN〉 OR 0〈EN〉? 1 [ENTER]
READING SUBRECORD# 1 IN PHYSICAL RECORD# 1
NAME                 JOHNSON, J.R.
ADDRESS              1024 RAM DRIVE
CITY                 FORT WUMPUS
STATE                TX
LAST PURCHASE        $188.75
PRESS〈EN〉 TO GO ON? [ENTER]
```

```
TYPE 1 〈EN〉 TO WRITE, 2〈EN〉 TO READ,
   0〈EN〉 TO QUIT? 0 [ENTER]
READY
〉_
```

区段的重迭

如果希望用两种方法——整体或部份——来存取同一个区段。就可以给缓冲器的同一区段设置两个区段名。例如，6位数字的货栈号的前两个数字代表货物种类，就可以使用下面的区段结构：

```
FIELD 1, 6 AS STOCK$, ……
FIELD 1, 2 AS CTG$……
```

现在，STOCK $访问整个货栈编号区段，而CTR $ 只访问货栈编号区段的前两位。

6. 磁盘BASIC错误信息

代码	信　息	注　释
50	FIELD OVERFLOW	分配给随机存取缓冲器的内容太多，超过了255字节
51	INTERNAL ERROR	磁盘操作系统本身出错或磁盘输入输出失败
52	BAD FILE NUMBER	文件——缓冲器编号使用不当。OPEN语句没有把此编号赋与文件
53	FILE NOT FOUND	读磁盘内没有存放的文件。請检查指定的文件名/名扩展，看是否正确
54	BAD FILE MODE	对磁盘文件进行I/O的方式与打开文件时给的方式不符
57	DISK I/O ERROR	计算机和磁盘文件间传输数据中出错
61	DISK FULL	软盘上可用空间已用完
62	INPUT PAST END	由磁盘以顺序方式读出数值赋于变量时，还没有读到数据就遇到文件结束标志
63	BAD RECORD NUMBER	PUT语句中记录号码超出范围(1，340)
64	BAD FILENAME	文件标识符不合适，参看"TRSDOS概述"文件标识符一节
66	DIRECT STATEMENT IN FILE	要装入、运行或合并的磁盘文件不是BASIC程序
67	TOO MANY FILE	在一个磁盘上存放的文件太多，一个盘上最多只能存取48个
68	DISK WRITE—PROTECTED	企图在一个有写保护的磁盘上写文件
69	FILE ACCESS DENIED	使用错误的通行字来存取现有文件

注意：磁盘错误不能用ERROR语句来模拟。

八、附　录

1. 词　汇　表

access 存取

是一种方法，按这种方法从磁盘上读入信息，或把信息写入磁盘。请参看随机存取和顺序存取。

address 地址

是存储器内的一个单元，通常用两字节的十六进制数来表示。地址范围是〈0至FFFF〉，以十进制表示为〈0至32767〉〈-32768，…-1〉

alphabetic 字母

A 至 Z之间的字母。

alphanumeric 字母数字

是字母和数字，包括 A 至 Z， 0 至 9 。

argument 宗量、自变量

是字符串或者数值，把此量代入函数进行运算，得到一个结果，这个结果就称为函数的值。

array 数组

是一组元素，若使用数组的名称，表示整个数组，若用一个或几个下标变量，则表示单个元素。在 BASIC 中任何变量名称都可作为数组名称，数组有一维数组和二维数组。AR( ) 表示一个一维数组，名叫 AR；AR(,)表示一个二维数组，名叫 AR。

ASCII 美国标准信息交换码

这种编码方法用来存储文本数据。数值数据一般地是以压缩形式存储的。

ASCII format disk file ASCII格式的磁盘文件

是一种磁盘文件，在这种文件中每一个字节对应原始数据的一个字符。例如 BASIC 程序按 ASCII格式存储，这个文件除了每一个字符是 ASCII码以外，看起来象一个程序清单。与压缩格式的文件比较就明确了。

background task 后台任务

指相对不急需处理的任务。计算机把别的后台任务也放在一起运行，受中断的约束。一旦中断任务完成后就继续处理后台任务。请参看前台任务。

backup disk 复制磁盘

是原始磁盘的复制品，一种“安全付本”。你必须保存有一个 TRSDOS 的原件和其它重要的数据磁盘的付本。

BASIC BASIC 语言

是 Beginner All-purpose Symbolic Instruction Code (初学者通用符号指令) 的缩写，是一种程序语言，存储在 TRS-80 的 ROM 中，Radio Shack 公司有 LEVEL Ⅰ BASIC，LEVEL Ⅱ BASIC 及 DISK BASIC。LEVEL Ⅱ BASIC 是 DISK BASIC 的一部份。

baud 波特

信号传输的速度。以每秒比特为单位。LEVEL Ⅱ 的磁带接口是以 500 波特速度工作的。

binary 二进制

只有二种可能的状态，也就是二进制数 0 和 1 。二进制 (以二为基) 的计数制使用一串 0 和 1 来表示一个数量。这与计算机数据的内部表示，也就是用电位值表示 0 与 1 十分相似。

bit 比特

一位二进制，是计算机存储器的最小单位，它能表示数值 0 及 1 。

bootstrap program 引导程序

是一个最基本的程序，当计算机的电源一打开，它使计算机能装入并执行高级的程序，也就是通过“自身的引导”，计算机使自己充实起来。引导程序是一个使计算机初始化的程序。

break 间断

间断一个程序的进行。在BASIC中，语句STOP使程序执行间断，按BAEAK键也起相同的作用。

buffer 缓冲器

是RAM中的一个区域，把数据存放在那里作进一步的处理。例如，把数据从BASIC传送给磁盘文件，或者从磁盘传送给BASIC，数据都必须通过文件缓冲器。

buffer field 缓冲器区段

是缓冲器的一部份，把它定义为一个缓冲器区段变量的存储区。把缓冲器划分成若干个区段，能使多个数值与磁盘进行交换。

byte 字节

是计算机存储器中可寻址的最小单位，通常由相邻的8个比特所组成，能表示256种不同数值，也就是能表示十进制数0到255。

compressed-format 压缩格式

是一种占存储单元较少的信息存储方法，它比标准ASCII格式占有较少的存储单元。一个整数占两个字节，一个单精度数占四字节，一个双精度数占八字节，不管在文本中要用多少字符来表示这些数，占用的字节总是如此。字符串不能用压缩格式来存储。

在RAM中的BASIC程序，及非ASCII码的磁盘文件都用压缩格式存储，以单字节特殊编码的BASIC键盘字，也用压缩格式存储。

command file 命令文件

是一类名扩展为/CMD的TRSDOS磁盘文件。这类文件必须由Z-80可执行的程序所组成，因为当你打入

fillename [ENTER]

之后，TRSDOS就装入该文件，并立即执行它。

命令文件能存在任何磁盘上，实际上它们扩充了TRSDOS的库命令（虽然它们对TRSDOS的系统文件来说，仍旧是外部的）。

close 关闭

结束磁盘文件的存取。在对文件重新存取之前，必须重新打开文件。

data 数据

是一些信息，它们由程序传送到我们的输出设备。在LEVEL Ⅱ和DISK BASIC中有四种类型的数据：

- 整型数
- 单精度浮点数
- 双精度浮点数
- 字符——字符串序列，或者叫“字符串”

data/device control block (DCB) 数据/设备控制块

是在RAM中与I/O缓冲器有关的一块区域，在其中存有操作系统需要的信息，用以对外设进行I/O和对文件进行存取

debug 查错

从程序中检出和排除逻辑、语法错误。

decimal 十进制

能有十种状态，例如十进制数字 0，1，…9。十进制（以 10 为基）数是日常使用的体制，它使用一串十进制的数字。十进制数字在计算机中是以二进制的编码存储的。

default 空缺

在你不规定一个动作或使用的数值时，由计算机提供的一个动作或数据。

delimiter 分隔符

是一个字符，它表示数据项的开头或结尾，但它不是数据项的一部份。例如双撇号在 BASIC 中是字符串的分隔符。

destination 目的地

是设备或者地址，它们在数据传送中接受数据。例如，在 BACKUP 操作中，目的盘就是进行源盘复制的目的地。

device 设备、装置

是计算机系统中用于数据输入/输出的实体器件，例如键盘显示器，行式打印机，盒式磁带机，磁盘，语音合成器等。

directory 目录

是文件的清单，它保存在每个磁盘上。

disk drive or Mini Disk drive 小型磁盘驱动器

是一个实体器件，它把数据写到磁盘上去，或从磁盘上取回数据。

diskette or disk 磁盘

是一种磁性记录的物体，用来存储大量的数据。

drive specification or drivespec 驱动器标识符

是 TRSDOS 文件标识符中及某些 TRSDOS 命令中的一个可选用的字段，它由一个冒号，后面跟一个数字（0 至 3）所组成。驱动器标识符是用来规定使用哪一个驱动器进行磁盘的读写。

当读的命令中不选用驱动器标识符时，则 TRSDOS 为了寻找所需的文件，将从 0 号驱动器开始，搜查所有的磁盘。

当写的命令中不选用驱动器标识符时，TRSDOS 为了寻找所需的文件，一般只在没有写保护的驱动器中搜寻。

drive number 驱动器编号

是一个 0 至 3 的整数，代表某一个磁盘驱动器。0 号驱动器最接近扩展接口，3 号驱动器离扩展接口最远。在 0 号驱动器中必须始终插着 TRSDOS 的磁盘，只有二种例外情况。

dummy variable 哑变量

是一种变量的名称，把它用在表示式中，以满足语法上的要求，而它的数值对程序员来说是毫无意义的。

edit 编辑

改变已有的信息。

end of file 或 **EOF** 文件结束

是一个标志，表明盘磁文件的结尾，也就是在这个标志处，有意义的数据已经结束，而未确定的数据从此处开始。

entry point 入口点

是机器语言程序开始执行的地址。它与编制该程序的起始地址不必一致，入口点也称为转移地址。

expression 表达式

是一个或几个变量，常数，操作符和函数所组成的有意义的关系序列。

field 区段

是随机存取文件缓冲区中用户定义的更小的区间，可由 FIELD 语句来建立及命名这些小区间。

field name 区段名

是一个字符串变量，它用 FIELD 语句赋与随机存取文件缓冲区的区段。

file 文件

有关数据的有机集合。在 TRSDOS 中，一个文件是最大的信息块，可用一个命令分配它的地址。BASIC 程序和数据块在磁盘上都存储在不同的文件中。

file extension 文件名扩展

是文件标识符中一个可选用的字段，它由/后跟一个字母，或者再加上二个字母数字所组成，可用名扩展区别文件的类型，例如，/BAS，/TXT，/CIM，它们分别是 BASIC，文本及内存映象的程序。

filename 文件名

是文件标识符中一个必须有的字段，它由一个字母及其后面的字母数字（最多可加到七个）所组成。当文件建立或重新命令名时，文件名就被确定了。

file spacification 或 filespec 文件标识符

是一串字符，在 TRSDOS 中用来规定某一个特定的磁盘文件，它至少必须由一个文件名组成，此外，其后还可选用名扩展，通行字及驱动器标识符。

foreground task 前台任务

是指迫切需要执行的程序，计算机必须周期性地顺序执行一些前台任务。这种任务是中断启动的。请参阅 ***background task, task, interrupt.***

format 格式化

通过 TRSDOS FORMAT 实用程序，把一个消磁后的盘或新盘划分成磁道及扇面。BACKUP 也把一个空白磁盘格式化。格式化后的磁盘有 35 个磁道，每个磁道分为 10 个扇面。

granule 区

是磁盘中可分配空间的最小单位，它由五个扇面组成。

hexadecimal 或 hex 十六进制

能有十六种状态，例如十六进制的数字 0，1，2，…，9，A,B,C,D,E,F．十六进制（以 16 为基）数是由一串十六进制的数字所组成的。地址及字节的内容经常都用十六进制的形式来表示。在磁盘 BASIC 下，输入十六进制常数时，要在常数前加上前缀 &H．

increment 增量

是一个数值，在一个重复过程中，每经历一周，就把这个数值加到计数器上。

input 输入

把数据从计算机的外部（例如磁盘文件，键盘等）传送到RAM中。

interrupt　中断

是一个信号，它使计算机停止正在进行的工作，而去完成某一种特殊任务，只有当计算机把后一个任务完成后，才返回去执行前一个任务。在TRS-80扩展接口中，有一个25毫秒的“中心节拍”中断，用它可启动实时时钟和其它的前台任务。中断启动任务可以预定和安排不同的优先级，因此计算机在每一时刻都似乎在做二件或二件以上的事情。

kilobyte或K　千字节

等于内存的1024字节，因此12K ROM等于12•1024＝12288字节。

library commands　库命令

是一组互相覆盖的TRSDOS命令，在需要时把它们调入RAM的覆盖区(5200至6FFF)，若要看有多少个库命令，可使用TRSDOS的LIB命令：

LIB〈EN〉

logical expression　逻辑表达式

是一个表达式，它只计算出二个值：真（＝－1）及假(＝0)。

logical record　逻辑记录

是一个数据块，占有1至256个字节，并把它作为一个单位，进行寻址。而一个逻辑记录实际上可能占一个物理记录，也可能跨越二个物理记录，但这是没有关系的。

machine language　机器语言

Z-80的指令集合，通常使用十六进制代码表示。所有高级语言都要翻译成机器语言，以便使计算机能够执行。

null string　零字符串

是一个字符串，其长度为零。例如

A$＝""

A$就是零字符串。

object code　目标码　目的码

是从“源码”例如汇编语言转换过来的机器语言。

octal　八进制

能有八种状态，例如八进制的数字0，1…，7。八进制（以8为基）数是由一串八进制数字所组成。地址和字节中的内容经常用八进制的形式表示。在DISK BASIC下，输入八进制常数时，要在常数前加前缀 &O.

open　打开

准备好文件，以便进行顺序输入，顺序输出和随机输入/输出的文件存取。

outptut　输出

把数据从计算机内存中传送到某些外部领域中去，例如送往磁盘文件，行式打印机等。

overloy　覆盖

在RAM中以另一个代码块代替一个代码块。例如，TRSDOS系统程序都存储在磁盘中，是以覆盖的方式把它们装入RAM公用区中的。

parameter　参数

是一个可选用的信息，要和命令一起提供，以规定命令如何进行。TRSDOS的参数都

要放在括号内。

password　通行字

是文件标识符中一个可选用的字段，它由一个字母及其后的字母数字(最多可加到七个)所组成。如果在文件建立时没有规定通行字，则八个空格就成为空缺通行字。为了存取文件，你必须在文件标识符中规定通行字。

用TRSDOS的ATTRIB命令，可以规定文件的更新通行字和存取通行字，存取通行字只给予文件有限的存取等级，而更新通行字允许文件整体存取。

physical record　物理记录

是最小的数据单位，用它为单位写到磁盘文件中去，或者从磁盘文件中读出。在TRSDOS中，物理记录由256个字节组成。使用汇编语言的程序员可以不关心物理记录的长度，因为TRSDOS有逻辑记录，它的长度为1至256个字节。

prompt　提示符

是由计算机输出的一个字符或者一些信息，表明计算机正处于等待状态，准备接收键盘输入。

protected file　保护文件

是一个磁盘文件，它有非空白的通行字，因此这个文件只有使用这个通行字才能存取。

protected level　保护级

用存取通行字给予的存取级别：Kill，rename，write，read及execute（删除，重新命名，写，读和执行）。

random access memory 或 RAM　随机存储器

是一种半导体存储器，可以直接编址以便向它读写。“用户RAM”是RAM的一部份，是TRSDOS及DISK BASIC留出来未使用的区间，从7000H到内存最高端。

real-time clock　实时时钟

是一个中断起动的程序，不管当前执行什么后台任务，每25毫秒该程序就更新某些存储单元，以反映时间。电源开关一打开，实时钟置00:00:00。当禁止中断时，钟也停走。

reset　复位

表示按一下TRS-80左后面（就在扩展接口联接端的旁边）的复位键。按一下复位，相当于计算机的电源重新打开，只是用户RAM中的内容没有受影响。

resident system program　常驻系统程序

是TRSDOS的一部份，它一直驻在RAM中，它就是TRSDOS的管理程序，当需要其它TRSDOS程序时，都由它调入。

read-only memory 或 ROM　只读存储器

预先编好程序的半导体存储器，可以直接寻址，但只能读，不能写。TRS-80 LEVEL Ⅱ占用12K的ROM，永久性地存储着引导程序，LEVEL Ⅱ BASIC及其它程序。

routine　子程序

是一系列指令，可完成某一种功能，在一个程序中可多次调用它。例如：键盘输入子程序。

sector　扇面

是磁盘上一个磁道的十分之一，它含有256字节的存储单元，在TRSDOS中是一个物

理记录。

sequential access　顺序存取

从磁盘读出或写入磁盘时，只能从头往后顺序地进行，不能直接存取文件中某一个指定记录。

statement　语句

BASIC 中一条完整的指令。

string　字符串

是一串字符，但必须逐字考察它的含意，换句话说，字符串不代表一个数量。例如一个数 1234 与 1000 + 234 是代表同一数量，而字符串 "1234" 不代表量（字符串相加实际上是把字符串合并起来，故 "1234" 等于"1"+2"+"3"+"4")。

system file　系统文件

是 TRSDOS 的一类磁盘文件，有扩展/SYS．这类文件是读保护的。为了避免混乱，用户自己的磁盘文件不要用扩展/SYS。

syntax　句法

是命令和语句中必须的语法。句法一般指标点符号和语句中各成份的次序。参看“Notation Conventions”中常用信息一节，其中描述了本手册中使用的句法缩写。

task　任务

是一个相对基本的程序，计算机定期地运行它，或者根据请求而运行。

track　磁道

是磁盘上的同心园，一个磁盘有 35 个磁道，每个磁道有 10 个扇面，或者 256 字节的存储位。这些磁道不象唱片上的纹路那样实际存在，它们只是磁性的轨道而已。

transfer address　转移地址

参看 entry point。

TRSDOS　TRS 磁盘操作系统

是 TRS-80 磁盘操作系统的缩写，它存储在磁盘上，可装入 RAM 中。

user RAM 或 user memory　用户 RAM 或用户存储器

参看 rendnm access memory。

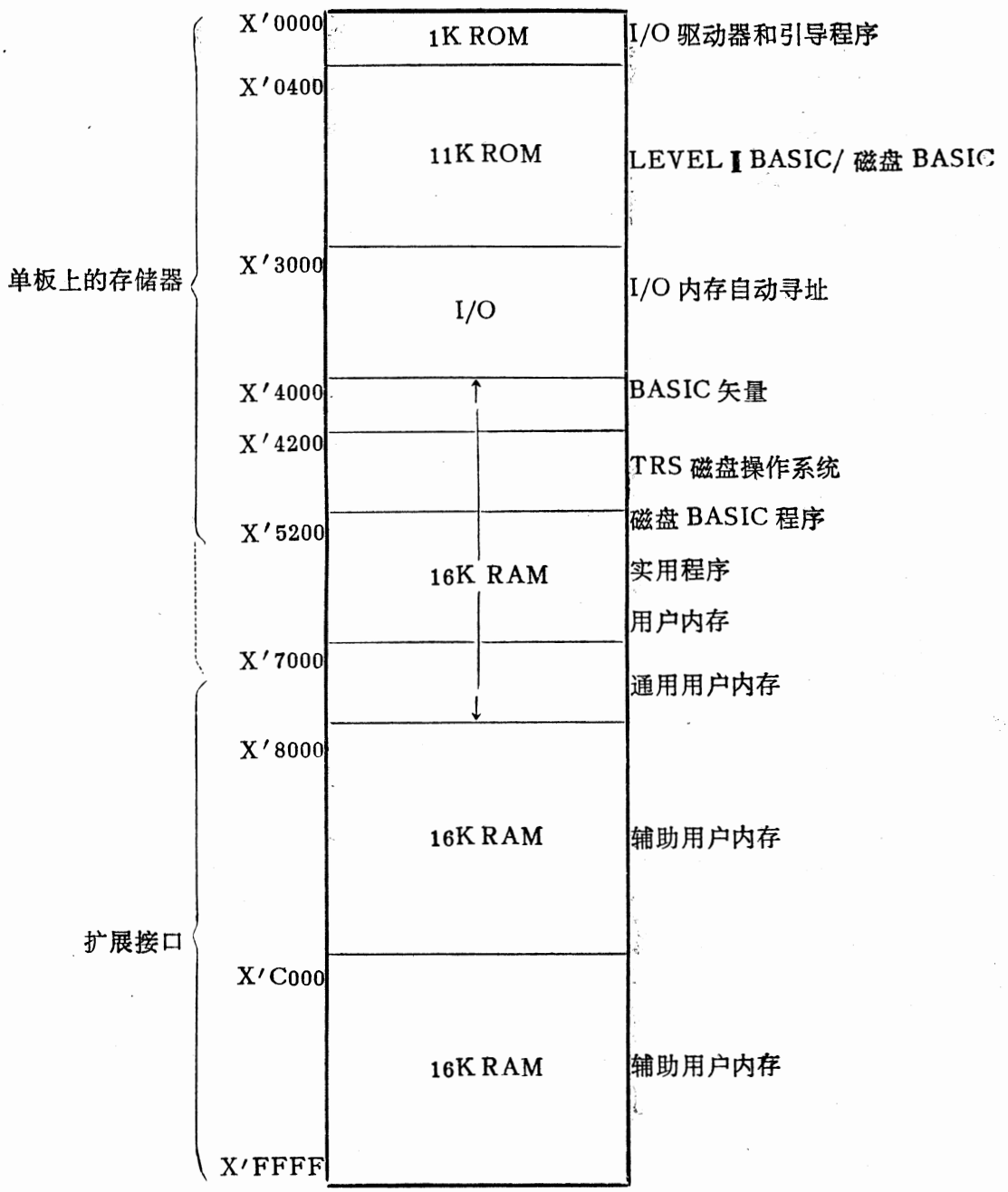
utility　实用程序

是一类程序或子程序，用于一些限定的特殊的目的。有二个扩展的 TRSDOS 实用程序，即 FORMAT和 BACKUP，还有二个非 TRSDOS 的实用程序，它们为 DISKDUMP/BAS 和 TAPEDISK。

write-protect　写保护

防止往磁盘上写任何信息而采取的具体措施。这个措施是在磁盘写保护的缺口上，贴上一片胶带。

2. 内存分配面


单板上的存储器
扩展接口
X′0000
1K ROM
I/O 驱动器和引导程序
X′0400
11K ROM
LEVEL Ⅰ BASIC/ 磁盘 BASIC
X′3000
I/O
I/O 内存自动寻址
X′4000
BASIC 矢量
X′4200
TRS 磁盘操作系统
磁盘 BASIC 程序
X′5200
16K RAM
实用程序
用户内存
X′7000
通用用户内存
X′8000
16K RAM
辅助用户内存
X′C000
16K RAM
辅助用户内存
X′FFFF

3. TRSDOS 符号表

代 码 图

下表包含 128 个 TRSDOS 使用的字符编码。另外还有128个表示图形符号和空格压缩符的代码，请参阅后面的“注解”。

此表的用法：把高（有效）位和低（有效）位结合在一起，就给出了符号的代码。例如，字符 Q 用代码 1010001（十进制81）表示。

高位有效位 (b_7 — b_5)

低位有效位 (b_4—b_1)	000	001	010	011	100	101	110	111
0000	NULL	DLE	SP	0	@	P	`	p
0001	BREAK	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BKSP	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	↑	k	↑
1100	FF	HOME	,	<	L	↓	l	↓
1101	CR	BOL	-	=	M	←	m	←
1110	CURON	EREOL	.	>	N	→	n	→
1111	CUROFF	EREOF	/	?	O	—	o	DEL

十进制/十六进制代码

十进制代码	十六进制代码	符号	十进制代码	十六进制代码	符号	十进制代码	十六进制代码	符号
0	00	NULL	32	20	SPACE	64	40	@
1	01	BREAK	33	21	!	65	41	A
2	02	STX	34	22	"	66	42	B
3	03	ETX	35	23	#	67	43	C
4	04	EOT	36	24	$	68	44	D
5	05	ENQ	37	25	%	69	45	E
6	06	ACK	38	26	&	70	46	F
7	07	BEL	39	27	'	71	47	G
8	08	BKSP	40	28	(	72	48	H
9	09	HT	41	29	)	73	49	I
10	0A	LF	42	2A	*	74	4A	J
11	0B	VT	43	2B	+	75	4B	K
12	0C	FF	44	2C	,	76	4C	L
13	0D	CR	45	2D	-	77	4D	M
14	0E	CURON	46	2E	.	78	4E	N
15	0F	CUROFF	47	2F	/	79	4F	O
16	10	DLE	48	30	0	80	50	P
17	11	DC1	49	31	1	81	51	Q
18	12	DC2	50	32	2	82	52	R
19	13	DC3	51	33	3	83	53	S
20	14	DC4	52	34	4	84	54	T
21	15	NAK	53	35	5	85	55	U
22	16	SYN	54	36	6	86	56	V
23	17	ETB	55	37	7	87	57	W
24	18	CAN	56	38	8	88	58	X
25	19	EM	57	39	9	89	59	Y
26	1A	SUB	58	3A	:	90	5A	Z
27	1B	ESC	59	3B	;	91	5B	↑
28	1C	HOME	60	3C	<	92	5C	↓
29	1D	BOL	61	3D	=	93	5D	←
30	1E	EREOL	62	3E	>	94	5E	→
31	1F	EREOF	63	3F	?	95	5F	—

注：96—127（十六进制60—7F）代码代表的符号是64—95（十六进制40—5F）代码代表的符号的小写体。只有大写体可以显示出来。

注解：

可以把 TRSDOS 符号按它们的功能分成下列几组：

十进制代码	十六进制代码	功能
0—31	00—1F	控制符号
32—95	20—5F	键盘/显示符号
96—127	60—7F	不打印字符
128—191	80—BF	图形符号
192—255	C0—FF	空格压缩代码

下列控制符号可以直接在键盘上打入：

符号	键
BREAK	[BREAK]
BKSP	[←]
HT	[→]
LF	[↓]
CR	[ENTER]
CAN	[SHIFT] [←]
EM	[SHIFT] [→]
SUB	[SHIFT] [↓]
ESC	[SHIFT] [↑]
EREOF	[CLEAR]
SP	[SPACE—BAR]

为了说明图形符号，可以执行下面的程序。如果没有连接行式打印机，就把 LPRINT 语句改为 PRINT 语句，用 SHIFT—@键暂停显示。

```
10 CLS: DEFINT A-Z
20 FORI=128 TO 191
30 POKE 15360, I
35 LPRINT CHR$(138)
40 LPRINT GRAPHICS CODE#"; I
45 LPRINT CHR$ (138)
50 A1=POINT (0, 0) : A2=POINT (1,0)
60 A3=POINT (0, 1) : A4=POINT (1,1)
70 A5=POINT (0, 2) : A6=POINT (1,2)
80 LPRINTTAB(8)CHR$(A1*(-40)+48); CHR$(A2*(-40)+48)
90 LPRINTTAB(8)CHR$(A3*(-40)+48); CHR$(A4*(-40)+48)
100 LPRINTTAB(8)CHR$(A5*(-40)+48); CHR$(A6*(-40)+48)
110 NEXT
```

空格压缩代码提供了一串空格的简洁的表示方法，可以表示从 0 到63个空格的字符串。例如，C0 表示零个空格，C1 表示一个空格，C2表示两个空格，FF表示63个空格。

4.基本转换表

下表列出单个字节值的基本转换关系

十进制	二进制	十六进制	八进制	十进制	二进制	十六进制	八进制
0	00000000	00	000	41	00101001	29	051
1	00000001	01	001	42	00101010	2A	052
2	00000010	02	002	43	00101011	2B	053
3	00000011	03	003	44	00101100	2C	054
4	00000100	04	004	45	00101101	2D	055
5	00000101	05	005	46	00101110	2E	056
6	00000110	06	006	47	00101111	2F	057
7	00000111	07	007	48	00110000	30	060
8	00001000	08	010	49	00110001	31	061
9	00001001	09	011	50	00110010	32	062
10	00001010	0A	012	51	00110011	33	063
11	00001011	0B	013	52	00110100	34	064
12	00001100	0C	014	53	00110101	35	065
13	00001101	0D	015	54	00110110	36	066
14	00001110	0E	016	55	00110111	37	067
15	00001111	0F	017	56	00111000	38	070
16	00010000	10	020	57	00111001	39	071
17	00010001	11	021	58	00111010	3A	072
18	00010010	12	022	59	00111011	3B	073
19	00010011	13	023	60	00111100	3C	074
20	00010100	14	024	61	00111101	3D	075
21	00010101	15	025	62	00111110	3E	076
22	00010110	16	026	63	00111111	3F	077
23	00010111	17	027	64	01000000	40	100
24	00011000	18	030	65	01000001	41	101
25	00011001	19	031	66	01000010	42	102
26	00011010	1A	032	67	01000011	43	103
27	00011011	1B	033	68	01000100	44	104
28	00011100	1C	034	69	01000101	45	105
29	00011101	1D	035	70	01000110	46	106
30	00011110	1E	036	71	01000111	47	107
31	00011111	1F	037	72	01001000	48	110
32	00100000	20	040	73	01001001	49	111
33	00100001	21	041	74	01001010	4A	112
34	00100010	22	042	75	01001011	4B	113
35	00100011	23	043	76	01001100	4C	114
36	00100100	24	044	77	01001101	4D	115
37	00100101	25	045	78	01001110	4E	116
38	00100110	26	046	79	01001111	4F	117
39	00100111	27	047	80	01010000	50	120
40	00101000	28	050	81	01010001	51	121

十进制	二进制	十六进制	八进制	十进制	二进制	十六进制	八进制
82	01010010	52	122	124	01111100	7C	174
83	01010011	53	123	125	01111101	7D	175
84	01010100	54	124	126	01111110	7E	176
85	01010101	55	125	127	01111111	7F	177
86	01010110	56	126	128	10000000	80	200
87	01010111	57	127	129	10000001	81	201
88	01011000	58	130	130	10000010	82	202
89	01011001	59	131	131	10000011	83	203
90	01011010	5A	132	132	10000100	84	204
91	01011011	5B	133	133	10000101	85	205
92	01011100	5C	134	134	10000110	86	206
93	01011101	5D	135	135	10000111	87	207
94	01011110	5E	136	136	10001000	88	210
95	01011111	5F	137	137	10001001	89	211
96	01100000	60	140	138	10001010	8A	212
97	01100001	61	141	139	10001011	8B	213
98	01100010	62	142	140	10001100	8C	214
99	01100011	63	143	141	10001101	8D	215
100	01100100	64	144	142	10001110	8E	216
101	01100101	65	145	143	10001111	8F	217
102	01100110	66	146	144	10010000	90	220
103	01100111	67	147	145	10010001	91	221
104	01101000	68	150	146	10010010	92	222
105	01101001	69	151	147	10010011	93	223
106	01101010	6A	152	148	10010100	94	224
107	01101011	6B	153	149	10010101	95	225
108	01101100	6C	154	150	10010110	96	226
109	01101101	6D	155	151	10010111	97	227
110	01101110	6E	156	152	10011000	98	230
111	01101111	6F	157	153	10011001	99	231
112	01110000	70	160	154	10011010	9A	232
113	01110001	71	161	155	10011011	9B	233
114	01110010	72	162	156	10011100	9C	234
115	01110011	73	163	157	10011101	9D	235
116	01110100	74	164	158	10011110	9E	236
117	01110101	75	165	159	10011111	9F	237
118	01110110	76	166	160	10100000	A0	240
119	01110111	77	167	161	10100001	A1	241
120	01111000	78	170	162	10100010	A2	242
121	01111001	79	171	163	10100011	A3	243
122	01111010	7A	172	164	10100100	A4	244
123	01111011	7B	173	165	10100110	A5	245

十进制	二进制	十六进制	八进制	十进制	二进制	十六进制	八进制
166	10100110	A6	246	208	11010000	D0	320
167	10100111	A7	247	209	11010001	D1	321
168	10101000	A8	250	210	11010010	D2	322
169	10101001	A9	251	211	11010011	D3	323
170	10101010	AA	252	212	11010100	D4	324
171	10101011	AB	253	213	11010101	D5	325
172	10101100	AC	254	214	11010110	D6	326
173	10101101	AD	255	215	11010111	D7	327
174	10101110	AE	256	216	11011000	D8	330
175	10101111	AF	257	217	11011001	D9	331
176	10110000	B0	260	218	11011010	DA	332
177	10110001	B1	261	219	11011011	DB	333
178	10110010	B2	262	220	11011100	DC	334
179	10110011	B3	263	221	11011101	DD	335
180	10110100	B4	264	222	11011110	DE	336
181	10110101	B5	265	223	11011111	DF	337
182	10110110	B6	266	224	11100000	E0	340
183	10110111	B7	267	225	11100001	E1	341
184	10111000	B8	270	226	11100010	E2	342
185	10111001	B9	271	227	11100011	E3	343
186	10111010	BA	272	228	11100100	E4	344
187	10111011	BB	273	229	11100101	E5	345
188	10111100	BC	274	230	11100110	E6	346
189	10111101	BD	275	231	11100111	E7	347
190	10111110	BE	276	232	11101000	E8	350
191	10111111	BF	277	233	11101001	E9	351
192	11000000	C0	300	234	11101010	EA	352
193	11000001	C1	301	235	11101011	EB	353
194	11000010	C2	302	236	11101100	EC	354
195	11000011	C3	303	237	11101101	ED	355
196	11000100	C4	304	238	11101110	EE	356
197	11000101	C5	305	239	11101111	EF	357
198	11000110	C6	306	240	11110000	F0	360
199	11000111	C7	307	241	11110001	F1	361
200	11001000	C8	310	242	11110010	F2	362
201	11001001	C9	311	243	11110011	F3	363
202	11001010	CA	312	244	11110100	F4	364
203	11001011	CB	313	245	11110101	F5	365
204	11001100	CC	314	246	11110110	F6	366
205	11001101	CD	315	247	11110111	F7	367
206	11001110	CE	316	248	11111000	F8	370
207	11001111	CF	317	249	11111001	F9	371

十进制	二进制	十六进制	八进制	十进制	二进制	十六进制	八进制
250	11111010	FA	372	253	11111101	FD	375
251	11111011	FB	373	254	11111110	FE	376
252	11111100	FC	374	255	11111111	FF	377

（卢传、宋知用、李士才、周宝兴译校）

第七篇 TRS-80编辑/汇编程序使用手册

一、前　　言

TRS-80 编辑/汇编程序是一个放置在 RAM（随机存贮器）内的文本编辑和汇编程序，供 16K RAM 的 TRS-80 微型计算机系统使用。这个编辑/汇编程序设计得使初学者很容易使用，而且有很强的功能，足以满足使用要求。LEVEL Ⅱ BASIC 能够直接把编辑/汇编程序（它是记录在盒式磁带上的）送入微型机。而 LEVEL Ⅰ BASIC 必须利用 SYSTEM 磁带（每台微型机都附有此带）作引导，才能把编辑/汇编程序送入微型机。

编辑/汇编程序的文本编辑功能，使得字母数字文本文件的处理得以简化。常把这种编辑功能用于编写和修改汇编语言的源程序。

编辑/汇编程序的汇编部分的功能是：把用符号语言写的源程序很容易地变换成微型机能执行的目的码。然而，对于 LEVEL Ⅰ BASIC 来说，这种目的码是用 SYSTEM 磁带来执行的，而 LEVEL Ⅱ BASIC 则直接用 SYSTEM 命令来执行。

汇编命令除了以下几点以外，是根据 Zilog 公司的 Z80 汇编语言程序手册来执行的。

1. 不能使用宏汇编命令。

2. 操作数表达式只能包括 + 和 −，&（逻辑与）和 <（移位）的操作，而且严格的按由左向右的顺序计算。不许使用括号！

3. 程序员想对源码文件中的某一部分进行汇编，这样的条件汇编命令不能使用。

4. 常数只能用十进制，十六进制和八进制来表示。不加标记者一律看作是十进制。

5. 唯一可用的汇编程序命令是 *LIST OFF 和 *LIST ON。

6. 标记区（LABEL）只能用字母数字符号（不能使用 − 和 ?）。标记区字符长度不得超过六个。第一个字符必须是英文字母，而其它的字符必须是字母数字。

二、本手册中所用符号的说明

[ ]　　方括号内是使用者选用的数据：

P[Line 1[:Line 2]]

其中 Line 2 是使用者选用的，P（在监视荧光屏上打出信息的命令）命令后面并不一定要跟随任何东西也能使用，在实际使用中方括号通常是不必打出来的。

……　　省略记号是表示前面的项要重复。如 A[[b 文件名称] [/开关说明符[/开关说明符]…]]

	/开关说明符被重复数次。
大写字母	是在输入输出的时候用的。
小写字母	表示是要用户自己决定的插入字符。如（inc．—增量，filename —文件名称，line—行编号）
下划线	在没有专门说明时，本说明书中标有下划线的信息是表示由编辑/汇编程序打出的信息。以便把用户的输入和计算机的输出区别开来，因而它不由用户来打印。
b	B的小写字母，表示用户必须在此留出空位（Space）。
Line	这是行编号，是0至65529的任意十进制数。
Line 1:Line 2	它规定了二个不同行编号之间的数量(Line 1通常是小于Line 2的)。
.	任何行编号处都能够用（.），它是表示正在汇编、打印和编辑的源码现行的行的一种标记。就是在监视荧光屏上看到的最后一行。
##	这个符号能够用在任一行编号处。它表示在本文缓冲器内第一个源码行（最低的行号）。
*	星号能够用在任一行编号处。它表示在文本缓冲器内最后的源码行（最高的行号）。
inc	这个数是表示连续的二行间的行增量。
filename	规定磁带文件的名称的字符串。详细情况参阅“盒式磁带”章节。

三、编辑/汇编程序

简单说，编辑/汇编程序是设计成供用户用键打入源汇编码的。对源码进行汇编，而得到的目的码可以记录在磁带上。编辑/汇编程序也可以读入、记录和编辑存贮在磁带上的其它的源码文件。当然，用编辑/汇编程序管理的源文件不必一定是汇编程序。文件可以是由编辑/汇编产生的任意的文本信息。BASIC程序带可以不用编辑/汇编程序来编辑，因为BASIC解释程序本身有编辑功能。

限制用户编写的汇编语言程序大小的是用户计算机系统的RAM存储器的总数。这个编辑/汇编程序保留一个“文本缓冲器”这个缓冲器起始于编辑/汇编程序的终点，且一直到RAM存储器结束为止。对于16KRAM的系统来说，通常留有8.5K字节的存储器，作为将来放置源文件的文本缓冲器。

1. 编辑/汇编程序送入微型机

1) LEVEL Ⅱ BASIC

因为编辑/汇编程序是一种用机器语言写的程序，所以它只能用SYSTEM命令送入机器。具体操作方法是：先把TRS-80与盒式录音机之间的联接电缆接好，黑色插头插入盒式机EAR插孔，灰色较粗的插头插入AUX插孔，而灰色较细的插头插入MIC插孔。把编辑/汇编磁带装入盒式录音机，然后按下录音机的PLAY开关。录音机音量调节旋钮放在5—6的位置（这是一种速率为500波特的磁带），打开TRS-80的电源和显示器电源。这

时监视荧光屏上将显示

MEMORY SIZE?

当按下白色的 ENTER 键时，将进入 LEVEL Ⅱ BASIC 的管理下，监视荧光屏上将显示

RADIO SHACK LEVEL Ⅱ BASIC

READY

>

其中>符号表示机器目前正处于 LEVEL Ⅱ BASIC 的管理之下。

用键打入 S Y S T E M，再按下 ENTER 键。微型机的监视屏上将打出 *? 作为响应。现在用键打入 E D T A S M（这是编辑/汇编程序的文件名称），按下 ENTER 键这时录音机开始转动，磁带上的信息将被读入微型机内的存储器。同时荧光屏右上角显出 * * 符号，其中第二个 * 的闪亮表示记录在磁带上的信息正在读入 TRS-80 的存储器。一旦输送完毕，荧光屏上右上角的闪动 * 号将停止不动，荧光屏上显示

*?

作为磁带信息输入完成的回答。

用键打入 / 记号，再按下 ENTER 键，计算机监视荧光屏上的字符将被清除，并显示出信息：

TRS-80 EDITOR/ASSEMBLER 1.2

*

这表示目前 TRS-80 是处于编辑/汇编程序的理管之下，星号 * 是编辑/汇编程序的起动符号。这是表示它正要求你给命令。除了正在读磁带、写磁带或编辑一行以外，按 BREAK 键以后，机器总是回答你一个星号 *。 BREAK 键是在汇编或打印输出进行中要中断时使用。

2). LEVEL Ⅰ BASIC

因为编辑/汇编程序是以 500 波特的速率记录在磁带上的，所以 LEVEL Ⅰ BASIC 不能直接把此磁带读入机器。你必须首先把 SYSTEM 磁带送入机器（SYSTEM 磁带随 TRS-80 微型机附带）。然后这个程序（指 SYSTEM 带上的）可以把 500 波特速率的编辑/汇编磁带读入机器。

操作方法：把 SYSTEM 磁带装入盒式录音机。把音量旋钮放在 8 或 9 处（这是一种 250 波特的带）。打入键 C L O A D，于是 BASIC Ⅰ 将把 SYSTEM 带读入机器。当输入一结束，程序即可开始。

计算机将打出：

*

现在把编辑/汇编带装入你的盒式录音机。把音量旋钮放于 5 — 6 的位置（这是 500 波特的磁带）。用键打入 E D T A S M，然后按下 ENTER 键。于是编辑/汇编程序带将被读入。当读完时，另外的 * 将被显示出来。现在用键打入斜道 / 然后再打入数字

18058。按下 ENTER 键，以执行编辑/汇编程序。数字 18058 是编辑/汇编程序的进入地址，进入后在荧光屏上将显示出

TRS-80 EDITOR/ASSEMBLER 1.2

*

现在你就可以使用编辑/汇编程序了，使用细节请参阅“汇编语言”章节。

BREAK 键的使用，同前面介绍是一样的。

2. 命　　令

TRS-80 编辑/汇编程序能够实现下述那些命令。在表示可以应用的符号 * 出现以后，这些命令就可以打入。这个星号表示是编辑/汇编“命令级”。下面的列表是带有简短说明的全部编辑/汇编命令级的指令。

A	汇编现在在文本缓冲器里的源码。
B	返回到放在 ROM （只读存储器）内的 BASIC。
D	删去指定的一行或数行。
E	这是一个编辑命令；几乎同 LEVEL Ⅱ BASIC 的 EDIT 命令一样。
F	在本文缓冲器内寻找指定的字符串。
H	除了要输出到行式打印机这点以外，同 P 命令相同。
I	把源码行插入具有确定增量的指定行里。
L	从盒式磁带上把源文件送入文本缓冲器。
N	对放在文本缓冲器内的源码行进行重新编号。
P	在监视荧光屏上打出现在放在文本缓冲器里指定区内的源码。
R	替换现在在文本缓冲器里的若干行。很像插入命令 I，但是不同的是只对行进行改写。
T	同 H 命令一样，所不同的是不打印出行编号——只是打印文本。
↑或↓	将在荧光屏上打出文本现行的行的后面的一行源码或者是前面的一行源码。
→	水平移动标记。
W	写现在放在文本缓冲器里的内容到磁带上。

1) 汇编命令(A)

格式：*A[[b 文件名称][/开关说明符[/开关说明符]……]]

这个命令是把文本缓冲区的源文件变换成机器目的文件，并存入盒式磁带上。

开关说明符可以是下面四种可供选用的内容中的任一个。

NL	不把列表打到荧光屏上。错误和不对的源码行仍旧被打出来。
NO	不要目的码。禁止把目的码录到磁带上。
NS	不要打印符号表。
LP	送列表、错误和符号表到 TRS-80 的行式打印机。

WE　　当发生错误时，使汇编处于等待状态。按下任何一个键，将继续进行汇编直到另一个错误被发现为止。而按 C 键将使汇编继续进行，就是有错误也不停止。在任何时候，只要按下 ENTER 键，就返回到命令级。

对编辑缓冲器的内容汇编。得到的目的码在确定的文件名称下写入盒式磁带（如果不指定文件名称，则文件名称就自动被置于 NONAME）。汇编错误通常立即在监视荧光屏上显示于发生错误的行的前面。

汇编完成以后，全部错误被打印出来。最后打出符号表，且计算机荧光屏显示出：

READY CASSETTE

为了记录目的码，准备好磁带和录音机，然后按下 ENTER 键。如果你不需要目的码，则只要按下 BREAK 键，这时星号(命令级)将显示出来。这些不是必需要执行的过程，它能由相应的开关说明符来完成。

举例：

*A	对文件名称是 NONAME 的文件汇编；在荧光屏上列表。
*AbIKKY	同上；目的文件的名称是 IKKY。
*A/NS	具有文件名称为 NONAME 的文件进行汇编，不要符号表。
*A/NS/LP	同上；然而全部输出到行式打印机。
*AbQ/NL	对文件名称为 Q 的文件汇编；不要列表，b 是一个必须留出的空位。

2) 返回 BASIC 命令(B)

格式： *B

打入 B 键，然后按下 ENTER 键，如果现在是处于 LEVEL Ⅱ BASIC 时，将回答你 MEMORY SIZE？（如同刚打开电源时的情况），而在 LEVEL Ⅰ BASIC 时，将回答你 READY。

举例：

*B

MEMORY SIZE?

3) 删除命令(D)

格式： *D [line 1 [:line 2]]

删去文本缓冲器内指定的某一行或数行。

举例：

*D100:500	从文本缓冲器内删去从 100 行到 500行（包括 500）。
D#:	删去写入文本缓冲器内的所有的行。即清除文本缓冲器。
*D.	删去现行的这一行。
*D105	单删去 105 行。

每打入上面的删除命令后，均要按 ENTER 键，以执行删除命令。

4) 编辑命令(E)

格式： * E[line]

允许用户编辑/修改源码行，和 LEVEL Ⅱ BASIC 中的 EDIT 命令一样。唯一不同之处是删除命令不包括删除惊叹号的信息。

举例：

* E.　　编辑现行的这一行。

* E211　　编辑 211 行。

进入编辑命令 (E) 以后，用于编辑一行的子命令有 A，C，D，E，H，I，K，L，Q，S，X 等。下面分别说明编辑子命令。

A　　重新开始编辑。在对某一行编辑的过程中，发生错误，要把刚才的编辑作废，则要先退出刚才的编辑子命令（同时按下 SHIFT ↑ 键），然后打入 A 键，于是荧光屏上重现刚才被编辑的那一行的行号，不承认刚才的编辑，可以对该行重新编辑。

nC　　更换 n 个字符。先用移动光标（键盘上最下面一行的长键）找出需要更换的字符，然后打入 n（你需要更换的字符数目），按下 C 键，打入更换的新字符（荧光屏上有显示）再按下 ENTER 键，于是就完成了更换工作（n 是表示更换连续的字符数目）。

nD　　删去 n 个字符。操作方法同上，只是打入 n 后，再打入 D 键，按下 ENTER 即可完成 nD 命令。

E　　结束编辑和进入更换，按 E 即结束现行的行的编辑工作，进入更换，荧光屏上出现命令级符号 *。

H　　删去光标后的字符，用键打入的字符保留。H 命令将不用于删除文本的整行。在一行上必须留有一个字符，否则将来用到这一行将出问题，这是因为这个子命令并不消去行号，此行整行内容虽已删去，但行号仍被占用。

I　　插入字符串。打入 I 键，用移位光标（长键）找到插入位置，再打入插入的字符，按下 ENTER 键，于是荧光屏显示的就是插入后的情况。

nKx　　消去第 n 个 x 以前的所有字符。

L　　打出现行的行，光标返回到起始位置。

Q　　在对某一行的编辑中，需要退出，且不要刚才执行的全部编辑。则先同时按 SHIFT ↑ 键，然后按下 Q，于是刚才对此行的全部编辑作废，荧光屏上出现命令级符号 *。

nSx　　寻找第 n 个 x 的所在地，以便编辑。

X　　移动到行的结束处，在行末插入你要的字符。

←　　移动编辑标返回一个空位。

(SHIFT)(↑)　　从任一编辑子命令中退出。

ENTER　　使这一行实现目前的编辑命令。

5) 寻找字符串(F)

格式： *F[字符串]

其中字符串是十六个字符的序列或者是少于十六个字符的序列。

当指定一个需寻找的字符串后，它是由现行的行编号的后一行开始对编辑缓冲器进行寻找。如果没有指定字符串，就寻找相同于最后一个F命令所找的那个字符串。如果寻找的字符串被找到了，那末包含有它的那一行要在荧光屏上显示出来。这时就把这一行看作为现行的行。如果字符串没有找到，于是荧光屏上将显示出：

STRING NOT FOUND

而现行行没有移动，还是开始寻找时的那一行看作为现行行。

在发出寻找命令(F)以前，通常要用P#移动现行行到缓冲器的起始处。所以当你要想由缓冲器的起始处来寻找某一字符串，均要使用P#命令。

举例：

*P#　　(用键打入P，#，按 ENTER 键)

显示 00100　ORG　7000H

*F3C00　　(用键打入 F 3 C 0 0，按 ENTER 键) 找字符串 3C00。

显示 00110　VIDEO　ORG　3C00H　　找到字符串 3C00 在00110。

*F　(用键打入F，按 ENTER 键) 再一次找 3C00 字符串。

显示 00211　LD　HL，3C00H　　找到在 00211 行还有 3C00 字符串。

*

6). 硬拷贝(H)

格式： *H[line 1 [:line 2]]

把一行或一组行在TRS-80的行式打印机上打印出来。现行行的标记。被修正到打印的最后一行处。这个命令同P命令完全一样。

举例：

H#：　　送文本缓冲器内所有的行到打印机。

*H100:500　　送100到500的行人打印机。

*H.　　送现行的一行到行式打印机。

*H　　从现行行开始的15行送人打印机。对于行式打印机并不常用。

7) 插入命令(I)

格式：　　*I line[, inc]

I命令是用于把用户编写的文本的一些行插人到编辑缓冲器内。所有源码行通常就是用I命令送进去的。发出I命令以后，打人起始行号和增量如 * I 100，10　　然后按下 ENTER 键，于是荧光屏上将再次显示出行编号

00100

这时就可以用键插入你需要的字符，当这一行插入完成后，再按下 ENTER 键，于是荧光屏将按给定之增量 10 显示出：

00110

供你插入。如此继续下去，即可把你的文本插入编辑缓冲器，直到下述条件之一发生为止：

(1) 按下 BREAR 键，命令级符号 * 被打出来（通常用于退出插入命令）。

(2) 编辑缓冲器已经充满。

注意：

(1) 如要在行间插入新的行时，当插入行的行编号大于或等于缓冲器内的下一个行的编号时，则荧光屏上将打出：

NO ROOM BETWEEN LINES

如：00100 ORG 5000H

00110 VIDEO EQU 3C00H

当在此二行间要插入的行为 00120 LD HL, VIDEO

或 00110 LD BC, 3FFH

在屏上就要显示 NO ROOM BETWEEN LINES

(2) 插入的下一行的行编号大于 65529 也是不允许的。

(3) 如果没有指定增量，则假定就是最后指定的那个增量值。现行行的标记．修正到插入的最后一行。

另外：

源码行可以多到 128 个字符长。但是通常并不需要这样长的行。推荐使用的每行字符数约为 60 个（这样打印输出和列出的表将是比较整洁的。）

8) 把磁带上的文件送入存贮器(L)

格式： *L[b 文件名称]

对于文件名称确定的文件，要搜索磁带。如果指定的文件被找到，则其内容就排列在编辑缓冲器已有内容的后面。注意，此时可能引起行编号排列上的错误，为了纠正，必须用N命令给予校正。如果用户不希望把带上的文件加到现行的文本缓冲器上去，则必须用 D并：* 命令来清除缓冲器。以便在完全空白的缓冲器内放入磁带上的文本。

如果不指定文件名称，则依次的将磁带上的下一个文件送入缓冲器。

当读入存储器时，荧光屏右上角将出现二个 * 号，其中第二个 * 在读入信息时要闪亮。L命令只能读入由编辑/汇编程序建立的源码文件。

举例：

*L 送下一个源码文件

*LbMYPROG 搜索和送入名称为 MYPROG 的源文件。b 是必须留出的空位。

9) 重编行号的命令(N)

格式： *N[line[, inc]]

N命令是用于重编编辑缓冲器内的行号。在重新编行号时，要给缓冲器内的首行一个确

定的编号，如果不给一个确定的行号，则由 00100 算起。在缓冲器内保留的行按给定的增量重新编号，如增量不给定，就按前面 N，R，或 I 命令中的增量为准。在 N 命令使用前的现行行仍旧看作是现行行，但行号能重新编号。

举例：

*N	根据前面定的增量，重新编号从 100 算起。
*N 5	根据前面定的增量，重新编号从第 5 行算起。
*N10,5	以 5 为增量，重新编的号由第 10 行算起。

当打入上述命令和按下 ENTER 键后，荧光屏显示的是命令级符号 * 。并不显示重新编号后内容。

10） 在荧光屏上打出文本缓冲器的内容(P)

格式： *P[line 1 [:line 2]]

在监视荧光屏上打出一行或一组行。表示现行行的标记(·)被修正到打出的最后一行。

举例：

P井：	打出文本缓冲器内的全部内容。
*P100:500	打出 100 行到 500 行（包括 500 行）。
*P.	打出现行的这一行。
*P	打出从现行的行起的 15 行。重复使用 P 命令，在屏全部打满没有空行时，可以把最早打出的源码行从荧光屏上方移出去。

11） 替换命令(R)

格式： *R[line[,inc]]

R 命令只替换一行，并进入插入工作方式。如果替换处有行，则把它删去，然后再插入。如果那儿没有行，就如同用 I 命令插入一样。如果增量不确定，那末就用最后一个 I，R 或 N 命令所确定的增量为准。替换的那一行作为现行行。

举例：

*R.	替换现行的那一行。
*R 100,10	从 100 行起开始替换，增量为 10。
*R 100	从 100 行起开始替换。用最后确定的增量为准。

替换命令 (R)完成以后，按 BREAK 键，于是退出 R 命令，恢复到命令级 * 。

12） 在行式打印机上打印(T)

格式： *T[line1[:line2]]

把一行或一组行在 TRS-80 行式打印机上打印出来。打印的最后一行作为现行行。这个命令很像 H 命令，所不同的仅是不打出行的编号来。

举例：

T井：	把文本缓冲器内的全部内容送入打印机。

*T100:500　　从文本的 100 行到 500 行送入打印机。

*T.　　把现行的这一行送入行式打印机。

13) 箭头和标记

编辑/汇编程序包括下面的一些特殊字符：

箭头向上(↑)

↑命令是用于打出现行行的前一行，并把打出的那一行作为现行的行。（如果现行的行是编辑缓冲器的第一行，则↑命令就把这一行打出来，并把这一行当作现行行）

箭头向下(↓)

↓命令是用于打出现行行的下一行，并把打出的那一行作为现行的行。（如果现行的行是编辑缓冲器的最后一行，则↓命令就把它打出来，并把这一行当作现行行）

注意：↑和↓命令必须是命令行的第一个字符。

→标记是表示光标右移到下面八个字符区。

称源码行的第一个字符为首位置（表示行编号的数字不计），标记是在 9, 17,25,33,41,49, 57 且按增量是 8 一直到 255 。标记(→)应用于插入空位(保存文本缓冲器的空位)，标记本身只占去一个字节。

删除字符(←)

(←)用于删去打入的最后一个字符。如果最后一个字符是空位标记，则活动删除标记就跳到更前一个非空位字符。

(SHIFT) (←)删除整行

(SHIFT) (←)将删去当前开始进入的整行。它对于源码行和命令二者均是适用的。

(SHIFT) @中止

在汇编或打印输出期间的任何时间，为了中止计算机的工作，可以打入 SHIFT ,@ 键。按下 ENTER 键将继续进行处理。当某行正在打印或汇编时，(SHIFT@)将不被接收；只有在二行之间才能接受。正在进行汇编期间接收的暂停将不被承认。

14) 写入磁带（W）

格式：*W[b 文件名称]

这个命令是把编辑缓冲器的内容写到盒式磁带上，记上文件名称。如果文件名称没有确定，也可以不用文件名称。现行行位置不改变。

举例：

*W　　不带文件名称，把文本缓冲器的内容记录到磁带上。

*WbDEMO　　文件名称定为 DEMO，把文本缓冲器的内容记录到磁带上。b 是必须留出的空位。

注意：文件名称必须是英文字母开始，字符最多可达 6 个。

在打入上述命令后，按下 ENTER 键，于是荧光屏上显出 READY CASSETTE 如果要录到磁带上，则再按下 ENTER 键，于是开始录入。

3. 盒式磁带的使用

所有由编辑/汇编程序建立的盒式磁带是按500波特速率写入的。录有编辑/汇编程序的盒式磁带也是500波特的。凡是读500波特磁带时，盒式录音机的音量电平旋钮必须放在5—6之间。

LEVEL Ⅱ BASIC能够直接读入500波特的编辑/汇编程序的磁带。当要执行存贮在磁带上的目的码程序，则要用键打入S Y S T E M命令后才能实现。

文件名称

录制在盒式磁带上的文件名称必须是由英文字母开始。而以后的字符可以是字母数字。文件名称的长度不得超过6个字符。对于A命令和W命令，不一定要确定文件名称，若文件名称不确定，则文件就赋于NONAME的文件名称。如果文件名称不确定，当使用L命令时，就把录制在磁带上的第一个文件送入机器的存储器内。

4. 应用举例

下面的例子是用编辑/汇编程序写的一个程序。为用户写的解释部分放在[ ]内，它并非是程序的一部分。

注意： 标记区（LABEL）不需留出空位。

```
TRS-80      EDITOR/ASSEMBLER1.2
*I100,10

00100(→)          ORG     5000H          [→是移动标记]
00110 VIDEO       EQU     3C00H
00120             LD      HL, VIDEO      ; SOURCE ADDRESS
00130             LD      DE, VIDEO+1    ; DEST.ADDRESS
00140             LD      BC,3FFH        ; BYTE COUNT
00150             LD      (HL), 0BFH     ; GRAPHICS BYTE
00160             LDIR                   ; WHITE OUT SCREEN
00170; DELAY LOOP TO KEEP WHITE OUT SCREEN ON
00180             LD      B,5
00190 LP1         LD      HL,0FFFFH      ; VALUE TO DECREMENT
00200 LP2         DEC     HL
00210             LD      A,H
00220             OR      L              ; HL=0?
00230             JP      NZ,LP2         ; IF NO DEC AGAIN
00240             DJNZ    LP1            ; DEC.B- -B
00250             JP      0H             ; RETURN TO BASIC
00260             END
00270 [BREAK]
*
```

对源程序进行汇编的一个例子

*AbXXX [汇编] [下面全部是计算机的输出]

```
5000              00100          ORG       5000H
3C00              00110 VIDEO    EQU       3C00H
5000 21003C       00120          LD        HL,VIDEO
5003 11013C       00130          LD        DE,VIDEO+1
5006 01FF03       00140          LD        BC,3FFH
5009 36BF         00150          LD        (HL),0BFH
500B EDB0         00160          LDIR
                  00170
500D 0605         00180          LD        B,5
500F 21FFFF       00190 LP1      LD        HL,0FFFFH
5012 2B           00200 LP2      DEC       HL
5013 7C           00210          LD        A,H
5014 B5           00220          OR        L
5015 C21250       00230          JP        NZ,LP2
5018 10F5         00240          DJNZ      LP1
501A C30000       00250          JP        0H
0000              00260          END
0000 TOTAL ERRORS
LP2 5012
LP1 500F
VIDEO 3C00
READY CASSETTE [存入磁带；录音机放于录音状态]
[ENTER] [按下 ENTER 键，即可录下目的码。]
```

*

当你把编好的源程序用键打入文本缓冲器后，就能把此源程序保存在磁带上。

*W MYPROG

执行上述命令，信号就被送入磁带。

用编辑/汇编程序的 L 命令，就能把磁带上名称为 MYPROG 的文件读入机器的存贮器内。

对目的码的执行如下：

1) LEVEL I BASIC 系统

首先送入 SYSTEM 磁带。操作方法：把 SYSTEM 磁带装入盒式录音机。音量旋钮放在 8 — 9 之间。用键打入 C L O A D，于是读入 SYSTEM 磁带。送入完毕即显示出

*

现在可以送入你的目的带的文件名称，此目的带是由编辑/汇编程序产生的。如果文件名称不确定，则就打入 NONAME。对于所举的例子来说，打入文件名称 XXX。

*XXX

于是可以把目的带 XXX 装入盒式录音机内，按下盒式录音机的 PLAY 开关。音量旋钮必须放在 5 — 6 之间的位置（这是 500 波特的带）。在荧光屏的右上角将有星号闪烁。一旦送入完毕，计算机荧光屏上将再次打出 * 。现在你必须送入目的程序的起动地址。起动地址（ORG）是 5000H，相应于十进制的 20480。现在先打入 [/] 键，然后打入 20480。此命令形式如下

*/20480

按下 [ENTER] 键，于是就执行用机器码表示的程序。所举例子的程序要完成的工作是将由一些实图字符■拼起来的图形显示出来，使得显示屏全部变白，并停留约 5 秒钟。然后程序将返回到 BASIC 的 READY 状态。

2) LEVEL Ⅰ BASIC 系统

LEVEL Ⅰ BASIC 执行就比较简单。目的带重新再送入计算机，音量旋钮放在 5 — 6 的位置。按照下面方法按键：

READY

>SYSTEM

*? XXX　[读入目的带]（无文件名称时打入 NONAME）

*? /20480

现在将执行程序，然后返回到如电源刚打开时的情况一样。

MEMORY SIZE?

多个程序的工作方式：

你能够相继把几个机器语言程序送入计算机的存储器。这些指令的 ORG 地址必须是使每个目的程序不与另一个相冲突。如果你有下面的一些文件：

XXX	7000 到 70FF
YYY	7100 到 71FF
ZZZ	7200 到 72FF

然后，你就能用下面的方法进入这三个程序：

*? XXX

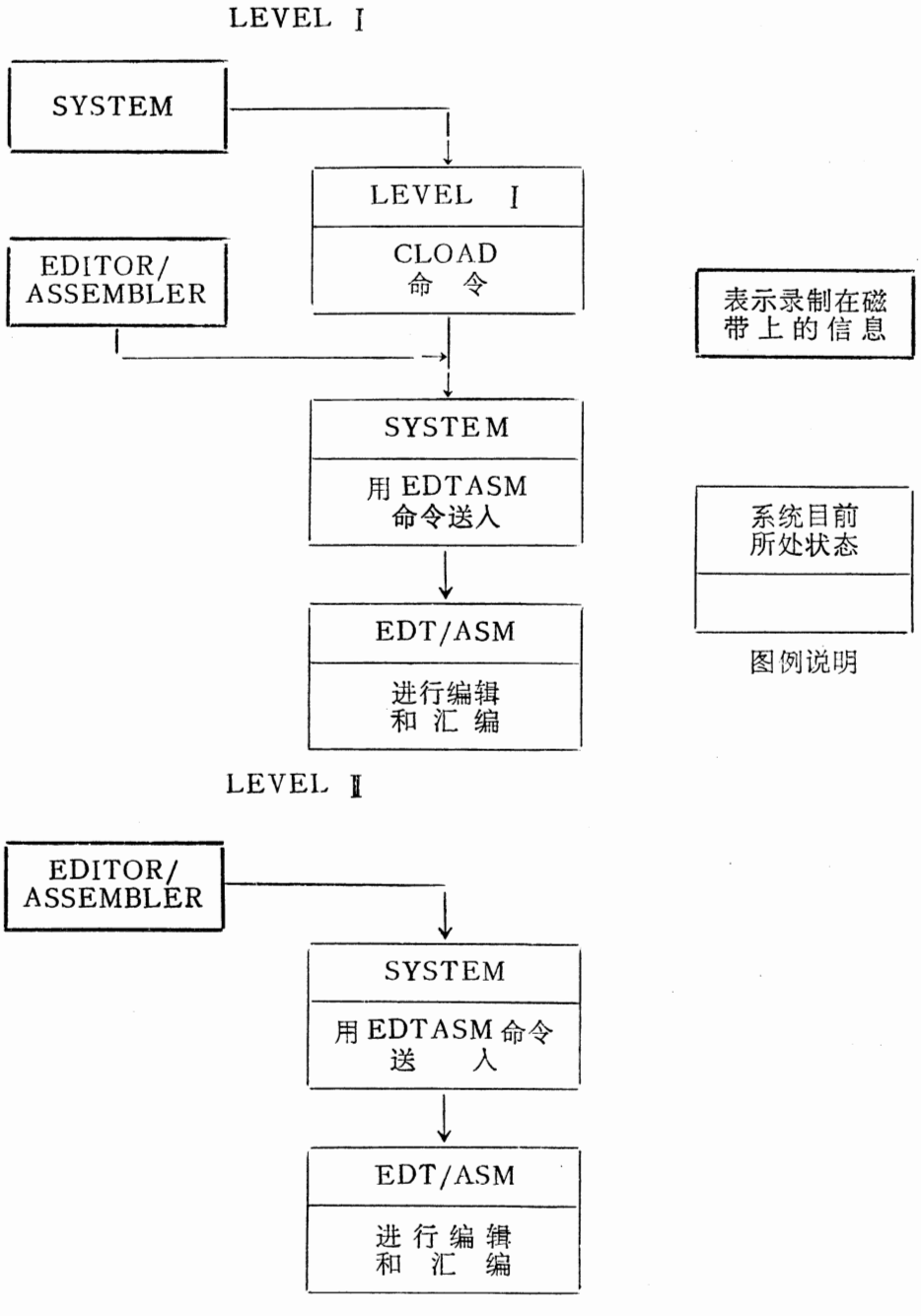
*? YYY

*? ZZZ

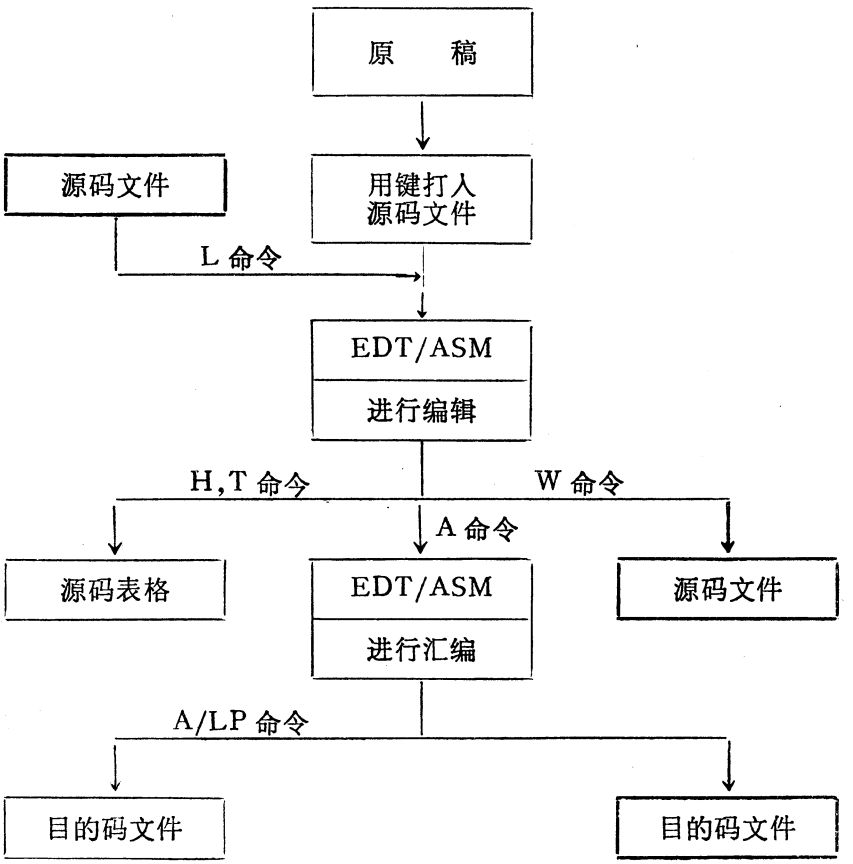
*? /28672 [跳到 XXX 程序]

5. 操作流程图

送入编辑/汇编程序的流程图

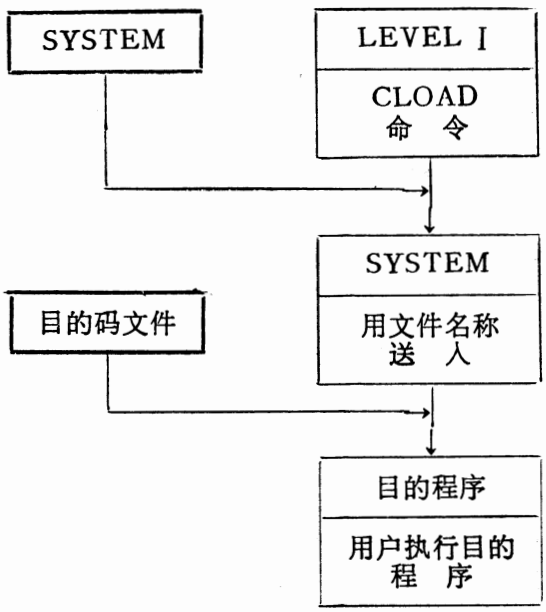


使用编辑/汇编程序的流程图

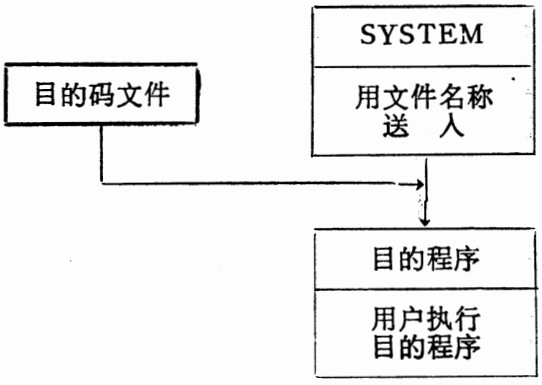


目的码执行流程图

LEVEL I



LEVEL II



四、汇 编 语 言

1. 语 法 (Syntax)

汇编命令的基本格式如下：

[LABEL]	[OPCODE]	[OPERAND(s)]	[COMMENT]
	ORG	7000H	
VIDEO	EQU	3C00H	
	LD	DE,VIDEO+1;	终点

LABEL（标记区）

LABEL 是行编号的符号名称。LABEL 是可以任选的。它是一个不大于 6 个字符的字符串。第一个字符必须是英文字母。LABEL 不能包括 $ 字符。$ 字符是留作为现行指令的参考记数器的值。所有的 LABEL 必须是由源码行的第一个位置开始。下面这些标记是留给寄存器用，不能作其它的用途：A,B,C,D,E,H,L,R,IX,IY,SP, PC, AF, BC, DE 和 HL 。

下面 8 个标记是留作分枝条件用，不能作其它的用途（这些条件加到状态标）：

状　态　标	条 件 置 位	条 件 不 置 位
Carry(进位)	C	NC
Zero (零)	Z	NZ
Sign (符号)	M	P
Parity(奇偶性)	PE	PO

举例：

```
JP  NZ, LOOP
```

解释：如果 Zero 标被清除（即不置位），则上述指令就跳到用 LOOP 指定的那条指令。

源码行的 LABEL 是供其它源码行作参考用，如果其它源码行中完全不需参考，则这个 LABEL 就无意义。

OPCODE（操作码）

TRS-80 编辑/汇编程序的操作码是严格符合 Z-80 编汇语言编程手册的。

OPERANDS（操作数）

操作数是用逗号分隔开的一个或者二个值。同样的指令可以要求全然不同的操作数。

操作数一般是供指令执行时必须参照的数据值或以符号表示的地址值。

举例：

```
LD    HL,  3C00H
XOR   A
LD    (HL), 20H
```

在括号中之值表示是间接寻址。

常数可由下列任一文字来结束：

H—十六进制　　D—十进制　　O—八进制

假定是十进制，常数可以不以这些字母来表示。常数必须以数字开始。表示成 FFH 是错误的，而应该写成 0FFH。

+，－ 和 & 符号的应用，请参看 Expression 章节。

COMMETS（注解）

所有的注解必须由分号（；）开始。如果源码行开始的第一列就是分号（；），则此行就是注解。

2. 表达式（Expression）

操作数的值可以是包含有 +，-，& 或 < 符号的表达式。这些操作数是从左向右执行。不允许用括号。所有这四种标符是二元的，而 + 和 - 这二种算符也可以一元使用。（即 +、-、& 和 < 是可以联系二个操作数如 op1 + op2，op1 & op2……，而 +，- 这二个算符也可以只与一个操作数相联系如 +op1，-op2……）

加（+）

加号表示二个常数相加，二个符号值相加或者是常数和符号值相加。将加号当作一元算符用时，它就仅表示这个值的符号。

LABEL	OPCODE	OPERAND	COMMENT
CON30	EQU	30	；=001EH
CON16	EQU	10H	；=0010H
CON3	EQU	3	；=0003H
VIDEO	EQU	3C00H	；=3C00H
A1	EQU	VIDEO + CON3	；=3C03H
A2	EQU	CON30 + CON16	；=002EH
A3	EQU	+ VIDEO	；=3C00H

减（-）

减号算符是表示二个常数相减，二个符号值相减，或者是常数和符号值相减。一元算符负号是表示 2 的补数。举例：

举例：

```
A1      EQU     VIDEO - CON3        ；=3BFDH
A2      EQU     CON30 - CON16       ；=000EH
A3      EQU     - VIDEO             ；=C400H
```

逻辑与（&）

逻辑与算符是表示二个常数的布尔代数乘法，二个符号赋值的布尔代数乘，或者是常数和符号赋值的布尔代数乘。

举例：

```
A1      EQU     3C00H&FFH           ；=3C00H
A2      EQU     0&15                ；=0000H
A3      EQU     0AAAH&555H          ；=0000H.
```

移位（<）

移位算符能用于把值左移或右移。

格式是：VALUE<AMOUNT

如果 AMOUNT 是正的，则 VALUE 被左移。如 AMOUNT 是负的，则 VALUE 被右移。

举例：

```
A1    EQU    3C00H<4              ；=C000H
```

```
A2      EQU     3C00H<-4          ;=03C0H
A3      EQU     3CBBH<8+255       ;=BBFFH
A4      EQU     15+3C00H<-4       ;=03C0H
```

3. 伪指令（Pseudo-OPS）

汇编程序能识别的伪指令有九种，这些汇编的控制译码指令(写得很像微处理器的指令)是作为汇编用的命令，并非是处理器的指令。在汇编进行期间，它们直接汇编以完成指定的工作，并不是对 Z-80 处理器而言的。这些伪指令如下：

ORG nn	置地址参考计数器为 nn 值。
EQU nn	在程序内置标记区值为 nn，在一个程序内任何一个标记只能置一个值。
DEFL nn	给标记区置 nn 值。在某一程序中可以重复的对同一标记给不同的值。
END	它表示源程序结束，这样下面的语句就可以忽略不计。如果没有找到 END 语句,则产生一个警告信号。END 语句能够确定一个起始地址，如 END LABEL，END 6000H。如果随着 [/] 记号没有给起始地址，则起始地址就自动转到系统程序。
DEFB n	能够为现行参考计数器定义一字节的内容为 n。
DEFB'S'	定义存储器的一字节内容为字符 'S' 所表示的 ASCII 码。
DEFW nn	能定义二字节的字的内容。把低阶的字节放在现行参考计数器内，而把高阶的字节放在现行参考计数器加 1 的地方。
DEFS nn	保留下存储器的 nn 字节，它开始于参考计数器的现行值。
DEFM'S'	定义存储器的 n 字节的内容,使其成为字符串 'S' 表示的 ASCII码。其中 n 是 S 的长度，它必须是处于 0≤n≤63 范围之内。

4. 汇编命令

TRS-80 编辑/汇编程序只能使用二个汇编命令。每个汇编命令必须起始于源码行的纵列第一个位置，且一定要由星号（*）开头。汇编命令是：

*LISTbOFF	由下一行开始使汇编列表中止。错误和不合格的源码行将仍被打出来。
*LISTbON	从这一行开始，使得汇编列表继续下去。

五、 错 误 信 息

TRS-80 编辑/汇编程序能够识别二种型式的错误：

1）命令错误——错误被打出来，控制转向命令级。

2）汇编错误——在执行汇编命令时能产生三种型式的错误。

(1) 终止：汇编被终止；控制转向命令级。

(2) 卡死：包含有错误的行不再进行汇编，对于这一行不产生目的码。汇编由下个源码行着手进行。

(3) 报警：错误信息被打出来，包含有报警的行的汇编继续进行。这样得到的目的码可能不是编程者所需要的。

下面列出全部的错误信息，每个都有解释。

1. 命令错误

1) 坏的参数（BAD PARAMETER (S)）

原因——

- 增量定为零

  I100,0
- 分隔和终止的参数不适用

  P1000,2000　　（逗号应该是冒号）

  P10:20L　　（命令的结束字符不需要）
- 指定的行编号或增量大于 65529。

  E66000
- 该放行编号的地方不是数字，或者不是特殊字符 #，.，或 *。

  P@:200
- 第二行的编号比第一行的编号小。

  P200:100
- 给定盒式磁带的文件名称不合规格。

(1) 大于六个字符 LbTESTFILE

(2) 不是以英文字母开始 Wb1TEST

(3) 包含有非字母数字字符 WbTE*LN

- 指定了不能使用的汇编开关说明（Assembly swich）或者斜号（/）放错了位置，也可能是遗漏了。

  A/NO/N

  ANO

  AZZ
- 企图送入不是用编辑程序写的磁带，或者是其它一些原因不适合读的文件。

2) 缓冲器已满

在编辑缓冲器内没有地方可供放入文本。

3) 非法命令（ILLEGAL COMMAND）

命令行的第一个字符没有使用按照编辑/汇编程序规定的命令。

*Z1000:1200

4) 行编号太大

原因——

当使用 N 命令重编行号时，根据确定的起始行和增量，重编产生的行编号将超过 65529，

于是就会出错。

```
N 60000, 100  (如果在编辑缓冲器内文本的行数比 6 大)
    则最后一行就是 66000，超过 65529。
```

用插入命令（I）或替换命令（R）时，如有行号超过 65529 也将出错。

```
*I65000, 600
  65000 HELOO
  LINE NUMBER TOO LARGE
*                    (下一个编号将是 65600)
```

5） 行间没有空位（NO ROOM BETWEEN LINES）

原因——

由插入或替换命令产生的下一个行编号将大于或等于编辑缓冲器内文本的下一行的行编号。解决办法是减少增量或者重编缓冲器的行号。

```
*P100:115
 00100 HEY
 00114 YOU
*I112, 2
 00112 TEST
  NO ROOM BETWEEN LINES
                    (下一个编号 114 早已存在)
```

6） 没有这样的行

用命令指定的行不存在。

```
*P100:115
00100 HEY
00114 YOU
*E112
NO SUCH LINE        (没有 112 行)
```

7） 在缓冲器内没有文本

产生这种错误的原因：当编辑缓冲器是空的时候，发出向缓冲器要求文本的命令。但是在缓冲器是空的时候，Load、Insert、Basic 和 System 等命令能够执行。而所有的其它命令就要求缓冲器内至少要有一行。

```
*D#:*          (清除缓冲器)
*P
NO TEXT IN BUFFER
```

8） 字符串没有找到

用 F 命令找字符串，但在现行的行和缓冲器结束行之间没有能找到。

STRING NOT FOUND

2. 汇编错误

1) 引起汇编终止的错误：

符号表溢出，没有足够的存储器供汇编符号表用。

2) 造成卡死的错误：

- 不合适的标记
  在源码的标记区发现有如下字符串：
  a) 由非英文字母开始。
  b) 标记长度超过六个字符。
  c) 有的字符不是字母或数字。
- 表达式错误
  操作数区有不符合格式的表达式。
- 非法的寻址方式
  操作数区未确定寻址方式，或者此寻址方式对于指定的操作码是非法的。
- 非法的操作码
  在源语句的操作码区找到的字符串不是约定的指令助记符号或汇编伪指令。
- 遗漏信息
  没有提供为了正确进行汇编源码行所不可缺少的信息。如遗漏了操作码，或者操作数不全。

3) 造成警告的错误：

- 分枝超出了范围
  相对跳跃指令（JR,DJNZ）的终点（D）不在范围内（LC—128）$\leqslant$D$\leqslant$（LC+127），其中LC是分配给跳跃指令的第一字节的地址。如果指令硬要取位移量为十六进制的FE，那末，最终就分枝到此指令本身。
- 区域溢出
  对于确定指令的操作数，在操作数区确定的数目或表示式的结果太大。结果要截尾。例如，BIT_9，就将产生这种错误。
- 多定义符号
  操作数区有多定义符号。符号的第一个定义能够用于汇编。
- 多定义性
  源码行企图非法的重复定义一个符号。而这个符号的第一个定义是被保留的。仅当符号开始由DEFL定义时，符号才能由DEFL伪操作来重复定义。
- 无结束语句
  程序的结束语句遗漏了。
- 非定义符号
  操作数区有未经定义的符号，零用作为非定义符号。

附录， **LEVEL Ⅱ BASIC 存储器分配**

地址 十进制	地址 十六进制	内容
0.	0000	
		LEVEL Ⅱ BASIC ROM
12288	3000	保留区
14302	37DE-	——通讯状态地址
14303	37DF-	——通讯数据地址
14304	37E0-	——中断锁存器地址
14305	37E1-	——磁盘驱动选择锁存器地址
14308	37E4-	——盒式录音机选择锁存器地址
14312	37E8-	——行式打印机的地址
14316	37EC-	——软磁盘控制器的地址
14336	3800	
		TRS-80 鍵盘存储器
15360	3C00	
		TRS-80 CRT 显示存储器
16383	3FFF	
16384	4000	
		LEVEL Ⅱ BASIC 固定 RAM 中断矢量 (RST 1-7)
16402	4012-	
16405	4015-	——鍵盘装置的控制块
		—DCB+0=DCB 型式
		+1=驱动地址
		+2=驱动地址
		+3=0
		+4=0
		+5=0
		+6='K'
		+7='I'
16413	401D-	——显示控制块
		—DCB+0=DCB 型式
		+1=驱动地址(LSB)
		+2=驱动地址(MSB)
		+3=光标位置 N(LSB)
		+4=光标位置 N(MSB)
		+5=光标字符
		+6='D'
		+7='0'

地址 十进制	地址 十六进制	内容
16421	4025	行式打印机控制块
		DCB+0=DCB型式
		+1=驱动地址(LSB)
		+2=驱动地址(MSB)
		+3=行/页
		+4=行计数器
		+5=0
		+6='P'
		+7='R'
16429	402D	保留区
16463	404F	
16464	4050	FDC 中断矢量
16466	4052	通信中断矢量
16468	4054	保留区
16476	405C	
16478	405E	25毫秒实时钟中断
16512	4080	[保留区
		LEVEL Ⅱ BASIC 可供使用 RAM
		[保留区
16870	41E6	I/O 缓冲器
17127	42E7	
17128	42E8	始终保持零
17129	42E9	
		[↓程序文本
		[↓简单变量
		[↓数 组
		[↓字符串变量名称和整理操作
		[↓空闲存储器
		[↑堆 栈
		[↑字符串空间
		给出 MEMORY SIZE? 这就是留给与 BASIC 混用的机器语言子程序的保留区
20479	4FFF	4K 存储器终点
32767	7FFF	16K 存储器终点
49151	BFFF	32K 存储器终点
65535	FFFF	48K 存储器终点

(梁祖威编译)

第八篇 TRS-80 监视、纠错和调试程序使用手册

一、引　　言

TBUG 是一种功能很强的，机器语言的监控程序，它设计用来使你能直接存取 Z-80CPU —这是 Radio Shack TRS-80 微型计算机的心脏。TBUG 有如下功能：

（1）建立和修改机器语言程序。

（2）用断点和寄存器显示来调试机器语言程序。

（3）检查和修改 RAM 存储器的内容和 Z-80 寄存器的内容。

（4）用盒式磁带存取机器语言程序。

（5）执行用 TBUG 建立的机器语言程序。如果你有 LEVEL Ⅱ TRS-80 的微型机，你也能（经 TBUG）执行用编辑/汇编程序建立的程序。

这本手册将告诉你如何使用 TBUG。这本手册还有 Z-80 指令表的摘要。但是这本手册并不能教你如何写机器语言程序。而你要学会写机器语言程序，你就至少要一本篇幅很长的书。在附录 G 中列出几本这样的书。

关于此手册的一些情况：

TBUG 对于 LEVEL Ⅰ 和 LEVEL Ⅱ 这两种机器都有效。因为，LEVEL Ⅰ 和 LEVEL Ⅱ 的存储器位置和 ROM 子程序是不同的，所以在这两种机器中，TBUG 的存入和操作略有不同。

送入和使用 TBUG 有两节，每种型号（指 LEVEL Ⅰ，Ⅱ）的机器各有一节。另外，尚有很少的几节也用这种方法分开写。读这些章节的时候，就要看你有什么型号的 TRS-80 而决定。

1. 符号约定

在使用以前，你应该熟悉下面这些符号约定，这会使你在用此手册时感到方便。

* [ENTER] 表示“按 ENTER 键”。
* 大写字母和标点：表示它一出现就进入机器。

  例如　　B nnnn

  只有命令 B 是逐字进入的；nnnn 的值是由你给定的。
* 显示出的大写字母：表示你提供的输入，在计算机显示屏显示出来。这种规定仅是用于需要区别是计算机给出的，还是由你输入的。

  例如

  ## M 3C00 20

  计算机给出的是 ##，20，以及所有的空位；你要提供的是带有点线方框的字母。
* 小写斜体字母：表示若干个字（Words）若干字母（Letters）或者是由你提供的在某种情况下允许的一组数值。

  例如

  P aaaa bbbb cccc name。 aaaa，bbbb，cccc 是由用户提供的地址，name 是由用

户提供的名称。

* nnnnH：跟随在数字后面的字母 H 表示此数是十六进制格式。

例如：

3C00H 是十六进制数，它等于十进制数 15360。0AH 是十六进制数。它等于十进制数 10。

* ddddd：　没有 H 词尾的数是十进制的，它不要其它的标记。
* lsb：　表示在一个寄存器对和地址中的最低有效字节。

  例如：

  IX(lsb) 看作为 IX 寄存器的最低有效字节。
* msb：　表示在一个寄存器对和地址中的最高有效字节。

  例如：

  IX (msb) 看作为 IX 寄存器的最高有效字节。

2. 送入 TBUG 之前……

这个 TBUG 监控程序只用荧光屏最前面的十六列来显示计算机的输出和用户的输入。而其余的 48 列不受 TBUG 的影响（例外：LEVEL Ⅱ Punch 命令）。

为了执行这本手册中的几个例子，在进入 TBUG 之前，你应该清除荧光屏的显示（CLS 语句或 CLEAR 键）。

注意：TBUG 提供一个“活的键盘”——发每个命令以后，你不必按 ENTER 键。

二、送入和使用 TBUG（LEVEL Ⅰ）

1. 如果你的 TRS-80 是 LEVEL Ⅰ BASIC，送入 LEVEL Ⅰ TBUG 带就像送入 BASIC 程序带一样，利用 CLOAD 命令。因为磁带上的信息送入，星号就会一亮一灭。当程序送入后，控制就转向 TBUG 监控程序，记号 # 会出现在显示屏的左边顶端。这个记号取代了 BASIC 的 > 起动符号。

注意：如果送带过程中有了错误，将会显示如下信息

```
WHAT?
>
```

退回 TBUG 带并重新开始。你应该试着把录音机音量旋钮的位置稍许调高一些或调低一些。

退出 TBUG 并返回 LEVEL Ⅰ BASIC：

最简便的方法是打入：

J 0000

强迫机器进入电源刚打开的状态。

请记住，你只要打入标有点线方框的字符就可以了。

另外一个方法是：你可以按下本机左后方的复位钮，利用手动复位功能。

（J 命令请看 TBUG 命令部分）

要从 LEVEL Ⅰ BASIC 返回 TBUG，你必须重新送入 TBUG 带。

2．在LEVEL Ⅰ 机器上使用 TBUG

在 LEVEL Ⅰ TRS-80 中，TBUG 是放入存储器位置是 4000H 到 43FFH。因此，你的机器语言程序必须写入 4400H 到存储器的终点（4K 为 4FFFH，16K 为 7FFFH）。堆栈指示器是 43FDH，如果你有需要的话，当然可以很容易地把它改变到存储器的另外的地址。

使 TBUG 成为如此有效和多用途的监控程序，是因为它采用了寄存器保留区（Register Save Area）。这个区域用来在 TBUG 控制时，保存 Z-80 的寄存器的内容。这就保证了寄存器的内容不受 TBUG 的影响。

存取寄存器保留区的 TBUG 命令是：断点（Breakpoint），寄存器（Register），转移（Jump）和转向（Go）。

当通过断点命令，由用户程序进入 TBUG 时，Z-80 寄存器的内容是被放在寄存器保留区的。寄存器命令用调出寄存器保留区的地址，以显示这些寄存器的内容。在用户程序执行以前（经 Jump 和 Go 命令），由寄存器保留区把数据送入 Z-80 的寄存器。更详细的内容请参阅命令解释部分。

在 LEVEL Ⅰ TRS-80 中，寄存器保留区安排在存储器地址 43B7H 到 43CEH。表 1 表示 Z-80 寄存器的每一个存放位置。

注意：为了修改寄存器保留区内的地址，你可以在执行 Jump 或 Go 命令以前，用 Memory 命令来规定各 CPU 寄存器的初始值。

表1. LEVEL Ⅰ 寄存器保留区

寄存器	存储单元	寄存器	存储单元
A′	43B8H	A	43C0H
F′	43B7H	F	43BFH
B′	43BAH	B	43C2H
C′	43B9H	C	43C1H
D′	43BCH	D	43C4H
E′	43BBH	E	43C3H
H′	43BEH	H	43C6H
L′	43BDH	L	43C5H
IX(MSB)	43C8H	SP(MSB)	43CCH
IX(LSB)	43C7H	SP(LSB)	43CBH
IY(MSB)	43CAH	PC(MSB)	43CEH
IY(LSB)	43C9H	PC(LSB)	43CDH

三、送入和使用 TBUG（LEVEL Ⅱ）

如果你的 TRS-80 是 LEVEL Ⅱ BASIC，你就在 SYSTEM 命令下送入 TBUG。具体操作是：把磁带装入录音机，然后把音量旋钮放于 LEVEL Ⅱ 规定的位置（对于 CTR-41型录音机约为 4 — 6）。

现在打入

> SYSTEM ENTER

*? TBUG ENTER

磁带开始送入，在荧光屏上要显出闪亮着的星号。当磁带送入完毕，SYSTEM 起动符号将重新出现。打入：

*? / ENTER

号将出现在荧光屏的最左边。这是 TBUG 的起动符号——相当于 BASIC 的 > 号。

注意：如果在送入 TBUG 过程中有错误，则在显示屏的右上角，星号的旁边要出现一个 C。这时要退回磁带，并重新开始，需把音量旋钮的位置稍许调高一点或稍许调低一点。

1. 退出 TBUG 并返回 LEVEL Ⅱ BASIC：

只要顺序地打入

J 0000

就能使计算机成为电源刚打开时的状态。请注意，你只要打入用点线所框出的那部分，不必按 ENTER。

其中 J 命令在 TBUG 命令章节中有说明。或者，你可以按下键盘的左边后面的复位按钮，以使计算机复位。

注意：如果你的机器语言程序已经改变了 4000H 到 42F8H 单元的内容，则按复位按钮就不能达到使计算机复位的功能，也不能返回到 BASIC。在这种情况下，你必须关掉计算机的电源，然后重新开机。

从 LEVEL Ⅱ BASIC 再进入 TBUG：

假如你没有改变 TBUG 驻留区的内容，则你打入：

> SYSTEM ENTER

*? /17312 ENTER

就能由 BASIC 返回到 TBUG。

请注意：相对于存储器容量来说，这个地址（十进制 17312）是太低了。所以执行BASIC 程序将更改 TBUG 的驻留区，故而当你执行过 BASIC 程序，就必须重新再送入 TBUG。

2. 在 LEVEL Ⅱ 机器中的 TBUG 用法：

在 LEVEL Ⅱ TRS-80 微型机中，TBUG 是放入 4380H 到 497FH 存储单元。这样，你的机器语言程序可以用地址4980H到存储器的终点这一个区间来写（对于4K的机器来说，终点是 4FFFH，16K的机器是 7FFFH，32K 的机器是 BFFFH，48K 的机器是 FFFFH）。堆栈预定是在 4980H，然而，你如有需要也可以很容易地改变到存储器的其它的地方（大于 4980H）。

使 TBUG 成为如此有效和多用途的监控程序，是因为它采用了寄存器保留区（Register Save Area）。这个区域用来在 TBUG 控制时，保存 Z-80 的寄存器内容。这就保证了寄存器的内容不受 TBUG 的影响。

存取寄存器保留区的 TBUG 命令是：断点（Breakpoint），寄存器（Register），转移

(Jump) 和转向 (Go)。

当通过断点命令，由用户程序进入 TBUG 时，Z80 寄存器的内容是被放在寄存器保留区的。寄存器命令用调出寄存器保留区的地址，以显示这些寄存器的内容。在用户程序执行以前（经 Jump 和 Go 命令），由寄存器保留区把数据送入 Z80 的寄存器。更详细的内容请参阅命令解释部分。

在 LEVEL Ⅱ TRS-80 微型机中，寄存器保留区是安排在存储器地址为 4825H 到 483CH。表 2 表示 Z-80 寄存器的每一个存放位置。

注意：为了修改寄存器保留区内的地址，你可以在执行 Jump 或 Go 命令以前，用 Memory 命令来规定 CPU 各寄存器的初始值。

表2. LEVELⅡ 寄存器保留区

寄存器	存储单元	寄存器	存储单元
A′	4826H	A	482EH
F′	4825H	F	482DH
B′	4828H	B	4830H
C′	4827H	C	482FH
D′	482AH	D	4832H
E′	4829H	E	4831H
H′	482CH	H	4834H
L′	482BH	L	4833H
IX(MSB)	4836H		
IX(LSB)	4835H		
IY(MSB)	4838H		
IY(LSB)	4837H		
SP(MSB)	483AH		
SP(LSB)	4839H		
PC(MSB)	483CH		
PC(LSB)	483BH		

四、 TBUG 命 令

在下面的范例中，要记住键盘输入（由你打入的）是用点线方框标出的。由计算机产生和输出的，则不带点线方框。注意：当单独一个 TBUG 提示符号 # 出现于荧光屏的下一行时，你才能打入命令。

1. 存储器命令 (Memory)：

存储器命令使你能够检查任何一个存储单元的内容，可以是 ROM 也可以是 RAM。它还能让你修改任何 RAM 单元的内容。为了开始执行这个命令，打入字母 M，后面跟着你

想改变其内容的那个地址：

M nnnn

其中 nnnn 是一个四位十六进制的地址。

（所有从 TBUG 输入或输出的数均为十六进制。）

举例：假如你想检查一下存储单元 3C20H 的内容。你只要打入 M3C20H：

```
#  M  3C20  20
```

注意：你打入第四位地址数以后，TBUG 就会立即显示 3C20H 的内容；为了容易读，TBUG 在这一行中插入一些空位。

现在你已经开始 M 命令了，你可以不改变现行地址的内容，而去检查存储器内下一个较高的地址，也可以改变现行地址的内容，然后再进到下一个地址。

为了改变所显示的那个单元的内容，你可以在显示出的数值后面打入一个新的 2 位数（十六进制）。例如为了把 3C20H 的内容由 20H 改为 4CH，则打入：

```
#  M  3C20  20 4C
   3C21  20
```

如果你试过了上例，则会注意到两个问题：

1)TBUG 自动显示存储器内下一个地址及其内容。

2)字母 L 出现在显示屏顶行的中间处。为了知道为什么会有 L 出现，你必须知道地址 3C00 到 3FFF 是用于显示屏的存储区。换句话说，凡是存贮在这些单元内的值，将会自动地按 ASCII 码的形式在监视屏上显示出来。

因为 4CH 是 ASCII 码中相应的字母 L，这个字母要出现在显示屏上相应于地址 3C20H 的地方。

你为某一地址打入新的两位数值后，TBUG 就要对此地址的内容作相应的修改，还将显示下一个地址。为了使下一个地址的内容保持不变，你只要按下 ENTER 键，TBUG将移动到下一个较高的地址。

例如

```
#  M  3C20  20  4C
   3C21 20  ENTER
   3C22 20  ENTER
   3C23 20  X
#
```

任何时候要从存储器命令退出时，只要打入 X。这个字符将不被显示出来，因为它有控制功能，TBUG 返回到起动状态。你就能接着送入任何 TBUG 命令。

2. 转移命令（Jump）：

转移命令使你能在指定的地址开始执行机器语言程序。

其格式是：

J nnnn

其中 nnnn 是一个 4 位十六进制的地址，程序将由此开始执行。

当你打入转移命令时，TBUG 将进行如下操作：

1）把 nnnn 送入程序计数寄存器（PC）。

2）从寄存器保留区把数据送到 Z80 的一些寄存器内。

3）开始执行。

例如：

为了从 4A00H 开始执行一个程序，打入：

```
#  J  4A00
```

你应该注意：所指定的地址要有一条可以执行的指令。它应该是一字节指令或多字节指令的第一个字节。

你可以在打入第四个地址字符以前的任意时刻，用打入字母 X 的办法来取消转移命令。这样就可以使控制返回到 TBUG，然后就能进入其它的命令。

3. 断点命令（Breakpoint）：

这个命令可以使你在预先确定的点或指令处中断程序的正常执行。这是一个极为有用的手段，可用于调试程序，以及很简便地终止程序。

为了开始断点命令，打入 B，后面跟有你希望产生运行中断的十六进制格式的地址。

B nnnn

其中 nnnn 是指定的断点地址。这个地址应该有一个单字节指令或多字节指令的第一个字节。

B nnnn 的作用是用断点序列来代替 nnnn 到 nnnn + 2 的内容。这些存储单元的原来的内容被保存起来，以后可以用安置（Fix）命令恢复（请看后面的内容）。

注意：你可以在打入地址第四位之前，用打入 X 来取消断点命令。

当程序的执行到达你所送入的断点时，控制将转向 TBUG，Z80 寄存器的内容将被放置在寄存器保留区。在那里你就可以检查或修改它们（请看寄存器命令）。

不要企图在一个程序中一次放置一个以上的断点，因为只有最后进入的那个断点能用安置（Fix）命令恢复其原来的内容。

例

```
#  M  5200   C3  ENTER
   5201      A0  ENTER
   5202      52  X
#  B  5200
#  M  5200   CD
```

在这个例子中，地址 5200H 到 5202H 包含有一条 Z80 的指令（Jump to 52A0H）。断点插入在 5200H。如果你检查 5200H，你会看到它现在的内容是 CDH，它是断点序列的第一个字节。第二个字节为 80H，第三个字节为 43H。

当你想在 5200H 开始执行，立刻就会碰到断点，断点序列是 TBUG 的入口地点，所以

TBUG 重新获得控制并返回 TBUG 状态，当返回 TBUG 状态后就会出现起动符号 #。为了说明这些，打入：

```
# J 5200
```

这时你可以检查寄存器和存储单元，也可以打入其它任意的 TBUG 命令。然而，如果你打算用安置命令来恢复断点的原来的内容，你应该在现在或者用户的 PC 内容修改以前来做这一点。

```
# F
# M 5200 C3 ENTER
  5201  A0      ENTER
  5202  52
```

注意：5200H 到 5202H 已经恢复到它们原来的内容。你现在能够用 Go 或 Jump 命令继续执行。

4. 安置命令（Fix）：

安置命令是与断点命令结合起来用的。在执行机器语言程序期间，到达断点以后，控制返回到 TBUG，打入 F，于是 F 命令告诉 TBUG 去恢复断点的原来的内容，并将它放入用户 PC 所指定的地址。请参阅断点命令的例子。

F 命令用于执行断点之后和改变用户 PC 内容之前。如果你在任何其它的时刻用 F 命令，则它会破坏你的一部分程序。

5. 转向命令（Go）：

转向命令告诉 TBUG，在由用户 PC 所规定的地址开始执行。只有当你知道用户的 PC 寄存器是什么内容时，或者是你刚用安置命令来恢复断点处内容时，才能用它。否则，转向命令能够使计算机“驶往不知道的地方”。

为了用转向命令，只要打入字母 G。这时 TBUG 将立即从寄存器保留区把数据送入 Z-80 的寄存器，并开始执行。

6. 寄存器命令（Register）：

寄存器命令使你能检查寄存器保留区的内容。只要打入字母 R，TBUG 将显示寄存器保留区如下：

```
A'F'    B'C'
D'E'    H'L'
AF      BC
DE      HL
IX(msb,lsb)  IY(msb,lsb)
SP(msb,lsb)  PC(msb,lsb)
```

为了修正任意的寄存器的内容，只要把所要的值放在相应的寄存器保留区地址。下次你

执行 Jump 和 Go 命令时，将会把保留区的内容送入 Z80 寄存器，这时你所作的修改就完成了。

7. 录制命令（Punch）：

录制命令让你能把存储器的内容转储到磁带上。这个命令对于 LEVEL Ⅰ 和 LEVEL Ⅱ 机器是不同的，必须由机器型式确定。由 P 命令录制的程序磁带能够像 TBUG 一样的送入机器。

注意：在执行 P 命令以前，应该把盒式录音机放于录音工作方式。

LEVEL Ⅱ 的录制命令

在 LEVEL Ⅱ TBUG 中，录制命令有如下形式：

P aaaa bbbb cccc name

其中 aaaa 是十六进制转储数据块的起始地址。

bbbb 是转储数据块的结束地址。

cccc 是入口点——这是磁带上数据送入机器后，开始执行的地址。

name 是你存在磁带上的程序的文件名称。

用 LEVEL Ⅱ TBUG P 命令录制的磁带能在 SYSTEM 命令下送入机器 —— 如同送入TBUG 本身一样。所有的 SYSTEM 磁带都有一个文件名称，这就是为什么 P 命令也要你确定一个文件名称。

这个文件名称能够是任意的字符序列——从 1 个到 6 个字符。如果你希望用比六个字符短的名称，你必须按 ENTER 键以便开始录制。如果你打入 6 个字符的文件名称，录制工作就自动开始。我们要用既容易记住又容易打入的文件名称。

例如：假定你的程序放在 5000H 到 521FH，你想用 TEST 的名称把程序存在磁带上。则打入：

\# P 5000 521F 5000 TEST ENTER

如你想以 TESTO1 的名称存入同样的程序，则打入：

\# P 5000 521F 5000 TESTO1

在第一种情况下，P 命令在按下 ENTER 键后立即执行。而在第二种情况下，要等你打入文件名称的第六个字符时，P 命令才执行。在你打入这个命令以前请把录音机放在录音工作方式。

当你正在打入起始地址，终止地址或入口地址时，你可以用打入字母 X 来取消 P 命令。然而，如果你正在打入文件名称，你就必须按下 BREAK 键，因为 X 对于文件名称是一个有效的字符（译注：因为文件名称不排除用字母 X，所以当你打下 X 时，机器会把它看作是你定义的文件名称中的一个字母，而不看作是一个取消命令的功能键）。

为了执行录制磁带命令，TBUG 打开录音机的电源，把指定的存储单元的内容写到磁带上，然后关上录音机的电源并返回到起动状态。输出的磁带格式是：

1）用零组成的 255 个字节的引导部分。

2）同步码（A5H）。

3）文件名称引导码（55H）。

4）带有后置空位（20H）的六字节文件名称。

5）有如下的一个或若干个数据块：

（1）数据引导码（3CH）。

（2）在这个数据块内的字节数目（从1到256，其中256是由00H来表示）。

（3）这一数据块的开始地址（lsb，msb）。

（4）数据（如在5b中确定的1到256个字节）。

（5）一字节的检查和，它是开始地址和数据的和。

6）入口点的引导码（78H）。

7）入口点的地址（lsb，msb）。

至于被写的数据块（上面第5项）的多少不受限制；TBUG将根据你给的开始地址和终止地址来计算所需要的量。

当读到入口地址时，送入操作就停止。

注意：这个磁带格式只能用TBUG送入命令(Load)或者LEVEL Ⅱ BASIC的SYSTEM命令才能被读出。

8. 送入命令（Load）：

送入命令是用于把先前录制在磁带上的机器语言传送到TRS-80的存储器内。用打入字母L来确定这个命令。TBUG将立即执行此命令。

在执行L命令期间，TBUG将不接受其它的键盘输入。只有一种方法可以取消L命令，那就是按下TRS-80键盘左后方的复位按钮。

在磁带信息送入期间，星号将出现在显示屏上且闪烁，表示数据在输往计算机。这个星号对于LEVEL Ⅰ是在显示屏的左边较高的地方，而对于LEVEL Ⅱ是在左边较低的地方。随着送入工作的完成，起动符号 ## 将出现在下一个显示行。

在送入期间，如发生检查和（Checksum）错误，TBUG将在L命令的下一行显示字母E。在这种情况下，试着再次送入磁带，产生这种错误多半是音量旋钮的位置放得稍许高了或低了一些。

五、有关变换的一些考虑

如果你有一台LEVEL Ⅰ TRS-80，但是你计划把它提高级别成LEVEL Ⅱ的机器，你或许想把由TBUG建立的若干程序也转换成LEVEL Ⅱ的。这时你应该了解LEVEL Ⅱ和LEVEL Ⅰ之间有几个重要的不同之点，这些不同点要求你修改你的LEVEL Ⅰ的机器语言程序，使得这些程序将可以在LEVEL Ⅱ机器上运行。

这一节将为你提供一些信息和提示，以使变换能简化一些。

由LEVEL Ⅰ转换成LEVEL Ⅱ时，要注意三点，

1）程序的位置。

2）对于ROM子程序调用（CALL）语句。

3）盒式磁带格式和传送速率。

第1项指的是LEVEL Ⅰ TBUG是放置在从4000H到43FFH的地址，而LEVEL Ⅱ是放置在从4380H到497FH的地址。所以，如果你有一个LEVEL Ⅰ机器语言程序，且由低于4980H的地址开始，则你必须在LEVEL Ⅱ把它“重新定位”。这是有点儿麻烦，因为所有的CALL和JP语句（还有一些其它的）必须改变。避免这些问题的最简单的方法是由4980H以上开始写你的LEVEL Ⅰ的程序。

第2项指的是改变利用ROM子程序的CALL语句。在这本手册的附录中，你将能看到关于如何用某些子程序的指令。你还将看到LEVEL Ⅰ的CALL程序是稍许不同于LEVEL Ⅱ的CALL程序。例如我们说，你想用键盘输入子程序来由键盘输入一个字节。在LEVEL Ⅰ中，你应该作如下的编码：

```
4A00  CD400B  00100AGN  CALL  0B40H; SCAN  KEYBOARD
4A03  28FB    00120     JR    Z, AGN
                        ; DO AGAIN IF KB CLEAR
```

然而，在LEVEL Ⅱ中，同样的编码看起来会成如下这样：

```
4A00  D5      00110     PUSH  DE      ; MUST SAVE DE
4A01  FDE5    00120     PUSH  IY      ; AND IY
4A03  CD2B00  00130AGN  CALL  2BH     ; SCAN ROUTINE
4A06  B7      00140     OR    A       ; A=0 IF KB CLEAR
4A07  28F7    00150     JR    Z, AGN  ; BRENCH IF NO BYTE
4A09  FDE1    00160     POP   IY      ; RESTORE IY
4A0B  D1      00170     POP   DE      ; AND DE
```

正如你所看到的，指令的数目和指令本身均有显著的差别。

把程序的形式由LEVEL Ⅰ变换成LEVEL Ⅱ的最简易的方法是：记住变换来编制你的LEVEL Ⅰ的程序。即在你编制你的LEVEL Ⅰ TBUG程序和你希望调用LEVEL Ⅰ ROM子程序来实现某些功能时，你必须看一下LEVEL Ⅰ和LEVEL Ⅱ子程序两者的调用有什么不同。如果LEVEL Ⅱ比LEVEL Ⅰ需较多的字节，则你应该用NOP(00H)指令来“填充”你的LEVEL Ⅰ程序，直到它们有同样的大小。在上面的例子中，你的LEVEL Ⅰ程序的地址4A03H到4A0AH要包含有若干NOP指令——在这个程序中的下一个有用的指令应该是在4A0BH的位置。另一方面，如果你想更进一步地减少你的变换工作，你应如下那样来编你的LEVEL Ⅰ程序：

```
4A00  D5      00100     PUSH DE       ; REQ'D FOR LEVEL Ⅱ
4A01  FDE5    00120     PUSH IY       ; TO SAVE DE&IY
4A03  CD400B  00130 AGN CALL 0B40H    ; CHANGE CALL FOR LEVEL Ⅱ
4A06  00      00140     NOP           ; USE'OR A' FOR LEVEL Ⅱ
4A07  28F7    00150     JR   Z,AGN    ; DO AGAIN IF KB CLEAR
4A09  FDE1    00160     POP IY        ; RESTORE IY
4A0B  D1      00170     POP DE        ; AND DE
```

注意：只有源码行130和140需要改变得适合LEVEL Ⅱ程序。当然，LEVEL Ⅰ能

自动显示字节；而 LEVEL Ⅱ 却不能够。如果你有此需要，则必须加上显示字节的程序。

变换的最后一项是为了解决把 LEVEL Ⅰ 的磁带送入机器。解决此问题的明显结论是在 LEVEL Ⅱ 的管理下用存储（Memory）命令重新把你的机器语言程序送入机器。这是假定你有这些程序的硬拷贝复制件。

还有一种显而易见的办法是——你可以写在 LEVEL Ⅱ 管理下能够让你读 LEVEL Ⅰ 带的机器语言程序。实现此功能的程序示于表 1 。你可以用 TBUG 或 TRS-80 EDITOR/ASSEMBLER 来送入这个程序。把这个程序和 TBUG 一起送入存储器。利用 Jump 命令，打入

```
# J 4100
```

把由 LEVEL Ⅰ TBUG 编制的磁带装入盒式磁带录音机，按下录音机 PLAY 键，然后打入 ENTER 。因为程序由磁带送入机器，所以大家熟悉的星号将出现在显示屏的左上方。送入完成，控制将返回到 TBUG。现在就可按你的需要来修改你的 LEVEL Ⅰ 程序；然后利用 TBUG 按 LEVEL Ⅱ 的格式把程序输出到磁带上。

注意：你将会发现“浮动定位”功能装进了你的程序。在4130位置的指令是：

```
04130  010000  04320  LD  BC,0
```

它能用来改变你的程序的送入地址。例如，假定你的 LEVEL Ⅰ 的程序是放置在存储器的 4400H 到 4623H，如果你想由磁带送入这个程序，这个程序将会冲掉一部分 TBUG。然而，你可以用放置在 4131H 和 4132H 单元的位移量，把送入地址改变到存储器的另外一个地方。例如把 4130H 处的指令改变为：

```
4130  01000C  04320  LD  BC,  0C00H
```

你的程序就可以放入 5000H 到 5223H 单元。

这是因为对于被写的程序在数据送入以前已经把 0C00H 加到 4400H 和 4623H 上去了。

这将使你能在 LEVEL Ⅱ 机器上读你的 LEVEL Ⅰ 磁带，如有需要还可以完成程序的再定位。但是你必须做上面所介绍的那些变换。

注意：当你为了转移而送入 LEVEL Ⅰ 程序时。你必须记住要把录音机的音量旋钮调到 7 — 9 之间。

如果在送入 LEVEL Ⅰ 磁带时发生错误，则最左边的星号将会被一个 E 来代替，而 TBUG 将恢复控制。再重新送入 LEVEL Ⅰ 磁带（试着把音量旋钮放在不同的位置）。

清单1. 供 LEVEL Ⅱ 机器用的 LEVEL Ⅰ 装入程序

```
04000 ; LEVEL I CASSETTE LOAD
04005 ;
04010 ; THIS PROGRAM LOADS A LEVEL I OBJECT FILE TAPE
04020 ; INTO A LEVEL II TRS-80
04025 ;
04027 ; VOLUME ON THE RECORDER MUST BE
04028 ; AT A LEVEL I SETTING (7-9)
04029 ;
04030 ; ADDRESSES 4131H (LSB) AND 4132H (MSB) MAY BE
04040 ; CHANGED TO RELOCATE INPUT PROGRAM
```

```
                04050 ;
                04055 ; ROM AND TBUG ADDRESSES
                04065 ;
4380            04070 TBUG    EQU   4380H          ; RETURN TO TBUG ADDRESS
3C00            04080 VIDEO   EQU   3C00H          ; UPPER LEFT CORNER VIDEO
002B            04090 KEY     EQU   2BH            ; KEYBOARD SCAN ROUTINE
                04093 ;
4100            04095         ORG   4100H
4100 310041     04100 START   LD    SP, $          ; SET STACK POINTER
4103 CD2B00     04110 CHKEY   CALL  KEY            ; SEE IF THERE IS
4106 A7         04120         AND   A              ;  ANYTHING FROM KEYBOARD
4107 28FA       04130         JR    Z, CHKEY       ; NO-CHKEY
4109 FE0D       04140         CP    0DH            ; YES-CARRIAGE RETURN?
410B 20F6       04150         JR    NZ, CHKEY      ; NO-CHKEY
410D CD8F41     04160         CALL  CTON           ; YES-TURN ON CASSETTE
4110 AF         04170         XOR   A
4111 CD6741     04180 CLOAD1  CALL  GETBIT         ; HAVE WE FOUND
4114 FEA5       04190         CP    0A5H           ;  THE SYNC CHARACTER?
4116 20F9       04200         JR    NZ,CLOAD1      ; NO-CLOAD1
4118 3E2A       04210         LD    A,'*'          ; TURN ON
411A 32003C     04220         LD    (VIDEO),A      ;  BOTH ASTERISKS
411D 32013C     04230         LD    (VIDEO+1),R
4120 CD8741     04240         CALL  GTBYTE         ; GET
4123 57         04250         LD    D,A            ;  START
4124 CD8741     04260         CALL  GTBYTE         ;    ADDRESS
4127 5F         04270         LD    E,A            ;      FOR DE
4128 CD8741     04280         CALL  GTBYTE         ; GET
412B 67         04290         LD    H,A            ;  ENDING
412C CD8741     04300         CALL  GTBYTE         ;    ADDRESS
412F 6F         04320         LD    L,A            ;      FOR HL
4130 010000     04320         LD    BC,0           ; CHANGE CONSTANT
4133 09         04330         ADD   HL,BC          ;  TO RELOCATE***
4134 EB         04340         EX    DE,HL          ;  RELOCATION
4135 09         04350         ADD   HL,BC          ;    AMOUNT
4136 EB         04360         EX    DE,HL          ;      TO DE & HL
4137 0E00       04370         LD    C,0            ; ZERO OUT CHECKSUM
4139 23         04371         INC   HL             ; INCLUDE CHECKSUM
                04380;
                04390; NOW WE WILL LOAD EACH BYTE OFF THE TAPE AND
                04400; STORE IN MEMORY AS INDICATED BY DE AND HL
                04410;
413A CD8741     04420 CLOAD2  CALL  GTBYTE         ; GET BYTE FROM TAPE
413D 12         04430         LD    (DE),A         ; STORE IT IN MEM
```

```
413E 13       04440          INC    DE              ;   AND STEP COUNTER
413F FE0D     04450          CP     0DH             ; IS IT A CARRIAGE RETURN?
4141 200A     04460          JR     NZ,CLOAD3       ; NO-CLOAD3
4143 F5       04470          PUSH   AF              ; YES-CONTINUE
4144 3A013C   04480          LD     A, (VIDEO+1)
4147 EE0A     04490          XOR    0AH             ; BLINK ASTERISK
4149 32013C   04500          LD     (VIDEO+1),A
414C F1       04510          POP    AF
414D 81       04520 CLOAD3   ADD    A,C             ; ADD BYTE TO CHECKSUM
414E 4F       04530          LD     C,A             ;   AND SAVE IT
414F 7C       04540          LD     A,H             ; SEE
4150 BA       04550          CP     D               ;   IF
4151 20E7     04560          JR     NZ,CLOAD2       ;     WE
4153 7D       04570          LD     A,L             ;       ARE
4154 BB       04580          CP     E               ;         DONE
4155 20E3     04590          JR     NZ,CLOAD2       ; NO-CLOAD2
4157 CD9441   04600          CALL   CTOFF           ; YES-TURN OFF CASSETTE
415A 79       04610          LD     A,C             ; IF CHECKSUM
415B A7       04620          AND    A               ;   EQUALS ZERO
415C CA8043   04630          JP     Z,TBUG          ;     RETURN TO TBUG
415F 3E45     04640          LD     A,'E'           ; NO-TURN FIRST ASTERISK
4161 32003C   04650          LD     (VIDEO),A       ;   INTO AN "E"
4164 C38043   04660          JP     TBUG            ;     AND GO TO TBUG
              04670 ;
              04680 ; THIS SUBROUTINE READS A SINGLE BYTE FROM THE TAPE
              04690 ; PORT AND 'ADDS' INTO THE A REGISTER
              04700 ;
4167 D9       04710 GETBIT   EXX                    ; USE ALTERNATE REGISTERS
4168 08       04720          EX     AF,AF'
4169 DBFF     04730 GB1      IN     A, (255)        ; INPUT SYNC BIT
416B 17       04740          RLA                    ; DO WE HAVE ONE?
416C 30FB     04750          JR     NC,GB1          ; NO-GB1
416E 067C     04760          LD     B,7CH           ; DELAY
4170 10FE     04770          DJNZ   $               ;   LOOP
4172 CD9941   04780          CALL   GTSTAT          ; CLEAR INPUT LATCH
4175 06F8     04790          LD     B,0F8H          ; DELAY
4177 10FE     04800          DJNZ   $               ;   LOOP
4179 DBFF     04810          IN     A, (255)        ; GET DATA BIT
417B 47       04820          LD     B,A
417C 08       04830          EX     AF,AF'          ; DATA BIT IS IN BIT?
417D CB10     04840          RL     B               ;   OF B REGISTER
417F 17       04850          RLA                    ; ADD IT INTO A REGISTER
4180 F5       04860          PUSH   AF
```

```
4181 CD9941    04870          CALL    GTSTA T          ; CLEAR INPUT LATCH
4184 F1        04880          POP     AF
4185 D9        04890          EXX                      ; REGISTERS NORMAL AGAIN
4186 C9        04900          RET                      ; FINISHED
               04910 ;
               04920 ; THIS SUBROUTINE CALLS GETBIT 8 TIMES
               04930 ; TO GET A FULL BYTE
               04940
4187 0608      04950 GTBYTE   LD      B,8              ; SET COUNT
4189 CD6741    04960 GB2      CALL    GETBIT           ; GET A BIT
418C 10FB      04970          DJNZ    GB2              ; LOOP' TILL WE HAVE A BYTE
418E C9        04980          RET
               04990 ;
               05000 ; THIS SUBROUTINE TURNS ON THE CASSETTE
               05010 ;
418F 2104FF    05020 CTON     LD      HL,0FF04H        ; SET BIT 2
4192 1808      05030          JR      CSTAT
               05040 ;
               05050 ; THIS SUBROUTINE TURNS OFF THE CASSETTE
               05060 ;
4194 2100FB    05070 CTOFF    LD      HL,0FB00H        ; CLEAR BIT 2
4197 1803      05080          JR      CSTAT
               05090 ;
               05100 ; THIS SUBROUTINE CHECKS THE PORT STATUS
               05110 ;
4199 2100FF    05120 GTSTAT   LD      HL,0FF00H        ; CLEAR READ LATCH
419C 3AA741    05130 CSTAT    LD      A, (STATUS)
419F A4        05140          AND     H                ; SET AND
41A0 B5        05150          OR      L                ;   CLEAR BITS
41A1 D3FF      05160          OUT     (255),A          ; OUTPUT TO PORT
41A3 32A741    05170          LD      (STATUS), A
41A6 C9        05180          RET
41A7 00        05190 STATUS   DEFB    0
4100           05200          END     START
```

六、附　录

1. 按助记符号字母为序的 Z80 指令表

操作码	源　码		操作码	源　码	
8E	ADC	A,(HL)	FD19	ADD	IY,DE
DD8E05	ADC	A,(IX+IND)	FD29	ADD	IY,IY
FD8E05	ADC	A,(IY+IND)	FD39	ADD	IY,SP
8F	ADC	A,A	A6	AND	(HL)
88	ADC	A,B	DDA605	AND	(IX+IND)
89	ADC	A,C	FDA605	AND	(IY+IND)
8A	ADC	A,D	A7	AND	A
8B	ADC	A,E	A0	AND	B
8C	ADC	A,H	A1	AND	C
8D	ADC	A,L	A2	AND	D
CE20	ADC	A,N	A3	AND	E
ED4A	ADC	HL,BC	A4	AND	H
ED5A	ADC	HL,DE	A5	AND	L
ED6A	ADC	HL,HL	E620	AND	N
ED7A	ADC	HL,SP	CB46	BIT	0,(HL)
86	ADD	A,(HL)	DDCB0546	BIT	0,(IX+IND)
DD8605	ADD	A,(IX+IND)	FDBC0546	BIT	0,(IY+IND)
FD8605	ADD	A,(IY+IND)	CB47	BIT	0,A
87	ADD	A,A	CB40	BIT	0,B
80	ADD	A,B	CB41	BIT	0,C
81	ADD	A,C	CB42	BIT	0,D
82	ADD	A,D	CB43	BIT	0,E
83	ADD	A,E	CB44	BIT	0,H
84	ADD	A,H	CB45	BIT	0,L
85	ADD	A,L	CB4E	BIT	1,(HL)
C620	ADD	A,N	DDCB054E	BIT	1,(IX+IND)
09	ADD	HL,BC	FDCB054E	BIT	1,(IY+IND)
19	ADD	HL,DE	CB4F	BIT	1,A
29	ADD	HL,HL	CB48	BIT	1,B
39	ADD	HL,SP	CB49	BIT	1,C
DD09	ADD	IX,BC	CB4A	BIT	1,D
DD19	ADD	IX,DE	CB4B	BIT	1,E
DD29	ADD	IX,IX	CB4C	BIT	1,H
DD39	ADD	IX,SP	CB4D	BIT	1,L
FD09	ADD	IY,BC	CB56	BIT	2,(HL)

操作码	源 码		操作码	源 码	
DDCB0556	BIT	2,(IX+IND)	DDCB0576	BIT	6,(IX+IND)
FDCB0556	BIT	2,(IY+IND)	FDCB0576	BIT	6,(IY+IND)
CB57	BIT	2,A	CB77	BIT	6,A
CB50	BIT	2,B	CB70	BIT	6,B
CB51	BIT	2,C	CB71	BIT	6,C
CB52	BIT	2,D	CB72	BIT	6,D
CB53	BIT	2,E	CB73	BIT	6,E
CB54	BIT	2,H	CB74	BIT	6,H
CB55	BIT	2,L	CB75	BIT	6,L
CB5E	BIT	3,(HL)	CB7E	BIT	7,(HL)
DDCB055E	BIT	3,(IX+IND)	DDCB057E	BIT	7,(IX+IND)
FDCB055E	BIT	3,(IY+IND)	FDCB057E	BIT	7,(IY+IND)
CB5F	BIT	3,A	CB7F	BIT	7,A
CB58	BIT	3,B	CB78	BIT	7,B
CB59	BIT	3,C	CB79	BIT	7,C
CB5A	BIT	3,D	CB7A	BIT	7,D
CB5B	BIT	3,E	CB7B	BIT	7,E
CB5C	BIT	3,H	CB7C	BIT	7,H
CB5D	BIT	3,L	CB7D	BIT	7,L
CB66	BIT	4,(HL)	DC8405	CALL	C,NN
DDCB0566	BIT	4,(IX+IND)	FC8405	CALL	M,NN
FDCB0566	BIT	4,(IY+IND)	D48405	CALL	NC,NN
CB67	BIT	4,A	CD8405	CALL	NN
CB60	BIT	4,B	C48405	CALL	NZ,NN
CB61	BIT	4,C	F48405	CALL	P,NN
CB62	BIT	4,D	EC8405	CALL	PE,NN
CB63	BIT	4,E	E48405	CALL	PO,NN
CB64	BIT	4,H	CC8405	CALL	Z,NN
CB65	BIT	4,L	3F	CCF	
CB6E	BIT	5,(HL)	BE	CP	(HL)
DDCB056E	BIT	5,(IX+IND)	DDBE05	CP	(IX+IND)
FDCB056E	BIT	5,(IY+IND)	FDBE05	CP	(IY+IND)
CB6F	BIT	5,A	BF	CP	A
CB68	BIT	5,B	B8	CP	B
CB69	BIT	5,C	B9	CP	C
CB6A	BIT	5,D	BA	CP	D
CB6B	BIT	5,E	BB	CP	E
CB6C	BIT	5,H	BC	CP	H
CB6D	BIT	5,L	BD	CP	L
CB76	BIT	6,(HL)	FE20	CP	N

操作码	源 码		操作码	源 码	
EDA9	CPD		ED58	IN	E,(C)
EDB9	CPDR		ED60	IN	H,(C)
EDA1	CPI		ED68	IN	L,(C)
EDB1	CPIR		34	INC	(HL)
2F	CPL		DD3405	INC	(IX+IND)
27	DAA		FD3405	INC	(IY+IND)
35	DEC	(HL)	3C	INC	A
DD3505	DEC	(IX+IND)	04	INC	B
FD3505	DEC	(IY+IND)	03	INC	BC
3D	DEC	A	0C	INC	C
05	DEC	B	14	INC	D
0B	DEC	BC	13	INC	DE
0D	DEC	C	1C	INC	E
15	DEC	D	24	INC	H
1B	DEC	DE	23	INC	HL
1D	DEC	E	DD23	INC	IX
25	DEC	H	FD23	INC	IY
2B	DEC	HL	2C	INC	L
DD2B	DEC	IX	33	INC	SP
FD2B	DEC	IY	EDAA	IND	
2D	DEC	L	EDBA	INDR	
3B	DEC	SP	EDA2	INI	
F3	DI		EDB2	INIR	
102E	DJNZ	DIS	E9	JP	(HL)
FB	EI		DDE9	JP	(IX)
E3	EX	(SP),HL	FDE9	JP	(IY)
DDE3	EX	(SP),IX	DA8405	JP	C,NN
FDE3	EX	(SP),IY	FA8405	JP	M,NN
08	EX	AF,AF'	D28405	JP	NC,NN
EB	EX	DE,HL	C38405	JP	NN
D9	EXX		C28405	JP	NZ,NN
76	HALT		F28405	JP	P,NN
ED46	IM	0	EA8405	JP	PE,NN
ED56	IM	1	E28405	JP	PO,NN
ED5E	IM	2	CA8405	JP	Z,NN
ED78	IN	A,(C)	382E	JR	C,DIS
DB20	IN	A,N	182E	JR	DIS
ED40	IN	B,(C)	302E	JR	NC,DIS
ED48	IN	C,(C)	202E	JR	NZ,DIS
ED50	IN	D,(C)	282E	JR	Z,DIS

操作码	源 码		操作码	源 码	
02	LD	(BC),A	78	LD	A,B
12	LD	(DE),A	79	LD	A,C
77	LD	(HL),A	7A	LD	A,D
70	LD	(HL),B	7B	LD	A,E
71	LD	(HL),C	7C	LD	A,H
72	LD	(HL),D	ED57	LD	A,1
73	LD	(HL),E	7D	LD	A,L
74	LD	(HL),H	3E20	LD	A,N
75	LD	(HL),L	46	LD	B,(HL)
3620	LD	(HL),N	DD4605	LD	B,(IX+IND)
DD7705	LD	(IX+IND),A	FD4605	LD	B,(IY+IND)
DD7005	LD	(IX+IND),B	47	LD	B,A
DD7105	LD	(IX+IND),C	40	LD	B,B
DD7205	LD	(IX+IND),D	41	LD	B,C
DD7305	LD	(IX+IND),E	42	LD	B,D
DD7405	LD	(IX+IND),H	43	LD	B,E
DD7505	LD	(IX+IND),L	44	LD	B,H
DD360520	LD	(IX+IND),N	45	LD	B,L
FD7705	LD	(IY+IND),A	0620	LD	B,N
FD7005	LD	(IY+IND),B	ED4B8405	LD	BC,(NN)
FD7105	LD	(IY+IND),C	018405	LD	BC,NN
FD7205	LD	(IY+IND),D	4E	LD	C,(HL)
FD7305	LD	(IY+IND),E	DD4E05	LD	C,(IX+IND)
FD7405	LD	(IY+IND),H	FD4E05	LD	C,(IY+IND)
FD7505	LD	(IY+IND),L	4F	LD	C,A
FD360520	LD	(IY+IND),N	48	LD	C,B
328405	LD	(NN),A	49	LD	C,C
ED438405	LD	(NN),BC	4A	LD	C,D
ED538405	LD	(NN),DE	4B	LD	C,E
228405	LD	(NN),HL	4C	LD	C,H
DD228405	LD	(NN),IX	4D	LD	C,L
FD228405	LD	(NN),IY	0E20	LD	C,N
ED738405	LD	(NN),SP	56	LD	D,(HL)
0A	LD	A,(BC)	DD5605	LD	D,(IX+IND)
1A	LD	A,(DE)	FD5605	LD	D,(IY+IND)
7E	LD	A,(HL)	57	LD	D,A
DD7E05	LD	A,(IX+IND)	50	LD	D,B
FD7E05	LD	A,(IY+IND)	51	LD	D,C
3A8405	LD	A,(NN)	52	LD	D,D
7F	LD	A,A	53	LD	D,E

操作码	源 码		操作码	源 码	
54	LD	D,H	6A	LD	L,D
55	LD	D,L	6B	LD	L,E
1620	LD	D,N	6C	LD	L,H
ED5B8405	LD	DE,(NN)	6D	LD	L,L
118405	LD	DE,NN	2E20	LD	L,N
5E	LD	E,(HL)	ED7B8405	LD	SP,(NN)
DD5E05	LD	E,(IX+IND)	F9	LD	SP,HL
FD5E05	LD	E,(IY+IND)	DDF9	LD	SP,IX
5F	LD	E,A	FDF9	LD	SP,IY
58	LD	E,B	318405	LD	SP,NN
59	LD	E,C	EDA8	LDD	
5A	LD	E,D	EDB8	LDDR	
5B	LD	E,E	EDA0	LDI	
5C	LD	E,H	EDB0	LDIR	
5D	LD	E,L	ED44	NEG	
1E20	LD	E,N	00	NOP	
66	LD	H,(HL)	B6	OR	(HL)
DD6605	LD	H,(IX+IND)	DDB605	OR	(IX+IND)
FD6605	LD	H,(IY+IND)	FDB605	OR	(IY+IND)
67	LD	H,A	B7	OR	A
60	LD	H,B	B0	OR	B
61	LD	H,C	B1	OR	C
62	LD	H,D	B2	OR	D
63	LD	H,E	B3	OR	E
64	LD	H,H	B4	OR	H
65	LD	H,L	B5	OR	L
2620	LD	H,N	F620	OR	N
2A8405	LD	HL,(NN)	EDBB	OTDR	
218405	LD	HL,NN	EDB3	OTIR	
ED47	LD	I,A	ED79	OUT	(C),A
DD2A8405	LD	IX,(NN)	ED41	OUT	(C),B
DD218405	LD	IX,NN	ED49	OUT	(C),C
FD2A8405	LD	IY,(NN)	ED51	OUT	(C),D
FD218405	LD	IY,NN	ED59	OUT	(C),E
6E	LD	L,(HL)	ED61	OUT	(C),H
DD6E05	LD	L,(IX+IND)	ED69	OUT	(C),L
FD6E05	LD	L,(IY+IND)	D320	OUT	N,A
6F	LD	L,A	EDAB	OUTD	
68	LD	L,B	EDA3	OUTI	
69	LD	L,C	F1	POP	AF

操作码	源 码		操作码	源 码	
C1	POP	BC	CB95	RES	2,L
D1	POP	DE	CB9E	RES	3,(HL)
E1	POP	HL	DDCB059E	RES	3,(IX+IND)
DDE1	POP	IX	FDCB059E	RES	3,(IY+IND)
FDE1	POP	IY	CB9F	RES	3,A
F5	PUSH	AF	CB98	RES	3,B
C5	PUSH	BC	CB99	RES	3,C
D5	PUSH	DE	CB9A	RES	3,D
E5	PUSH	HL	CB9B	RES	3,E
DDE5	PUSH	IX	CB9C	RES	3,H
FDE5	PUSH	IY	CB9D	RES	3,L
CB86	RES	0,(HL)	CBA6	RES	4,(HL)
DDCB0586	RES	0,(IX+IND)	DDCB05A6	RES	4,(IX+IND)
FDCB0586	RES	0,(IY+IND)	FDCB05A6	RES	4,(IY+IND)
CB87	RES	0,A	CBA7	RES	4,A
CB80	RES	0,B	CBA0	RES	4,B
CB81	RES	0,C	CBA1	RES	4,C
CB82	RES	0,D	CBA2	RES	4,D
CB83	RES	0,E	CBA3	RES	4,E
CB84	RES	0,H	CBA4	RES	4,H
CB85	RES	0,L	CBA5	RES	4,L
CB8E	RES	1,(HL)	CBAE	RES	5,(HL)
DDCB058E	RES	1,(IX+IND)	DDCB05AE	RES	5,(IX+IND)
FDCB058E	RES	1,(IY+IND)	FDCB05AE	RES	5,(IY+IND)
CB8F	RES	1,A	CBAF	RES	5,A
CB88	RES	1,B	CBA8	RES	5,B
CB89	RES	1,C	CBA9	RES	5,C
CB8A	RES	1,D	CBAA	RES	5,D
CB8B	RES	1,E	CBAB	RES	5,E
CB8C	RES	1,H	CBAC	RES	5,H
CB8D	RES	1,L	CBAD	RES	5,L
CB96	RES	2,(HL)	CBB6	RES	6,(HL)
DDCB0596	RES	2,(IX+IND)	DDCB05B6	RES	6,(IX+IND)
FDCB0596	RES	2,(IY+IND)	FDCB05B6	RES	6,(IY+IND)
CB97	RES	2,A	CBB7	RES	6,A
CB90	RES	2,B	CBB0	RES	6,B
CB91	RES	2,C	CBB1	RES	6,C
CB92	RES	2,D	CBB2	RES	6,D
CB93	RES	2,E	CBB3	RES	6,E
CB94	RES	2,H	CBB4	RES	6,H

操作码	源 码		操作码	源 码	
CBB5	RES	6,L	CB03	RLC	E
CBBE	RES	7,(HL)	CB04	RLC	H
DDCB05BE	RES	7,(IX+IND)	CB05	RLC	L
FDCB05BE	RES	7,(IY+IND)	07	RLCA	
CBBF	RES	7,A	ED6F	RLD	
CBB8	RES	7,B	CBIE	RR	(HL)
CBB9	RES	7,C	DDCB051E	RR	(IX+IND)
CBBA	RES	7,D	FDCB051E	RR	(IY+IND)
CBBB	RES	7,E	CB1F	RR	A
DBBO	RES	7,H	CB18	RR	B
CBBD	RES	7,L	CB19	RR	C
C9	RET		CB1A	RR	D
D8	RET	C	CB1B	RR	E
F8	RET	M	CB1C	RR	H
D0	RET	NC	CB1D	RR	L
C0	RET	NZ	1F	RRA	
F0	RET	P	CB0E	RRC	(HL)
E8	RET	PE	DDCB050E	RRC	(IX+IND)
E0	RET	PO	FDCB050E	RRC	(IY+IND)
C8	RET	Z	CB0F	RRC	A
ED4D	RETI		CB08	RRC	B
ED45	RETN		CB09	RRC	C
CB16	RL	(HL)	CB0A	RRC	D
DDCB0516	RL	(IX+IND)	CB0B	RRC	E
FDCB0516	RL	(IY+IND)	CB0C	RRC	H
CB17	RL	A	CB0D	RRC	L
CB10	RL	B	0F	RRCA	
CB11	RL	C	ED67	RRD	
CB12	RL	D	C7	RST	0
CB13	RL	E	D7	RST	10H
CB14	RL	H	DF	RST	18H
CB15	RL	L	E7	RST	20H
17	RLA		EF	RST	28H
CB06	RLC	(HL)	F7	RST	30H
DDCB0506	RLC	(IX+IND)	FF	RST	38H
FDCB0506	RLC	(IY+IND)	CF	RST	8
CB07	RLC	A	9E	SBC	A,(HL)
CB00	RLC	B	DD9E05	SBC	A,(IX+IND)
CB01	RLC	C	FD9E05	SBC	A,(IY+IND)
CB02	RLC	D	9F	SBC	A,A

操作码	源 码		操作码	源 码	
98	SBC	A,B	CBD4	SET	2,H
99	SBC	A,C	CBD5	SET	2,L
9A	SBC	A,D	CBD8	SET	3,B
9B	SBC	A,E	CBDE	SET	3,(HL)
9C	SBC	A,H	DDCB05DE	SET	3,(IX+IND)
9D	SBC	A,L	FDCB05DE	SET	3,(IY+IND)
DE20	SBC	A,N	CBDF	SET	3,A
ED42	SBC	HL,BC	CBD9	SET	3,C
ED52	SBC	HL,DE	CBDA	SET	3,D
ED62	SBC	HL,HL	CBDB	SET	3,E
ED72	SBC	HL,SP	CBDC	SET	3,H
37	SCF		CBDD	SET	3,L
CBC6	SET	0,(HL)	CBE6	SET	4,(HL)
DDCB05C6	SET	0,(IX+IND)	DDCB05E6	SET	4,(IX+IND)
FDCB05C6	SET	0,(IY+IND)	FDCB05E6	SET	4,(IY+IND)
CBC7	SET	0,A	CBE7	SET	4,A
CBC0	SET	0,B	CBE0	SET	4,B
CBC1	SET	0,C	CBE1	SET	4,C
CBC2	SET	0,D	CBE2	SET	4,D
CBC3	SET	0.E	CBE3	SET	4,E
CBC4	SET	0,H	CBE4	SET	4,H
CBC5	SET	0,L	CBE5	SET	4,L
CBCE	SET	1,(HL)	CBEE	SET	5,(HL)
DDCB05CE	SET	1,(IX+IND)	DDCB05EE	SET	5,(IX+IND)
FDCB05CE	SET	1,(IY+IND)	FDCB05EE	SET	5,(IY+IND)
CBCF	SET	1,A	CBEF	SET	5,A
CBC8	SET	1,B	CBE8	SET	5,B
CBC9	SET	1,C	CBE9	SET	5,C
CBCA	SET	1,D	CBEA	SET	5,D
CBCB	SET	1,E	CBEB	SET	5,E
CBCC	SET	1,H	CBEC	SET	5,H
CBCD	SET	1,L	CBED	SET	5,L
CBD6	SET	2,(HL)	CBF6	SET	6,(HL)
DDCB05D6	SET	2,(IX+IND)	DDCB05F6	SET	6,(IX+IND)
FDCB05D6	SET	2,(IY+IND)	FDCB05F6	SET	6,(IY+IND)
CBD7	SET	2,A	CBF7	SET	6,A
CBD0	SET	2,B	CBF0	SET	6,B
CBD1	SET	2,C	CBF1	SET	6,C
CBD2	SET	2,D	CBF2	SET	6,D
CBD3	SET	2,E	CBF3	SET	6,E

操作码	源 码		操作码	源 码	
CBF4	SET	6,H	CB3F	SRL	A
CBF5	SET	6,L	CB38	SRL	B
CBFE	SET	7,(HL)	CB39	SRL	C
DDCB05FE	SET	7,(IX+IND)	CB3A	SRL	D
FDCB05FE	SET	7,(IY+IND)	CB3B	SRL	E
CBFF	SET	7,A	CB3C	SRL	H
CBF8	SET	7,B	CB3D	SRL	L
CBF9	SET	7,C	96	SUB	(HL)
CBFA	SET	7,D	DD9605	SUB	(IX+IND)
CBFB	SET	7,E	FD9605	SUB	(IY+IND)
CBFC	SET	7,H	97	SUB	A
CBFD	SET	7,L	90	SUB	B
CB26	SLA	(HL)	91	SUB	C
DDCB0526	SLA	(IX+IND)	92	SUB	D
FDCB0526	SLA	(IY+IND)	93	SUB	E
CB27	SLA	A	94	SUB	H
CB20	SLA	B	95	SUB	L
CB21	SLA	C	D620	SUB	N
CB22	SLA	D	AE	XOR	(HL)
CB23	SLA	E	DDAE05	XOR	(IX+IND)
CB24	SLA	H	FDAE05	XOR	(IY+IND)
CB25	SLA	L	AF	XOR	A
CB2E	SRA	(HL)	A8	XOR	B
DDCB052E	SRA	(IX+IND)	A9	XOR	C
FDCB052E	SRA	(IY+IND)	AA	XOR	D
CB2F	SRA	A	AB	XOR	E
CB28	SRA	B	AC	XOR	H
CB29	SRA	C	AD	XOR	L
CB2A	SRA	D	EE20	XOR	N
CB2B	SRA	E	NN	DEFS	2
CB2C	SRA	H	IND	EQU	5
CB2D	SRA	L	M	EQU	10H
CB3E	SRL	(HL)	N	EQU	20H
DDCB053E	SRL	(IX+IND)	DIS	EQU	30H
FDCB053E	SRL	(IY+IND)		END	

2. 按操作码顺序排列的 Z-80 指令表

操作码	源 码		操作码	源 码	
00	NOP		27	DAA	
018405	LD	BC,NN	282E	JR	Z,DIS
02	LD	(BC),A	29	ADD	HL,HL
03	INC	BC	2A8405	LD	HL,(NN)
04	INC	B	2B	DEC	HL
05	DEC	B	2C	INC	L
0620	LD	B,N	2D	DEC	L
07	RLCA		2E20	LD	L,N
08	EX	AF,AF'	2F	CPL	
09	ADD	HL,BC	302E	JR	NC,DIS
0A	LD	A,(BC)	318405	LD	SP,NN
0B	DEC	BC	328405	LD	(NN),A
0C	INC	C	33	INC	SP
0D	DEC	C	34	INC	(HL)
0E20	LD	C,N	35	DEC	(HL)
0F	RRCA		3620	LD	(HL),N
102E	DJNZ	DIS	37	SCF	
118405	LD	DE,NN	382E	JR	C,DIS
12	LD	(DE),A	39	ADD	HL,SP
13	INC	DE	3A8405	LD	A,(NN)
14	INC	D	3B	DEC	SP
15	DEC	D	3C	INC	A
1620	LD	D,N	3D	DEC	A
17	RLA		3E20	LD	A,N
182E	JR	DIS	3F	CCF	
19	ADD	HL,DE	40	LD	B,B
1A	LD	A,(DE)	41	LD	B,C
1B	DEC	DE	42	LD	B,D
1C	INC	E	43	LD	B,E
1D	DEC	E	44	LD	B,H
1E20	LD	E,N	45	LD	B,L
1F	RRA		46	LD	B,(HL)
202E	JR	NZ,DIS	47	LD	B,A
218405	LD	HL,NN	48	LD	C,B
228405	LD	(NN),HL	49	LD	C,C
23	INC	HL	4A	LD	C,D
24	INC	H	4B	LD	C,E
25	DEC	H	4C	LD	C,H
2620	LD	H,N	4D	LD	C,L

操作码	源 码		操作码	源 码	
4E	LD	C,(HL)	76	HALT	
4F	LD	C,A	77	LD	(HL),A
50	LD	D,B	78	LD	A,B
51	LD	D,C	79	LD	A,C
52	LD	D,D	7A	LD	A,D
53	LD	D,E	7B	LD	A,E
54	LD	D,H	7C	LD	A,H
55	LD	D,L	7D	LD	A,L
56	LD	D,(HL)	7E	LD	A,(HL)
57	LD	D,A	7F	LD	A,A
58	LD	E,B	80	ADD	A,B
59	LD	E,C	81	ADD	A,C
5A	LD	E,D	82	ADD	A,D
5B	LD	E·E	83	ADD	A,E
5C	LD	E,H	84	ADD	A,H
5D	LD	E,L	85	ADD	A,L
5E	LD	E,(HL)	86	ADD	A,(HL)
5F	LD	E,A	87	ADD	A,A
60	LD	H,B	88	ADC	A,B
61	LD	H,C	89	ADC	A,C
62	LD	H,D	8A	ADC	A,D
63	LD	H,E	8B	ADC	A,E
64	LD	H,H	8C	ADC	A,H
65	LD	H,L	8D	ADC	A,L
66	LD	H,(HL)	8E	ADC	A,(HL)
67	LD	H,A	8F	ADC	A,A
68	LD	L,B	90	SUB	B
69	LD	L,C	91	SUB	C
6A	LD	B,D	92	SUB	D
6B	LD	L,E	93	SUB	E
6C	LD	L,H	94	SUB	H
6D	LD	L,L	95	SUB	L
6E	LD	L,(HL)	96	SUB	(HL)
6F	LD	L,A	97	SUB	A
70	LD	(HL),B	68	SBC	A,B
71	LD	(HL),C	99	SBC	A,C
72	LD	(HL),D	9A	SBC	A,D
73	LD	(HL),E	9B	SBC	A,E
74	LD	(HL),H	9C	SBC	A,H
75	LD	(HL),L	9D	SBC	A,L

操作码	源 码		操作码	源 码	
9E	SBC	A,(HL)	C620	ADD	A,N
9F	SBC	A,A	C7	RST	0
A0	AND	B	C8	RET	Z
A1	AND	C	C9	RET	
A2	AND	D	CA8405	JP	Z,NN
A3	AND	E	CC8405	CALL	Z,NN
A4	AND	H	CD8405	CALL	NN
A5	AND	L	CE20	ADC	A,N
A6	AND	(HL)	CF	RST	8
A7	AND	A	D0	RET	NC
A8	XOR	B	D1	POP	DE
A9	XOR	C	D28405	JP	NC,NN
AA	XOR	D	D320	OUT	N,A
AB	XOR	E	D48405	CALL	NC,NN
AC	XOR	H	D5	PUSH	DE
AD	XOR	L	D620	SUB	N
AE	XOR	(HL)	D7	RST	10H
AF	XOR	A	D8	RET	C
B0	OR	B	D9	EXX	
B1	OR	C	DA8405	JP	C,NN
B2	OR	D	DB20	IN	A,N
B3	OR	E	DC8405	CALL	C,NN
B4	OR	H	DE20	SBC	A,N
B5	OR	L	DF	RST	18H
B6	OR	(HL)	E0	RET	PO
B7	OR	A	E1	POP	HL
B8	CP	B	E28405	JP	PO,NN
B9	CP	C	E3	EX	(SP),HL
BA	CP	D	E48405	CALL	PO,NN
BB	CP	E	E5	PUSH	HL
BC	CP	H	E620	AND	N
BD	CP	L	E7	RST	20H
BE	CP	(HL)	E8	RET	PE
BF	CP	A	E9	JP	(HL)
C0	RET	NZ	EA8405	JP	PE,NN
C1	POP	BC	EB	EX	DE,HL
C28405	JP	NZ,NN	EC8405	CALL	PE,NN
C38405	JP	NN	EE20	XOR	N
C48405	CALL	NZ,NN	EF	RST	28H
C5	PUSH	BC	F0	REN	P

操作码	源 码		操作码	源 码	
F1	POP	AF	CB1A	RR	D
F28405	JP	P,NN	CB1B	RR	E
F3	DI		CB1C	RR	H
F48405	CALL	P,NN	CB1D	RR	L
F5	PUSH	AF	CB1E	RR	(HL)
F620	OR	N	CB1F	RR	A
F7	RST	30H	CB20	SLA	B
F8	RET	M	CB21	SLA	C
F9	LD	SP,HL	CB22	SLA	D
FA8405	JP	M,NN	CB23	SLA	E
FB	EI		CB24	SLA	H
FC8405	CALL	\M,NN	CB25	SLA	L
FE20	CP	N	CB26	SLA	(HL)
FF	RST	38H	CB27	SLA	A
CB00	RLC	B	CB28	SRA	B
CB01	RLC	C	CB29	SRA	C
CB02	RLC	D	CB2A	SRA	D
CB03	RLC	E	CB2B	SRA	E
CB04	RLC	H	CB2C	SRA	H
CB05	RLC	L	CB2D	SRA	L
CB06	RLC	(HL)	CB2E	SRA	(HL)
CB07	RLC	A	CB2F	SRA	A
CB08	RRC	B	CB38	SRL	B
CB09	RRC	C	CB39	SRL	C
CB0A	RRC	D	CB3A	SRL	D
CB0B	RRC	E	CB3B	SRL	E
CB0C	RRC	H	CB3C	SRL	H
CB0D	RRC	L	CB3D	SRL	L
CB0E	RRC	(HL)	CB3E	SRL	(HL)
CB0F	RRC	A	CB3F	SRL	A
CB10	RL	B	CB40	BIT	0,B
CB11	RL	C	CB41	BIT	0,C
CB12	RL	D	CB42	BIT	0,D
CB13	RL	E	CB43	BIT	0,E
CB14	RL	H	CB44	BIT	0,H
CB15	RL	L	CB45	BIT	0,L
CB16	RL	(HL)	CB46	BIT	0,(HL)
CB17	RL	A	CB47	BIT	0,A
CB18	RR	B	CB48	BIT	1,B
CB19	RR	C	CB49	BIT	1,C

操作码	源 码		操作码	源 码	
CB4A	BIT	1,D	CB72	BIT	6,D
CB4B	BIT	1,E	CB73	BIT	6,E
CB4C	BIT	1,H	CB74	BIT	6,H
CB4D	BIT	1,L	CB75	BIT	6,L
CB4E	BIT	1,(HL)	CB76	BIT	6,(HL)
CB4F	BIT	1,A	CB77	BIT	6,A
CB50	BIT	2,B	CB78	BIT	7,B
CB51	BIT	2,C	CB79	BIT	7,C
CB52	BIT	2,D	CB7A	BIT	7,D
CB53	BIT	2,E	CB7B	BIT	7,E
CB54	BIT	2,H	CB7C	BIT	7,H
CB55	BIT	2,L	CB7D	BIT	7,L
CB56	BIT	2,(HL)	CB7E	BIT	7,(HL)
CB57	BIT	2,A	CB7F	BIT	7,A
CB58	BIT	3,B	CB80	RES	0,B
CB59	BIT	3,C	CB81	RES	0,C
CB5A	BIT	3,D	CB82	RES	0,D
CB5B	BIT	3,E	CB83	RES	0,E
CB5C	BIT	3,H	CB84	RES	0,H
CB5D	BIT	3,L	CB85	RES	0,L
CB5E	BIT	3,(HL)	CB86	RES	0,(HL)
CB5F	BIT	3,A	CB87	RES	0,A
CB60	BIT	4,B	CB88	RES	1,B
CB61	BIT	4,C	CB89	RES	1,C
CB62	BIT	4,D	CB8A	RES	1,D
CB63	BIT	4,E	CB8B	RES	1,E
CB64	BIT	4,H	CB8C	RES	1,H
CB65	BIT	4,L	CB8D	RES	1,L
CB66	BIT	4,(HL)	CB8E	RES	1,(HL)
CB67	BIT	4,A	CB8F	RES	1,A
CB68	BIT	5,B	CB90	RES	2,B
CB69	BIT	5,C	CB91	RES	2,C
CB6A	BIT	5,D	CB92	RES	2,D
CB6B	BIT	5,E	CB93	RES	2,E
CB6C	BIT	5,H	CB94	RES	2,H
CB6D	BIT	5,L	CB95	RES	2,L
CB6F	BIT	5,(HL)	CB96	RES	2,(HL)
CB6F	BIT	5,A	CB97	RES	2,A
CB70	BIT	6,B	CB98	RES	3,B
CB71	BIT	6,C	CB99	RES	3,C

操作码	源 码		操作码	源 码	
CB9A	RES	3,D	CBC2	SET	0,D
CB9B	RES	3,E	CBC3	SET	0,E
CB9C	RES	3,H	CBC4	SET	0,H
CB9D	RES	3,L	CBC5	SET	0,L
CB9E	RES	3,(HL)	CBC6	SET	0,(HL)
CB9F	RES	3,A	CBC7	SET	0,A
CBA0	RES	4,B	CBC8	SET	1,B
CBA1	RES	4,C	CBC9	SET	1,C
CBA2	RES	4,D	CBCA	SET	1,D
CBA3	RES	4,E	CBCB	SET	1,E
CBA4	RES	4,H	CBCC	SET	1,H
CBA5	RES	4,L	CBCD	SET	1,L
CBA6	RES	4,(HL)	CBCE	SET	1,(HL)
CBA7	RES	4,A	CBCF	SET	1,A
CBA8	RES	5,B	CBD0	SET	2,B
CBA9	RES	5,C	CBD1	SET	2,C
CBAA	RES	5,D	CBD2	SET	2,D
CBAB	RES	5,E	CBD3	SET	2,E
CBAC	RES	5,H	CBD4	SEL	2,H
CBAD	RES	5,L	CBD5	SET	2,L
CBAE	RES	5,(HL)	CBD6	SET	2,(HL)
CBAF	RES	5,A	CBD7	SET	2.A
CBB0	RES	6,B	CBD8	SET	3,B
CBB1	RES	6,C	CBD9	SET	3,C
CBB2	RES	6,D	CBDA	SET	3,D
CBB3	RES	6,E	CBDB	SET	3,E
CBB4	RES	6,H	CBDC	SET	3,H
CBB5	RES	6,L	CBDD	SET	3,L
CBB6	RES	6,(HL)	CBDE	SET	3,(HL)
CBB7	RES	6,A	CBDF	SET	3,A
CBB8	RES	7,B	CBE0	SET	4,B
CBB9	RES	7,C	CBE1	SET	4,C
CBBA	RES	7,D	CBE2	SET	4,D
CBBB	RES	7,E	CBE3	SET	4,E
CBBC	RES	7,H	CBE4	SET	4,H
CBBD	RES	7,L	CBE5	SET	4,L
CBBE	RES	7,(HL)	CBE6	SET	4,(HL)
CBBF	RES	7,A	CBE7	SET	4,A
CBC0	SET	0,B	CBE8	SET	5,B
CBC1	SET	0,C	CBE9	SET	5,C

操作码	源 码		操作码	源 码	
CBEA	SET	5,D	DD7005	LD	(IX+IND),B
CBEB	SET	5,E	DD7105	LD	(IX+IND),C
CBEC	SET	5,H	DD7205	LD	(IX+IND),D
CBED	SET	5,L	DD7305	LD	(IX+IND),E
CBEE	SET	5,(HL)	DD7405	LD	(IX+IND),H
CBEF	SET	5,A	DD7505	LD	(IX+IND),L
CBF0	SET	6,B	DD7705	LD	(IX+IND),A
CBF1	SET	6,C	DD7E05	LD	A,(IX+IND)
CBF2	SET	6,D	DD8605	ADD	A,(IX+IND)
CBF3	SET	6,E	DD8E05	ADC	A,(IX+IND)
CBF4	SET	6,H	DD9605	SUB	(IX+IND)
CBF5	SET	6,L	DD9E05	SBC	A,(IX+IND)
CBF6	SET	6,(HL)	DDA605	AND	(IX+IND)
CBF7	SET	6,A	DDAE05	XOR	(IX+IND)
CBF8	SET	7,B	DDB605	OR	(IX+IND)
CBF9	SET	7,C	DDBE05	CP	(IX+IND)
CBFA	SET	7,D	DDE1	POP	IX
CBFB	SET	7,E	DDE3	EX	(SP),IX
CBFC	SET	7,H	DDE5	PUSH	IX
CBFD	SET	7,L	DDE9	JP	(IX)
CBFE	SET	7,(HL)	DDF9	LD	SP,IX
CBFF	SET	7,A	DDCB0506	RLC	(IX+IND)
DD09	ADD	IX,BC	DDCB050E	RRC	(IX+IND)
DD19	ADD	IX,DE	DDCB0516	RL	(IX+IND)
DD218405	LD	IX,NN	DDCB051E	RR	(IX+IND)
DD228405	LD	(NN),IX	DDCB0526	SLA	(IX+IND)
DD23	INC	IX	DDCB052E	SRA	(IX+IND)
DD29	ADD	IX,IX	DDCB053E	SRL	(IX+IND)
DD2A8405	LD	IX,(NN)	DDCB0546	BIT	0,(IX+IND)
DD2B	DEC	IX	DDCB054E	BIT	1,(IX+IND)
DD3405	INC	(IX+IND)	DDCB0556	BIT	2,(IX+IND)
DD3505	DEC	(IX+IND)	DDCB055E	BIT	3,(IX+IND)
DD360520	LD	(IX+IND),N	DDCB0566	BIT	4,(IX+IND)
DD39	ADD	IX,SP	DDCB056E	BIT	5,(IX+IND)
DD4605	LD	B,(IX+IND)	DDCB0576	BIT	6,(IX+IND)
DD4E05	LD	C,(IX+IND)	DDCB057E	BIT	7,(IX+IND)
DD5605	LD	D,(IX+IND)	DDCB0586	RES	0,(IX+IND)
DD5E05	LD	E,(IX+IND)	DDCB058E	RES	1,(IX+IND)
DD6605	LD	H,(IX+IND)	DDCB0596	RES	2,(IX+IND)
DD6E05	LD	L,(IX+IND)	DDCB059E	RES	3,(IX+IND)

操作码	源 码		操作码	源 码	
DDCB05A6	RES	4,(IX+IND)	ED68	IN	L,(C)
DDCB05AE	RES	5,(IX+IND)	ED69	OUT	(C),L
DDCB05B6	RES	6,(IX+IND)	ED6A	ADC	HL,HL
DDCB05BE	RES	7,(IX+IND)	ED6F	RLD	
DDCB05C6	SET	0,(IX+IND)	ED72	SBC	HL,SP
DDCB05CE	SET	1,(IX+IND)	ED738405	LD	(NN),SP
DDCB05D6	SET	2,(IX+IND)	ED78	IN	A,(C)
DDCB05DE	SET	3,(IX+IND)	ED79	OUT	(C),A
DDCB05E6	SET	4,(IX+IND)	ED7A	ADC	HL,SP
DDCB05EE	SET	5,(IX+IND)	ED7B8405	LD	SP,(NN)
DDCB05F6	SET	6,(IX+IND)	EDA0	LDI	
DDCB05FE	SET	7,(IX+IND)	EDA1	CPI	
ED40	IN	B,(C)	EDA2	INI	
ED41	OUT	(C),B	EDA3	OUTI	
ED42	SBC	HL,BC	EDA8	LDD	
ED438405	LD	(NN),BC	EDA9	CPD	
ED44	NEG		EDAA	IND	
ED45	RETN		EDAB	OUTD	
ED46	IM	0	EDB0	LDIR	
ED47	LD	I,A	EDB1	CPIR	
ED48	IN	C,(C)	EDB2	INIR	
ED49	OUT	(C),C	EDB3	OTIR	
ED4A	ADC	HL,BC	EDB8	LDDR	
ED4B8405	LD	BC,(NN)	EDB9	CPDR	
ED4D	RETI		EDBA	INDR	
ED50	IN	D,(C)	EDBB	OTDR	
ED51	OUT	(C),D	FD09	ADD	IY,BC
ED52	SBC	HL,DE	FD19	ADD	IY,DE
ED538405	LD	(NN),DE	FD218405	LD	IY,NN
ED56	IM	I	FD228405	LD	(NN),IY
ED57	LD	A,I	FD23	INC	IY
ED58	IN	E,(C)	FD29	ADD	IY,IY
ED59	OUT	(C),E	FD2A8405	LD	IY,(NN)
ED5A	ADC	HL,DE	FD2B	DEC	IY
ED5B8405	LD	DE,(NN)	FD3405	INC	(IY+IND)
ED5E	IM	2	FD3505	DEC	(IY+IND)
ED60	IN	H,(C)	FD360520	LD	(IY+IND),N
ED61	OUT	(C),H	FD39	ADD	IY,SP
ED62	SBC	HL,HL	FD4605	LD	B,(IY+IND)
ED67	RRD				

操作码	源 码		操作码	源 码	
FD4E05	LD	C,(IY+IND)	FDCB052E	SRA	(IY+IND)
FD5605	LD	D,(IY+IND)	FDCB053E	SRL	(IY+IND)
FD5E05	LD	E,(IY+IND)	FDCB0546	BIT	0,(IY+IND)
FD6605	LD	H,(IY+IND)	FDCB054E	BIT	1,(IY+IND)
FD6E05	LD	L,(IY+IND)	FDCB0556	BIT	2,(IY+IND)
FD7005	LD	(IY+IND),B	FDCB055E	BIT	3,(IY+IND)
FD7105	LD	(IY+IND),C	FDCB0566	BIT	4,(IY+IND)
FD7205	LD	(IY+IND),D	FDCB056E	BIT	5,(IY+IND)
FD7305	LD	(IY+IND),E	FDCB0576	BIT	6,(IY+IND)
FD7405	LD	(IY+IND),H	FDCB057E	BIT	7,(IY+IND)
FD7505	LD	(IY+IND),L	FDCB0586	RES	0,(IY+IND)
FD7705	LD	(IY+IND),A	FDCB058E	RES	1,(IY+IND)
FD7E05	LD	A,(IY+IND)	FDCB0596	RES	2,(IY+IND)
FD8605	ADD	A,(IY+IND)	FDCB059E	RES	3,(IY+IND)
FD8E05	ADC	A,(IY+IND)	FDCB05A6	RES	4,(IY+IND)
FD9605	SUB	(IY+IND)	FDCB05AE	RES	5,(IY+IND)
FD9E05	SBC	A,(IY+IND)	FDCB05B6	RES	6,(IY+IND)
FDA605	AND	(IY+IND)	FDCB05BE	RES	7,(IY+IND)
FDAE05	XOR	(IY+IND)	FDCB05C6	SET	0,(IY+IND)
FDB605	OR	(IY+IND)	FDCB05CE	SET	1,(IY+IND)
FDBE05	CP	(IY+IND)	FDCB05D6	SET	2,(IY+IND)
FDE1	POP	IY	FDCB05DE	SET	3,(IY+IND)
FDE3	EX	(SP),IY	FDCB05E6	SET	4,(IY+IND)
FDE5	PUSH	IY	FDCB05EE	SET	5,(IY+IND)
FDE9	JP	(IY)	FDCB05F6	SET	6,(IY+IND)
FDF9	LD	SP,IY	FDCB05FE	SET	7,(IY+IND)
FDCB0506	RLC	(IY+IND)	NN	DEFS	2
FDCB050E	RRC	(IY+IND)	IND	EQU	5
FDCB0516	RL	(IY+IND)	M	EQU	10H
FDCB051E	RR	(IY+IND)	N	EQU	20H
FDCB0526	SLA	(IY+IND)	DIS	EQU	30H

3. TRS-80LEVEL I 存储器分配表

地址 十进制	十六进制	内　容
0	0000	LEVEL I BASIC ROM
4095	0FFF	LEVEL I BASIC ROM
4096	1000	不　用
14335	37FF	
14336	3800	键盘存储器
15359	3BFF	
15360	3C00	显示器存储器
16383	3FFF	

存储器的一些起点

16384	4000	Begin LEVEL I Scratchpad Storage
16488	4068	Pointer to Current Cursor Position(LSB)
16489	4069	Pointer to Current Cursor Position(MSB)
16490	406A	Pointer to End of RAM Memory(LSB)
16491	406B	Pointer to End of RAM Memory(MSB)
16492	406C	Pointer to Beginning of Free MEM(LSB)Preset to
16493	406D	Pointer to Beginning of Free MEM(MSB) 42001
16528	4090	I/O Status Byte
16894	41FF	End of LEVEL I Scratchpad Storage
16895	4200	BASIC Program Storage
20479	4FFF	End of 4K Memory
32767	7FFF	End of 16K Memory
49151	BFFF	End of 32K Memory
65535	FFFF	End of 48K Memory

4. LEVEL Ⅰ 存储器分配表

地址 十进制	十六进制		内　容
0	0000		
			LEVEL Ⅱ BASIC ROM
12288	3000		
			Reserved
14302		37DE	Communication Status Address
14303		37DF	Communication Data Address
14304		37E0	Interrupt Latch Address
14305		37E1	Disk Drive Select Latch Address
14308		37E4	Cassette Select Latch Address
14312		37E8	Line Printer Address
14316		37EC	Floppy Disk Controller Address
14336	3800		
			TRS-80 Keyboard Memory
15360	3C00		
			TRS-80 CRT Video Memory
16383	3FFF		
16384	4000		
			LEVEL Ⅱ BASIC Fixed RAM
			Vectors(RST'S 1 Through 7)
16402		4012	
16405		4015	Keyboard Device Control Block
			DCB + 0 = DCB Type + 1 = Driver Address + 2 = Driver Address + 3 = 0 + 4 = 0 + 5 = 0 + 6 = 'K' + 7 = 'I'
16413		401D	Video Display Control Block
			DCB + 0 = DCB Type + 1 = Driver Address(LSB) + 2 = Driver Address(MSB) + 3 = Cursor POS N(LSB) + 4 = Cursor POS N(MSB) + 5 = Cursor Character + 6 = 'D' + 7 = 'O'
16421	4025		

地址 十进制	十六进制		内　容
		——	Line printer Control Block
			DCB+0=DCB Type +1=Driver Address(LSB) +2=Driver Address(MSB) +3=Line/Page +4=Line Counter +5=0 +6='P' +7='R'
16429	402D	——	
			Reserved
16463	404F	——	
16464	4050	——	FDC Interrupt Vector
16466	4052	——	Communications Interrupt Vector
16468	4054	——	Reserved
16476	405C	——	
16478	405E	——	25MSec Heartbeat Interrupt
16512	4080		Reserved
			LEVEL II BASIC Free RAM
			Reserved
16870	41E6	——	
			I/O Buffer
17127	42E7	——	
17128	42E8	——	Always Zero
17129	42E9	——	
			↓ Program Text
			↓ Simple Variables
			↓ Arrays
			↓ String Variable Names and Overhead
			Free Memory
			↑ Stack
			↑ String Space
			Space Reserved For Machine Language Routines–If Memory Size Set,d uring Initialization Dialog
		——	
20479	4FFF		End of 4K Memory
32767	7FFF		End of 16K Memory
49151	BFFF		End of 32K Memory
65535	FFFF		End of 48K Memory

5. 只读存储器地址和子程序

Level I Basic 地　址

键盘扫描	WAIT	CALL 0B40H	;SCAN
		JR Z,WAIT	;Z=1 IF KB CLEAR
在光标处		PUSH DE	;MUST SAVE
显示字节		PUSH IY	; DE & IY
		LD A,20H	;BYTE TO DISPLAY
		RST 10H	;DISPLAY BYTE
		POP IY	;RESTORE
		POP DE	; DE & IY
开盒式机		CALL 0FE9H	;TURN ON CASSETTE
电　源		(On board cassette is turned on via remote plug)	
内存内容		CALL 0FE9H	;TURN ON CASSETTE
录入磁带		LD HL,7000H	;START ADDRESS
		LD DE,7100H	;LAST+1 ADDRESS
		CALL 0F4BH	;SAVE IT
由磁带		CALL 0EF4H	;TURN ON & READ
存入内存			
返　回		Press Reset	
LEVEL I BASIC		JP 0	;POWER UP
		JP 01C9H	;RE-ENTRY WITH READY
返　回 TBUG		Set a Breakpoint to next opcode address	
		JP 40B1H	;RE-ENTRY TBUG

Level II Basic 地　址

显　示		PUSH DE	;MUST SAVE
光标字符		PUSH IY	; DE & IY
		LD A,0EH	0EH IS CURSOR BYTE
		CALL 33H	;DISPLAY ROUTINE
		POP IY	;RESTORE
		POP DE	; DE & IY
键盘扫描	AGN	PUSH DE	;MUST SAVE
		PUSH IY	; DE & IY

```
                CALL    2BH             ;SCAN ROUTINE
                OR      A               ;A=0 IF KB CLEAR
                JR      Z,AGN           ;BRANCH IF NO BYTE
                POP     IY              ;RESTORE
                POP     DE              ;   DE & IY

在光标处        PUSH    DE              ;MUST SAVE
显示字节        PUSH    IY              ;   DE & IY
                LD      A,20H           ;BYTE TO DISPLAY
                CALL    33H             ;DISPLAY
                POP     IY              ;RESTORE
                POP     DE              ;   DE & IY
              ;A-REGISTER SPECIFIES
                CASSETTE(0 OR 1)
开与主机相连的  LD      A,0             ;ON BOARD CASSETTE
盒  式  机      CALL    0212H           ;DEFINE DRIVE

写引导码和
同步字节        CALL    0278H

关盒式机        CALL    01F8H
电  源

内存内容        LD      A,0             ;ON BOARD CASSETTE
录入磁带        CALL    0212H           ;DEFINE DRIVE
                CALL    0287H           ;WRITE LEADER
                LD      A,20H           ;BYTE TO RECORD
                CALL    0264H           ;OUTPUT BYTE
                CALL    01F8H           ;CASSETTE OFF

寻找引导码      CALL    0296H
和同步字节

从磁带送        LD      A,0
入内存          CALL    0212H           ;DEFINE DRIVE
                CALL    0296H           ;FIND SYNC BYTE
                CALL    0235H           ;READ ONE BYTE

返  回          Press   RESET
LEVEL Ⅱ BASIC   JP      0               ;LIKE POWER UP
                JP      1A19H           ;RE-ENTRY

在LEVEL Ⅱ BASIC
管理下，返回TBUG                        et a Breakpoint to next opcode address.
                JP      43A0H                   ;RE-ENTER TBUG
```

输出到行
式打印机
（只有 LEVEL Ⅱ 才有）

```
PRTOUT EXX                      ;SAVE REGS.
       LD  HL,47E8H             ;LOAD LP POINTER IN HL
PRTLP8 LD  D,(HL)               ;LOAD LP STATUS BYTE
       BIT 7,D                  ;IS THE PRINTER BUSY?
       JP  NZ,PRTLP8
       LD  (HL),A               ;OUTPUT BYTE TO
                                  PRINTER
       EXX
       RET
```

;PUT ASCII BYTE IN

;A-REGISTER AND
  CALL PRTOUT
;BUSY CONDITION
  TESTED FOR

6. 基　本　变　换

二进制	十六进制	十进制	二进制	十六进制	十进制
00000000	00	0	01001000	48	72
00000001	01	1	01010000	50	80
00000010	02	2	01011000	58	88
00000011	03	3	01100000	60	96
00000100	04	4	01101000	68	104
00000101	05	5	01110000	70	112
00000110	06	6	01111000	78	120
00000111	07	7	10000000	80	128
00001000	08	8	10001000	88	136
00001001	09	9	10010000	90	144
00001010	0A	10	10011000	98	152
00001011	0B	11	10100000	A0	160
00001100	0C	12	10101000	A8	168
00001101	0D	13	10110000	B0	176
00001110	0E	14	10111000	B8	184
00001111	0F	15	11000000	C0	192
00010000	10	16	11001000	C8	200
00011000	18	24	11010000	D0	208
00100000	20	32	11011000	D8	216
00101000	28	40	11100000	E0	224
00110000	30	48	11101000	E8	232
00111000	38	56	11110000	F0	240
01000000	40	64	11111000	F8	248
			11111100	FC	252
			11111110	FE	254
			11111111	FF	255

7. 几 本 参 考 书

The Z-80 Microcomputer Handbook, by William Barden, Jr. Howard W .Sams & Co.,Inc.

Z-80 Assembly Language Programming. A Zilog, inc., publication

Z-80 Programming for Logic Design, by Adam Osborne et al. Osborne & Associates, inc. Berkeley CA

（梁祖威译、陶笃纯校）

第二部分硬件部分

第一篇 TRS-80硬件手册

一、序 言

TRS-80作为一个微型计算机系统，很早就已经开始出售。这种产品不仅在美国国内，就是在其它许多国家都已成为畅销品。之所以如此，是基于它具有优越的硬件结构和丰富多变的软件操作，而特别是使用了BASIC Ⅰ这一语言形式，它的使用简单易懂；程序编制容易可靠等优点，已受到广泛的重视。

TRS-80微型计算机所使用的是Z80部件。一般来说，要想使微型计算机具有广泛的用途，如像TRS-80那样的机器作为个人计算机来使用的话，在它的外围就必须连接相当多的硬件设备。

到目前为止，作为个人计算机已有许多种类型，但是，有关以微处理器为主的外围硬件设备的详细说明书，实际上几乎没有。

本书是为使用者提供关于微型计算机的硬件方面的知识而写的。在书中对有关TRS-80的硬件的各个部分的详细工作原理都作了说明。并且还介绍了在TRS-80中发生某种故障时，如何查找故障的原因和排除故障的方法。关于查找和排除故障的方法，这不仅是对TRS-80机，而对其它类似相同的硬件设备也都是适用的。

同时，为了让使用者能够广泛地利用TRS-80机，在这里也提供了简单的控制电源电路的硬件设计例子。

TRS-80是使用功能较强的BASIC Ⅰ语言，在此基础上再附加上一些外部接口的硬件设备，就能够完成很多种工作。

本书在讲述个人计算机TRS-80硬件设备的同时，还进一步为使用者提供了广泛地应用TRS-80机的多种途径。

二、系统的结构与工作原理

1. TRS-80的系统结构

TRS-80微型计算机所使用的是Z80器件。整体的系统结构图参看图1。由图1可以看出，TRS-80是分别由10个主要部分所组成的。

中央处理单元即CPU(Central Processing Unit,Z80)，是最核心的部件，是实现运算、控制等的部件。从CPU里分别引出8根数据线，也称为数据总线（或称数据母线）；16根地

址线，也称为地址总线（或称地址母线）。这些线将机器的各个主要部分连接起来。CPU利用这些线，一方面指定出数据的存放地址，另一方面用它实现数据的传输过程。

CPU经地址总线给出信息（命令或数据）的存放地址，并定时送出单向信号，通过加到16根线上的二进制形式的信号进行操作指示。为了与外部接口相连接，该地址总线是还连接到所设计的扩展接口引出脚的插头上。

数据总线是8根双向传输线。它一方面是将由CPU所指示的地址的数据送到CPU里；另一方面把CPU要送出的数据放到所指定的地址里去。这8根数据总线为了同接口相连接，也送到扩展接口引出脚的插头上。

CPU还用来控制输入和输出的信号，所有这些问题将在以后介绍。

1) ROM

ROM是只读存贮器的简称(Read Only Memory)，它的意思是表示只能读出的单向存贮器。用来起动TRS-80机的信息要预先放到这里面来。BASIC的程序也要放到这里面。然而通过更换ROM，TRS-80机还可以用于其它目的。

接通TRS-80的电源开关，再按下复位开关时，该ROM的具体地址就能自动地置定下来，从而建立该机的初始状态。在ROM当中，除了存有BASIC的程序之外，也存放用来操纵这个CPU和外部设备的控制程序。

虽然所有的运算或控制是由CPU来实现的，但是，涉及到作怎样的运算或控制的信息，全部都放在ROM里的。所以如果没有ROM，CPU就什么工作也做不成。

2) RAM

RAM是随机存取存贮器(Random Access Memory)的简称。它能自由地进行数据的改写，所以用它来暂时地存贮数据，或者把用BASIC语言所编的程序存放到这里面。

因为RAM容量的大小决定了它所能够处理的数据量，而且各种程序的大小也是不同的，所以在TRS-80里必须要相应地增设一些RAM。

但是，有时候需要将ROM里的数据转移到RAM里。从ROM里直接转送到RAM里是不行的，所以要想把数据从ROM转移到RAM里时，就要通过CPU先从ROM里把数据读出来，然后再写进RAM。这是一个确定地址总线的过程，读出和写入都只能由CPU来进行控制。

3) 键　盘

对CPU来说，键盘虽然不是特别必要的部件，但是，因为TRS-80机要由人工进行操作，所以缺少键盘也是不行的。

该键盘的输入，与ROM同样被接到地址总线上。它按照所指定的地址范围（3800～38FF)对部分的存贮器进行相应地处理。根据从键盘来的输入信号，CPU接着进行下面操作，即借助于存贮在ROM里的系统程序用已确定的模式进行工作。

4) 视　频 RAM(Video RAM)

视频RAM也与键盘一样，都是涉及到人的作用，不同点只是前者是对人作输出表示的器件，后者却是以人工进行操作的器件，而CPU则是用于一方面给出此刻要进行的是什么操作；另一方面把输入进去的程序或者数据，甚至其计算结果通过显示器显示出来。

该视频RAM也与一般的RAM一样的被连接到地址总线和数据总线上，按照已被指定的地址(3COO～3FFF)被读写。

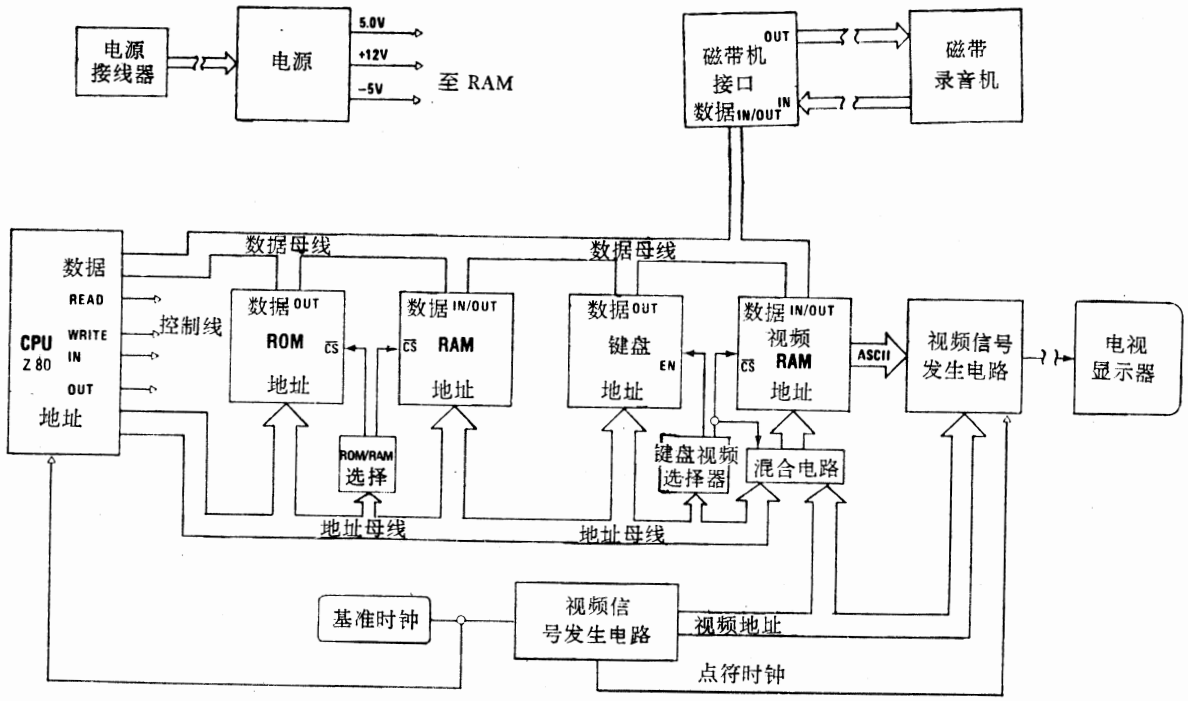


图1 TRS-80 的系统方块图

与视频 RAM 相对应的输入形式必须使用 ASCII 代码。该视频 RAM 的输出是根据 ASCII 代码被变更成它所对应的字符图形，之后再变换成电视图象信号。从 ASCII 代码变换成电视图象时的字符图形所使用的器件是 ROM。

5）视频信号发生器(Video Divider Chain)

为了使一般家庭用的电视机能够作为 TRS-80 的输出显示器，则需要合成一个包含同步信号的视频信号。为此，必须首先合成稳定的水平同步信号和垂直同步信号，而后再进一步合成来自视频 RAM 的字符图形的图象信号。

如果使用 CPU 直接去作合成这样的视频信号的话，是相当困难的，所以要通过视频信号发生回路来产生视频信号。视频 RAM 的地址是与电视显示器上的位置有关系的。就是说，要想在显示器的某一个特定的位置上显示一个字符时，那么就要用 ASCII 代码把字符写到相当于那个特定位置的地址号码上。写入的操作是由 CPU 执行的。显示器上的显示，是通过另外的地址发生器按所指定的顺序依次地被读出来。

用来作显示的地址，它的产生方式是通过把 CPU 用于读写 ASCII代码的地址作必要的转换，而作这种地址转换的部件就是 MUX(Multiplexer) 多路转换器。

只有在需要从 CPU 直接地向视频 RAM 里读写 ASCII 代码时，视频 RAM 的地址才被转接到来自 CPU 的地址总线的那一侧，而其它时间则是把电视显示用的地址加到视频 RAM 的地址里。

6）存贮器的分配形式（Memory Map）

在下面的图 2 里表示采用 TRS-80 的 BASIC-Ⅰ时所给出的存贮器的分级配置。从 0000 到 0FFF 的地址是为 BASIC-Ⅰ ROM 用的地址。键盘用的地址是从 3800 到 38FF，而作为视频表示用的地址则是从 3C00到 3FFF。

RAM 的地址是从 4000 到 7FFF 的范围，按需要容量的不同而定。要注意，上述 RAM 中从 4000 到 41FF 的地址是供 BASIC-Ⅰ的内部处理用的，所以实际上只能使用 4200 以后的地址。

注意：在 TRS-80 中主要是采用 TTL 逻辑元件，所以在本书中把逻辑“0”只是简单地写成 0，而它所呈现的电压值为 0.2 伏左右。另外，逻辑“1”约为 3 伏电压，本书中称它为逻辑 1。

2. TRS-80 的工作原理

这里对有关 TRS-80 的工作原理作一说明。对照说明并请参看书后面的线路总图（图24）。

该线路图分为两张，第一张称为分图 1 (Sheet 1)；第二张称为分图 2 (Sheet 2)。

在分图 1 里标记 SHT.2 的地方，表示该接点用导线连接到分图 2 上。分图2内亦同。

1）系统的时钟

用来使 TRS-80 动作的时钟电路，在分图 2 的左上方。

这里的 Y1 是一个频率为 10.6445MHz 的石英振子，它被装在反馈振荡器的回路里，该反馈振荡器是由 2 块 Z_{42} 的非门 (NOT) 元件组成的。47PF 的电容器 C_{43} 是反馈回路的一部分；电阻 R_{46} 与 R_{52} 是用作起振的元件。

地址 (16进制)	內　　容
0000 ≀ 0FFF	BASIC Ⅰ ROM
1000 ≀ 37FF	不　用
3800 ≀ 38FF	鍵　盘
3900 ≀ 3BFF	不　用
3C00 ≀ 3FFF	视频 RAM
4000 ≀ 41FF	供BASICⅠ使用
4200 ≀ 4FFF	从这里开始使用
5000 ≀ 5FFF	
6000 ≀ 7FFF	
8000 ≀ FFFF	不　用

4KRAM

8KRAM

16KRAM

图2　BASICⅠ 存贮器分配单

Z_{42} 引出脚 4 的输出是形成一个频率接近 10.6445MHz 的正弦信号波形。当该系统的时钟振荡器尚未起振时，系统是不工作的。在有故障的情况下，首先要用示波器等仪器检查一下该振荡器是否仍在工作，接着在 Z_{42} 的引出脚 4 上作频率测量，但有时由于测试装置的负载匹配问题，其频率并非是 10.6445MHz，这时就要在缓冲器 Z_{42} 的输出引出脚 6 上进行测量。因为这个输出就是送给 CPU 和视频信号发生器回路的。

2) CPU 的定时脉冲

要使 Z80 工作就必须有时钟的输入。Z80 不是采用类似于 8080 或 6800 双相时钟脉冲的，而是采用单相时钟脉冲进行工作的。

频率约为 10MHz 的时钟脉冲，通过在分图 1 的左上方的计数器 Z_{56}，作 1/6 的分频之后，得到的 1.774MHz。该信号在 Z_{56} 的输出端 8 上输出，并通过缓冲器 Z_{72} 最后被加到微处理器的输入端 6 上。

因为 Z_{72} 的引出端 15 是加有 0 电平，所以保证了时钟脉冲信号总是被加到 Z80 上。计数器 Z_{56} 的引出端 7 与 6 分别是用作打入计数和清除数据的，所以当这些引出端都处于 0 时，机器正常工作。

为此在 Z_{42} 的引出端 9 上通过 4.7 千欧的电阻 R_{67} 把 +5 伏的电源加上去。由于已预先限定为逻辑 1，所以 Z_{42} 的引出端 8 就变为 0 了，当然与其相连的 Z_{56} 的引出端 6 与 7 也都处于 0。Z_{42} 在车间里是作为测试元件用的。

3) 接通电源时的清除与系统的复位

接通电源之后，CPU 就把放在 ROM 里的某一个程序取出来，这时的机器就应开始执行程序。但是可以设想，当接通电源时，程序的执行要从某一规定的地址开始的，所以从接通电源开始，稍后一段时间才将 Z80 复位。

复位电路是由连接到 Z_{53} 的引出脚 12 与 13 上的电阻 R_{47} 与电容器 C_{42} 组成的。电容器 C_{42} 在未接通电源时的电压为 0 伏。接通电源时，经过电阻 R47 的 5 伏电压在一定时间内使电容器 C_{42} 的两端电压达到 1.4 伏，我们把它看作状态 1。

因此，当 Z_{53} 的引出端 12 与 13 的输入为“1”之后的一段时间（数微秒）内，Z80 的复位端 26 就变成 0，即 Z80 已被复位。这样，如果 Z80 微处理器已复位的话，则 16 根地址总线的值就变成以 16 进制表示的 0000，而 CPU 为取 ROM 里的初始数据作准备。

这里应注意，虽然逻辑元件 Z_{53} 也是与非门(NAND)，但是我们所使用的是类似于或门的符号(OR)，这是为了使逻辑方面的功能易懂，以便于灵活地掌握。如果在与非门的输入侧标记的是非端(NOT)，把它画成或门那样符号的话，则这个逻辑元件就其功能而言表示的是或门的逻辑。

复位开关 S_2 是与接通电源开关时的清 0（或复位）具有同样的功能。在该开关的回路上所加的电容器 C_{57} 的容量比电源开关用的要小。这是为了使它建立逻辑 1 的状态比电源开关快的缘故。

按下开关 S_2 时，Z_{53} 的引出脚 1 呈现 0 伏，于是 Z_{37} 的引出脚 12 与 11 的值变为 1，其 Z_{37} 的输出端则变为 0。因此 CPU 的引出脚 17 的 $\overline{NMI}$ 的值呈现 0。这时候从 CPU 引出来的地址总线反映出的值即为 0066。当手离开开关之后，Z_{53} 的引出脚 1 的值约超过 1.4 伏，CPU 就从 ROM 的地址 0066 开始执行程序。

开关 S_2 的另一个作用是，当CPU由于某种原因失去控制，机器已不能继续执行工作时，

用它来使机器返回去重新执行正确的程序。譬如，在使用盒式磁带来传送程序的过程中，由于某种原因使磁带继续不停地走下去时，这时候按下开关S_2，传送程序即可中断，重新返回到初始的准备状态。

Z80的CPU的引出脚18的输出$\overline{HALT}$就是表示暂停CPU的一个状态信号。在采用BASIC-Ⅰ的情况，该引出脚18为0，而它反映的并非是暂停这一概念。这里所以要把它变为0状态是因为有时在程序的执行过程中有暂停的命令存在。但在该系统的ROM当中却未包括用作停止的命令。

然而，由于某种原因常常需要加进停止命令。而这时当你即使按下开关S_2，机器也达不到初始准备状态。在发生这样的情况下，我们就得切断电源开关再从头开始。

Z_{53}的引出脚11与另一个Z_{53}的引出脚3，分别连接到Z_{37}的引出脚2与3上，而Z_{37}的引出脚1的输出是作为系统复位用的信号，叫做SYSRES*信号。该信号通过插头与外部接口相连。但是该信号用BASIC-Ⅰ语言在TRS-80里是不能使用的。

由于CPU失控，在不得已的情况下必须切断电源开关时，要至少停10秒钟后方能重新起动开关。这正是保留在C_{42}中的电荷消除而变成0伏时所需要的时间。

如果断开电源不到10秒钟就又起动电源时，则从CPU引出的地址总线就不能出现所谓初始地址0000，而且CPU又会重新出现失控现象。

4) WAIT*、INT*、TEST*

在TRS-80里，Z80的CPU的引出脚24$\overline{WAIT}$、引出脚16$\overline{INT}$以及引出脚25$\overline{BUSRQ}$(TEST*)都是通过4.7千欧电阻接到+5伏的电源上。而且它们都作为输入端接到与外部接口相连接的插头上，所以可以充分利用这些引出脚。

下面将介绍有关这些引出脚的功能。

Z80的CPU的引出脚24（对应于插头上为33）的WAIT*，它的作用是当我们采用外部的速度慢的存贮器读出、写入时要利用它。假如把WAIT*，变为0伏的话，CPU就处于“等待”状态，从它再次出现逻辑1的状态之后，CPU就移到下一个工作状态。

例如，一个长达100μs的存贮器系统被接上作为外部接口的部件。如果此时表示要求把数据读出来的话（譬如有RD*信号示出），那么就使该WAIT*在100μs的间隔之内保持为0状态，之后该WAIT*再出现逻辑1时，CPU就能从外部存贮器里把数据读出来。

Z80的CPU的引出脚16(对应于插头为21)的INT*是表示“中断请求”(Interrupt Request)。只要把该输入点变成0即可实现中断请求，执行中断所需要的程序。这一点是供TRS-80的外部扩展接口方面使用的。

TEST*输入，是当TRS-80中产生故障时，为了便于故障的诊断而加进去的。接口的插头23是被连接到Z80的CPU的引出脚25的$\overline{BUSRQ}$上。

这样一来，如果使引出脚25为0时，则从CPU引出来的数据总线、地址总线以及一些控制线等都处于切断状态（即呈现高阻抗状态）或称为无输出状态。这一点在TRS-80里是不采用的。但是可以用它作为“切断”CPU的输入。

5) CPU的地址总线

Z80的CPU从A_0到A_{15}的输出，是用来指定ROM、RAM、视频RAM以及键盘等的地址的输出端。

为了把地址总线上的信号送到相应的地址里，CPU上的地址输出所需要的电流是不充

裕的，所以必须附加一些缓冲器Z_{55}、Z_{22}、Z_{39}、Z_{38}.另外，由于这些缓冲器采用的是具有三种状态的元件，所以从外部接口输入进来的地址总线信号，也有可能把数据传送到TR-S-80的RAM里。

前面已经提到，这时候如果信号TEST*为0的话，由CPU来的地址总线就在这些缓冲器的位置上被“切断”。该切断地址总线的功能，是由Z_{52}的引出脚4信号ENABLE*出现逻辑1的状态来实现的。

产生故障要进行测试时，须将TEST*置于0状态，检查该ENABLE*的电压值，确认从A_0到A_{15}的端点已被切断之后，就可进行其它的各种检查修理等。

6）CPU的数据总线

Z80的CPU的数据总线是从D_0到D_7共8根。它与地址总线一样都接有缓冲器。地址总线从CPU只能单方向向外输出信号，而数据总线的情况却不同，它除了输出之外还向CPU输入。所以Z_{75}、Z_{76}的一部分用作输出缓冲器，而向CPU里输入的缓冲器，在这里所使用的是Z_{55}及Z_{76}的一部分。切换这些输出输入缓冲器的是由Z_{53}的引出脚9、10的输入DBOUT*信号来执行的。它处于0状态时，数据就选择CPU的输出方式，如果信号DBOUT*变为逻辑1的状态时，DBIN*就变为0状态，于是选择向CPU里输入数据。

由于Z_{53}的引出脚4是与TEST*相连接的，所以如果TEST*为0的话，Z_{53}的引出脚6就变为逻辑1，而输出缓冲器被切断。这时的DBIN*呈现0状态，输入缓冲器工作是正常的。由此可以认为没有产生什么特别的故障。

当TEST*出现逻辑1时，CPU的引出脚21，即$\overline{RD}$就不会是0，而此时的DBOUT*变为0，于是数据线可切换到输出状态。

7）读出线（RD,Read）

所谓读出，即表示CPU能够接受数据的状态。该RD信号是通过Z_{23}的引出脚6输出。信号RD是取Z80中CPU的引出脚21的$\overline{RD}$与引脚19的$\overline{MREQ}$（Memory Request）信号相与（AND）的结果。所以Z_{23}的引出脚6的输出，就是它们两者都为0时而输出是0状态的RD信号。

该RD信号输入到Z_{22}的引出脚6上，再通过Z_{22}缓冲器被输出到外部接口的插头15的RD*上。该RD*信号当TEST*处于0时也同样地被切断。

8）写入线（WR,Write）

所谓写入，就是表示CPU能够把数据写入到存贮器等部件里去的状态。所以送到Z_{23}的输入引出脚12,13的数据，必须当CPU的引出脚19的输出$\overline{MREQ}$和引出脚22的输出$\overline{WR}$（Memory Write）的信号两者都为0时，方能从Z_{23}的引出脚11送去WR为0状态的信号。它与RD信号同样要通过缓冲器Z_{22}输出到外部接口插头13的WR*上。

9）输出线（OUT,Output）

该输出是用来向盒式磁带中写入数据的信号。当Z80的CPU的引出脚20的$\overline{IORQ}$（I/O Request）与引出脚22的$\overline{WR}$两者的输出都为0时，Z_{23}的引出脚3的OUT变为0输出。

这个输出，通过缓冲器Z_{22}作为OUT*信号输出到外部接口插头12上。该信号就是盒式磁带接口Z_{25}的引出脚9的输入。

10）输入线（IN,Input）

该输入是用来从盒式磁带机里读出数据的信号。所以当CPU的引出脚20的$\overline{IORQ}$与

引出脚21 的输出的 $\overline{RD}$ 两者信号都为 0 时，Z_{23} 的引出脚 8 的 IN 变为 0 状态输出。

这个输出也通过缓冲器 Z_{22}，作为 IN* 信号输出到外部接口插头 19 上。同时该信号用来作为读入盒式磁带的控制信号，就成为盒式磁带机接口 Z_{25} 的引出脚 4 的输入。

上面的 WR*、OUT*、IN* 等信号与信号 RD* 一样，当 TEST* 的输出为 0 时，它们都在其缓冲器的位置上被切断。

3. 地址译码器

关于地址的如何分配问题，在图 2 中已经作了说明。为了给出这样的地址分配形式，就要通过地址译码器来实现。

当由 CPU 里送出地址信号的时候，要选择哪一个地址的问题，这要根据地址译码器的输出而定。我们把地址译码器的一部分取出来画在图 3 里。

下面所列出的表不是采用 16 进制的数，而是以 2 进制的形式表示如何定出各存贮器对应的地址。图 3 就是该表中从 A_{10} 到 A_{15} 的译码器的电路。

A_{12}、A_{13}、A_{14} 是接到两输入——四线的译码器 Z_{21} 上。由于 Z_{21} 的 C_1 与 C_2 组成公共的输入端，并且 C_1 与 C_2 又作为司动门输入，所以 Z_{21} 可以认为是 3 个输入，8 个输出的译码器。

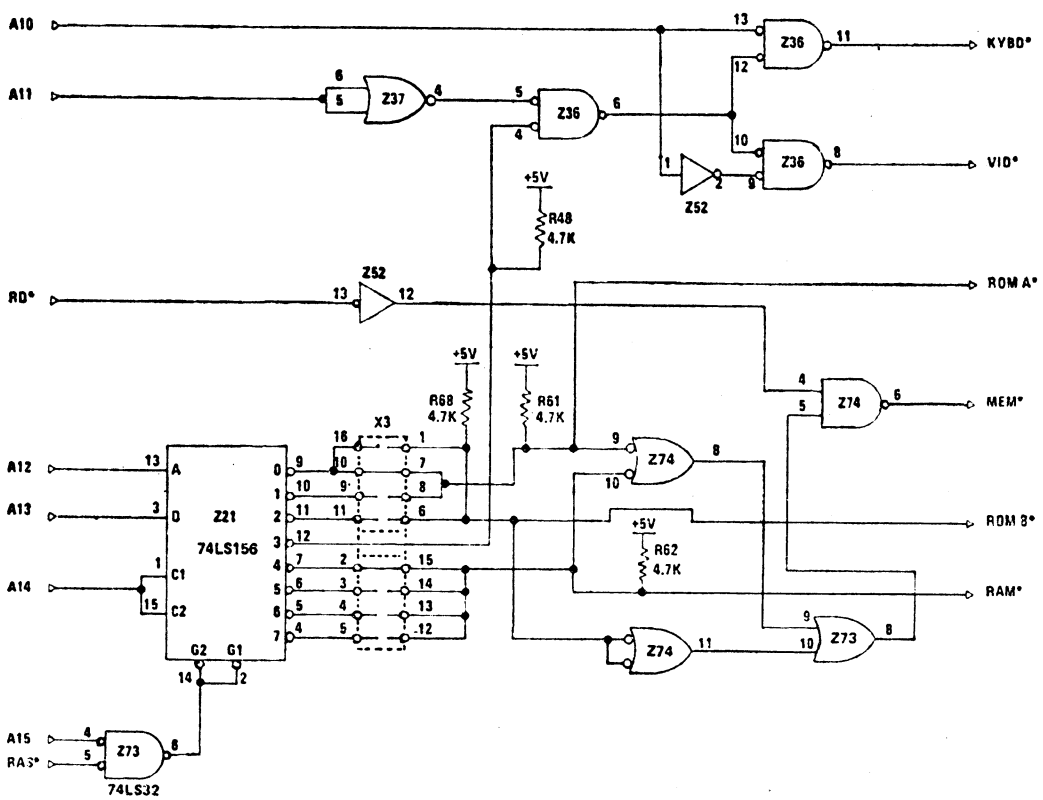


图 3　地址译码电路

	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	
0000	0	0	0	0	0	0	0	0	BASIC-I ROM
⌇									
0FFF	0	0	0	0	1	1	1	1	
3800	0	0	1	1	1	0	0	0	键盘
⌇									
38FF	0	0	1	1	1	0	0	0	
3C00	0	0	1	1	1	1	0	0	显示器 RAM
⌇									
3FFF	0	0	1	1	1	1	1	1	
4000	0	1	0	0	0	0	0	0	4K RAM
⌇									
4FFF	0	1	0	0	1	1	1	1	
4000	0	1	0	0	0	0	0	0	16K RAM
⌇									
7FFF	0	1	1	1	1	1	1	1	

司动门的输入即为 A_{15} 与信号 RAS*（Z80 CPU 的 $\overline{\text{MREQ}}$的输出），所以存贮器请求（MREQ）和 A_{15} 都为 0 时，Z_{21} 的译码器就开始动作。Z_{73} 是或门电路，但是根据实际的逻辑功能就用与门符号（AND）表示。

当 A_{15} 的值为 0 时，通过 A_{14}、A_{13}、A_{12} 的值的组合，使 Z_{21} 译码器的几个输出端的值变成 0。例如，把 011 的值加到 Z_{21} 的 A_{14}、A_{13}、A_{12} 上时，在 Z_{21} 的输出端 12 上就呈现 0 状态。以上的组合情况说明已选定的地址正是键盘和视频 RAM 的。

从这个 Z_{21} 的引出脚 12 来的输出通过逻辑回路 Z_{36} 被译成信号KYBD*和信号 VID*。这两个信号分别在下列条件成立时才有输出。

	RAS*	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}
KYBD*	0	0	0	1	1	1	0
VID*	0	0	0	1	1	1	1

1) ROM 的译码器

关于 Z_{21} 的输出，即涉及到 16 脚 IC 插座 X_3。它是放在印刷板 Z_3 的位置上。根据这些连线就能够清楚地知道 CPU 所指出的 RAM 与 ROM 的地址。

在图 3 中，X_3 有 6 根线已被切断。如果把 Z_{21} 的输出端相互连接起来的话，它就变成与或门相同的作用。譬如，把 Z_{21} 的引出脚 9 与 10 连接起来，那么选择这两个引出脚中任何一个都可作为输出表示。

因而，如果把 X_3 的接点 10 与 7；9 与 8 连接，则按下列条件在 ROMA* 上才有输出，并且所选择的是 8K 字节的 ROM。

	RAS*	A_{15}	A_{14}	A_{13}	A_{12}
ROMA*	0	0	0	0	X

这里的符号X是表示为 0 或者为 1 都可以。在图 3 里，Z_{21} 的引出脚 9 在 X_3 里是被连到接点 10 和 7 上。这样的话，ROMA* 只能被指定到 4K 字节的 ROM 里。

该 ROMA* 信号经过 Z_{74} 的引出脚 9 以后，从引出脚 8 输出，然后再作为或门 Z_{73} 的

输入，从 Z_{73} 的引出脚 8 输出，这个输出又通过与门 Z_{74} 与信号 RD* 相与。当信号RD*为 0 时，通过 Z_{82} 变换，使 Z_{74} 的引出脚 4 变为逻辑 1，Z_{74} 的引出脚 6 则呈现 0 状态，于是此刻被指定的信号为 MEM*。ROM 或 RAM 的缓冲器 Z_{67}、Z_{68} 已畅通的话，ROM 的输出当再一次出现数据总线的信号 DBIN* 为 0 时，它就通过 Z_{76}、Z_{75}、Z_{55} 将数据信息送到 Z80的CPU 里。

2）键盘译码器

键盘的地址是从 3800 到 38FF 。当 Z_{21} 的输入被选为下列情况，即

RAS*	A_{15}	A_{14}	A_{13}	A_{12}
0	0	0	1	1

而且 A_{11} 为 1 时，Z_{37} 的输出端 4 就为 0，而 Z_{36}的输出端 6 作为选择输出即为 0 。与此同时 A_{10} 为 0 时，Z_{36} 的输出端 11 就变成 0，从而选择了信号 KYBD*，所以 KYBD* 信号的选择条件为

	RAS*	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}
KYBD*	0	0	0	1	1	1	0

该 KYBD* 正是键盘输出缓冲器 Z_3 与 Z_4 的门控信号。当它为 0 时，就使这些输出缓冲器接通了。

关于键盘的动作问题，在以后将作详细介绍。这里只要按下按键就把从 A_0 到 A_7 的地址及从 D_0 到 D_7 的数据等线接通。该键盘的输出如同 ROM 情况一样都是经过向 CPU 里读入的缓冲器，把数据送到 CPU 里去。

3）视频显示用的 RAM 译码器

视频显示用的 RAM 译码器，与以前所述的键盘译码器用的回路大致相同，只是 A_0 信号通过 Z_{82} 的变换而反相。所以信号 VID* 的选择条件为

	RAS*	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}
VID*	0	0	0	1	1	1	1

根据 VID* 呈现 0 状态这一条件，视频 RAM 就能够工作，而它工作的详细情况以后再作介绍。

4）4K RAM 译码器

在存贮器的分配上用来表示 4K RAM 的地址是从 4000 到 4FFF .如图 3 中所表示的连线，就是选定 4K RAM的连线方式。所以根据 X_3 中接点 2 与接点 15 之间的连接情况，选择 RAM*的条件就变为如下形式。该 RAM* 通过分路器 X_{71} 被连到所有的RAM接点$\overline{CE}$。

	RAS*	A_{15}	A_{14}	A_{13}	A_{12}
RAM*	0	0	1	0	0

选择 RAM 与选择 ROM 的情况一样。从 RAM 里把数据读出来的情况，是将 RD*变为 0，因而信号 MEM* 也就呈现 0 状态。当从 CPU 向 RAM 里写数据时，信号 RD* 不为 0，而信号 WR*为 0 。若是 ROM(或RAM)的缓冲器被切断的话，则从 CPU 来的数据，就将被转送到 RAM 里。

在IC插座X_3里，要选择8KRAM时，就得把接点3与14连接起来；要选择12KRAM时，就得把接点4与13连接起来；要选择16K的RAM时，把接点5与12连接起来。如果象这样选择不同字节的地址的话，则RAM*信号从4000到4FFF这段范围之内都呈现0状态。

由于Z_{21}是一个开路的接线器，所以把它们的输出端相互连接起来是不会出现任何问题的。但是请注意，处于原有的开路状态，即使用示波器也观察不到输出的。

例如在图3里，在原来接点11上只接上示波器的探头是什么也看不到的。要作输出观察，就须在观察点与+5伏电源之间加上5千欧姆大小的电阻。

4. 系统RAM

由框图可以看到RAM与ROM都同样地被接到数据总线和地址总线上。RAM数据的输入或输出是由信号MEM*来控制的。

不过就RAM的地址的输入来说，虽然是按4KRAM连接的，而我们注意到的只是使用了7根地址线。其原因是，从CPU里所送出来的地址信号线，被多路转换器Z_{35}与Z_{51}作了时间的分割，它们向RAM里各送一半的地址线，而在RAM里又把这些地址重新排列一下增加一倍，即又变成14根。这是为了减少RAM的引出脚并使其小型化的缘故。

为了实现这些控制，需要三个控制信号，即信号RAS*(Row Address Select、行地址选择)、信号CAS*(Column Address Select、列地址选择)以及信号MUX (Multiplexer、多路转换)。这些信号都产生于分图1的左下部分。

1) MUX、CAS*、RAS*

Z80 CPU的引出脚22与21的输出是接到Z_{74}的与非门上。Z_{74}就其功能而言是属于逻辑或的。所以我们这里就采用或逻辑符号。Z_{74}的输出端3，当CPU的输出$\overline{WR}$或$\overline{RD}$为0状态时，就是说由CPU来进行写入或是读出的时候，它呈现逻辑1状态。我们把这个信号记为MREQ.要注意它与Z80的输出$\overline{MREQ}$的区别。

Z_{69}和Z_{70}的D型触发器是用来产生定时信号的部件，用以控制动态RAM的。它的每个引出脚输出的时间关系如图4所示。

信号MREQ是接到Z_{69}与Z_{70}的CLR（清除）回路里的,所以QMUX是0状态时，QCAS就为逻辑1状态。Z_{69}的输出端5是信号NEXT，它是随下一个时钟波形的前沿被输出出去的。它的波形在图4中由E示出。该信号经过下一级D型触发器，在引出脚9上得到信号QMUX.

信号QMUX的波形如图4中以F所示。它成为Z_{72}的引出脚2的输入并以信号MUX在该引出脚3上输出（图4中的J）。信号MUX接到Z_{35}、Z_{51}多路转换器的选择器的输入端1上。当该输入端1的输入信号为0状态时,该选择器就选择输入引出脚2、5、11、14的信号并分别送给引出脚4、7、9、12. 因此把地址信号A_0,A_1,A_2、A_3、A_4、A_5、A_6送给RAM约需250ns的时间。在这段时间内，对RAM来说所给出的地址即为它的行地址(Row Address)。

信号QMUX还通过D型触发器Z_{70}被延迟一个脉冲间隔而变成信号QCAS,该QCAS信号在图4中为H. 它再通过缓冲器Z_{72}之后被表示成信号CAS*（图4中的K）。信号CAS*通过Z_{67}送到Z_{13}～Z_{20}的$\overline{CAS}$里。RAM中的信号$\overline{CAS}$就是用于给出它的列地址(Column Address)的输入端。

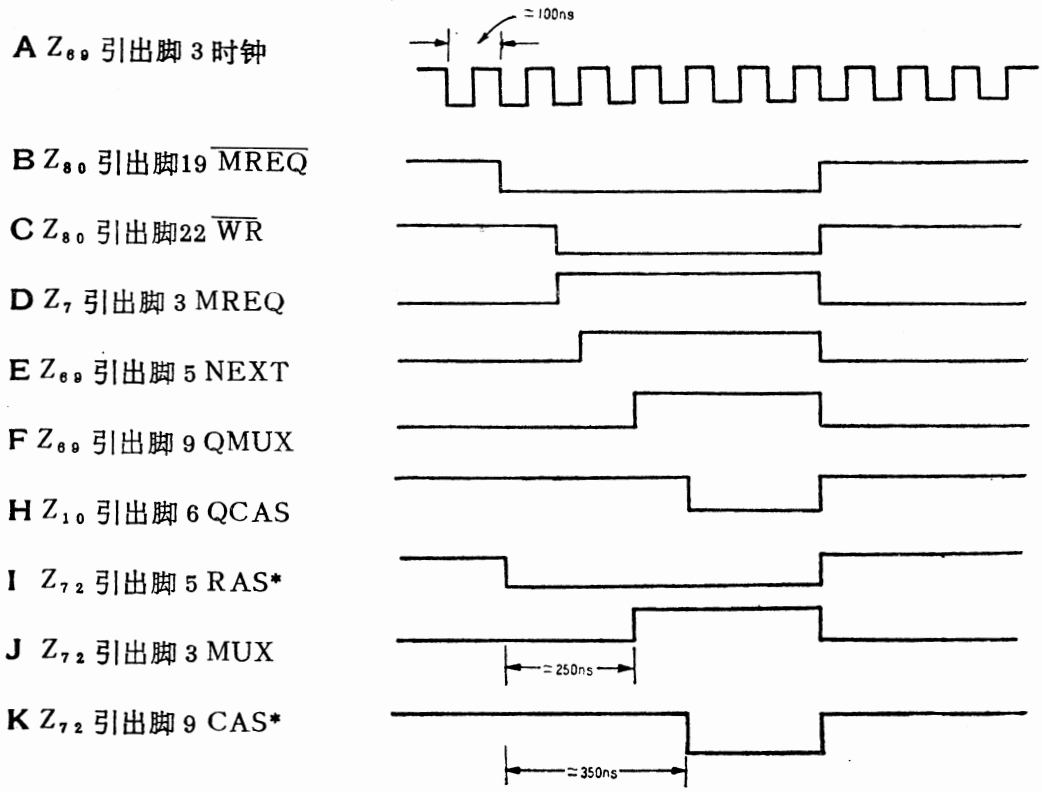


图4 控制动态 RAM 的时间波形

MUX 信号是在信号 $\overline{CAS}$ 出现之前 100ns 就从 0 状态翻转到逻辑 1。根据这个信号，Z_{35} 与 Z_{51} 就能把 A_7、A_8、A_9、A_{10}、A_{11}、A_{12} 等地址的输出信号送到 RAM 里。

如果信号 $\overline{WR}$ 或者 $\overline{RD}$ 变为逻辑 1 的话，由于 Z_{74} 的输出端 3 变为 0 状态，并使触发器 Z_{69} 与 Z_{70} 清零，上述的信号 MUX 及 CAS*就没有输出。

这些 RAM 的信号，就是用来控制动态 RAM 动作的定时信号。

2）RAM 的地址选择

有关 TRS-80 里的 RAM 被设计成采用动态 RAM 的问题，在前节的介绍里只是大致地了解一下。

RAM 最好设计成 4K 字节与 16K 字节两者都能够采用的形式。采用 4K 字节的情况有 12 根地址线，即有从 A_0 到 A_{11} 的地址线就足够了。根据 X_{71} 的接线方式来进行由 4K 字节到 16K 字节的转换。如果把 X_{71} 中记有 4K 的部分连接起来，那么给出的就是 4K 的动态RAM.对 4K 的动态 RAM 来说，RAM 的引出脚 13 就是信号 $\overline{CE}$（芯片选择 Chip Enable）。

另外，对 16K 的动态 RAM 来说，该 RAM 的引出脚 13 就变成作为地址 A_6 来使用。为了采用 16K 的动态 RAM,其地址线必须是从 A_0 到 A_{13} 的 14 根地址线。因此要把 X_{71} 中记有 16K 的地方连接起来，这样即可以作为 16K 的动态 RAM 来使用。要注意，采用 16K 的动态 RAM 也必须把 X_3 中的接点 2 与 15、3 与 14、4 与 13、5 与 12 等都连接上。

3）从 RAM 里读出

这里以 4KRAM为例说明。信号 $\overline{MREQ}$和 $\overline{RD}$ 是作为 Z_{80} 的输出送出去的。就是说当它们的信号都变成 0 状态时，在图 3 中所表示的地址译码器上的输出 MEM* 和 RAM*也都呈现 0 状态。MEM* 信号把 RAM 或 ROM 的数据缓冲器切换到读出方式，同时信号RAM* 把 0 状态的信号送给了 RAM 的 CE(chip Enable)以驱使 RAM 进行工作。

在 CPU 向外传送地址信号的同时，信号 MUX 变成 0 状态，于是 RAM 通过多路转换器接收 $A_0 \sim A_5$ 的地址信号。

信号 RAS*是从 Z80的引出脚19的 $\overline{MREQ}$输出，再经过缓冲器 Z_{72} 得到的，它又经过缓冲器 Z_{68} 变成信号 MRAS 之后加到 RAM 引出脚 4 的 $\overline{RAS}$ 上。所以从 A_0 到 A_5的地址信号是作为行地址被接收的。

当信号 MUX 出现逻辑 1 的状态时，通过多路转换器 Z_{35}和 Z_{51} 把地址通道接向 $A_6 \sim A_{11}$ 高位地址信号的一侧，并把它们输送到 RAM 的每个地址中去。这时候的信号 CAS* 是 0 状态。

信号 CAS* 经过缓冲器 Z_{67} 变成了信号 MCAS 之后被送到 RAM 的引出脚 15 上。这时候，RAM 接收了 $A_6 \sim A_{11}$ 的高位地址信号并且确定出了所有的地址之后，就把对应的地址数据值通过缓冲器送到 CPU 里。

4）向 RAM 里写入

写入与读出的不同点，只是把用来代替 $\overline{RD}$ 的信号 $\overline{WR}$从 CPU 里送出就行，而其它一些地址信号的动作情况完全与读出一样。

ROM 或者 RAM 的缓冲器，如果只用来作读出的话，就不送出信号 MEM*. CPU送出信号 $\overline{WR}$ 时，它经过 Z_{23} 门和缓冲器 Z_{22}，加到 RAM 的引出脚 3 $\overline{WR}$ 上。根据这一情况，从 CPU 来的数据就可以写到 RAM 里。

5）关于 RAM 的更新问题

关于 TRS-80，前面已经提到它是采用动态 RAM 的。与该动态 RAM 方式对应的就是静态 RAM 方式。在静态方式的 RAM 里，一旦把数据写进去，除非切断电源或重新改写，否则是消除不了的。

另一方面在动态 RAM 里，由于是周期性地选择地址，常常会出现这样一种情况，即还没有来得及更新就已经被所谓“忘掉了”。所以在这样的动态 RAM 里，就有必要选择那些不会忘而具有“想起”这一功能的地址。

TRS-80 里的动态 RAM 所采用的更新方式只要借助于信号 $\overline{RAS}$ 就能实现。因此只要信号 RAS* 为 0 状态时，就能自动地对给定的行地址进行更新。RAS* 是通过 CPU 的引出脚 19（信号 $\overline{MREQ}$ ）给出的。

TRS-80 所使用的 CPU，为了更新它的动态 RAM，只要每隔一定的时间(2ms)，输出 $\overline{MREQ}$ 信号并给出所需要的地址信号，而不采用专门的更新回路。

这里要特别注意的是，仔细看图即可以发现，当使信号 TEST* 为 0 状态的话，信号 $\overline{MREQ}$ 在缓冲器 Z_{72} 处被切断，因而动态 RAM 的内容也被消除掉了。

5. 同步信号发生器

在 TRS-80 里，是采用电视形式的显示器。因此就需要有一个用来合成电视信号的信

号发生器。电视画面上所反映出来的信息，都保存在视频 RAM 里。所以必须把视频 RAM 里的数据变换为视频信号。同时在电视机里也必须有水平同步信号与垂直同步信号。视频 RAM 的地址与电视画面是一一对应的。为此也必须产生用于视频信号的地址。

另外，在产生该视频信号的回路里，还要有每行 32 个字符的产生回路。但是，用 BASIC Ⅰ是不能作 32 个字符显示的，而用 BASIC-Ⅱ是可以的。

1）主同步信号发生器

TRS-80 所显示的字符行数是固定不变的，即16 行然而其终端的字符显示形式有两种。第一种形式，在 16 行里每行最多显示 64 个字符。所以在这种情况下所使用的是具有 1024 个字节的视频 RAM.而另一种形式，由于其字符的大小约为上述字符的两倍。所以在 16 行里每行最多能显示 32 个字符，而使用的是具有 512 个字符的视频 RAM。

主要的波形起振回路是由触发器 Z_{70}、1/12 的分频器 Z_{58} 以及多路转换器 Z_{43} 所组成。Z_{70} 把约为 $10MH_Z$ 振荡频率的脉冲信号作 1/2 分频，变成约 $5MH_Z$频率。

多路转换器 Z_{43} 是对原有的 $10MH_Z$ 与 $5MH_Z$ 频率作出选择。当 Z_{43} 的引出脚 1 的输入为信号 MODESEL（模式选择，Mode Select）并且为 0 状态时，则它选定的是 32 个字符的显示方式。相反地 Z_{43} 的引出脚 1 的输入为逻辑 1 状态时，则它所选定的是 64 个字符的显示方式。在 Z_{43} 的输入上接的是经过 Z_{58} 的 1/12 分频之后的输出。

这里我们首先说明显示 64 个字符时的操作情况。现在由于信号 MODESEL 出现逻辑 1 状态，Z_{43} 的引出脚 3 被接到引出脚 4 上，在该引出脚 4 上的输出就是频率为 $10MH_Z$ 的脉冲信号。引出脚 6 与 10 是互相连起来的。而当引出脚 1 为逻辑 1 时，Z_{43} 的引出脚 7 和 9 上的输出就是原来 Z_{58} 的引出脚 8 的输出。

图 5 中的 A 是表示频率为 $10MH_Z$ 的系统时钟的波形。B 的波形是表示通过 D 触发器 Z_{70} 作了 1/2 分频之后的输出脉冲。

在 Z_{43} 的引出脚 4 上，送出 $10MH_Z$ 的脉冲信号输入到 Z_{58} 的引出脚 14 上。Z_{58} 的输出端 12、11、9、8 等的输出波形分别表示为图 5 中的 C、D、E、F.图中出现的“0”是表示分频器 Z_{58} 的输出全为 0 时的数值。

Z_{58} 的引出脚 6 与 7 是连接到分图 1 中的 CTR 端上。如前所述，该接线端只是在车间里进行测试时使用，通常情况皆为 0 状态。Z_{58} 的引出脚 9 称为信号 DOT2；引出脚 12 称为 DOT1。这两个信号在与非门电路 Z_{24} 内相与之后，得到像图 5 中那样的输出波形 G，我们把它叫作信号 LATCH.

Z_{43} 的引出脚 7 与 9 的输出就是 Z_{58} 的引出脚 8 的输出。Z_{43} 的引出脚 9 的输出我们称它为信号 CHAIN，这就是同步信号的主要输入。我们又把 Z_{43} 的引出脚 7 的输出称为信号 C_1.该信号被接到视频 RAM 的多路转换器 Z_{64} 的引出脚 10 上，用于表示视频 RAM 地址的最低位数。

在用 32 个字符的显示时，由于 Z_{43} 的引出脚 1 呈现为 0 状态，所以就分别把引出脚 2 与 4、5与7、11 与 9 相互连接起来。这样的话，Z_{58} 的输入端 14 上的脉冲频率就是 $5MH_Z$，而 Z_{43} 的引出脚 7 的输出就总是为接地的 0 状态。这时的信号 CHAIN 也就是 Z_{58} 的引出脚 9 的输出。

Z_{58} 的引出脚 12、11、9、8 等的输出波形表示为图 5 中的 H～K.

在采用 32 个字符的形式中，计数器 Z_{58} 是被用作 1/6 分频的分频器。因此，无论采用

32 个字符显示，还是采用 64 个字符显示，信号 CHAIN 的周期都相同，即为 887.041KHz。

Z_{24} 的引出脚 3 的 LATCH 的输出是不同的。在采用 64 个字符显示时，LATCH 的周期是 6 个系统时钟周期，其信号宽度为一个时钟的脉冲宽度。而在采用 32 个字符的显示里，其周期就变成 12 个系统时钟周期，其信号宽度为 2 个时钟的脉冲宽度。

另外就信号 C_1 来说，对 64 个字符显示的情况它与信号 CHAIN 是同相工作的，但是对 32 个字符显示的情况，它总是处于 0 状态。根据这一情况，当用作 32 个字符显示时，由于其最低位的地址总是为 0 的，所以就只能用 512 个字节进行显示。信号 LATCH 是用作从 RAM 里取出字符信号的。

2）同步信号发生器（Divider Chain）

产生同步信号的计数器回路，是由 Z_{65}、Z_{50} 以及 Z_{32} 所组成。每一个计数器都采用16进制的计数回路，其计数过程的变动是由外部控制的。

图 6 是表示产生同步信号回路的框图。根据从 Z_{65} 的引出脚 1 所输入的信号，分别在其引出脚 9 和引出脚 8 上把经过 1/2 和 1/4 分频的时钟脉冲信号输送出去。因此，经 Z_{65} 的1/4分频后的信号，从其输出端 8 以 221.760KHz 的频率值送出。然后在 Z_{50} 里再作1/14的分频。现在 Z_{50} 的输出如果为下列情况的话，则根据二进制数来看它表示的是 14。

A=0　　B=1　　C=1　　D=1

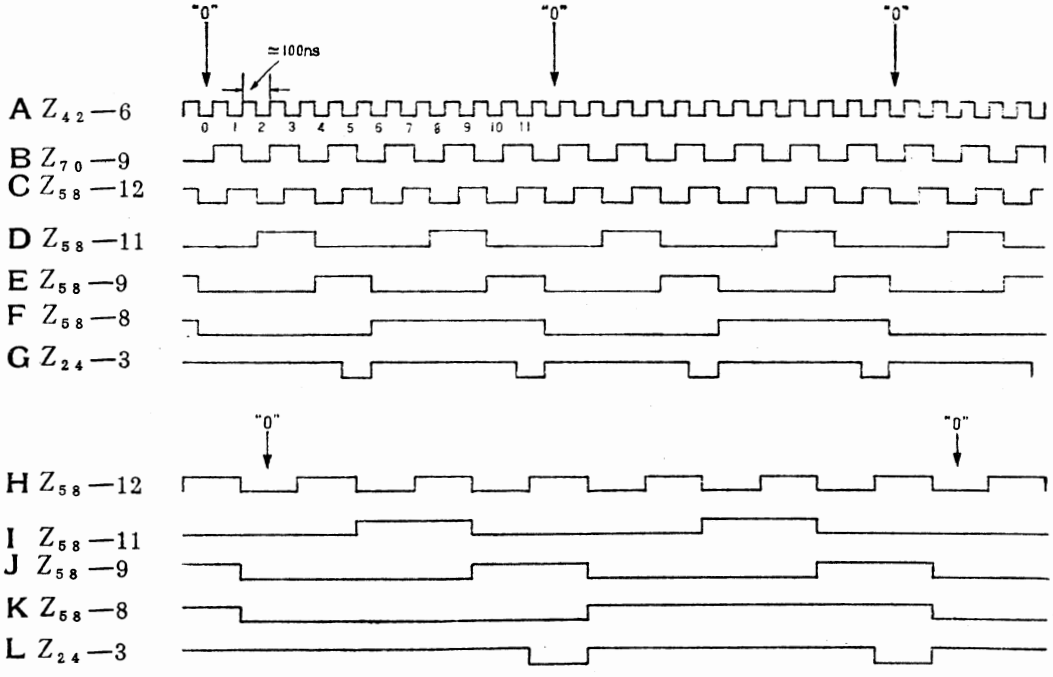


图5　视频发生回路的时间波形

Z_{66} 的引出脚 3、4、5 分别接到 Z_{50} 的 B、C、D 上。B、C、D的输出全部为 1 时，即 2 进制数 14 的输出时，那么 Z_{66} 的输出为 1，同时这个输出又将 Z_{50} 清除，因此该 Z_{50} 即成为 14 进制的计数器了。

Z_{50} 的输入频率为 221.760KHz，而 Z_{50} 的输出端 11 作为水平同步信号的输入，其频率为15.840KHZ。

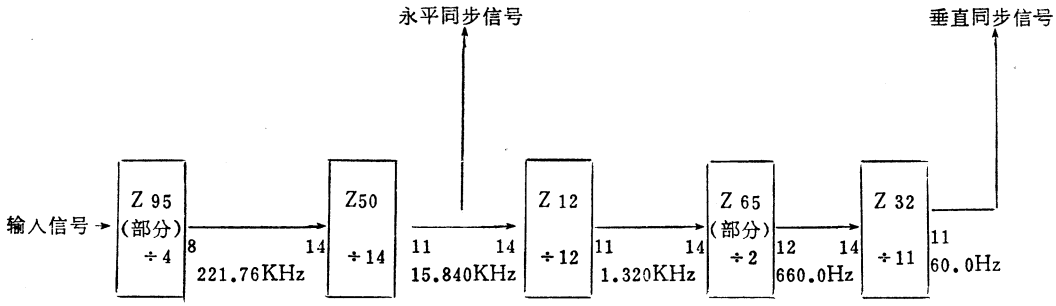


图6 同步信号发生器方块图

Z_{50} 的引出脚 11 的输出，接到 Z_{12} 的输入端 14 上，同样经过1/12 的分频。该 Z_{12} 的输入门使用的是 Z_{66} 的一部分。Z_{12} 的输出端 11 的频率降到 1.32KHz。该输出是连接到 Z_{65} 的引出脚 14 上。Z_{65} 是被分成两个部分的。从引出脚 14 进来的信号又被 Z_{65} 作了 1/2 的分频之后移到其引出脚 12 上输出出去，该点频率即变为 660Hz，它又被接到 Z_{32} 的引出脚14 上。

Z_{32} 是终端计数器，它将 660Hz 频率又作了 1/11 的分频，使 Z_{32} 的最后输出频率即变成 60Hz。Z_{32} 的输出作为信号 VDRV 用在垂直同步信号上。

3）视频 RAM 的地址选择方法

如前所述，我们已经知道视频 RAM 必须有两种地址信号；

一种是用来通过 CPU 把数据从视频 RAM 中读出来或者写进去的地址信号；另一种就是用来读取电视显示数据的地址信号。

上述这些信号都是通过多路转换器 Z_{64}、Z_{49}、Z_{31} 给出的。从同步信号发生器送出来的供选择视频 RAM 用的共有 10 根地址信号。它们以下列对应关系分别把信号传送出去，即从 Z_{43} 送出信号 C_1；从 Z_{65} 送出信号 C_2 和 C_4；从 Z_{50} 送出 C_8、C_{16}、C_{32}；还有从 Z_{65} 的另一部分里送出 R_1；从 Z_{32} 送出 R_2、R_4、R_8 等等。

现在考虑一个在方形图里有 16 行、64 列个字符的情况，其字符的总数即为 1024 个。把它们从左上方开始由左到右排列下去的话，那么就可以把 64 列由左到右如 0、1、2、3……等等编上号码。为此需要 6 根地址线。

上述的 6 根地址线是 C_1、C_2、C_4、C_8、C_{16}、C_{32}。用它们来标记从画面的左端起 64 个列的号码。另外在考虑到 16 个行数时，这只需要 4 根地址线，它们是 R_1、R_2、R_4、R_8 地址线。

如果象这样地把电视画面看成由 16 行×64 列个小四方块组成的话，那么我们也可以把这些小方块都标记上号码。最左上端的即为 0 行 0 列，而最右下端的即为 15 行 63 列了。

在 TRS-80 里从 CPU 输出的信号也都同样地标有地址号码。视频 RAM 的地址是从 3C00 到 3FFF，而 3C00 在画面上是以 0 行 0 列表示，3FFF 则以 15 行 63 列表示。

视频 RAM 地址的多路转换器，当每个引出脚 1 上都为逻辑 1 的时候，就把由同步信号发生器来的地址信号送出去。现在由 CPU 所选择的是相当于视频 RAM 的地址，这时的信号 VID* 要变成 0 状态，于是从 CPU 来的地址信号就送到视频 RAM 的各个地址中去，从而进行视频 RAM 的写入或者读出。

CPU 向视频 RAM 里作写入或者读出结束之后，信号 VID* 重新变成逻辑 1 状态，把从同步信号发生器来的地址信号送出去。由于 CPU 进行这样的操作所占的时间非常之短，所以其余大部分时间是用在把原来的 RAM 的数据变换成电视上的字符显示。

这里要提及的是 Z_{49} 的输入端 13 和 6，它们是通过电阻接到电源 V_{cc} 上的。

当采用同步信号发生器来驱动视频 RAM 的地址时，上述 Z_{49} 的输出端 12 和 7 的输出最好是理解为只用于显示的信号。

另一方面当 CPU 的控制信号到达时，则它们就用来读写视频 RAM。

写入时，信号 WR* 从引出脚 14 进去，当引出脚 1 为 0 状态时，就是说由 CRU 来控制 Z_{49} 作切换的时候，从 Z_{49} 的引出脚 12 输出信号 VWR* 把写入信号送到视频 RAM 的引出脚 3 上，由 CPU 来进行写入操作：

读出时，RD* 信号由 Z_{49} 的引出脚 5 进去，由引出脚 7 输出变成信号 VRD*。该信号把视频 RAM 的输出缓冲器 Z_{66} 和 Z_{44} 的门打开，从而将数据总线上的数据送到 CPU 里。

4）字符的显示方法

在电视的显示器上，电子束从上到下；从左到右在不断地运动着(我们称为电子扫描)，其结果就可扫成一幅画面。我们把从左向右运动的一根线称为扫描线。在一幅画面上共用264根扫描线。其中有 192 根扫描线用来作画面显示；剩下的 72 根扫描线被用在画面的上下空白部分及各行字符之间的垂直间距上。所以这 72 根扫描线什么也不表示。

要作出 1024 个字符显示的话（如果出现信号 MODESEL 即作 512个字符显示），每一行就有 64 个字符（或者是 32 个字符），在整个画面上共有 16 行。每一行共用 12 根扫描线，其中用来作字符显示的有 7 根，其余 5 根使用在行与行之间的间距空白上。

同步信号发生器中的计数器 Z_{65} 与 Z_{50} 是产生电视画面上的纵向地址信号的。Z_{32} 则是用来产生字符的 16 行信号的。而 Z_{12} 是用来产生一行字符的扫描线的。

Z_{12} 是一个 1/12 分频的计数器，它的输出形成所谓信号 L_1、L_2、L_4、L_8。这四根输出线并非用在视频 RAM 的地址线上，而是用于在每一行上产生字符的。

信号 L_1、L_2、L_4 分别被送到产生字符用的 ROM Z_{29}(MCM 6670P)的 RS1、RS2、RS3 上，以用来确定字符的图形。L8 则是用来给出各行字符之间的扫描线的间隔。

信号 HDRV 与 VDRV 从门 Z_{30} 的引出脚 8 与 9 进去，其输出由引出脚 10 出来。这个输出称为信号 BLANK*，是用来给出 72 根空白扫描线的信号。与此同时，它也作为信号用来给出在画面的左右两边所存在的空白部分。

5）视频 RAM

因为视频 RAM 是属于静态方式，所以就不必进行类似动态方式那样的更新。它与系统中的 RAM 同样地被接到数据总线上。

在视频 RAM 里共使用 7 个 1024 比特的 RAM。其中有 6 个用在 ASCII 代码上，剩下的一个是作为图符与字符之间的转换用的。

沿着所画出的线路可以发现信号 BIT6，它是 Z_{30} 的引出脚 13 的输出。该输出只有当信号 BIT5 与 BIT7 为 0 时，其输出端 13 才出现状态 1，这是一个使用 6 个 RAM 来产生 ASCII 代码的回路。

6. 视频信号产生概述

用来产生视频信号的共有五个部分，它们分别为数据闩锁、字符或图符的产生部分，移位寄存器、同步发生器以及视频合成器等。

数据闩锁是用于暂时存放 ASCII 代码或者图符数据的部件，把视频 RAM 里的数据读出来放到数据闩锁里，之后视频 RAM 就转入到取下一个数据的周期。

产生字符的 Z_{29}（MCM6670P），可以认为是数据闩锁与信号 L_1、L_2、L_4 的地址 ROM，它按所指定的地址产生字符图形。

产生图形符号的是由一个六进一出的多路转换器 Z_8 来实现的。其过程是把放在视频 RAM 里的 6 个比特的信号变换成图形符号的图象。

移位寄存器是并行接收产生字符用的 ROM 的信号，并且将它们变换成串行信号。

同步发生器部分。它首先是接收用于产生同步信号的计数器的输出，然后再把它合成作电视显示所需要的同步信号。

这样作的结果即把同步信号以电气条件合成为视频信号，于是通过阻抗为 75 欧的电缆输送出去。该信号供电视显示器作字符或者图示符号的显示。

1）数据闩锁

ASCII 代码和图形符号的数据是从视频RAM里读出，暂时存放在闩锁回路Z_{28}、Z_{27}里。

Z_{28} 的输入为 6 个比特的数据。Z_{28} 的闩锁输出是送给产生字符图形用的 Z_{29}和作图形符号用的Z_8。

Z_{28} 的输入控制即为它的引出脚 9（闩锁时钟）和引出脚 1（信号VCLR*）。引出脚 9 的输入是由 Z_{24} 的引出脚 3 的输出 LATCH 加上去的(图 5 中的信号LATCH即G、L)。当信号LATCH 从低电平变成高电平时，就把视频 RAM 的输出读入到 Z_{28} 里。

视频 RAM 的输出，在这时就移到下一个字符读出的周期里。一般来说，RAM 为了进行下一个数据的读出，需要有一定的延迟时间。我们把这个时间称为“存取时间”。它指的是从改变地址开始到它所要显示的下一组的地址数据为止的时间间隔。

这样，把视频 RAM 的输出闩锁之后，同步信号发生器的地址就开始改变。因此在采用 64 个字符显示的情况下，用在存取上所需要的时间约为 560ns 以上。

Z_{27} 的闩锁，相对于 Z_{28} 的输入为 6 个比特来说，它只有 4 个比特。Z_{27} 的引出脚 4 的输入是视频 RAM Z_{63} 输出的倒相信号。Z_{27} 的引出脚 5 是与信号 BLANK* 相连接的。其引出脚 12 是与信号 L8 相连。Z_{27} 的引出脚 13 被接到产生 ASCII 代码回路的输出 Z_{30} 上，即为信号 BIT6 。

这里由于 ROM 的输入形式也必须是 ASCII 代码才能实现字符图形乃至视频信号的变换。对此要运用一点技巧。

Z_{27} 的输入端 4，是决定 Z_{28} 中的数据是作 ASCII 代码用还是作图示符显示用的信号。Z_{27} 的引出脚 5 的输入是 BLANK* 信号，如同前面所说的那样它是用来给出画面的上下，左右空白的信号。当 BLANK* 信号为逻辑 1 的时候，就作画面的显示；而当 BLANK* 信号出现 0 的时候，则在这个时间内在画面上不显示任何信息。Z_{12} 的输出端 11 被接到Z_{27} 的引出脚 12 上，即信号 L_8 线，它的作用类似于信号 BLANK*。

信号 L8 是控制字符行间扫描线的。当信号 L8 为 0 状态时，就可以作字符显示，而当

它呈现逻辑 1 的状态时，则行间就出现空白。Z_{27} 的输出端 15 是接到产生字符的 ROMZ_{29} 的引出脚 1 上。

在两个数据闩锁的引出脚 1 上，它们的输入信号若为 0 的话，则闩锁里的内容就被请除掉。我们把这个信号称为 VCLR*（Video CLeaR、视频清除）。它是由触发器 Z_7 的输出端 6 接过来的。由于该触发器的引出脚 4 是信号 VID*，所以信号 VID* 为 0 状态时，即 CPU 向视频 RAM 里写入或读出时，Z_7 的输出端 6 也变成 0 状态，因此使两个数据闩锁内容都被清除。

CPU 的操作结束时，Z_7 的引出脚 4 出现逻辑 1 的状态，又重新转入正常工作。

像这样的回路由 CPU 来进行操作，在画面上是不会出现“浮动”现象的，同时用它又可以消除字符显示上的重影之类的现象。

2）字符的产生

在电视画面上所显示的字符都是由点子所组成的。一个字符是由横 5 个，纵 7 个点子组成。字符与字符之间的间隔，是根据所显示的字符数而有区别。

由 5×7 个点子所组成的这些字符的图形数据是放在 ROM Z_{29} 里的。

ASCII 代码的地址，要对应于它的代码选择字符。相当于 5 个列上的点子，要同时地从 Z_{29} 里读出信号 D_0、D_1、D_2、D_3、D_4。由于一个字符是由 7 行、5 列的点子所组成，所以要选择那一行的问题，是根据信号 RS1、RS2、RS3 来决定的。这些信号线分别接到 Z_{29} 的引出脚 8、10、11 上。当要显示字符的第一行的点子时，它们都为 0 信号；当要显示字符的第二行的点子时，它们的输入信号就为 0、0、1 值。

因此，就电视信号的扫描来说，首先形成字符列中的起始行扫线，然后从 5×7 的字符上开始把每列的点子从 RAM 里一行一行地读出来。这样，按 64 个字符显示的时候，每回扫一行需要变换 64 次地址，方能从视频 RAM 里把一行ASCII 代码读出来。

为了完整地显示出每一个字符，就需要 7 根扫描线，所以一个字符要经过 7 次扫描才能从视频 RAM 里把每一行的 ASCII 代码读取出来。

至于取出字符的哪一行的问题。这里为了与电视扫描线相一致，我们可以利用 ROM 的信号 RS1、RS2、RS3 的输入端，按顺序地把二进制的地址数据输入到 ROM 里。当扫描线完成 7 行的扫描之后，即认为每个字符的图形被完整地显示出来。

3）图形符号的显示

在 TRS-80 里也能作图形符号的显示。它类似于前面所提到的那样用 16 行×64 列个字符的显示方法。但是在图形的显示里，是把整个画面分成 16×64个四角形，而每个四角形又进一步被分解成如图 7 所示的 6 个小四角形。

这个相当小的四角单元，其大小是由横向 3 个点，纵向 4 根扫描线组成的。所以其整个画面就变成具有16行×3＝48 行；64 列×2＝128 列的图形显示单元。

Z_8 是用作图形发生器的门电路。

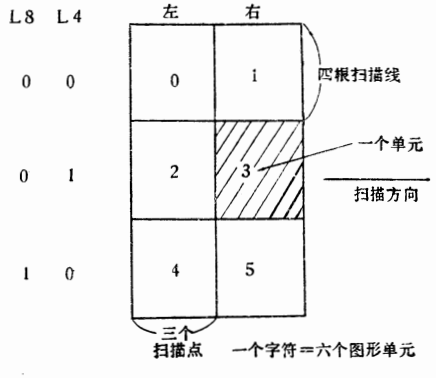


图7 图符显示单元

放在闩锁 Z_{28} 里的 6 个比特的数据构成 Z_8 的输入。Z_8 的输入端 2 与 14(A、B)上还输进信号 L_8 与 L_4。由于信号 L_8、L_4 加给 Z_8 的是二进制 00、01、10、11 的图形信号，所以根据这些信号来进行输入数据的转换。但是，11 的图形信号在 Z_{12} 里是不输出的，结果分别给与 Z_8 的就只有信号 00、01、10 值。详细表示由图 7 中示出。

图形单元上的图形是根据数据闩锁的值是 0 还是 1 来作显示的。对于 Z_8 的输入信号我们令它们为信号 LB0、LB1……LB5。Z_8 的输入端 6 是相当于图形单元的左侧；而输入端 10 则相当于图形单元的右侧。

因此，由 CPU 送来的数据最高位的比特为 1 时，那么从最低位的比特起在图 7 所示的具体位置上，出现逻辑 1 或者 0 时，就定出每一个字符以 6 个图形单元进行表示。

4）移位寄存器

Z_{10} 是字符图形用的移位寄存器，而 Z_{11} 是图形符号用的移位寄存器。它们两者都是并行地接收数据，并且把数据变换成串行信号，再通过称为 SHIFT 的信号，逐个地送出去形成视频信号。

这里首先介绍字符图形用的移位寄存器。

产生字符图形的条件是：

(1) 画面必须具有消隐的条件，即在垂直方向与水平方向的空白位置上不存在图象(BLANK)。

(2) 要有字符产生 (BIT7 信号为逻辑 0) 的条件。

(3) 要产生 7 根字符的扫描线(L_8)。

使这些显示条件得到满足的部件是门 Z_{26}。在门 Z_{26} 上还得加上信号 LATCH，该信号是把数据从数据闩锁转移到移位寄存器里。

这时候，在 Z_{10} 的输入端上送进来脉冲信号 SHIFT，Z_{20} 的输出信号就立刻移到移位寄存器里。当信号 LATCH 为逻辑 1 时，移位寄存器没有输出；当信号 LATCH 为 0 时，即 Z_{26} 的输出变为逻辑 1 时，从 Z_{10} 的输出端 13 上把该移位寄存器的数值依次输出出去。这时的输出速度是由其引出脚 7 的时钟脉冲决定的。

Z_{10} 的引出脚 9，即清除端是通过 4.7KΩ 电阻接到 +5 伏的电源上。如果该引出脚 9 的状态为 0 的话，则在画面上什么也不显示。引出脚 14、3、2、1以及 6 都被串在一起接地。引出脚 14 是为了空出字符与字符之间的间隔而使用的。而引出脚 2、3 在这里没使用。

Z_{11} 是图形显示用的移位寄存器。它的动作大致与 Z_{10} 相同，但其产生图形的条件有些不同。

Z_{11} 与 Z_{10} 的情况相同的是，必须满足上述的条件 (1)。这就是加到门 Z_{26} 的输入端 1、2 上的 BLANK 条件。此外，与条件 (2) 正相反的，即当信号 BIT7 为 1 时的条件，这就是将信号 BIT7 变成门 Z_{26} 的引出脚 4 的输入，用于产生图形显示。Z_{26} 的引出脚 6 的输出是接到 Z_{11} 的引出脚 15 上，这样就为 Z_8 中的数据向 Z_{11} 中转移作准备。

该图形符号如前所述，每个字符是由 6 个比特的图形来决定的。但是在每个图形符号的上下左右是不存在间隙的。图符的图形是从 Z_{11} 的引出脚 13 上被送出去的，在门 Z_{30} 上与字符图形进行合成。在 Z_{11} 中所空余的引出脚，与 Z_{10} 的移位寄存器一样被接到电源 +5 伏或串在一起接地。

5）同步信号

对电视信号来说，所需要的同步信号是利用产生同步信号的计数器的输出合成出来的。

CMOS 变换器Z_6是用于产生水平同步信号的回路，而Z_{57}是用于产生垂直同步信号的。

Z_{50}的引出脚 11 的输出信号 HDRV，经过缓冲器 Z_6 加到 100KΩ 的可变电位器 R_{20} 上。当电阻值很小时，也就是说使它接近于 Z_6 的引出脚 2 时，信号 HDRV 的脉冲前沿就越陡；相反地增加电阻值时，则其脉冲的建立时间就越长。

这个延长的时间不仅仅决定于电位器 R_{20}，也与电容器 C_{20} 有关。如果信号 HDRV从 0 值变到逻辑 1 时，Z_6 的引出脚 2 上的输出就变得接近于 5 伏了。

这个电压通过电位器 R_{20} 的电流给电容器 C_{20} 充电。于是 Z_6 的引出脚 3 上的电压从 0 开始逐渐上升，当它达到逻辑 1 的状态时，使 Z_6 的引出脚 4 成为 0 状态，并使引出脚 6 呈现逻辑 1，于是加速了 C_{20} 的充电。

信号 HDRV 在为逻辑 1 的状态之间，Z_6 的引出脚 2 上的输出继续保持高电平，一旦信号HDRV 再次变为 0 时，C_{20} 即可开始放电，与前述过程正好相反。

这样所形成的水平同步信号位置是由电位器 R_{20} 来决定的，因而电视画面上的图形左右位置变动，根据这一点就能够确定的。同步信号要改变它的相位，则是通过 C_{21} 和 R_{42} 作微分，这样就使改变后的信号比信号 HDRV 延迟了一定的脉冲宽度。

将这个脉冲宽度相当窄的信号加到 Z_6 的引出脚 11 上。该信号经过两次倒相而变成的 4μs脉宽的水平同步信号加到 Z_5 的引出脚 13 和 2 上。

关于垂直同步信号的产生与水平同步信号的产生方法完全一样。这时的时间延迟是由 R_{21} 和 C_{26} 来决定的，而输出的同步脉冲宽度则是由 R_{44} 和 C_{27} 来决定的。由于所使用的电容器的值不同，其输出的脉冲宽度也不同。

6）水平同步信号与垂直同步信号的合成

上述所形成的水平同步信号与垂直同步信号是通过门电路 Z_5 合成的。图 8 中给出门电路 Z_5 的各部分的信号。

图 8 中的波形 A 是 Z_6 的引出脚 8 的输出信号。波形 B 是 Z_{57} 的引出脚 8 的输出信号。它们分别表示水平同步信号和垂直同步信号。

Z_5 的输出端 3 是取 A 与 B 波形的与非结果，其输出即为图 8 中的 C 波形。Z_5 的引出脚 11 即为 D 波形，引出脚 6 即为 E 波形。把这些输出送到引出脚 10 和 9 上，最后在 Z5的引出脚 8 上就得到象图中的 F 所表示的合成同步信号的波形。

7）视频信号的合成

视频信号的合成回路，是用来把字符图形或者图形符号与同步信号进行合成的回路。其合成的结果再通过 75 欧的电阻送到电视显示器上。

从移位寄存器 Z_{10} 与 Z_{11} 上输出的信号在或门电路 Z_{30} 里进行合成。当然这两个信号并非同时地进入 Z_{30} 里。

Z_{30} 的输出端 1 的输出送到 Z_{41} 的输入端 6、7 上。Z_5 的输出端 8 所合成的同步信号被加到晶体管 Q_2 的基极上。一旦 Q_2 导通，则通过电阻 R_{28} 使 Z_{41} 与晶体管 Q_1 的基极都被加到约 5 伏的电源上（实际上由于 Q_2 的饱和电压输出比 5 伏稍微低一些）。

Z_{30} 的输出端 1 的数据经过 Z_{41} 倒相之后，其信号在引出脚 5 上反映出来。元件 Z_{41}根据由两个移位寄存器输出的数据，起到类似开关的动作。有关上述这些关系，让我们简单地

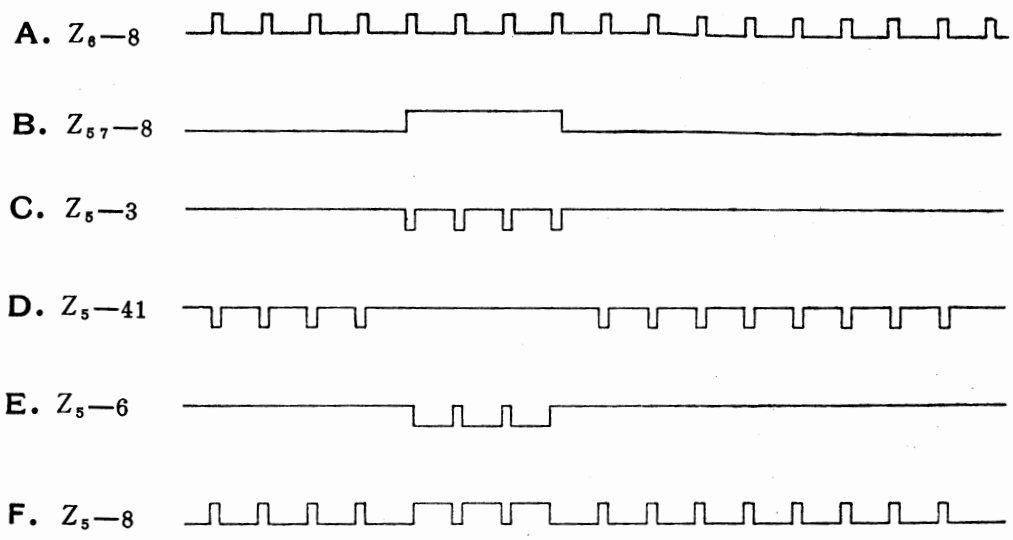


图8 同步信号的合成

用图 9A 所示出的合成视频信号的阻抗网络来加以说明。

用图 9A 来说明信号合成的话，Q_2 与元件 Z_{41} 就表示为简单的开关。如果 Q_2 是截止的，由于 R_{28} 上没有电压，所以其输出即为 0 。而当 Q_2 导通时，Z_{41} 也导通时，其输出就变成约为 1.23 伏电压，即相当于“暗电平”。这时候把用于显示的信号向 Z_{41} 输入进来时，且 Z_{41} 是导通的，那么其输出就约为 2.75 伏。

若是把这个信号送出作为电视显示的话，由于阻抗太高而采用 Q_1 的射极跟随器形的缓冲器作为它的输出。该 Q_1 的射极端的输出波形在图 9B 中示出。

电容器 C_7、C_2 以及电阻 R_{30} 是 Q_1 的滤波电路，即使电路的输出部分的偏压有变化，Q_1 的损耗也不至于过大，故此处插入电阻 R_{30}。

7. 键　盘

TRS-80 的键盘，共有 53 个单极开关。通常都是处于开启状态，只要按下塑料按键即可接通。它们都安装在键盘的印刷板的上面。

该键盘输出的 ASCII 代码与普通所使用的方式不一样。它的开关被安排成一个矩阵的形状。按下某一键钮，就通过 ROM 中的软件来作调动，再由 CPU 把键盘的输入动作变换成对应于那个键的 ASCII 代码。

键盘的选择，是由译码器的输出信号 KYBD* 来执行的。这个信号为 0 时，Z_3 与 Z_4 的三态元件都处于等待状态。但是在未按下按键时，它们的输入全部通过 4.7KΩ 的电阻接到 5 伏电源上。

一旦信号 KYBD* 出现 0 时，全部地址线都变为 0 状态。这时候无论按动哪一个按键其数据的输出端都表示为“逻辑 1”，所以，下面为了检查按哪一个键钮，在哪一

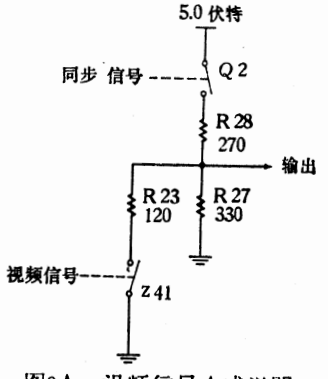


图9A　视频信号合成说明

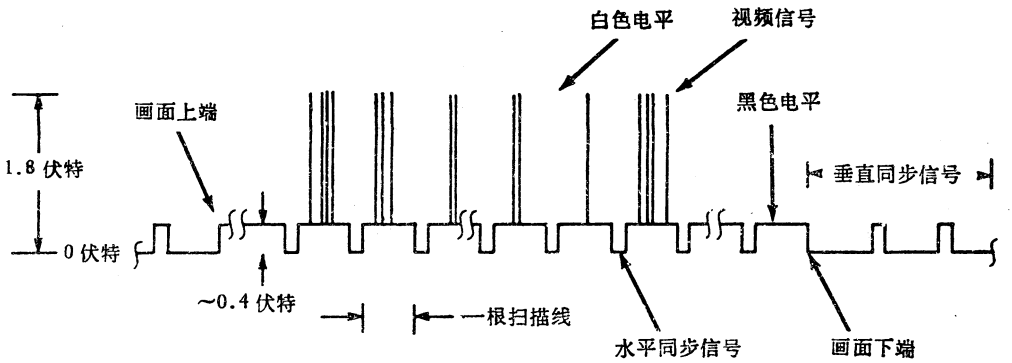


图9B　视频信号合成说明

个数据线上有信号输出这一情况，只要顺次地改变地址的输出就可以知道。

同时要检查一下按动移位键的情况，是否由CPU把它变换成与其相对应的ASCII代码。

这里必须注意的是，在 Z_1、Z_2 上所使用的是集电极开路的集成回路。因此，只要不按动按键，地址线上就不会有输出。

8. 输入与输出接口（Port）

TRS-80 的存贮器是采用分割方式的。不过也可以采用输入或输出接口方式。存贮器的分割方式与接口方式之不同点，表现在存贮器的分割方式里它可直接与 CPU 进行数据交换，而采用接口方式时，CPU 则不能直接区分使用哪一个接口送数。

如果在 TRS-80 里直接地把存贮器划分成输入和输出的话，那么 CPU 就能够知道输入数据是产生在哪一个接口上，并把它们的数据接到数据总线上，而且通过数据总线 CPU 就能把数据取出来。

TRS-80 最多能接 256 个输入或输出的接口，而在该 TRS-80 机系统里只使用一个接口是连接在盒式磁带录音机上，以 16 进制数码分配在 FF 的地址范围里（输入输出接口是由低 8 位的地址决定的）。

1）接口地址的规定

TRS-80 的输入或输出的接口只采用一个，所以只要一个接口译码器就足够了。该接口译码器由分图 2 上面的 Z_{54} 表示。

Z_{54} 通常是用来检查从 A_1 到 A_7 的地址数据的；Z_{52} 的引出脚 5 则是用于检查地址 A_0 的。如果采用 16 进制的数码输出地址 FF 的话，即 A_0～A_8 全都是逻辑 1 的话，那么 Z_{54} 的引出脚 8，Z_{52} 的引出脚 6 就呈现 0 状态。Z_{36} 的引出脚 3 的输出也呈现 0 状态。由此即实现对接口的译码。

这时候信号 OUT* 为 0 状态时（即 CPU 把信号送到输出接口时，该信号为 0），Z_{25} 的引出脚 8 的信号 OUTSIG* 就变成 0 状态。

信号 IN* 为 0 时（接收 CPU 送到输入接口的信号时，该信号为 0），Z_{25} 的引出脚6的信号 INSIG* 就变成 0 状态。由于信号 IN* 和信号 OUT* 不能同时为 0，所以信号 INSIG* 或 OUTSIG* 就不会同时动作。

2）信号 OUTSIG*

信号 OUTSIG* 是用来实现对盒式录音机和视频信号功能的两种控制的。

在盒式录音机的控制里有一个信号CSAVE,它是用来控制盒式录音机的马达的。而在视频控制方面有一个控制信号MODESEL(Mode Select)，它是被用来选择64个字符表示或是32个字符表示的模式选择信号。

信号OUTSIG* 也被使用在Z_{24}的与非门的闩锁控制上，而这一点将在后面介绍。

Z_{59}是通过信号OUTSIG* 控制的闩锁器。在该闩锁器上连有数据信号D_0～D_3。D_0和D_1被接到Z_{59}的引出脚4与5上。这两个数据的输入是根据信号CSAVE的控制而使用在磁带录音机的记录上面。D_2被接到Z_{59}的引出脚12上。它是用于控制录音机的马达的。

Z_{59}的输入闩锁是用信号OUTSIG* 的前沿工作的。当输入信号D_2为逻辑1时，信号OUTSIG* 从0上升到逻辑1，而Z_{59}的输出端10就变为逻辑1，于是录音机的马达就开始旋转。相反，输入信号D_2为0时，同样的信号OUTSIG* 作正跃变送来，录音机的马达即停止旋转。

3）盒式录音机的马达控制

一旦开始执行信号CSAVE的功能，盒式录音机的马达就一定开始旋转。

CPU给出信号OUTSIG* 为0；D2上输出数据为“1”。OUTSIC 信号从0变到逻辑1时，信号D_2的“1”就从Z_{59}的引出脚10上输出。这个信号被送到Z_{41}继电器的前置放大器上。由于Z_{41}的引出脚3的电平下降到0，在继电器的线圈K_1内流过电流，使K_1的触点闭合。这样一来插座J_3上的接点1和3被接通，与该插座相连接的录音机马达就开始旋转。

当线圈K_1上的电流被切断时，会产生高电压，这时用二极管CR_3可以保护Z_{41}不致被损坏；另一方面用二极管CR_3也是为了防止在继电器的触点上出现电振荡现象。

齐纳二极管CR_9、CR_{10}是与CR_3同样的原因被接到回路里。当把录音机的马达开关接通时，由于会产生高电压，有可能使继电器的触点燃毁。为此我们加进去齐纳二极管CR_9、CR_{10}，用以保持稳定的电压值。

4）盒式录音机的音频输出

录音机的马达开始旋转时，CPU向磁带里送出要作记录的信号。用来进行这种记录的定时信号，完全由软件来控制的。

Z_{59}接收来自CPU的输出，把该输出信号进行合成。从CPU来的数据是从数据线D_0与D_1送到Z_{59}的引出脚4和5里，然后再转到引出脚2和6上，由R_{53}、R_{54}、R_{55}、R_{56}所组成的阻抗网络输出。该输出记为信号CASSOUT，其信号波形表示在图10中。

图10是Z_{59}的引出脚2与6两者输出的合成信号波形。

T_1所示的区间约为0.46伏，这是表示引出脚2为0，而引出脚6为逻辑1时的电平($D_0=0$，$D_1=0$)。

T_2所示的区间是引出脚2与6都为逻辑1时的情形。($D_0=1$，$D_1=0$)。而T_3的区间是引出脚2，引出脚6都为0时的电平($D_0=0$，$D_1=1$)。

我们把这种波形叫作“数字正弦波”。

在图10里，从某一个比特到下一个比特的间隔约为2ms；信号是“0”还是“1”，是决定相邻两个比特之间有无信号的。

当CPU的输出为“1”的时候，从第一个起动比特开始，经1ms之后才出现另外的信号。紧接1ms产生第二个起动比特脉冲。当下一个比特信号为“0”的时候，经2ms就把第二

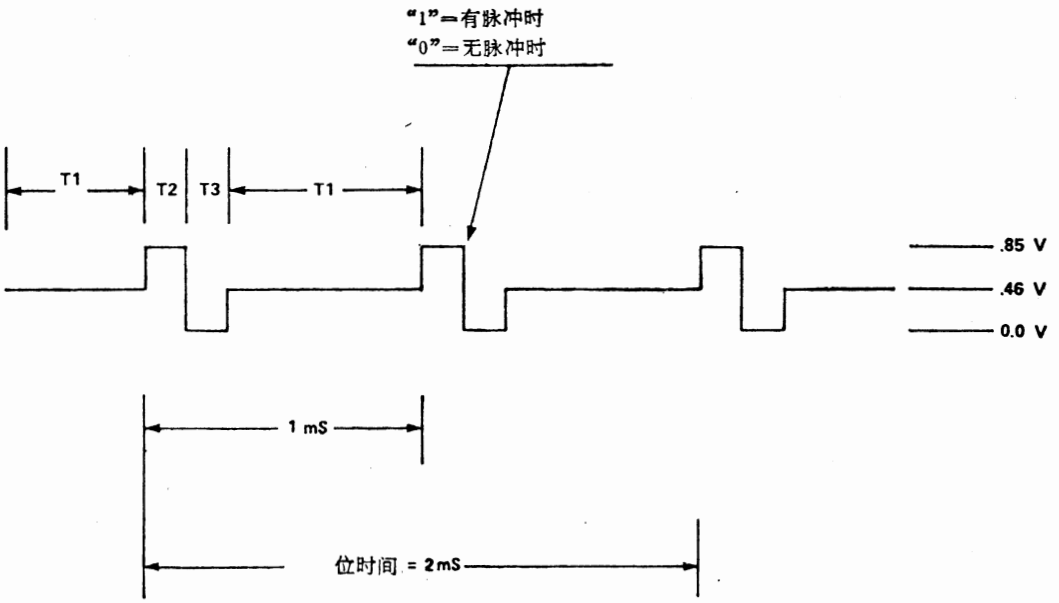


图10　信号 CASSOUT 的波形

个起动比特脉冲送出去。

这个信号是从 J_3 的接点 5 上被输出去，所以从盒式磁带录音机的输入AUX 上即可以作录音。

根据信号 CSAVE 的功能，CPU 对 RAM 的操作命令就能够转送到盒式磁带录音机里。如果转送结束，则 D_2 的输出为 0，录音机的马达即可停止。

把数据从 CPU 里送出去时，要进行如下操作。

首先给出信号 CSAVE，通过 Z_{59} 的输出先把 128 个"0"比特信号送出去。其次送出用在信号 CLOAD 上的同步代码 A_5。再次送出 2 个字节的起始地址和 2 个字节的终止地址。此后把所需要的数据送出去。送数据结束时，最后要附加一个字节的校验总和。

该校验总和是取全部数据的简单和，所以当出现信号 CLOAD 时，要检查该校验总和以确定它与信号 CSAVE 是否一致。如果出现错误，不是信号 CSAVE 就是信号 CLOAD 产生的。

5）盒式录音机的音频输入

如果正确地记录实际的输出波形，采用的是盒式机的读出方式来表示的话，那么这里所叙述的回路似乎是不必要的。但是，在一般情况下记录实际的信号受盒式机输出的限制。因为在盒式录音机里经常加进去一些电源的哼声以及噪声等干扰。

当执行信号 CLOAD 时，盒式机的马达旋转，从插头 J3 的引出脚 4 上得到盒式机的音频信号。该信号即称为信号 CASSIN，它被送到电容器 C_{24} 与电阻 R_{67} 的交点上并经音频信号回路 Z_4 进行处理。

Z_4 的引出脚 1、6、5 的回路是由音频放大器组成的有源滤波器回路，在该回路里将滤掉噪声以及哼声等不需要的信号。这是一个带宽大约为 $2KH_2$ 的高通滤波器。

将信号 CASSIN 放在示波器上进行观察时，就能发现到上面所提到的噪声和哼声。通过这个高通滤波器之后，就能把这些哼哼声等不良现象消除掉。这时的信号电平大约以 2 伏为

中心线的。图 11 中表示了盒式机信号的各部分的处理波形。

A 波形为 Z_4 的引出脚 5 的波形。Z_4 的后一部分是作为有源整流器来使用的，即用二极管 CR_5、CR_4 作全波整流。C07 点的波形即为图中的 B。

上述的 B 信号波形进一步送到 Z_4 的引出脚 8 与 13 上，经过倒相放大之后的波形在引出脚 9 上得到。R_{41} 与 R_{42} 的电阻是为了获得大约为 2 的增益而加进去的。该输出波形表示为图中的 C。

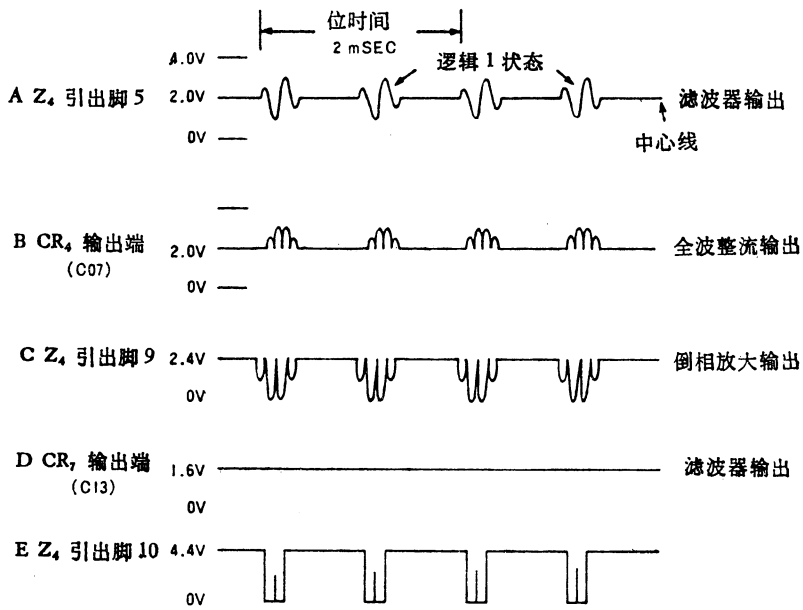


图11　音频输入信号的处理

Z_4 的最终端是电平检波器 CR_5、CR_4、CR_9。它是相当于某种电源。Z_4 的引出脚 9 的输出被加到 CR_6 上。这时候 CR_6 与 CR_7 把输入电压降到 0.8～0.9 伏。

C_{39} 是起滤波作用的，象图 11 中的 D 所表示的那样它形成具有一定的直流电平。当 Z_4 的引出脚 9 的输出信号下降时，信号从 C_{39} 的标准电平开始降下来，于是 Z_4 的引出脚 10的输出也下降。当 Z_4 的引出脚 12 的输入在标准电平以下时，Z_4 的引出脚 10 的输出保持在原有的电平以下。

由于 Z_4 的引出脚 10 的电平降低，使 Z_{24} 的触发器工作，此后数据的读出则由编入ROM里的软件内容通过 CPU 来执行的。

6) 信号 INSIG*

在 TRS-80 里用于读取盒式机数据的硬件要尽可能少些。

Z_{25} 的引出脚 4 被接到信号 IN* 上。CPU 把用于作读写的地址指定在 FF 范围内。信号IN* 有输出时，信号 INSIG* 就变成 0 。

Z_{25} 的引出脚 6 的输出信号是仅用于控制 Z_{44} 的。Z_{44} 的输入端被接到由与非门 Z_{24}所组成的触发器输出端 8 上。当 Z_{24} 的引出脚 9 出现 0 时，引出脚 8 就为逻辑 1，于是该触发器回路就处于置位状态。

要使触发器 Z_{24} 复位，即是信号 OUTSIG* 出现 0 的时候。Z_{44} 是用信号 INSIG*的控制来检查 Z_{24} 的状态。

下面介绍信号 CLOAD 是怎样进行工作的。

现在把从键盘来的 CLOAD 的命令作为输入并使信号 OUTSIG* 为 0，起动盒式录音机马达同时使 Z_{24} 触发器复位。

当最早的比特信号到达时，Z_{24} 的引出脚 9 出现 0 使触发器置位。

图 12 中的 A 是表示 Z_{24} 的引出脚 9 的输入波形。D 表示闩锁的输出波形。B 是信号 INSIG* 的波形。C 是信号 OUTSIG* 的波形。信号 INSIG* 与信号 OUTSIG* 是根据 ROM 中的软件作控制输出信号。

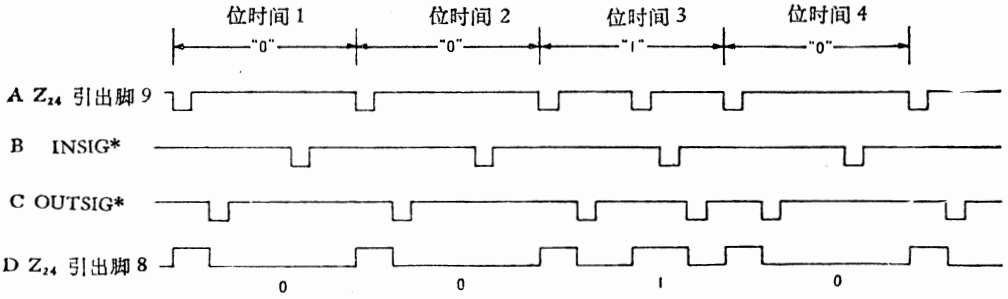


图12 数据闩锁的时间图

CPU 给出信号 INSIG* 是对 Z_{24} 的引出脚 8 的输出状态作检查的。在 D 的波形里，引出脚 8 为 0 即可以知道其数据即为 0。在位周期的起始点上，触发器被置位，而当出现信号 OUTSIG* 时它又被复位。如果用信号 INSIG* 通过 Z_{44} 作检查时发现仍为 0，则 CPU 就判定它的值为 0。

但是，在位周期 3 里面给出信号 INSIG* 作检查时，由于此时 Z_{24} 的引出脚 8 变为"1"，所以 CPU 即可知道它的数据为"1"。这时候 CPU 必须使触发器 Z_{24} 复位，于是在位周期 3 里便送出表示复位的信号 OUTSIG*。在位周期 4 里又重新读出 0 信号值。

以上这种操作一直执行到读完 CLOAD 的校验总和为止。

9. 系统的电源

在 TRS-80 里，需要如下三种类型的电源。

(1) +12 伏，350 毫安 (2) +5 伏，1.2 安培 (3)−5 伏，1 毫安

+12 伏与−5 伏的电源是系统中 RAM 所需要的电源。其余大部分都是采用 +5 伏的电源。+12 伏与 +5 伏的电源，都是采用稳压电源并且必须有电源的短路保护。−5 伏电源与上面那两种电源比较起来不那么重要，但都得通过齐纳二极管进行稳压。

交流电源或者是未经稳压的直流电源分别由交流耦合或整流后供给的。

1) 变压器

变压器是由一个初级线圈和两个次级线圈组成。该变压器是放在一个塑料盒子里。初级侧的工作范围是在 105～135V，其保险丝放在初级一侧。

次级的两个线圈一定要有中心抽头，其中一个输出为 1 安倍 14 伏的电压，被用来作 +5 伏和−5 伏的电源。另一个输出是在其内部接上二极管，其输出为 19.8 伏电压，350 毫安电流。该回路用作 12 伏电源。

这个电源连同中心抽头一起从插座 J_1 输入进去。

2）＋12 伏电源

所用的＋12 伏的输入电源虽然经过整流，但并未被稳压。该电源是从插座 J_1 的接点 2 送进去的。当接通开关 S_1 时，在滤波电容器 C_8 上加有约 20 伏的电压，该电压还加到 Q_6 上，同时也加到稳压用的 Z_2 的引出脚 12 上。

Z_2 的引出脚 12 的输入被加到恒流用的齐纳二极管 Z_a 上。Z_2 的引出脚 6 约输出7.15伏的齐纳电压。

图 13 是表示稳压器 Z_2、Z_1 用的标准化的集成电路。其引出脚 6 是通过电阻 R_{15} 被加到引出脚 5 上，而该引出脚 5 是用 Z_b 作表示的运算放大器的正端输入。

另外，运算放大器的负端输入是被接到电位器 R_{10} 的调节端上。在 R_{13}、R_{10}、R_{15}的阻抗网络上产生一个 ＋12 伏的稳定电压。

R_{18} 是限流电阻。

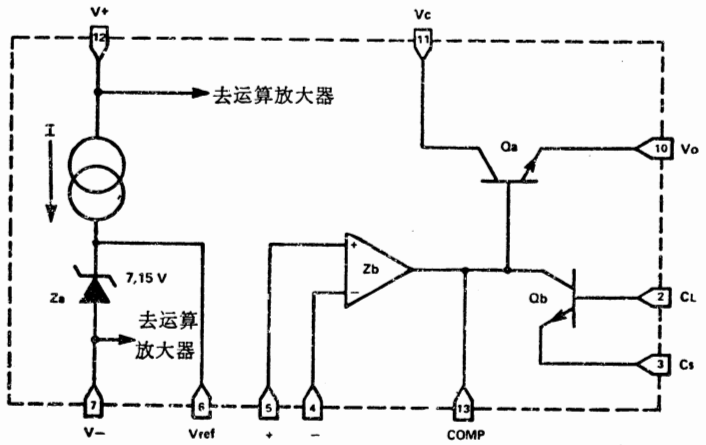


图13　723 稳压器的方块图

图 13 中的晶体管 Q_b 是用来保护晶体管 Q_6 的。

流经电阻 R_{18} 上的电流越多其两端的电压越高。当 Z_2 的引出脚 2 上的电压超过 12.6 伏时，Q_b 导通而 Q_a 截止，Q_6 也截止。

为了达到这样的工作状态，Z_2 的引出脚 10 上的电压将接近 14.7 伏，要把它稳压在 12 伏上，最大电流可达 480 毫安。

输出端短路时，由于瞬时电流使 Q_b 导通，而 Q_a 截止，Q_6 的基极电压接近其发射极电压，因而 Q_6 也处于截止状态。如果输出端未被短路即复原的话则重新返回到原来的状态。

在引出脚 13 上所接的电容器是作补偿频率用的，以防止运算放大器产生振荡现象。电容器 C_{11}～C_{15} 是起滤波与噪声抑制作用的元件。

电容器 C_{28}、C_{30}、C_{32}、C_{34}是分布在印刷电路板上。

3）＋5 伏电源

在＋5 伏的电源上也采用 723C 的稳压调整器。它使用的件数比 ＋12 伏的要多些，但其原理没有什么区别。

在 5 伏电源里，加在插座 J_1 的接点 1 和 3 两端的交流电压约为 17 伏。该交流电压通过 CR_3 全波整流。当接通开关 S_1 时，滤波电容器 C_9 的两端加上大约 7 伏电压。

供给 Z_1 的电源是从 12 伏电源的 R_{18} 的一侧取出。从引出脚 6 输出的齐纳 7.15 伏的电压，在电阻 R_6，R_5，R_{11} 的网络上作分压，通过 R_5 调整到 5 伏之后加到运算放大器的正端上。

运算放大器负端是通过电阻 R_7 加到 +5 伏上。运算放大器控制了晶体管 Q_a，而 Q_a 则控制 Q_3 偏压。

晶体管 Q_3 是用来限制晶体管 Q_2 的大电流，它是通过限制 Q_2 的集电极电流的电阻 R_4 而加到 +5 伏电源上。

电流的限制，在这里是由晶体管 Q_5 进行监视。当电阻 R_4 上的电流变大时，使 Q_5 导通，通过电阻 R_3、R_9 使 Q_b 上的电流增大。

Q_a 的输出使晶体管 Q_3 上的电流降低，其结果又使晶体管 Q_2 的电流降低，于是达到了限流目的。这一动作使电阻 R_4 的两端电压开始降到 0.6 伏，因而使工作电流限制到 1.82 安培。

C_{12} 电容器是被接到 Z_1 的引出脚 13 与 4 之间，它与电容器 C_{13} 起同样的作用。电容器 C_{10} 与 C_{14} 是输出滤波器。有 32 个 0.01μF 的电容器是用来抑制噪声的，它们分别被固定在印刷板的各处。

5 伏总线上的齐纳二极管 CR_1 是为防止 RAM 产生故障而加上的元件。如果由于某种原因使 12 伏和 5 伏的总线形成短路的话，则 CR_1 导通，使 5 伏总线处于限流状态。

由于齐纳二极管 CR_1 为 6.2 伏，所以把所加上的 12 伏电压也限制在 6.2 伏上，这样就不会损坏 TTL 元件。

通常 CR_1 是处于截止状态的。

当 +12 伏的电源不能正常工作时，+5 伏的电源就不能正常工作，这时在作调整的时候，首先要调整 +12 伏然后再调整 +5 伏。

4）－5 伏电源

－5 伏电源的电流很小。从 CR_8 的负端取出电压。接通开关 S_1 时在 C_1 的两端出现约 －11 伏电压。

该电压通过电阻 R_{19} 后被齐纳二极管 CR_2 稳压到 5.1 伏。电容器 C_4、C_3 是用来滤波和消除噪声等元件。从 C_{16} 到 C_{19} 的四个电容器也是一个噪声滤波器。

三、机器的调整和故障分析

1. 分解机器的方法

(1) 把计算机放在软垫上，再把键盘取下来。

(2) 取下箱底的六个螺钉。由于螺钉长度不一致，应记清楚，哪个螺钉应放在哪个位置。

(3) 将计算机恢复原来位置，小心地取下机箱的上半部分。

根据发光二极管（即指示灯）的安装形式有两种不同的底盘，分别用①和②两种办法来拆取：

① 发光二极管安装在面板上，用两根导线把发光二极管连接到键盘的印刷板上。借助尖嘴钳取下发光二极管的插座环，用类似铅笔的橡皮头将露出的发光二极管按下，并从塑料

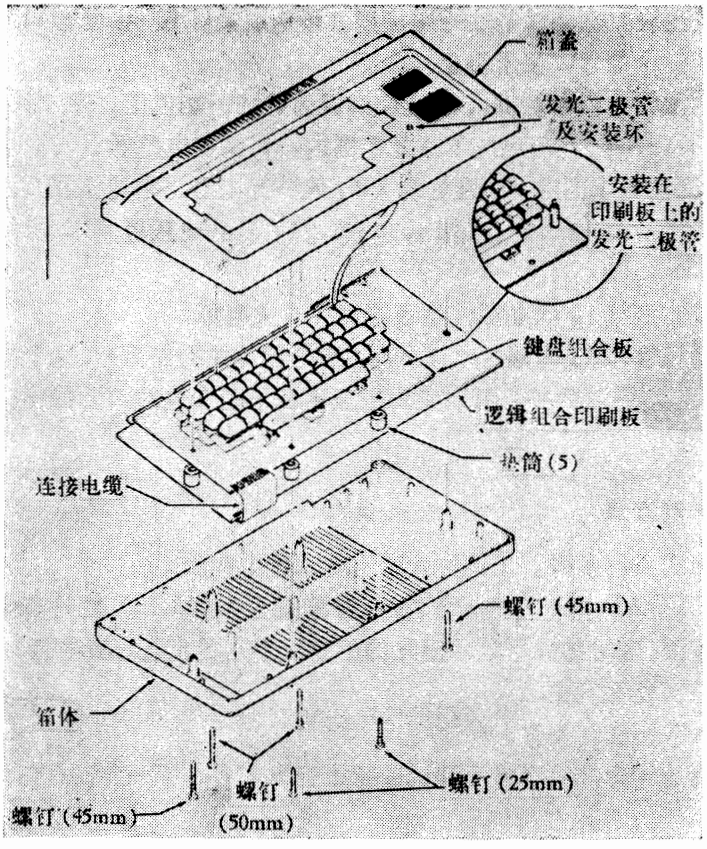


图14 TRS-8 分解图

箱上取下来。为小心保存插座环，发光二极管的引线要稍微弯曲一点为好。

② 发光二极管是直接焊接在键盘印刷板上的。在塑料面上恰好有一个窟窿露出发光二极管的头部。

(4) 将面板放在旁边，从底盘上慢慢的把键盘提取出来。这时请注意不要用力牵拉连接用的电缆。

(5) 在两块印刷板之间有五个橡皮垫筒把它们隔开。取下时请记住每个垫筒所对应的位置。

(6) 从箱内拿出这两块印刷板，一定要注意不要使连接用的电缆受力。

(7) 为了便于操作，两块印刷板要并排放置，但要注意不要使电缆受力。

2. 电源的检查及调整步骤

从箱体内取出印刷板以后，将电源与显示用的插头相连接。

(注：按照正确的位置，揭起键盘印刷板，底下的 CPU 印刷板就显露出来了。电源用的插头 J_1 与盒式磁带机用的插头 J_3 请不要接错，J_1 插头与电源开关挨得最近。)

请接通 CPU 印刷板的开关和显示器电源开关。 根据显示器有没有显示、有哪些显示等情况来估计故障的状况。

首先需要测试电源电压（请参考图 15 ）。

(1) 用数字电压表，把测试笔的一端接地。(接地端最好使用电容 C_9 的负端，因为 C_9 是印刷板内最大的一个电容，便于测量)。

(2) +12V 电源要检查电阻 R_{18} 靠近印刷板边缘的一端电压。该端电压必须为 12.0V ±5%（即11.4V～12.6V)。若电压不在此范围内，可调整电位器 R_{10}，使其成为正确值。

(3) +5V 电源请测量电阻 R_4 的左端（R_4 为一瓦电阻，它连接在 C_8 和 C_9 两个大电容之间)。该电压必须为 5V ± 5%（即 4.75～5.25V)。电压范围不准时，调整电阻 R5 使其达到正确值。

注意一定要首先调整 12 伏电源，然后再调整 5 伏电源。

(4) 请检查稳压管 CR_2 两端的电压(CR_2 在电源开关的左面)。这个电压值必须为 −5V ±5%。由于该电压不可以调整，所以实测电压不在此范围内时，要找出有故障的部件，并将其更换。

3. 寻找故障的方法

故障的表现形式多种多样。不同的故障出现不同的症状。

例如，执行 CLOAD 指令和给一下程序，在画面上观察 LIST 的进行情况时，不知什么原因只呈现一半画面。这时如果所显示出来的部分程序的目录是正确的话，应考虑下列情况：

(1) 磁带的数据有无问题，

(2) 由于 RAM 出错，使输入数据有误。

听听磁带产生的声音，或者输入测试磁带检查 RAM。下一步要仔细地检查其中哪儿有问题。

可是对麻烦的故障会发生这样的情况：接通电源时，显示器整个画面上出现予想不到的字符与图象。

前面已经介绍过，电源接通时，CPU 立刻执行初始设置的程序。象这样出现不明原因的显示时，说明机器没有执行所规定的初始程序。

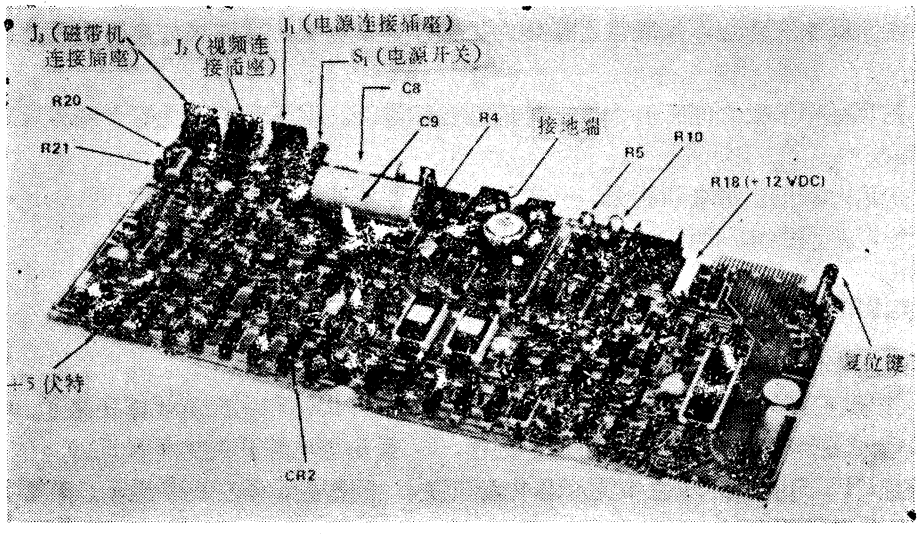


图15 逻辑汇总印刷板

对这种情况，故障不仅限于电源输入逻辑电路；RAM，ROM，同步信号发生器甚至Z80 CPU都应加以考虑。

这时从何处着手呢？将所有部件：RAM，ROM，甚至包括CPU都拿出来检查固然可以的，但化费时间太多。

除了由于焊锡等物引起短路原因外，即使把插在插座里的部件都拿下来，仍不能弄清它们发生故障的原因。

下面给出一个检查步骤，用于寻找发生故障的具体位置。

1）故障的分段判别法

根据图16，取下部件的同时，参考流程图，用它作为检查哪个部分存在故障的步骤。

在这个流程图中，最容易引起故障的是电源。当接通电源时，显示器上出现莫明其妙的数字或图形。这种情况写在流程图的第一个方块中。

在第二方块中，暂时拔掉电源线及视频电缆，然后再接通电源。

方块图3是表示判断的，在方块2操作以后，对显示器是否还有莫明其妙的显示，还是恢复了正常状态作出判断。

若恢复了正常状态，由方块4表明是由于电缆连接不良，例如，是否有断线的可能，或者考虑连接电缆焊接处发生短路。

在这种情况下，要检查电缆等的连接上有无异常。有时只不过是电缆与插座的接触不良而已。

方块图5是切断电源开关，等待十秒钟再接通电源。这十秒钟是初始设置用的逻辑电路达到正常动作时所需要的时间。由方块6可以知道，若显示器这时显示出“READY”，则考虑方块7；这时，可能是开关S_2，或者是C_{42}等元件已经损坏。

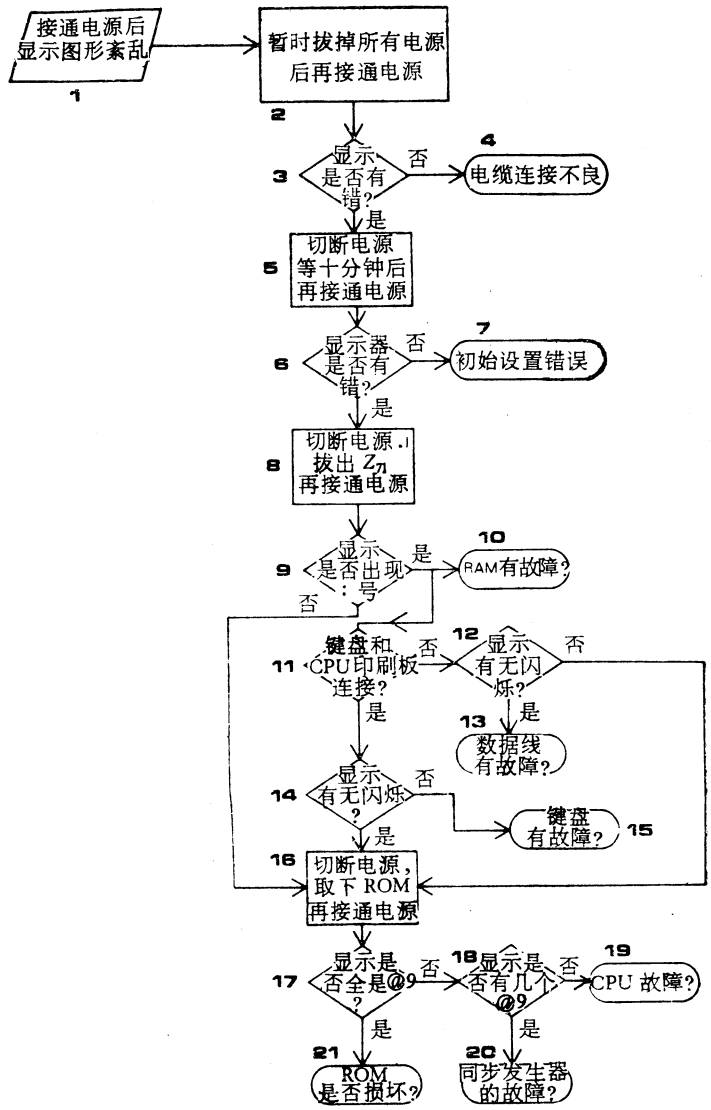


图16　寻找故障的判别流程图

根据方块 6，如果显示的是莫明其妙的东西，则参见方块图 8；在 Z_{71} 中拔出双列直插式的分路器（X_{71}）。这样从系统的电气角度切断了 RAM，使机器不受 RAM 故障的影响。这时接通电源后，只有 CPU 和 ROM 在工作。

方块图 9 表明，可根据在显示器中是否出现 16 行 32 个“：”字符号来判断故障的情况。若显示出“：”字符，则可认为 RAM 或键盘上有异常。

方块图 11 至方块图 15 所示的步骤是用来发现某些故障部位的。

根据第 11 个方块图，把键盘的插头拔下来试一试。在未连接键盘的情况下，显示器上仍有闪烁的图形，说明数据线上有异常；而在连上键盘时有闪烁显示，那么可根据方块 14 的判据，应考虑方块图 15 所示的键盘上有故障。

接着由方块图 16，取下 ROM。请注意在取下 ROM前必须切断电源！

取下 ROM 以后，对 CPU 任何程序都送不进去，这时在显示屏面上以每行 64 个字符的形式全部由@ 9 显示，由于连续地送出 VID* 信号，所以这时会出现闪烁现象。

如果没有看到@ 9，或者只看到一部分@ 9，说明同步发生器或视频 RAM 有问题。若此时显示器显示仍莫明其妙，则 CPU 工作不正常。

这个流程图只是给出一个检查顺序，可按图寻找。图中由于不能断定故障所在的确切位置，故用“?”来表示。

例如，就方块图 19 所示那样，若在显示器上能显示几个@ 9，则不能断定 CPU 就没有故障。所以图 16 所示的流程图不是绝对的东西，而仅作为寻找故障的一个参考。

2）使用信号状态的检查方法

作为检查故障部位的手段，首先要找出哪个地方产生了错误的信号，然后由此，朝相反方向边走边找，直至找到出现正常信号的位置，那末在正确信号和错误信号之间，就能确定故障的部位。

这种办法在一般无线电和电视修理中是常用的。TRS-80 这样的计算机也可以用这种方法。然而，在 TRS-80 中能够给出多种类型的信号。至于有哪些信号的状态则叙述如下：

（1）脉冲信号

所谓脉冲信号是指在逻辑 1（约 3 伏以上）和逻辑 0（0.2 伏以下）之间互相变换的重复信号。它包括持续工作的脉冲信号和瞬态工作的脉冲信号。

持续工作的脉冲信号称为时钟脉冲，瞬态工作的脉冲信号称为单脉冲。

为了检查脉冲信号必需使用示波器。对于时钟脉冲等信号也可用三用表测量其电压，但这只能简单地检查它是否在工作，而不能得到它的正确情况。

例如，Z_{42} 的脚 6，它是振荡器的缓冲级输出端，它通常给出约 100 毫微秒重复周期的时钟脉冲。

（2）恒定持续信号

对恒定电压电平的信号来说，除了时钟脉冲以外由逻辑 1（3 伏以上）或逻辑 0（0.2伏）所构成的信号称之恒定持续信号。

如CPU 工作时所产生的单脉冲信号，而在 CPU 停止工作之后，还在一段时间间隔内所发出的恒定持续信号。

例如，Z_{80} 的引出脚 16 的逻辑 1 的恒定持续信号 $\overline{INT}$，它平时通过电阻 R_{50} 接到 +5V上。CPU 时钟脉冲分频用的计数器 Z_{66}，其引出脚 6、7 的输入为逻辑 0 的恒定持续信

号。如果将与电阻 R_{67} 相连的 Z_{42} 输入端 9 定为逻辑 0 的恒持信号的话，输出端 8 则为逻辑 1 的恒持信号。

(3) 浮动信号

作为恒定持续信号，其电平为逻辑 1 （3 伏)或逻辑 0 （0.2 伏），取其中间值的电平信号称之浮动信号。

不论在 CPU、ROM、RAM 数据母线还是在地址母线缓冲器中都使用三态器件。（所谓三态器件是一种包含逻辑 1、逻辑 0 和断开三种信号电平的器件，其中断开状态是指高阻抗的意思）。

这种元件处于断开或者未进行门选状态时，构成了浮动的信号电平。用同步示波器等仪器来检查时，就可以看到混杂着噪声。这种噪声电平约为 1.5 伏左右。

由于 TTL 电路的逻辑 0 为 0.8 伏以下，逻辑 1 为 2.0 伏以上，这样的中间浮动信号就不能被认为是逻辑 1 或逻辑 0 .

在浮动信号情况下，通过高阻抗（50KΩ或100KΩ)与 +5 伏相连时，该信号电平则变成逻辑 1 电平。若与 0 伏变连，则构成逻辑 0 .

噪声的平均电平大约为 1.5 伏。拔掉 ROM 以后测量其输出脚,该噪声电平为 0 伏。

这时，通过高阻抗进行测试，所得结果与前面相同。而噪声电平则不同。请注意拔掉 ROM时必须先切断电源。

3) CPU

CPU 中的 Z80 发生故障是非常难办的。它发生故障的现象如同地址母线、数据母线、缓冲器以及CAS/RAS 的信号线定时不良所发生故障的现象相似。

这时最方便的办法就是更换一个已知好用的 Z80CPU. 当然考虑其它部分也可能存在问题，但总的说来由于 Z80 容易从插座上取下来，所以首先更换 CPU 进行观察。

图 17 是用于 CPU 诊断有关故障的流程图。

因怀疑 CPU 有问题需作判断时，可参考方块图所示检查步骤。该图最左边一列方块给出了各种主要信号的检查途径。

首先请检查 CPU 的地址线和数据线有无脉冲信号输出，若这些线上无脉冲信号输出，应考考CPU 的时钟脉冲输入电路故障。

由于从 CPU 出来的地址线和数据线在印刷板上的排线是平行的，而且靠的很近，所以这些线间的间隙往往被焊锡之类的东西挤得很小。当地址线的其中一些线之间被短路时，会产生与浮动信号一样的动作。

不过，在数据线中使用三态信号电平的元件，当数据线正常时，也有浮动信号电平，或为逻辑 1，或为逻辑 0 的值，所以要判断它们是否短路，那是相当困难的。

为了检查是否短路，将 Z80 的引出脚 25（或者是电阻 R_{56} 的一端）固定为 0 伏。这样 CPU 一端的地址线便出现浮动信号，地址缓冲器也会浮动。

数据输出端的缓冲器会浮动，而数据输入端的缓冲器则处于工作状态。由于这些缓冲器的输入全处于浮动信号状态，所以CPU 的数据信号均为逻辑 1 .

关于数据线有无短路的检查，可在到 CPU去的输入缓冲器的输入端上加上逻辑 1 或是逻辑 0 的 TTL 信号进行观察即可。该信号最好从同步信号发生器取出。

已输入的信号会出现在对应的输出端，如果信号没有出现在其它数据线上，那是正常的

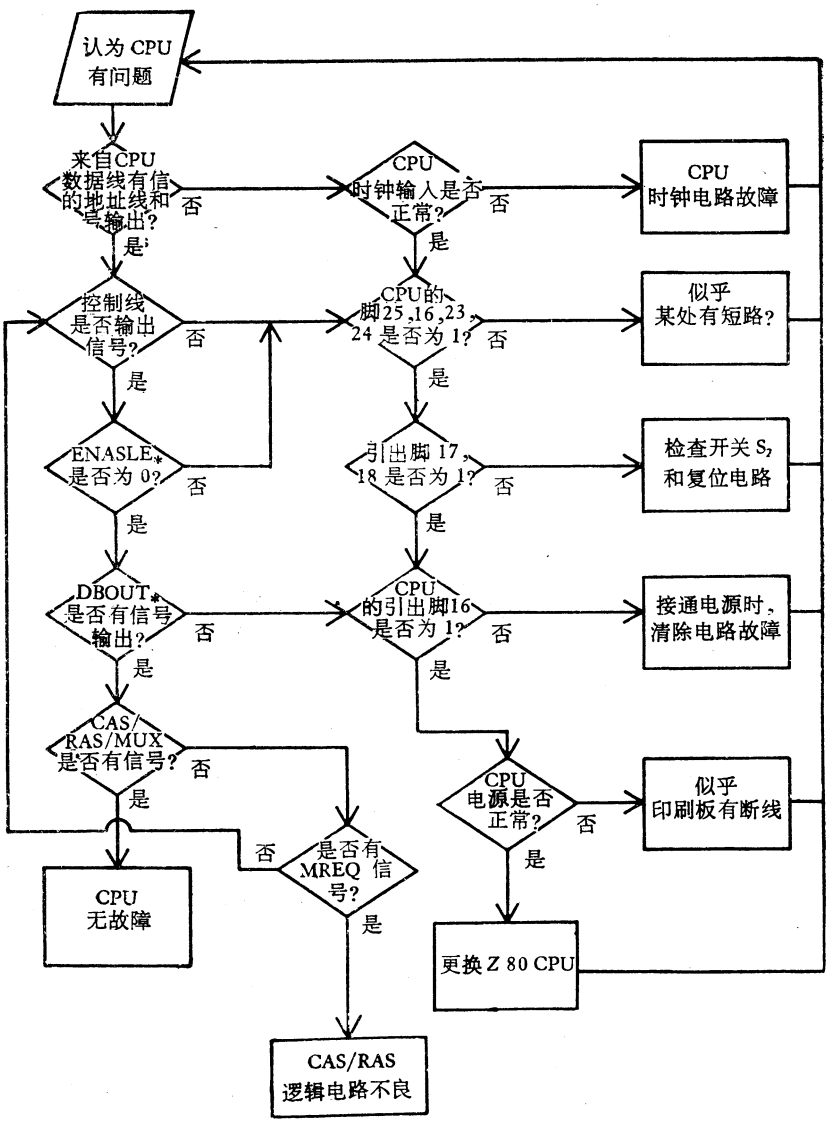


图17　CPU 故障诊断流程图

现象，如果输入信号出现在其它数据线上，说明这里有短路。

4) ROM

有关 ROM 的问题可分成下列三类：

(1) 地址回路　(2) 数据回路　(3) 芯片选择回路

上述这三个问题将在下面分别予以介绍：

(1) 地址回路

地址回路的连线由于十分靠近而且平行，有可能造成短路。除此还要考虑插头接触不良、断线等现象，以及由于某种冲击造成 ROM 松动等情况。

地址线的检查请从芯片引出脚开始。通常在地址线上输入脉冲型的信号。在 CPU 方面使 TEST* 信号线为 0，即在 CPU 那边切断地址线，这时就能对地址线的短路及断线进行测试。

(2) 数据回路

数据回路中引起的故障有两种原因：其一是 ROM 中数据位损坏，那是无法修理的。这一点可用 SCQATS 程序进行检查。这时，如果 SCQATS 程序加不进去的话，那末必须更换 ROM。

(SCQATS 程序是用机器语言书写的供测试用的程序带，如一定需要的话，请向 Radio shack 公司索取)

第二个原因是在数据线上可能有短路和断线。这时拔下 Z_3 的双列直插分路器，各个 ROM 一起出现浮动信号，即可用前述方法来测试数据线。

(3) 芯片选择回路

在选择芯片时，与 ROM*，MEM* 等信号有关。

ROM* 信号是用于选择哪个 ROM 芯片的信号，而 MEM* 信号是用来控制从RAM、ROM*读出的数据缓冲器的信号。双列直插式分路器应正确地连接到所需要使用的芯片上去（TRS-80 所用的双列直插分路器，其中如有损坏也可能引起短路）。

5) RAM

注：RAM 这种器件易受静电而损坏，请在使用时要特别小心。

在使用 RAM 时，首先用手接触一下附近的接地点，把积存在自己身体上的静电放掉，最好碰一碰电容 C_9 的一端。

在存放和运输 RAM 时，应把它插进浸过黑色导电涂料的泡沫垫里。绝对不能使用白色苯乙稀泡沫，它能产生很高的静电，从而使 RAM 损坏。

还有请不要用玻璃纸带等东西包装 RAM，取下玻璃纸带时因摩擦也会产生高电压。

RAM 的故障诊断，因为它与 ROM 的读写方式不同，在更换地址方面也比 ROM 来得难办。但检查的方法与 ROM 相同，可使用 SCQATS 程序来检查 RAM。

除此请检查一下各 RAM 的电源输入脚，看看供给 RAM 的电源电压是否符合要求：脚 1 为－5 伏，脚 8 为＋12 伏，脚 9 为＋5 伏。

还有由于某种冲击会使 RAM 脱离它的插座，请检查一下 RAM 是否还确实在插座上。

再请检查，RAS，CAS 的各个信号是否加上，还有多路调制器（前称多路转换器）输入的 MUX 信号是否在工作。这些信号都是以一定周期工作的脉冲信号。

还要确定地址信号是否真正加到每一个 RAM 上。特别是在从 4K RAM 更换成 16 K RAM时应该核准双列直插分路器内部连接是否正确。

当 RAM 有异常，或者 SCQATS 程序加不上时，请全部更换成正常的 RAM进行观察。这时若能正常工作，再依次插入被怀疑是坏的 RAM，从而就能找到不能正常工作的 RAM。

6) 地址译码器

地址译码器不良时，解决问题的根源在于存贮器是否失去工作能力。

例如，若 ROM 完全不能工作的话，是由于 ROM* 的信号没有被加入。

所以哪一个存贮器完全不能工作，就看地址译码器的哪个部分有问题，请对这个地址译码器的周围电路进行检查。

Z_{21} 解码电路由于与地址线 A_{12}、$_{13}$、$_{14}$、$_{15}$ 等相连接，所以首先检查这些地址线有无问题，再检查门 Z_{73} 的引出脚 5 的输入 RAS* 信号线是否加上。

双列直插分路器 Z_3 可用来改变输出情况。由于译码器 Z_{21} 是集电极开路器件，必须在

+5伏与Z_{21}之间加一约为5千欧的电阻，才能检查其输出端输出是否正常。

译码器和分路器如能正常工作，再检查其它输入。例如，请检查地址线A_{12}和RD信号线状态。

由CPU连续选择每个地址，观察数据读写。特别要尽量选择键盘对存贮器来作读出。这时，如果有什么假冒的输入信号，则CPU执行的操作不能如实反映从键盘来的输入内容。

7）键盘

键盘的故障大多数出自于机械部分。例如按键未安装好，弹簧折断，以致按一个键牵动了别的键，还有电缆断线等等情况。

键盘矩阵的短路是容易发现的。

紧接字符>的后面，如果出现了什么字符，那末与这个字符相应的键或者印刷板有短路。

在键盘完全不能工作的情况下，可能由于连接电缆断开没有接上电源，成者没有地址译码器输出端的KYBD*信号。

键簧的弹性减弱，以及在键帽套口处形变，使按下的键弹不上来，发生这种情况时，应更换弹簧，取下键帽衬套。

把键帽拿掉进行观察，开关接点若发生损坏时应拆下键盘整体，根据下列步骤，更换按键开关。

（1）拆下整个单元。

（2）摘掉键帽和键帽插棒。

（3）拿下弹簧。

（4）从印刷板上取掉接点焊锡，这时接点完全与印刷板脱离。

（5）认准接点的位置在哪个地方。在正确位置上把键盘提出来时，在右边即可露出接点。

（6）用尖细而有力的无线电尖嘴钳，从塑料基板上把接点拔出。

（7）在工具#773—10000或#773—10023上插进新的接点，保证接点的表面在正确的方向上相互对准。

（8）对键基板上用上述工具把新的接点插入，直到完全插进去为止。这时应将印刷板放在坚硬的表面上进行操作，把接点插进去，请注意要把接点尖端完全露出印刷板的表面。

（9）拿下工具，为保持接点的正确位置，请塞进焊接头。这时，把印刷板翻过来操作。

（10）焊好焊接点以后，再装弹簧、插棒、键帽等零件再确定工作是否正常。

下表是修理键盘用的工具表：

名　　称	工具编号	名　　称	工具编号
插棒	171—40103	弹簧（3盎斯）	173—10012—2
固定接点	173—30052—2	金属插入工具	773—10000
活动接点	173—30053—3	塑料插入工具	773—10023
弹簧（2盎斯）	173—10012—4		

8）连接用电缆

在更换连接电缆时，请注意电缆连接的正确操作。

9）同步信号发生器

同步信号发生器发生故障时，显示器屏面将出现不稳定状态。在没有垂直同步信号或水平同步信号时，可从相反方向追查这些信号，就能够发现计数器、复位电路的不良部位。

由于系统的主振荡器接到该同步发生器的输入端，如果主振荡器发生问题时，那末究其原因也将使显示器的显示出现不正常。

为了进行视频合成，要用到同步信号发生器的输出，这个电路的故障可根据显示器的显示情况来了解。产生水平以及垂直显示用的信号出现异常时，最好考虑该同步信号发生器的有关故障。

首先，第一个检查点是在Z_{32}的输出端11上，以确定频率计数器的60Hz输出信号是否有输出。

假如有60Hz输出，那么所有计数回路看来不会妨碍机器正常工作，如果不是60Hz，则寻找各计数器输出，请检查哪一级的频率不正常。

如下面图18所示表格给出了各计数器输出的频率值。该表所写的是标准频率值（误差小于−0.03%）。实际测量值可能略有差异。

信号名	信号输出	信号频率	分频比	备注
VDRV	Z32引出脚11	60.00Hz	177.3517×10^3	垂直同步信号
R8	Z32引出脚8	60.00Hz	177.3517×10^3	字符位置的行地址
R4	Z32引出脚9	180.0Hz	59.11722×10^3	字符位置的行地址
R2	Z32引出脚12	360.0Hz	29.55861×10^3	字符位置的行地址
R1	Z65引出脚12	660.0Hz	16.12288×10^3	字符位置的行地址
L8	Z12引出脚11	1.319kHz	8.067550×10^3	单个字符的行地址
L4	Z12引出脚8	2.639kHz	4.032247×10^3	单个字符的行地址
L2	Z12引出脚9	3.959kHz	2.687825×10^3	单个字符的行地址
L1	Z12引出脚12	7.917kHz	1.344082×10^3	单个字符的行地址
HDRV	Z50引出脚11	15.835kHz	671.9987	水平同步信号
C32	Z50引出脚8	31.670kHz	335.9993	字符的列地址
C16	Z50引出脚9	63.340kHz	167.9997	字符的列地址
C8	Z50引出脚12	110.840kHz	96.00414	字符的列地址
C4	Z65引出脚8	221.69kHz	47.99991	字符的列地址
C2	Z65引出脚9	443.38kHz	23.99995	字符的列地址
C1	Z43引出脚7	886.756kHz	12.0003	注2
Chain	Z43引出脚9	886.756kHz	12.0003	输入

注1. 主振荡频率为10.641099MHz（−0.03%误差）。

注2. 这里为64个字符显示时所用数值。

图18 同步信号发生器的各种频率

在分频比一栏中，它表示从基本频率中经过多少次分频的数目。例如Z_{42}的引出脚6的频率若为10.64100MHz，信号L_1的值则经1.344082×10^3次分频得到7916.93Hz。可能会有些差异，但差别太大的话，则应考虑那个部件有损坏的可能。

当频率值差别很大时，根据这个一览表中的值边查边找，直到出现基本准确的频率值的信号。但是测量时过于性急的话会得出错误的结论。

现在，假设 HDRV 信号通路出现意外，而在 C_{32} 线上的信号良好。这可能是 Z_{50} 计数器不良，也许是 Z_{66}、Z_{49}、Z_{6} 等大规模集成电路有故障。还有可能是受与外部设备连接的电路影响的缘故。

在更换这些大规模集成电路以前，先用刀尖割断印刷板上的连线。这里，首先割断 Z_{50} 的引出脚 11 的输出端，直接进行检查。如在这一点上所测得的值不准确，则更换 Z_{50}。

若 Z_{50} 输出脚 11 输出正确，则把脚 11 割断处焊上，小心地把它修复。再把 Z_{66} 的引出脚 5 断开，重新检查 Z_{50} 的脚 11，如果出现信号应更换 Z_{66}。

这样把切断的印刷板连线连接起来，请再一一观察找出哪一个是不良部件。

10）视频 RAM

假定在视频 RAM 有故障的情况下，可输入 SCQATS 程序来进行观察。

用该程序来检查视频 RAM 的哪一位有问题是一种最方便的办法。

在测试中，如果出现许多错误，应考虑同步信号发生器或者作视频 RAM 地址转换用的多路调制器可能出现意外。经常出现 READY 和 ＞的符号情况，则是视频 RAM 地址的周围有故障。

地址外部的故障多数原因是多路调制器上的短路，管脚和印刷板接触不良所致。在RAM 地址的输入端，用示波器可容易地进行检查这些故障。

全部地址线（V_0—V_9）用 64 个字符形式来表示时，呈现为脉冲型信号，而不是浮动或者某种信号。

视频 RAM 的 VWR* 信号是为写入从 CPU 来的视频信号，所以它通常处于逻辑 1。

取下 BASIC 的 ROM，接通电源，在显示器屏面上全部出现@ 9。

如果只出现少量@ 9 或几乎看不见时，应检查显示器部分。

如果在水平方向有某一行不出现@ 9，应检查行地址 V_6—V_9 线。

如在垂直方向不出现@ 9，请检查该列地址即 V_0—V_6 的地址线信号。

若显示器屏面出现闪动的图形(同样在显示@ 9 字符时)，那是 CPU连续地对视频RAM 的地址回路产生干扰引起的。

这时，观察视频 RAM 的地址线，即可看到同步信号的地址线信号和 CPU 的地址线信号交替出现，这是正常现象。

没有这样的闪动现象，则意味着 CPU 的地址信号没有真正被送到地址多路调制器里。

假定多路调制器 Z_{10}、Z_{11} 方面发生故障，首先要检查所有的地址线。若公共的输入脚1 和 15，电源输入脚 8 等等无异常现象，而在输出脚上又不出现真正从同步发生器和 CPU 来的信号时，那么请更换性能不良的多路调制器。

这里假定 VID* 信号经常不为 0 状态，若 VID* 信号被迫置为逻辑 1 时，即应怀疑 Z_{36} 已经损坏。

11）视频信号发生器

如果视频发生器存在着不良现象，那么包括字符在内完全没有任何东西出现。

由于视频发生器各部分是串联在一起的，所以比较容易发现故障。

例如，只显示字符而不出现图形时，应好好检查移位寄存器 Z_{11} 和图形发生器 Z_{8} 有关

的电路。

有图形，却没有字符的情况下，可检查字符发生器 Z_{29} 和移位寄存器 Z_{10} 有关的电路。字符和图形两种都没有信号显示时必须检查 Z_{26}、Z_{27}、Z_{30} 及 Q_1、Q_2 等等晶体管这些有关部件。

一般认为显示器问题中最恶劣的情况是在显示屏面上什么也看不见。这时要从整体着手来分析其原因。

首先请从电源开始检查，然后检查主振荡器。

如果这些都是正常的话，可查看 Q_1 射极的视频信号，从最终输出端开始由外朝里边查边找，直到找到产生信号的部件为止。

其次，采取继续地边查边找的方法向前深入，例如请查看在字符发生器上的信号是否存在。如果正常的话，即在这前后两点之间有故障。

现在来观察 TRS-80 系统中给出什么信号。例如系统如果用来产生字符的话，则 Z_{26} 的引出脚 8 上应有脉冲信号，并且从 Z_{10} 上也有输出信号的话，那么要检查 Z_{30}。

如果 Z_{30} 的引出脚 2 保持逻辑 1，则从脚 1 就不会有输出；Z_{26} 的引出脚 8 和 6 如果保持逻辑 1 的话，应考虑 Z_{27} 不工作。鉴于这种情况，可认为 Z_7 有故障。

显示器屏面上什么也没有时，可拔下 BASIC ROM，那么显示器应全部显示@9 的图形。

为了寻找故障，下面给出检查步骤，以供参考。

(1) 显示器的字符图形对比度差（影象淡薄）

组件 Z_{41} 的性能不良。Z_{41} 有发热现象请即更换。如器件发热，由于晶体管饱和电压变大则不能充分推动晶体管 Q_1，引起画面暗淡。

(2) 字符的点有短缺遗漏

整个字符的部分点行有遗漏的话，请检查字符发生器 Z_{29} 的 RS_1、RS_2、RS_3 输入端。

如输入信号正常，那么请更换 Z_{29}。如果字符的各个垂直方向上的点发生遗落的话，要检查 Z_{10} 的输入信号。若输入信号亦正常，则更换 Z_{10}。

(3) 字符显示的点不稳定

当 Z_{10} 的温度过高，数据进入 Z_{10} 时，由于提取数据被丢失，导致字符显示点不稳定。

Z_{10} 器件发热异常，停机少许冷却一下，再观察，如仍是发烫，那么更换 Z_{10}。

(4) 图形部分错误

请检查 Z_8 的输入端。若输入良好，则检查输出端。

在右边的图形或者左边图形的垂直方向上发生错误的话，那是 Z_8 的外部线路有问题。在图形中垂直方向上的线条正常的话，那是 Z_{10} 性能不良。

(5) 图象不稳定

由于此现象形成的原因与前面的 (3) 项相同，酌情更换 Z_{11}。

(6) 行间没有留下空隙

这与 L_8 的信号有关。应检查同步发生器甚至 Z_{29} 的闩锁电路等等。

(7) 画面出现干扰

这不是视频发生器的问题。请用示波器观察 +5 伏电源电路的信号。若有振荡现象，请检查 C_{12} 以及 C_{13} (12V处) 等电容器是否工作正常。

(8) 字符的拼写有错误

字符的拼写出现错误时，首先怀疑的应是视频 RAM 的数据是否正确，也许是 Z_{28}，从 Z_{28} 到 Z_{29} 的地址线有短路，或者断线。

12) 同步信号合成电路

该回路是机器中最容易对故障作出诊断的部分。

由于每个水平和垂直的同步信号分别用 Z_6 和 Z_{57} 来接收，最好从这里开始依次追查下去。

这里问题最多的地方发生在微调电位器 R_{20}、R_{21} 上，一是使用温度非常高而损坏，再就是接触不良。

电容 C_{20}，C_{26}，只要不是物理性损坏，就不会有太大问题。由于 C_{21} 以及 C_{27} 是釉质电容，应考虑有短路的可能。

组件 Z_6 和 Z_{57} 是 CMOS 器件。它与 TTL 不同之处是它的电路输入阻抗高，只能通过小电流。

所以，即使输入浮动，其输出端也不浮动。这是由于流经很小电流也能建立一定的状态。在检查这种电路时请注意与前面介绍的那种对 RAM 处理的方法完全相同。

13) 地址解码器

地址解码器由门电路构成，所以有时候问题比较清楚，处理也简单。

关于这一部分，不能采取把印刷板割断，然后再连接起来的那种检查方法。

根据原有的输入输出关系应该检查解码器的工作逻辑是否正确。

例如，在确定 ROM* 和 MEM* 的信号是否有输出的同时，还应该了解 ROM* 信号出现的准确时间，这是困难的。为了保证测得 ROM* 信号输出的情况，应采用下列方法：

(1) 全面检查地址线的输入端。从逻辑上分析其波形，以保证 ROM* 信号输出正常。

(2) 根据 CPU 把予先了解的 ROM 编号，用机器语言编写的程序指定出来，以此作为地址开关的编码。

地址解码器性能不良会使其它部分的工作性能不良。

接通电源时，即使已经显示出 READY，而键盘还不能工作，这时要检查 KYBD* 信号，这个信号出现与否，是容易了解的。

如果不出现这个信号时，应认为与 KYBD* 信号有关的解码器不正常，但是由于这时能够显示 READY，所以其它地方没有什么大问题。

接通电源后，如果不出现 READY 信号，那么应该考虑怎样检查才好呢？

碰到这种情况，拔下 Z_{71} 双列直插分路器，机器将显示大冒号：。如有这种显示图形，那么插回 X_{71}。然后检查双列直插分路器的哪个地方有什么问题。

如果这里没有别的故障，应该认为 RAM* 信号没有输出来。这也有可能使双列直插分路器 Z_3 不工作。

组件 Z_{73} 的引出脚 6 没有输出时，自 Z_{21} 出来的地址解码信号就没有输出。这时往往由于 Z_{73} 的不良所致，所以应检查其引出脚 6 的输出信号，也请检查其脚 5 的 RAS* 输入和 A_{15} 线的输入端。

按下复位开关，也是检查地址解码器外部电路的一种方法。

在按一下复位开关的一段时间内，CPU 从 ROM 中取出程序，并执行读写 RAM，消

影视频显示，写入视频 RAM 等动作。

接着等待接收来自键盘的信号。

所以在按下复位开关的一瞬间，就有 ROM*、VID*、RAM*、MEM* 等脉冲信号被送出来，因此可检查这些信号是否出现。

14）盒式录音机的控制

盒式录音机的输出控制所引起的故障，主要是马达不工作，这是 K_1 继电器在接通时接点接触不良所致。

继电器的故障，特别是在使用 BASIC Ⅱ 程序时该继电器的损耗过多。这是因为在 BASIC Ⅱ 程序里盒式机的使用次数特别多。

关于继电器故障主要考虑继电器接点接触不良，以及继电器线圈短路或者断线等等。

更换继电器是很容易的。把继电器取下来的时候，请不要搞错相应的标志。如果装配反了的话，Z_{41} 的引出脚 3 与电源 Vcc 构成短路状态，将引起发热而损坏。

由于这种发热，可使视频信号回路端的 Z_{41} 也被烧坏。

因此，视频显示迅速变暗，当看不见的时候，可能是盒式录音机马达控制回路的继电器线圈短路或者 CR_3 稳压管短路等等原因所造成的。

继电器 K_1 的接点如果一直处于接通状态，请用手指把 K_1 继电器拨一下看看，这样使接点脱离从而停止马达转动。

如果再次启动马达（用 CLOAD 等程序命令来执行），接点仍发生合在一起的情况，那么请更换 K_1 继电器。

15）盒式机的输入输出回路

当输入从盒式机来的数据或程序时发生问题的话，请检查运算放大器 Z_4 的引出脚 10 的信号。

这时输入尽可能长的程序，以观察引出脚 10。通常它为高电平，但每次输入信号时电平下降，每个位周期内如果连续有两个脉冲，即可认为 Z_4 工作正常。

在稳压管 CR_4、CR_5 和 CR_6、CR_7 等有问题的情况下，Z_4 的引出脚 10 通常出现低电平。而由磁带输入信号则往往构成高电平。

这时，由于 CR_4、CR_5 短路，或者由于 CR_6、CR_7 及 C_{39} 等器件不良是不能作电平检查的。

关于这样的工作情况，请参照前述的工作原理。

触发器 Z_{24} 的置位复位端，触发输入端如果工作不正常，可通过 CLOAD 命令加入一个长程序，然后检查 Z_{24} 的引出脚 8。如果该端工作正常，接着检查门电路 Z_{44} 引出脚 12 的输入和引出脚的输出等波形。

在执行 CLOAD 命令时，通过 OUT* 信号将 Z_{24} 触发器复位。如果 Z_{24} 仍处置位状态，应检查门电路 Z_{25} 的引出脚 8 的信号，然后再检查 Z_{25} 的引出脚 9 和 10 的输入端。

如果 INSIG* 不工作，可检查 Z_{25} 输入端 4、5 等信号。若 INSIG* 和 OUTSIG* 都没有信号，请检测构成地址线 A_0—A_7 的解码电路 Z_{54}、Z_{52}、Z_{36} 的每个输入输出信号。

这些信号全部正常，且 Z_{44} 门电路也正常，那么依次观察 RAM、CPU、ROM 有无问题。

用一种比较观察法来观察在正常工作情况下，TRS-80 中的 INSIG*、OUTSIG* 信号线，在执行 CLOAD 命令时将会是什么样子。

执行CLOAG命令的时候，可以了解RAM是否有问题。例如到某一部件以前，程序能正常运行，而从该部件以后窜进了混乱的数据。

碰到这种情况，可使用SCQATS程序进行检查，或者一次更换一个RAM，并注意观察恶劣现象是否消失。

在执行CSAVE命令时，ROM所出现的不良现象，实际上是CPU所造成的原因，多数不可能被完整地记录下来。

通过检查CPU，并排除CPU方面的问题以后，再检查IN*和OUT*信号，此后请着手ROM的检查。

若CSAVE命令有问题，应认为是软件(ROM/RAM)及闩锁电路Z_{59}所造成的原因。

若确知CLOAD命令工作正常的话，录音机中又未装磁带，则执行CSAVE命令来观察程序，并检查盒式机插头J_3的引出脚5的输出波形。假如这时不出现任何波形，可检查R_{53}、R_{54}、R_{55}等电阻，并检查Z_{59}的D触发器的引出脚9的OUTSIG*信号输入。

信号线D_0与D_1的值，当CSAVE命令工作时，由于使用了Z_{59}的闩锁功能，应检查这两根数据线有无输入。同时请检查Z_{59}的引出脚1是否为逻辑1，以确定其是否正常工作。

32个字符显示不工作，这常常是Z_{59}闩锁电路发生了故障。

从Z_{59}的脚14输出的MODESEL信号，是根据OUTSIG*的时钟脉冲来对D_3值进行闩锁的信号。

若CSAVE，CLOAD两命令都能正常工作，则可考虑与Z_{59}的脚14输出有关的闩锁电路是否损坏，以及该脚14是否对地短路。

闩锁电路Z_{59}被清除的情况下，Z_{59}的脚1如果接地良好，说明闩锁电路已被清零，且显示器可将32个字符显示改变成64个字符显示。

这时改变成64个字符后，令脚1脱离接地状态，机器又重新恢复显示32个字符的状态。这时考虑OUTSIG*信号是否作为脉冲信号加上去的，以及是否由于印刷板上线条断开以后处于噪声状态下工作的情况。

16) 电源

电源的故障是由焊接不良、元件短路，焊锡形成短路以及电源变压器断线等原因造成的。

电源输出线被短路时电流保护装置即刻工作，不会发生损坏。因此在电源外部，由于焊锡造成的短路，也不能出现所需要的电压。

+12伏和+5伏两个电源都没有电压的情况下，请测量R_{18}电阻的两端电压。如该电压为0.6伏，那么+12伏电源无电压输出，这是由于反馈回路造成了截止，故不能产生+12伏电压。

由于+12伏没有电压，导致+5伏电源也没有电压。这是因为+5伏用的稳压大规模集成电路使用+12伏的输出造成的，请检查+12伏在什么地方已被短路。

没有-5伏电源时，请测量电阻R_{19}两端电压。若两端电压承担了整个输入电压（-11伏）的话，说明在什么地方有短路。

由于RAM的电源要用到+12伏和-5伏，假如要找哪儿有短路，首先考虑RAM中可能短路。这种内部短路的RAM比其它RAM要热得多，所以很容易发现。

注：用手指检查RAM温度时，由于这种短路的RAM十分烫手，谨防烫伤，望十分注意。

切断电源取下所有 RAM 之后，再加上电源观察。这时若有电压输出，那就是某一块 RAM 已发生短路。切断电源，插入一块 RAM 再通电观察，这样重复数次，即可找出短路的RAM.这时，短路的 RAM 不一定只有一块，应全部核查。

+5 伏电源短路是很难检查的。凡是用 +5 伏的地方都是它供给的，所以也最容易被短路。

首先不要性急。在 +5 伏的支座上用眼睛检查一下确定有没有被短路。

这时如果还不能了解故障原因，可割断印刷板上的配线。这样，采用分割法能较快地明确被短路的部分：

那就是先切断 +5 伏电源线的靠中间部分。这里如果没有短路，则是从所切断处的后半部的地方有短路；若有短路的话，则从所切处的前半部地方有短路。

在有可能发生短路的那部分，继续在其一半的地方，用割断配电线的方法来检查。对明显知道没有短路的地方，用电烙铁把它们焊接起来。

在过分长的时间里用电烙铁加热，注意不要损坏印刷版。

请反复按上述步骤进行分割检查，找出故障。

+12 伏电源如果已经破坏，首先检查晶体管 Q_1 的散热片。由于螺钉松动基极或者是射极可能与散热片有短路，有时印刷板的引线脱落的地方也会引起短路。散热片的螺钉松动时应注意基极或射极与散热片不要短路，并进行固定螺丝。

交流变压器的故障多产生于终端接触不良。应该检查保险丝及套筒插头是否脱落。

如果保险丝容易烧毁的话，应检查整流器 CR8 中的二极管是否短路。

下面的图 19 是组件 Z_1 和 Z_2 的各引出脚在正常工作时所测定的值。这里 +12 伏写成 12.00 伏。+5 伏写成 5.00 伏。

17）画面位置的调准方法

在更换同步发生器的部件以后，显示器的画面应调准到处于中心的位置。为此，需加入下面所示的样本程序。

该程序无论是 BASIC-Ⅰ 语言，还是 BASIC-Ⅱ 语言都可使用。在显示器上，当画面超出边缘跑到外面去时，在中心位置上也露出了四角，这时一边观察该画面，一边调节 R_{20} 和 R_{21} 电位器，使边框恰好移动到适当位置。

这种调节不用金属螺丝刀亦能方便地进行调节。

```
10  CLS
20  FOR  X=0 TO 127
30  SET(X, 0); SET (X, 47)
40  NEXT X
50  FOR  Y=0 TO 47
60  SET(0, Y); SET(127, Y)
70  NEXT Y
80  FOR  X=62 TO 65
90  SET(X, 23); SET(X, 24)
100 NEXT X
110 GO TO 110
```

Z_1		Z_2	
引出脚编号	电　　压	引出脚编号	电　　压
1	0.00	1	0.00
2	5.30	2	10.60
3	5.00	3	11.99
4	5.00	4	6.92
5	5.00	5	6.92
6	7.46	6	6.92
7	0.00	7	0.00
8	0.00	8	0.00
9	0.33	9	5.72
10	5.89	10	12.31
11	11.99	11	21.16
12	11.99	12	21.69
13	7.05	13	13.48
14	0.00	14	0.00

这些电压是用数字电压表在电容 C_9 的接地端与各脚之间测得

图19　Z_1 和 Z_2 的各脚的电压值

4. 有时不能正常工作的故障

由于 TRS-80 发生故障，那么就要修理，使其恢复工作正常。如果用 TRS-80 干活，有时可能出现令人难以明白的显示。此时，有人喜欢敲打机器，但请不要过分。

由于没有办法，不妨在这里用 TRS-80 连续运行检查存贮器的 SCQATS 程序。

这时，TRS-80 稍微有些发热是允许的。几小时以后，也许 SCQATS 程序能找出那些不良的存贮器。

如运行一天还没有希望的话就请更换全部 RAM 。

即便用 SCQATS 程序也找不出什么结果的时候，那么制作一个专为反映故障的程序。为了找出只是偶然发生的故障，可以试试在正常情况下它产生故障的情况。

例如，在循环 (FOR) 程序的程序环中，如果显示出现错误，请编写一个用循环程序构成显示错误的程序。

由于虚焊，可能没有完全焊牢。这时，用铅笔头上的橡皮轻轻敲打印刷板进行查看，就能发现接触不良的部位。

焊锡粒往往会滴落在印刷板上，轻轻敲打，锡粒就会掉出来。但有时也许锡粒牢牢地粘在插座上也看不出来。

总之，假如发现了焊锡粒，可认为它是引起故障的原因，因此要把它取出来。

把印刷板颠倒过来，请检查 RAM 插座的焊接部分。是引线弯折，还是焊接位置不正。对认为可疑的地方稍用力按一下，如引起工作不良，那与焊接不良有关。

有关焊接不良情况不仅发生在 RAM 插座上，还要检查一下其它部分是否有类似情况。对于大规模集成电路的焊接点，还有电阻、电容等等，这些焊接方法各不相同。

在偶而发生的故障中，本来印刷板的正面连线和反面连线以通孔连在一起，但也会发生所谓通孔中间断裂。请认真检查焊锡是否透过孔的中心。

假如发现了这种通孔，请用细导线穿过通孔，把两头焊在一起。像这样的故障，经常是接二连三地出现。因此如果发现一处有不良现象，就要仔细地检查其周围情况。

5. 故障查找参考

数字计算机有一个特点，它由基本电路组成，且反复地使用那几个基本电路。因此如果能够充分了解它的某些电路的话，碰到相同的电路就可以完全一样处理。

不管什么场合，对相同形式的逻辑门，如输入条件不同，输出值则不同，而输入条件相同，那么输出也相同。

在查找故障时，不利的因素就是计算机使用机器语言编写程序。通过按钮，机器就按照对应的程序进行操作，这是由于用机器语言写入的程序，不经过相当麻烦的思考也就不可能理解其全部操作。

在键盘操作说明中已经大致地介绍了键盘是用哪些硬件来构成的，并根据软件程序了解到按下哪个键来传送什么样的信息。然而关于其详细内容这里就不一一介绍了。

如果在软件方面有什么缺陷的话，那么用什么办法来补救呢？首先，要了解键盘的软件由哪几部分组成，这就要下点功夫仔细地分析操作程序中的每个步骤。

只要你不嫌浪费时间，拿出全部软件表，把软件存在故障的那一部分查出来。

因此，本节只能提出一个检查办法：

即使在软件中存在故障，也把软件作为正常的状态下，检查硬件上有那些错误操作。

这一方法实际上，对确定哪个部分发生故障的判断是有用的。如果肯定了硬件没有故障，那么就能知道包含软件的 ROM 有问题。

1）具有恒定输出的门电路

使用示波器等工具检查门电路，在脉冲信号多的情况下不那么容易详细了解它们是否都处于正常工作。特别是地址解码器部分，它给出复杂的脉冲信号。

若有同时可观测两种波形的示波器的话，就能分析两个输入门来的信号，但若是八个输入端的与非门的话，就会使你束手无策了。

诚然最近以这种信号分析为目的的逻辑示波器已上市，然而利用这种检测器作信号分析也并非容易。

关于进行故障诊断，下面介绍一种有效的方法：

将输入端接地进行观察，这样做对 TTL 电路没有特别害处。例如，对或门和或非门如果没有输出，请注意其中一个脚的输入信号可能对地短路。

对两个输入端的或非门，如将其中某个输入脚接地，输出端有正常信号出现的话，那么可以知道至少那个没有被短路的门输入能正常工作。

再换一个脚予以短路进行检查，同样有输出信号的话，说明该门两个输入都能正常工

作。这样继续做下去。

当检查与非门和与门的输出时，用输入端接地的办法来观察是不恰当的。（但是：与门的输出通常是逻辑 1 或者与非门输出通常是逻辑 0 的时候，即负逻辑应用时不受此限制）。

输入脚不可与 5V 电源相连。输入端与自己的输出端相连会损坏 TTL 电路。

只要有可能，可以通过输入端接地的检查方法来寻找出现故障的门电路。

以视频信号发生器为例。Z_9 的脚 4 是脉冲发生器 Z_{26} 的输入倒相器的输出脚。为使 Z_{26} 的输入脚 13 和脚 5 都是逻辑 1，则 Z_9 的脚 3 应处于地电位。同样，Z_{27} 的闩锁输入接地的话，Z_{26} 的脚 12 、脚 4 等的输入则为逻辑 1 。

这样，把输入接地，在输出端假如不出现信号的话，那怎么办好呢？而且即使对所有有关输入进行检查，怎么也得不到输出，则要考虑下列两种情况：

① 门电路损坏。

② 输出端与＋5 伏电源连通或对地短路。

为了检查上述内容，可切断印刷板上的配电线，只需检查输出端的波形；切断连线后如果得到了正常输出，说明输出端连线在哪个地方有短路。例如，用作其它输入的大规模集成电路也许有短路。

如果检查结果表明印刷板上连线没有短路，那么其原因是大规模集成电路损坏，这时把它更换掉。请记住把割断的线焊接起来恢复原状。印刷板连线与＋5 伏电源线短路也许就是大规模集成电路损坏的原因。

当怀疑可能有短路时，最好是检查门输出电压。

门电路与 ＋Vcc 之间有短路时为＋5 伏，而 TTL 上的高电平只有 3.7V 左右。（对 CMOS 类型器件的高电平输出只比电源电压低几个毫伏）若知道器件与 ＋5 伏短路，那么请沿印刷板上的连线查找。

与接地端短路时。检查方法与上相同。这时用示波器探头接到被认为有短路的导线上，切断电源开关，分段进行验证。如果电压幅度不变的话，那么就可以认为那根线已被短路。

如果在切断电源时，电压瞬时上升，但过一会又降下来的话，则可认为那根线与别的 TTL 电路的输出端有短路。

2）关于短路的种类

根据引起短路的原因，要考虑下列五种情况：

（1）焊接处松动引起短路。

（2）焊锡粒引起的短路。

（3）由焊锡尖刺引起的短路。

（4）在印刷板线条之间没有完全被腐蚀掉的铜箔残余引起短路。

（5）由于元件质量引起的短路。

焊接处松动所引起的短路经常会碰到。诸如焊锡用得过多，焊接操作不十分小心，烙铁头过分粗大使大规模集成电路的相邻脚被短路，松动的引出脚之间的短路等等。

还有烙铁焊接时间过长，焊锡滴落在印刷板上引起线间短路，这种短路是极易发现的。

锡粒引起的短路是由于在计算机制造车间里采用了自动焊接装配线进行焊接所致，下面讲讲由于什么原因，使锡粒会溅落在印刷板的插座之间的。

这种自动焊接装置容易产生焊锡粒，在焊完印刷板之后，经过清洗，微小的焊粒偶而会

存留下来。

这样，加电检查时仍能正常工作，但在运输中锡粒受到震动，于是脱落下来造成想象不到的短路原因。对这样的事情，由于工厂制造方面的疏忽，却给用户增添了麻烦，请加以注意。

关于焊锡的尖刺，要去掉焊锡的多余部分，需作表面修整，有时，像针状的焊锡物造成了线与线之间的短路；还有某种冲击加到被焊接的面上，产生了细小的碎片，这些碎片可能使印刷板上的连线短路。

为了找出故障，用刀子把连接线割断时，请注意不要把金属碎末残留在机器上。

在 TRS-80 中使用的烙铁头要干净，不要用刮刀之类的工具去干别的事情。焊锡等金属碎片可用橡皮擦，轻轻一搓就会掉下来。

在腐蚀过程中存留在线间的短路现象显然是印刷板制造工艺上的差错。还有印刷板上被损伤的箔线之间也会引起短路。

焊接时间过长，印刷板导线发生过热，使连线从印刷板上剥落，这种剥落的线会导致短路。

部件损坏发生在大规模集成电路不良，二极管有问题等等容易造成短路的地方。对这种情况，及电源 Vcc 与地短路等有关问题已经作过介绍，这里不再赘述。

3）逻辑电路的短路

TTL 门的输出端相互短路会引起很大程度的变化。有时对某些功能尚能正常工作，而有时却使工作状态非常不正常。

输出端短路的个数较多的时候，虽然没有使用三态器件，却出现了类似三态器件的工作情况。如前所述，三态器件的信号电平，除了处于逻辑 1 和逻辑 0 以外，还可出现第三种逻辑电平。由于数据母线用三态器件来工作，欲观察该器件输出波形的话，它所执行的操作大致上就是所说的那些状态。

TTL 的输出与三态器件的工作方式完全不同，这种波形假如出现在输出端的话，那末要仔细地追查连接线在那个器件附近的配电线之间有无短路。

在印刷板上出现短路现象是经常的事。但也有极少情况发生在同一个大规模集成电路相邻输出脚之间，这是由于在组件封装时出现短路现象，这时应予更换。

同步发生器的计数器所使用的组件是 74LS93 。该计数器的输出端波形在示波器上观察时可以发现它是一种多值电平。

这种电平在计算数的时钟脉冲的间隙时间里为一恒定值。且在这样的计数情况下同时也构成多值电平，TTL 的输入端工作在逻辑 1 时，如工作电平在 2.4 伏以下属于不正常状态。

这个计数器的输出变化在 0.5 伏左右。构成最低电平时也应在 3.0 伏以上。这种现象在计数器中是特有的，也许正由于这种现象，可以不必怀疑计数器的输出。

4）地址母线以及数据母线的短路

地址母线与数据母线在印刷板上是并行走线的。在这些线之间经常会出现短路，但这样的故障是非常不好办的。

这些母线与系统的每一处都有关系。母线可以从印刷板插头引到外面来。

不管地址母线还是数据母线，在发生短路时都是麻烦的事情，数据母线则远远比地址母线还要难办。数据母线的状态如前所述是三态器件构成的，是一种双向数据流。

地址母线只在输出缓冲器上使用三态元件。在通常工作中，则不用三种状态进行工作。

它可与普通的 TTL 组件一样进行处理。(TEST*信号线的输入端接地时，那末地址母线即浮动。)

要了解地址母线的情况，可逐根用示波器检查地址线的波形；要特别注意在检查这样的情况：即至少有两根地址线波形是一样的，而且它们有点象三态器件那样浮动操作。

如果看到了这种信号的话，大多数原因是由于地址母线之间形成短路，请彻底进行检查，甚至要查及印刷板的每个角落，看这两根线之间有无短路。

在找不到短路所在的时候，实在不得已可用切割印刷板连线的办法来寻找。割断连线后，千万不要忘记把它们全部修复。

数据母线之间有短路时，与地址母线的情况一样，两根数据线同时被加上一样的信号，这时应分析其原因。本来数据线就以三种状态工作，所以在数据母线上出现相同信号的情况很多，故不容易进行分辨。

在检查数据母线时，将 TEST* 信号接地，数据母线全部处于浮动状态（即呈现高阻抗）。

这样，数据母线即处于象被切断一样的状态，在数据线上出现了浮动电压(约 1.5伏)。如果所测量的各母线状态为逻辑 0 或者逻辑 1 的话，那末就是用作缓冲器的大规模集成电路的性能不良。如果全部是浮动状态，那就在 +5 伏与母线之间接入 5 千欧姆的电阻，观察各条母线，并检查在其它母线上有没有出现高电压。

如果在与 +5 伏电压相连的母线以外的地方出现高电压的话，可认为这些母线中间有短路的地方。

5）引出脚弯曲

在 TRS-80 中，更换器件是很容易的，大规模集成电路用的插座数量很多。取下器件来查找故障，十分简便。但如果操作疏忽会使大规模集成电路的引出脚弯折，特别是 RAM 的大规模集成电路的引出脚容易弯曲，请多加注意。

如引出脚本来已经弯曲，再插入插座会使两者非常不吻合。这时用力按入会使大规模集成电路更加弯曲，导致与插座接触不良。

还有取出大规模集成电路时，特别是大型的四十个脚的大规模集成电路等器件，在拔出过程中也可能把引出脚搞弯。

要使 RAM 的引出脚不至弯曲，请从上向下瞄准后插进去。还有把 RAM 插入插座时应以一样高度，水平地安插 RAM 。

假如高度不一致的话，就会出现动摇，并且也不能正常工作。这时只要高度一致地按下 RAM，才能纠正故障。

6）双列直插式分路器

关于如前所述的双列直插式分路器，这里择要再作一些说明。首先请再次确定一下双列直插分路器所执行的程序是否正确。

还有，双列直插分路器如施加太大的压力往往会产生裂缝，在其中间会引起接触不良。用双列直插分路器编制程序时，分路器插入插座后应切断那些必须切断的地方。又，双列直插分路器的脚 1 与脚 8 比较脆弱容易断裂，务请注意。

若双列直插分路器产生了裂纹，把必须连接的部分用铅笔头轻轻地把接线杆压进去。

这样如果操作正确的话，即可更换双列直插分路器。如果更换有困难，也可以用导线把

断开的地方焊接在一起。

四、关于 TRS-80 的应用方法

在使用计算机之前，应明确使用的各种目的，即在计算机上干些什么。下面介绍关于驱动机器工作的一种方法。

在各种各样的场合下都可使用计算机。TRS-80 中，只用软件来做相应的工作，还可以做各种各样的游戏。但，并不仅仅如此，如果需要的话，利用硬件与外部设备连接起来，计算机的应用将会是怎么样呢？

为此首先有必要进一步熟悉弄懂有关软件。

使用 BASIC Ⅰ 语言，必须精通到根据机器语言编制程序的方法 那样的程度。

使用 BASIC Ⅱ 语言，由于采用了比较简单的“POKE”，“PEEK”以及“OUT”，“INP”等语言，使 BASIC 语言比较容易地控制与外部连接的 硬件接口。

使用机器语言是一件复杂的事情。使用编辑程序和汇编程序比用机器语言编写程序来得轻松方便。

归根到底，如果在外部与某种设备相连的话，为了控制那些外部电话，就要使用必要的软件。

TRS-80 还可以连接扩展接口设备，这样除了磁盘和打印机设备外，无须再加接控制外部特殊设备的硬件。

例如，每天到了某个时间，自动煮饭机开关接通，煮完饭，用砂锅烧开汤。如果早餐准备就绪，那么就能够使用电视代替眼睛来控制开关，或使用无线电发出音响的系统，以示意就餐。这将多么方便啊！

但是为了实现这种控制，必须根据那些目的来设计相应的接口。

对硬件的设计，在 TRS-80 中，要决定在两大类硬件中用哪种形式进 行控制。一般用下列两种方式来连接控制设备。

(1) 连接部分存贮器。

(2) 连接输入输出设备

在使用部分存贮器时请恰当地分配存贮器各部分的地址。

为了控制外部接口，需使用一部分存贮器，包括 16 根地址线和 8 根数 据 线。

通过输出设备控制接口时，要使用 WR* 信号。由 CPU 来检查外部接口状态使其 变 成为所需要的状态时，则要使用 RD* 信号。

用机器语言以及汇编语言来编制程序时，为了控制与外部相连的接口，要用到 WR（写入）命令和 LD （读出）命令。在 BASIC Ⅱ 中却使用了“POKE”和“PEEK”的执行语言。

以接通 100 伏电源为目的的接口举例——控制 100 伏电源的操作根据继电器接点所允许电流大小担负不同的功能；不管什么目的都可使用该接口——接口用十六进制的地址表示，分配在 8FFF 。数据用 02 表示。要实现“接通”电源那种功能的接口，用汇编 语言可写成：

```
LD 8FFFH, 02H; DENGENON
```

该汇编语言需转换成机器语言后执行程序，指定存贮器地址 8FFF，且在数据线出现02时即构成电源“接通”。

在 BASIC Ⅱ 语言中，使用执行命令 POKE：

POKE 36863，2

其中 36863 是十六进制 8FFF 的十进制数。

外部接口电路使用输入输出口的控制方法来代替存贮器的分配如下：

通过 CPU 用作输入输出设备控制口的最大数目多达 255 个。使用输入输出口时，分配存贮器的工作方式必需使用 16 根地址线，而实际使用其中 8 根地址线就够了，对应由 CPU 选定的接口，用数据线输入输出数据。

例如用于控制电源电路的接口地址表示成十六进制 FE，数据为 02，以此参数进行设计。这时用汇编语言写出程序是：

OUT 0FE，02H;DENGENON

用 BASIC Ⅱ 来写这条程序，则为：

OUT 255，2

两者执行的功能完全一样。

用来执行输入输出口的动作将使用 OUT* 和 IN* 的信号线。

不管使用上述存贮器分配还是通过输入输出口都可以控制外部接口，在 TRS-80 中只能使用 RAM 来完成这类功能（最大为 48K 字节），而划分存贮器的位置几乎都用光了，所以只能使用十六进制的 3000-36FF，3900-3BFF 范围内的内存地址（其中 3700-37FF 用于扩展接口）。

另一方面，输入输出口最多可用到 256 个。用于控制盒式磁带机的控制口规定为十六进制的 FF，除此以外的地址都可用于各种控制。

对于存贮器地址分配的情况，由 16 条地址线来完成。而使用输入输出口的情况，只用八条地址线，这仅是为了适应硬件设计而确定的。

1）根据存贮器分配，控制电源电路的外部接口举例

图 20 所示电路是存贮器分配方式用于控制电源电路接口的例子，这里只是一个例子，还可以用别的方法进行设计。

相应软件的条件：

(1) 能控制继电器用来管理电源电路。

(2) 存贮器地址分配在十六进制 8FFF。

(3) 能检查继电器处于什么状态，“接通”还是“断开”。

用于存贮器地址解码器是 Z_1、Z_2、Z_3，当加入 8FFF 的地址码时，8FFF *的信号为逻辑 0，满足 Z_{2C}、Z_{2D} 输入门的输入条件。

用 POKE 执行命令时，十六进制 8FFF 用对应的十进制 36863 来表示，则有：

POKE 36863，2

在执行此指令时，WR* 的信号为逻辑 0；Z_{2C} 的输出，即在 WRITE 线上有信号输出，构成了 Z_6 的时钟信号。

十六进制数据 02，使数据线 D_1 信号为逻辑 1，触发器 Z_6 被置位。

Z_6 输出为高电平时，使 Z_{7A}～Z_{7C} 继电器驱动器工作，继电器 K_1 上流过电流，则将电源电路的接点闭合，完成了接通电源的操作。

假如使用下列指令，继电器 K_1 则切断电源：

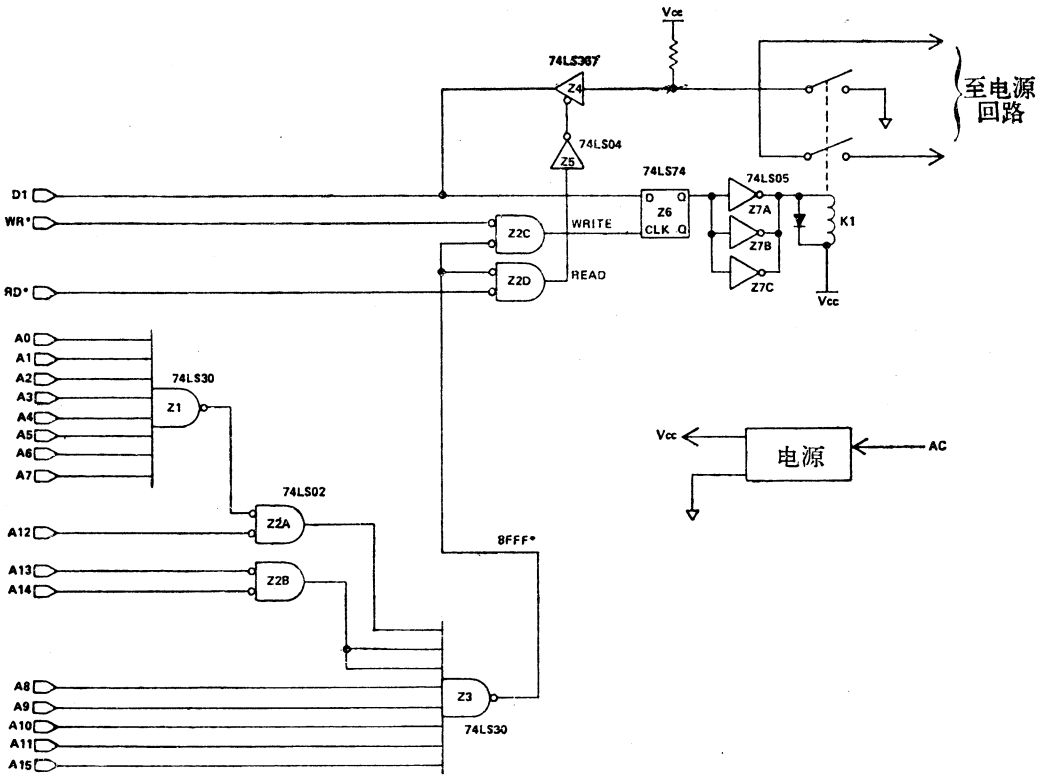


图20　根据存贮器分配控制电源电路

POKE 36863，0

为了检查电源电路控制接口继电器的工作情况，可使用 PEEK 执行指令。

若继电器接通，继电器 K_1 有电流流过，在电源电路继电器接点闭合的同时，另一个继电器接点闭合，将 Z_4 的输入端接地。

这时，Z_4 的输入端为 0 伏。

那末根据执行命令 PEEK，如使用下面指令：

A＝PEEK（36863）

在 8FFF* 的信号为逻辑 0，且 RD* 的信号为逻辑 0 的情况下，Z_4 的三态器件的输出端根据输入状态来确定输出。

因此，继电器工作为“接通”状态时，数据线 D_1 输出为逻辑 0；继电器切断时即切断电源电路，D_1 输出为逻辑 1。

由于这个输出接到 D_1，读出数据为下列所示：

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
继电器“接通”	×	×	×	×	×	×	0	×
继电器“断开”	×	×	×	×	×	×	1	×

其中×表示可以任意，或者 0 或者 1。

为了明确继电器状态，有必要使用下列的 BASIC 程序：

```
800 B=2
900 IF A AND B=0 THEN GOTO1980
```

当 A 和 B 相与的结果等于 0 时，表示继电器工作，此时执行 程序移到 1980。

下面指出在构成接口时所注意的事项：

① 地址完全与解码器相接。

② 在 WRITE 指令结束时，执行 READ 指令应确定动作是否正确。

③ 从数据线读入数据时，中间经过闩锁电路（不一定必须通过闩锁电路）。

④ 供电电源由内部提供。

⑤ 对应计算机侧面的输出，设置了一块大规模 TTL 集 成电路板。

在上述内容中，以④⑤两点为重要，不遵守这两点，就会造成误操作。

2）根据输入输出口控制电源电路的外部接口举例

图 21 是根据输入输出口控制电源电路的 电路图。控制门 Z_2 以右的电路与图 20 电路大致相同。与图 20 不同之处，是在地址译码器电路中 WR* 被 OUT*所取代，还有 RD* 换成了 IN*。

用输入输出口设计外部接口的 情况，只用 8 条地 址 线。根据 Z_1 和 Z_4 的地址，对十六进制的 FE 进行译码。这个 FE 相当于十进制 254。

用 BASIC Ⅱ 语言书写程序时，把 POKE 换成下式来表示：

OUT 254，2

PEEK 则改写成下述形式：

A＝INP(254)

对此写法，可执行与 POKE，PEEK 一样的操作。

根据OUT254,2执行字节，Z_1的FE*信号输出为逻辑0时，Z_{2A}或Z_{2B}才可能执行操作。

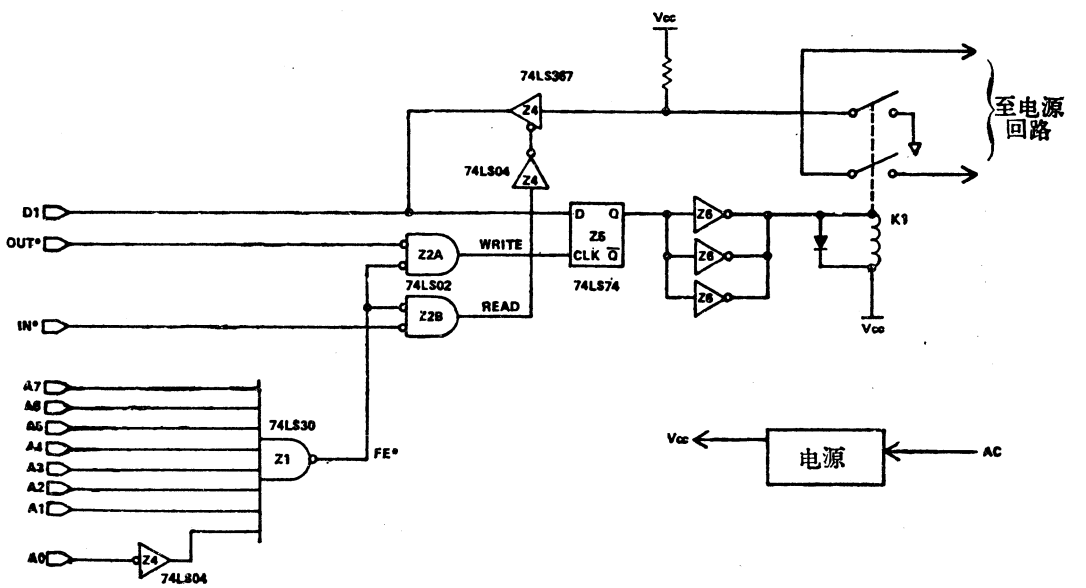


图21　根据输入输出口控制电源电路

输出 (OUT) 时，OUT* 的信号为逻辑 0，这时在 WRITE 线上有信号输出。

另外在输入 (INP) 时，IN* 的信号为逻辑 0，则输出 READ 信号，且把三 态 器件 Z_4 的输入条件送到 D1 数据线。

如上所述，在执行存贮器分配以及根据输入输出口执行控制中，使用地址母线的方法是

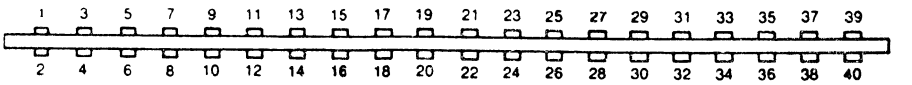
根据使用不同的 WD*，RD*，OUT*，IN* 等等信号而有所不同，但有关的操作没有什么特别的本质差别。

下面给出扩展接口引出脚信号及元件表。

扩展接口用的引出脚连接图

脚号	信号名称	内　　容
1	RAS*	动态 RAM 用的列地址选通信号
2	SYSRES*	系统复位输出
3	CAS*	动态 RAM 用的行地址选通信号
4	A10	地址输出线
5	A12	〃
6	A13	〃
7	A15	〃
8	GND	信号地线
9	A11	地址输出线
10	A14	〃
11	A8	地址输出线
12	OUT*	用于输出设备的选通输出
13	WR*	写入存贮器的选通输出
14	INTAK*	中断认可信号输出
15	RD*	读出存贮器的选通输出
16	MUX	16 脚动态 RAM 地址转换信号
17	A_9	地址输出线
18	D_4	数据出入线
19	IN*	用于输入设备的选通输出
20	D_7	数据出入线
21	INT*	中断输入（可以掩蔽）
22	D_1	数据出入线
23	TEST*	该输入端接地 A_0～A_{15} D_0～D_7 WR* RD* IN* OUT* RAS* CAS* MUX 等均为高阻抗
24	D_6	数据出入线
25	A0	地址输出线
26	D_3	数据出入线
27	A_1	地址输出线
28	D_5	数据出入线
29	GND	接地
30	D0	数据出入线
31	A_4	地址输出线
32	D_2	数据出入线
33	WAIT*	用于等待处理的输入信号，可供低速存贮器使用
34	A_8	地址输出线
35	A_5	〃
36	A_7	〃
37	GND	信号地线

脚号	信号名称	内　　容
38	A_6	地址输出线
39	＋5 V	＋5伏（限制输出电流、BASICⅡ 时接 地）
40	A_2	地址输出线



图为从键盘后面向前看的脚号

凡是与 AMP P/N88103-1同类产品的插头均可使用

图22　扩展接口的插头信号

3）TRS-80 元件表

BASICⅠ 印 刷 板

记号	内　　容	元件编号
	逻辑印刷板	1700069
	电　　容	
C1	200μF, 16V	1500059
C2	10μF, 16V	1500012
C3	0.01μF, 10%, 25V	1500047
C4	10μF, 16V	1500012
C5	10μF, 16V	1500012
C6	100μF, 16V	1500014
C7	0.01μF, 10%, 25V	1500047
C8	2.200μF, 35V	1500064
C9	10,000μF, 16V	1500058
C10	10μF, 16V	1500012
C11	10μF, 16V	1500012
C12	470pF, 50V	1500057
C13	470pF, 50V	1500057
C14	0.01μF, 10%, 25V	1500047
C15	0.01μF, 10%,25V	1500047
C16	0.1μF, 10%, 12V	1500052
C17	0.1μF, 10%, 12V	1500052
C18	0.1μF, 10%, 12V	1500052
C19	0.1μF, 10%, 12V	1500052
C20	330pf, 10%, 50V	1500062
C21	750pF, 10%, 50V	1500050
C22	0.1μF, 10%, 12V	1500052
C23	0.1μF, 10%, 12V	1500052
C24	220pf, 10%, 50V	1500061
C25	220pF, 10%, 50V	1500061
C26	0.047μF, 100V	1500051
C27	0.022F, 100V	1500023
C28	0.1μF, 10%, 50V	1500053
C29	0.1μF, 10%, 12V	1500052
C30	0.1μF, 10%, 50V	1500053
C31	0.1μF, 10%, 12V	1500052
C32	0.1μF, 10%, 50V	1500053
C33	0.1μF, 10%, 12V	1500052
C34	0.1μF, 10%, 50V	1500053
C35	0.1μF, 10%, 12V	1500052
C36	0.1μF, 10%, 12V	1500052
C37	0.1μF, 10%, 12V	1500052
C38	0.1μF, 10%, 12V	1500052
C39	0.1μF, 10%, 12V	1500052
C40	0.1μF, 10%, 12V	1500052
C41	0.1μF, 10%, 12V	1500052
C42	22μF, 16V	1500055
C43	47pF, 10%, 50V	1500048
C44	0.1μF, 10%, 12V	1500052
C45	0.1μF, 10%, 12V	1500052
C46	0.1μF, 10%, 12V	1300052
C47	0.1μF, 10%, 12V	1500052
C48	0.1μF, 10%, 12V	1500052
C49	0.1μF, 10%, 12V	1500052
C50	0.1μF, 10%, 12V	1500052

记号	内　　容	元件编号
C51	0.1μF, 10%, 12V	1500052
C52	0.1μF, 10%, 12V	1500052
C53	0.1μF, 10%, 12V	1500052
C54	0.1μF, 10%, 12V	1500052
C55	0.1μF, 10%, 12V	1500052
C56	0.1μF, 10%, 12V	1500052
C57	10μF, 16V	1500012

二　极　管

CR1	1N4735, 10%, 6.2V稳压管	4800021
CR2	1N5231, 5%, 5.1V稳压管	4800022
CR3	1N4148, 75V	4800002
CR4	1N4148, 75V	4800002
CR5	1N4148, 75V	4800002
CR6	1N4148, 75V	4800002
CR7	1N4148, 75V	4800002
CR8	桥式全波整流器 MDA202, 2A, 202V	4800023
CR9	1N982, 75V, 稳压管	4800026
CR10	1N982, 75V, 稳压管	4800026

插　座

J1	DIN制5脚插座	2100033
J2	DIN 5 脚插座	2100033
J3	DIN 5 脚插座	2100033

继　电　器

K1	5V 继电器	4500001

晶　体　管

Q1	2N3904, NPN	4822001
Q2	MPS3906, PNP	4822003
Q3	TIP29, 推动管	4820004
Q4	2N6594, 功率管	4824003
Q5	MPS3906, PNP	4822003
Q6	MJE34, 功率管	4824002

记号	内　　容	元件编号

电　阻

R1	68Ω, 1/2W, 5%	4708022
R2	2.7K, 1/4W, 5%	4704056
R3	750Ω, 1/4W, 5%	4704044
R4	0.33Ω, 2W, 5%	4717004
R5	1KTrimPot, 30%	4750019
R6	1.2K, 1/4W, 5%	4704049
R7	1.2K, 1/4W, 5%	4704049
R8	100K, 1/4M, 5%	4704087
R9	3.3K, 1/4W, 5%	4704058
R10	1K, 可变, 30%	4750019
R11	3.3K, 1/4W, 5%	4704058
R12	3.3K, 1/4W, 5%	4704058
R13	2.2K, 1/4W, 5%	4704054
R14	12K, 1/4W, 5%	4704070
R15	1.5K, 1/4W, 5%	4704050
R16	1.2K, 1/4M, 5%	4704049
R17	2K, 1/4W, 5%	4704053
R18	5.6Ω, 3W, 5%	4717003
R19	220Ω, 1/2W, 5%	4708032
R20	100K, 可变, 20%	4750018
R21	100K, 可变, 20%	4750018
R22	75Ω, 1/4W, 5%	4704023
R23	120Ω, 1/4W, 5%	4704027
R24	680K, 1/4W, 5%	4704100
R25	1.6MΩ, 1/4W, 5%	4704106
R26	1MΩ, 1/4W, 5%	4704102
R27	330Ω, 1/4W, 5%	4704036
R28	270Ω, 1/4W, 5%	4704034
R29	1.8K, 1/4W, 5%	4704052
R30	47Ω, 1/4W, 5%	4704019
R31	10Ω, 1/4W, 5%	4704011
R32	10K, 1/4W, 5%	4704068
R33	360K, 1/4W, 5%	4704098
R34	470K, 1/4W, 5%	4704097
R35	470K, 1/4W, 5%	4704097
R36	360K, 1/4W, 5%	4704098
R37	560K, 1/4W, 5%	4704099
R38	470K, 1/4W, 5%	4704097
R39	4.7K, 1/4W, 5%	4704061

记号	内　　容	元件编号
	电　阻	
R40	4.7K, 1/4W, 5%	4704061
R41	470K, 1/4W, 5%	4704097
R42	1.0MΩ, 1/4W, 5%	4704102
R43	10K, 1/4W, 5%	4704068
R44	10K, 1/4W, 5%	4704068
R45	470K, 1/4W, 5%	4704097
R46	910Ω, 1/4W, 5%	4704046
R47	10K, 1/4W, 5%	4704068
R48	4.7K, 1/4W, 5%	4704061
R49	4.7K, 1/4W, 5%	4704061
R50	4.7K, 1/4W, 5%	4704061
R51	4.7K, 1/4W, 5%	4704061
R52	910Ω, 1/4W, 5%	4704046
R53	1.2K, 1/4W, 5%	4704049
R54	7.5K, 1/4W, 5%	4704066
R55	7.5K, 1/4W, 5%	5704066
R56	220K, 1/4W, 5%	4704092
R57	4.7K, 1/4W, 5%	4704061
R58	4.7K, 1/4W, 5%	4704061
R59	4.7K, 1/4W, 5%	4704061
R60	4.7K, 1/4W, 5%	4704061
R61	4.7K, 1/4W, 5%	4704061
R62	4.7K, 1/4W, 5%	4704061
R63	4.7K, 1/4W, 5%	4704061
R64	330Ω, 1/4W, 5%	4704036
R65	10K, 1/4W, 5%	4704068
R66	4.7K, 1/4W, 5%	4704061
R67	100Ω, 1/4W, 5%	4704025
	开　关	
S1	4PDT	5102008
S2	DPDT	5102009
	散 热 片	
	Q4散热片	5300003
	Q6-14 散热片	5300002

记号	内　　容	元件编号
	组 件 插 座	
X3	16脚组件插座	2100037
X13	16脚组件插座	2100037
X14	16脚组件插座	2100037
X15	16脚组件插座	2100037
X16	16脚组件插座	2100037
X17	16脚组件插座	2100037
X18	16脚组件插座	2100037
X19	16脚组件插座	2100037
X20	16脚组件插座	2100037
X32	24脚组件插座	2100034
X33	24脚组件插座	2100034
X39	40脚组件插座	2100035
X71	16脚组件插座	2100037
	晶　体	
Y1	10.6445MHz, 0.004%	2300004
	集 成 电 路	
Z1	723, DIP, 电压调整用	3100001
Z2	723, DIP, 电压调整用	3100001
Z4	LM3900, 差分放大器	3100002
Z5	74C04 CMOS	3102026
Z6	74C04 CMOS	3102027
Z7	74LS 74	3102015
Z8	Z4LS153	3102019
Z9	74LS04	3102008
Z10	74LS166	3102021
Z11	74LS166	3102021
Z12	74LS93	3102017
Z21	74LS156	3102028
Z22	74LS367	3102024

记号	内　　容	元件编号
	集 成 电 路	
Z23	74LS32	3102014
Z24	74LS132	3102018
Z25	74LS32	3102014
Z26	74LS20	3102011
Z27	74LS175	3102023
Z28	74LS174	3102022
Z29	MCM6670，用于字符发生	3108001
Z30	74LS02	3102007
Z31	74LS157	3102020
Z32	74LS93	3102017
Z33	2K×8ROM A，450ns	3108011
Z34	2K×8ROM B，450ns	3108012
Z35	74LS157	3102020
Z36	74LS32	3102014
Z37	74LS02	3102007
Z38	74LS367	3102024
Z39	74LS367	3102024
Z40	Z80 微处理器	3110001
Z41	75452	3106002
Z42	74LS04	3102008
Z43	74LS157	3102020
Z44	74LS367	3102024
Z45	2102，AN-4L，1K静态RAM	3108002
Z46	2102，AN-4L，1K静态RAM	3108002
Z47	2102，AN-4L，1K静态RAM	3108002
Z48	2102，AN-4L，1K静态RAM	3108002
Z49	74LS157	3102020
Z50	74LS93	3102017

记号	内　　容	元件编号
Z51	74LS157	3102020
Z52	74LS04	3102008
Z53	74LS132	3102018
Z54	74LS30	3102013
Z55	74LS367	3102024
Z56	74LS92	3102016
Z57	74C04 CMOS	3102027
Z58	74LS92	3102016
Z59	74LS175	3102023
Z60	74LS367	3102024
Z61	2102，AN-4L，1K静态RAM	3108002
Z62	2102，AN-4L，1K静态RAM	3108002
Z63	2102，AN-4L，1K静态RAM	3108002
Z64	74LS157	3102020
Z65	74LS93	3102017
Z66	74LS11	3102010
Z67	74LS367	3102024
Z68	74LS367	3102024
Z69	74LS74	3102015
Z70	74LS74	3102015
Z71	不用	
Z72	74LS367	3102024
Z73	74LS32	3102014
Z74	74LS00	3102006
Z75	74LS367	3102024
Z76	74LS367	3102024

4K RAM明细表

记号	内　　容	元件编号
	集 成 电 路	
A3	双列直插分路器	2100041
A71	双列直插分路器	2100041
Z13	4096 bit，动态 RAM，450ns	3108003
Z14	4096 bit，动态 RAM，450ns	3108003
Z15	4096 bit，动态 RAM，450ns	3108003
Z16	4096 bit，动态 RAM，450ns	3108003
Z17	4096 bit，动态 RAM，450ns	3108003
Z18	4096 bit，动态 RAM，450ns	3108003
Z19	4096 bit，动态 RAM，450ns	3108003
Z20	4096 bit，动态 RAM，450ns	3108003

或

16K RAM 明细表

A3	双列直插分路器	2100041
A71	双列直插分路器	2100041
Z13	16384 bit，动态 RAM，450ns	3108009
Z14	16384 bit，动态 RAM，450ns	3108009
Z15	16384 bit，动态 RAM，450ns	3108009
Z16	16384 bit，动态 RAM，450ns	3108009
Z17	16384 bit，动态 RAM，450ns	3108009
Z18	16384 bit，动态 RAM，450ns	3108009
Z19	16384 bit，动态 RAM，450ns	3108009
Z20	16384 bit，动态 RAM，450ns	3108009

键　盘

	印 刷 电 路 板 键 盘	1700070

电　容

C1	0.1μF，10%，12V	1500052
C2	0.1μF，10%，12V	1500052

二　极　管

记号	内　　容	元件编号
CR1	LED，HP5082-4850	2400025

键　盘

KB1	DS5300.53个键，2个点键	5100013

电　阻

R1	4.7K，1/4W，5%	4704061
R2	4.7K，1/4W，5%	4704061
R3	4.7K，1/4W，5%	4704061
R4	4.7K，1/4W，5%	4704061
R5	4.7K，1/4W，5%	4704061
R6	4.7K，1/4W，5%	4704061
R7	4.7K，1/4W，5%	4704061
R8	4.7K，1/4W，5%	4704061
R9	330Ω，1/4W，5%	4704036

集　成　电　路

Z1	74LS05	3102009
Z2	74LS05	3102009
Z3	74LS368	3102025
Z4	74LS368	3102025

BASICⅡ 配套

	印刷电路板 BASIC Ⅱ ROM 适配器	1700081
J1	24脚组件插座	2100034
R1	4.7K，1/4W，5%	4704061
Z1	4K×8ROM，450ns，ROM A	3108013
Z2	4K×8ROM，450ns，ROM B	3108014
Z3	4K×8ROM，450ns，ROM C	3108015
Z4	74LS42	3102036

五、 BASIC Ⅱ 用的 ROM

由于可以改变 ROM，从而增强了 TRS-80 的功能。

TRS-80 的 BASIC Ⅰ 语言只能进行简单的数学处理，并具备程序显示的能力，而 BASIC Ⅱ 语言则较之有更强的功能。

BASIC Ⅰ 为 4K 字节的 ROM，而 BASIC Ⅱ 用了 12K 字节的 ROM。

通过增加 8K 字节的 ROM 以后，可用作程序语言编集，各式各样的函数，数据串的排列等处理。

就是不增加硬件，而只增加 ROM 就可发挥出意外优越的性能。

在 BASIC Ⅱ 的机器中，由 3 个 4K ROM 和 TTL 译码器以及扁平电缆组成。用扁平电缆输送 ROM 地址和数据。把这根电缆的插头插入到原来的 ROM 插座上，其中绿、橙、红、黄四根橡胶电缆请分别接到 CPU 印刷板上的 A_{11}、A_{12}、A_{13} 以及 ROM* 线上。参见图 23。

A_{11} 用于三块 ROM 的地址，与 Z_1、Z_2、Z_3 这些组件的脚 18 相连。A_{12} 和 A_{13} 与 Z_4 解码器的 A_0 和 A_1 相连，当 A_{12}、A_{13} 为 00 时，组件 Z_4 的引出脚 1 选择 ROM A 使其工作。

同样，A_{12}、A_{13} 为 01 时，选择 ROMB；A_{12}、A_{13} 为 10 时选择 ROMC，分别使它们工作。

ROM* 信号加到 Z_4 的脚 12 上。ROM* 为 0 时，把 A_{11}、A_{12}、A_{13} 地址线的这些输出送到 Z_4 的脚 1、2、3 作为主要输入去控制各 ROM。

ROM 的电源由主电缆供给。ROM 板的一部分电缆插到印刷板的 ROM 插座上，另一部分与 Z_{31} 的插座相连。

对于地址和数据，BASIC Ⅱ 与 BASIC Ⅰ 使用的情况完全一样。

（卢长津、李士才译　王余君、梁祖威校）

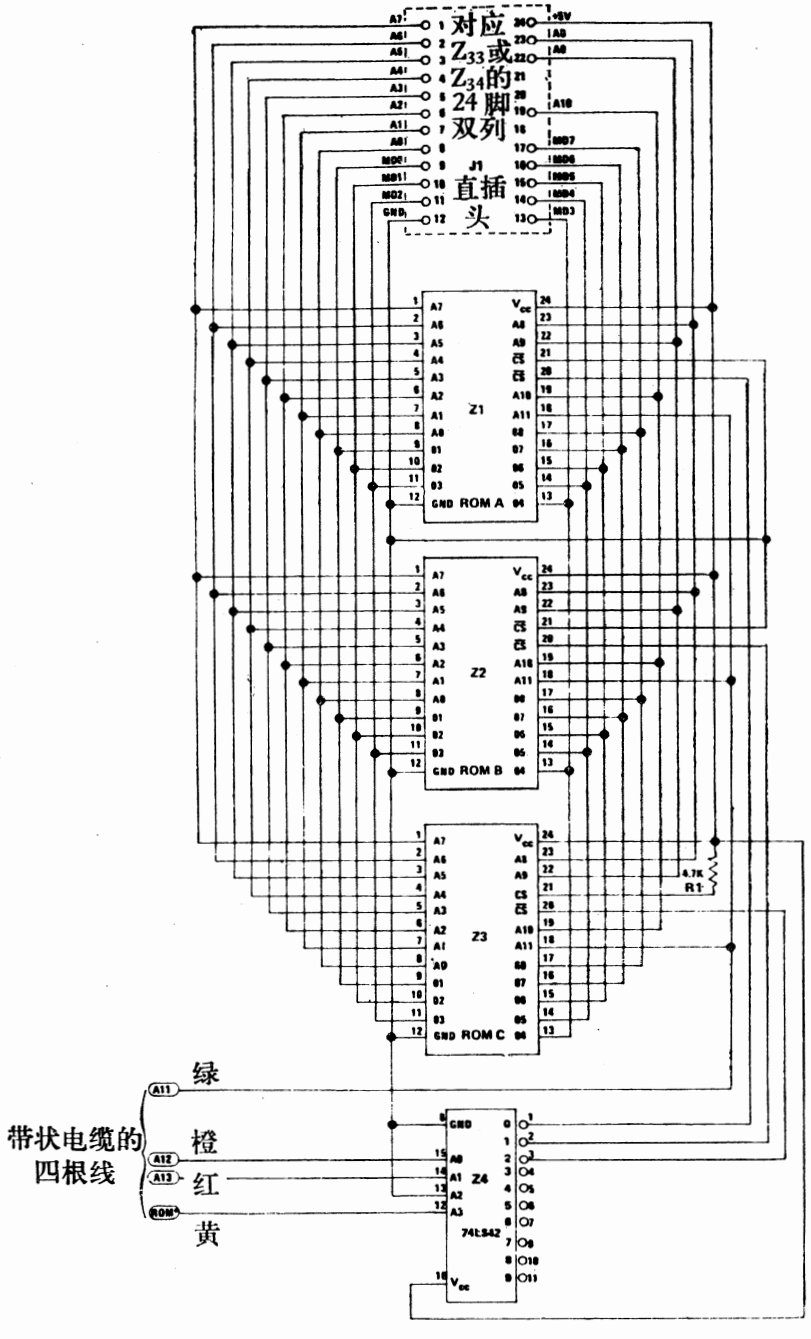


图23　BASICⅡ　的扩展印刷板

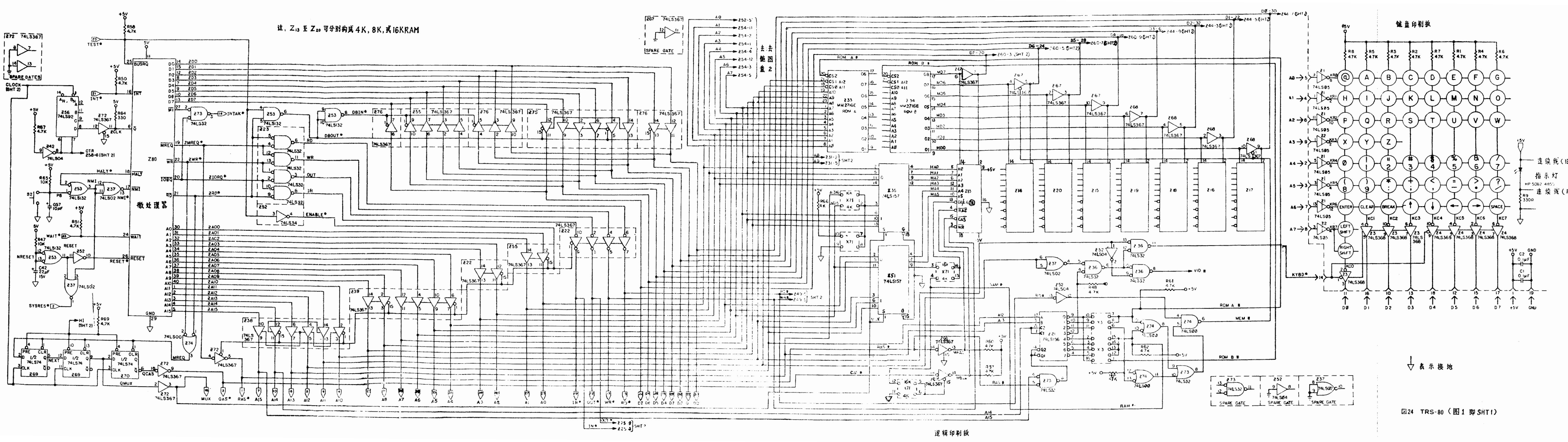


图24 TRS-80（图1 即SHT1）

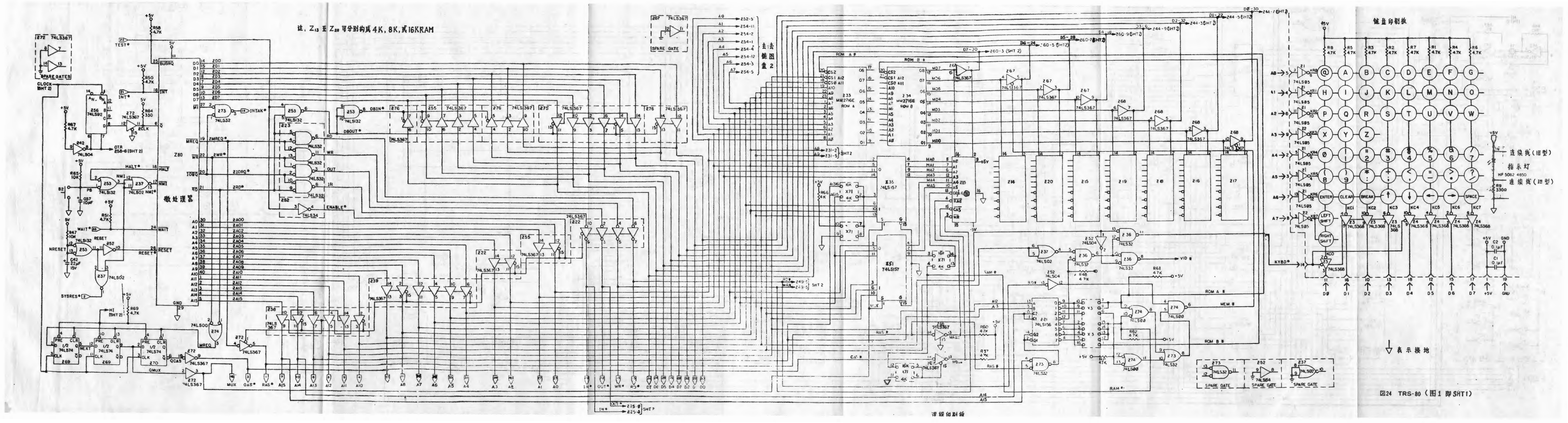
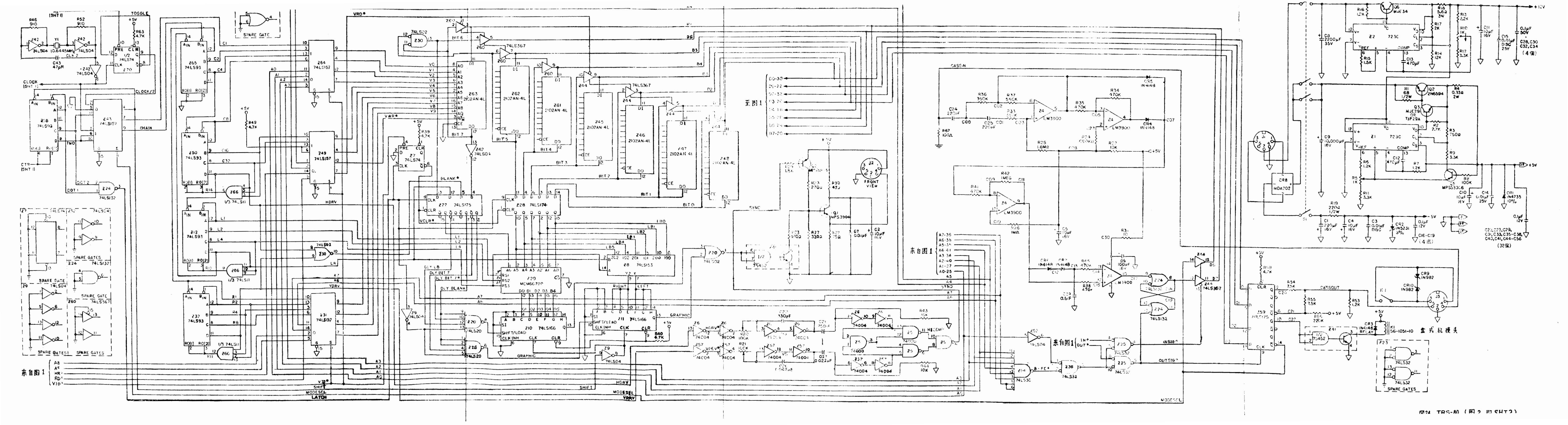


图24 TRS-80（图1 即SHT1）


TOGGLE
SPARE GATE
CLOCK (SHT 1)
CHAIN
CTR (SHT 1)
SPARE GATES
SPARE GATE
SPARE GATES
来自图1
VRD*
VWR*
BLANK*
HDRV
VDRV
SHIFT
MODSEL
LATCH
LOAD
DLY BLANK
DLY LD
GRAPHIC
SHIFT
MODSEL
VIDEO
SYNC
FRONT VIEW
至图1
来自图1
来自图1
CASCIN
CASCOUT
INSIG*
OUTSIG
WODSEL
TRS-80 (图2 的SHT2)

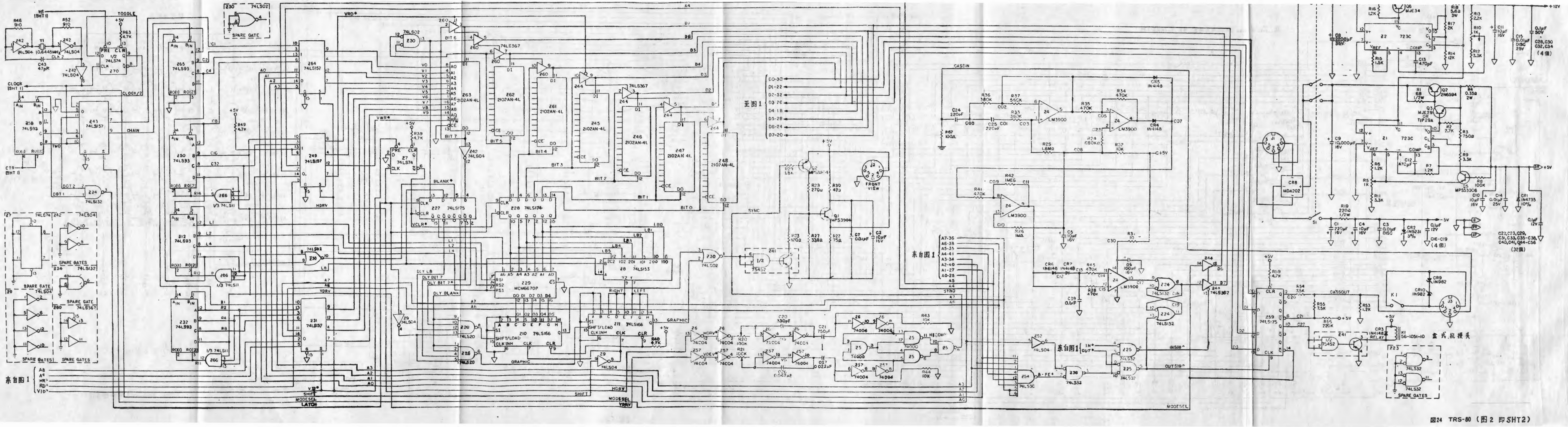


图24 TRS-80（图2 即SHT2）

第二篇　TRS-80 扩展接口操作员手册

引　言

TRS-80 扩展接口（见图 1）包括一个外壳，一个直流电源，一根带状电缆，一个盒式录音机的跨线和第 2 号盒式录音机用的附加的录音机电缆。注意，在到货时直流电源并没有安装在机壳内，它必须按照“安装”一节中所叙述的步骤安装好。如图 2 所示。

外壳中安装扩展接口的印刷电路板(PCB)，二个直流电源，另外还可以容纳一个附加的扩展 PCB。扩展接口使用一个实时钟，并有一些插座以便扩充 RAM 到 32 K（增量为 16K）。

一个直流电源给 PCB 供电。另一个电源是TRS-80 的电源，这两个电源是可以互换的。

带状电缆的两端都有 40 腿的接头，它用于扩展接口和 TRS-80 相联，你还收到这些接头的罩子，这些连接头以后还要述及。

盒式录音机跨线的两端都是五腿的 DIN 插头，它把扩展接口的磁带输入/输出 (I/O)和 TRS-80 微型机右后边的磁带(TAPE)接头相联。

盒式机电缆是用来把扩展接口联到第 2 号盒带机上去的。

功能与优点

此接口可以将下列 Radio Shack 的设备加到你的系统中：

1）网式打印机(26—1151)

2）行式打印机(26—1150)

3）小型磁盘系统（26—1160/26—1161）

4）第 2 号盒式录音机（14—841）

网式打印机或行式打印机可将 TRS-80 所产生的信息打印成硬拷贝。

TRS-80 小型磁盘系统是软盘的一种小型品种。它比磁带机的存取速度快而且可提供巨大的存储容量。第 1 号磁盘可容纳约 50,000 字节的空位供存储文件之用。每个附加的磁盘可供89,600字节的空位。磁盘系统有自己的命令组，可以用来处理文件及扩展使用文件的功能。TRS-80 小型磁盘系统采用顺序的或随机的访问方式。有了磁盘后可以使用几个附加的Ⅱ级命令。

重要的注意事项：由于扩展接口中有了磁盘控制器，计算机将试图输入更多的命令。在计算机接入扩展接口时，就认为小磁盘已经接好了。要在无小磁盘的情况下使用扩展接口，可以按TRS—80键盘的 BREAK 键。这时就可无视磁盘工作方式而进入正常的Ⅱ级 BASIC 操作。

使用两个盒式录音机可以更为有效和方便地修改存在磁带机上的数据。举例来说，你的磁带上有关于工资的数据，这上面的信息可以用 1 号盒带机逐项地读出，然后修改或增加其内容，再存入第 2 号盒带机。这里所举的例子是一种非常简单的应用；然而，同时使用两条

磁带可以构成数据输入及输出的功能极强的子程序。

小心：这个设备只能用于Ⅱ级 BASIC，**不适**用于Ⅰ级 BASIC。

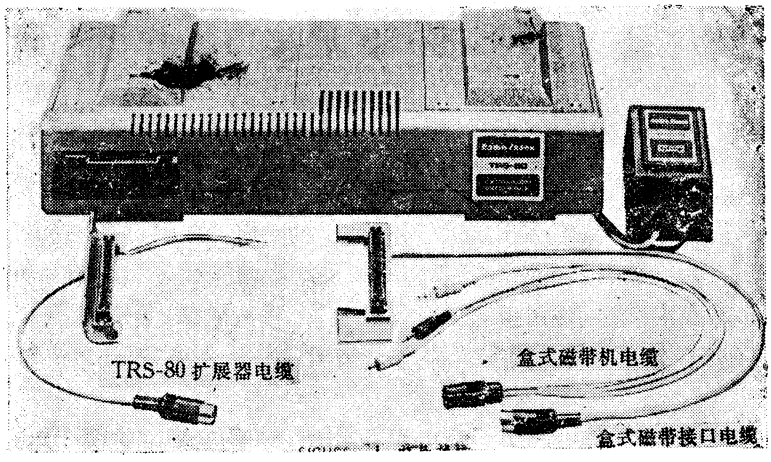


图1 扩 展 接 口

安 装

电源和 PCB 安装（见图 2）

打开电源盖（顶上右边）。先把直流电源的一根引线（DIN 插头）接到 PCB 的接头上，再安装直流电源，如图所示。把引线的剩余部分从外壳后面引出。在把盖板装好之前要把电源引线安放在盖的缺口部分。

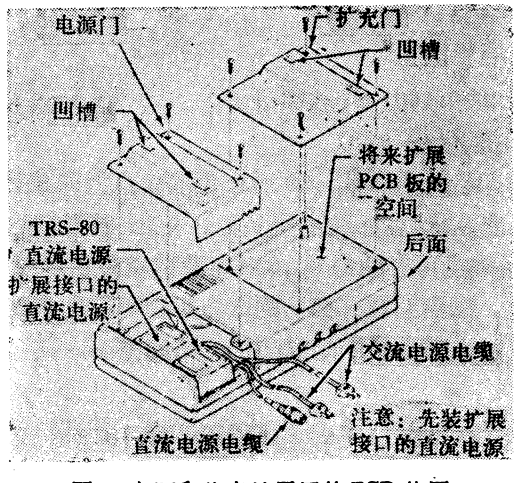


图2 电源和将来扩展用的 PCB 位置

为了把将来扩展用的 PCB 放入机壳部分，可把机壳顶上左边的盖板打开。

注意：开口(PORT) 这个词在本手册中是指机壳上的开口，它用于安装把 TRS-80 和扩展接口箱相联而用的接头座。

这些开口，除了扩展接口的开口外，都有可拆卸的门（或盖板）。为了取走这些门，可

以在门的右侧往下按，这样它就能稍微旋转一点，拿住它的左侧就可以把它拔下（位置见图3）。

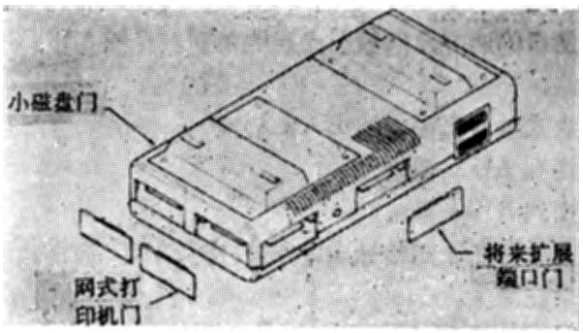


图3　扩展接口前视图——门已取走

电气连接（见图4）

转动TRS-80使它的后背面向你。找到开口门（1400083）；它在后盖板的右端。把它往上提并向右滑动再往上提就可以把它取下。

把TRS-80和扩展接口的罩子（14000217和14000214）放在带状电缆接头上，如图4所示。这两个罩代替了TRS-80上的门并盖住了扩展接口的开口，这些罩设计时就使它不能上下反插。它们用作接头的导向槽。把扩展接口的左前方的开口和TRS-80的开口用带状电缆相联。

把直流电源引线（DIN接头）联到TRS-80右后方的POWER（电源）接头上，把两根交流电源线都插到标准的120V* 交流插座上。

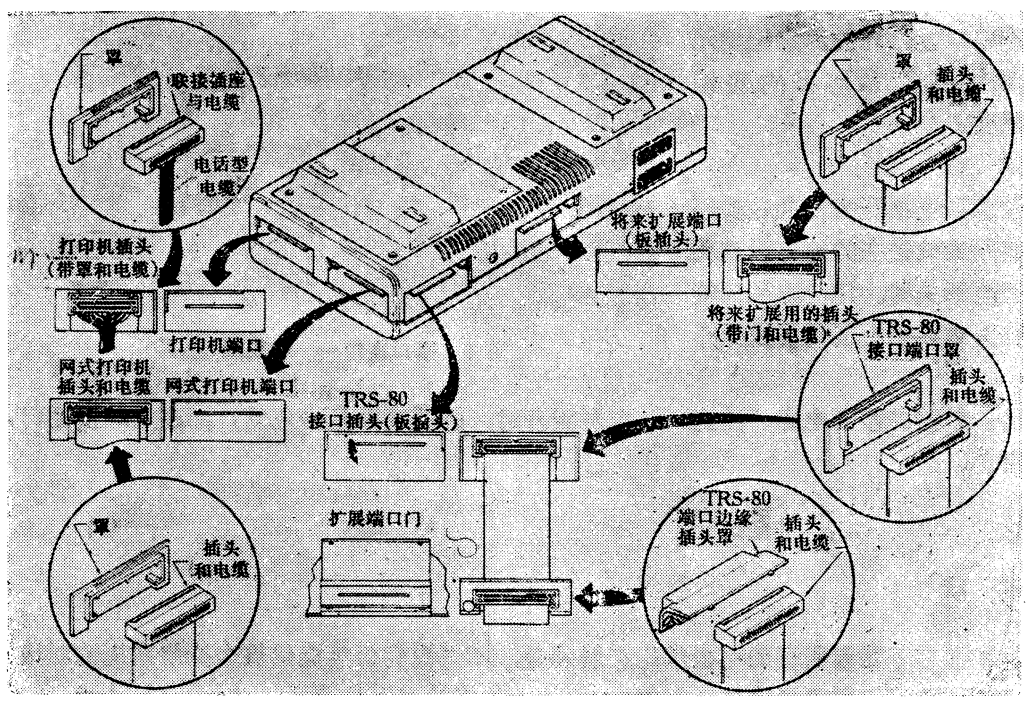


图4　接口连线——前视图

注：有些已改成220V——译者

扩展设备的互联电缆是随设备供应的。见图 4 可以了解罩的配置及安装。

把盒式录音机的电缆（一端为 DIN 插头，另一端是三个双芯插头）连接到 Tape I/O 接头，这个接头位于扩展接口的后方，电源引出线的对面（见图 5）。这根盒式录音机电缆的另一端的三个插头的接法：

1） 黑色插头接到盒式录音机侧面的 EAR 插座。

2） 大的灰色插头接到 AUX 插座。

3） 小的灰色插头接到 REM 插座。

注意：你的录音机上带有一个短路插头，把它插到 MIC 插座。这个插头可使内部传声器断开，以免它把声音录在带上。

把盒式录音机跨线接到扩展接口后面那个 DIN 插座，把另一端接到 TRS-80 右后方的 TAPE 插座。

把显示电缆从显示器接到 TRS-80 右后方的 VIDEO 插座。

注意：你的录音机可能是电池供电，也可能是交流 120V 供电的。因此交流电源线是任选件。

TRS-80 的扩展接口是设计成能安放显示器的。它把显示器的脚放在电源和 PCB 的盖板的凹槽处。

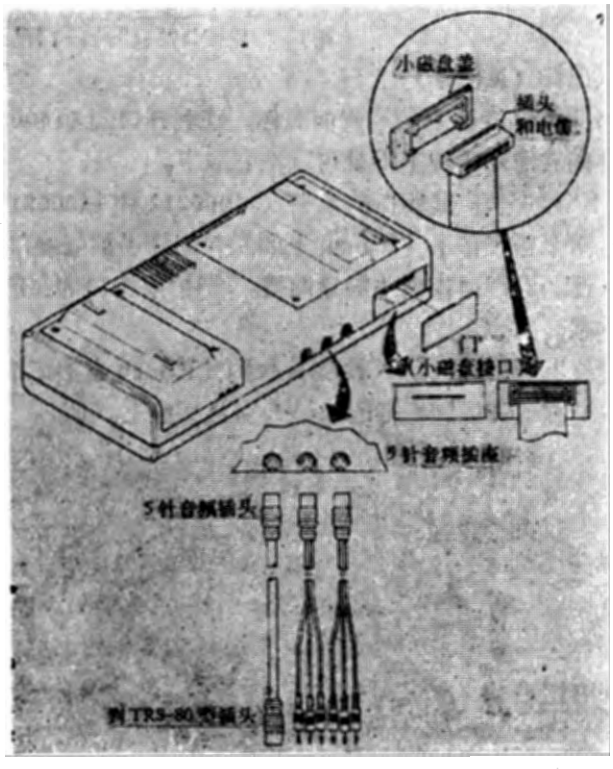


图 5 接口连线——后视图

操 作

注意：电源开关安装在扩展接口前面的缩进去的地方以避免偶尔把电源关掉。用橡皮头铅笔的橡皮头或其它小工具来按此开关。

将扩展接口接通电源。注意，在电源断开时，开关端面是白色的，而在电源接通时，它变成橙色。

结 语

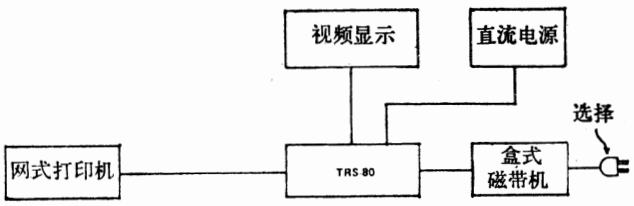
也许你并不需要所有的可提供的扩展设备，不过我们仍然提供整个扩展系统的联接电缆的罩子。请按图使用罩子以防止电缆与 PCB 上的接头之间的误插。

如果你丢失了门或罩子而要配一个，我们准备好了零部件一览表。你可根据零部件表和附图来确定它的零件号，通过当地 Radio Shack 的零售店可定购有关零件。

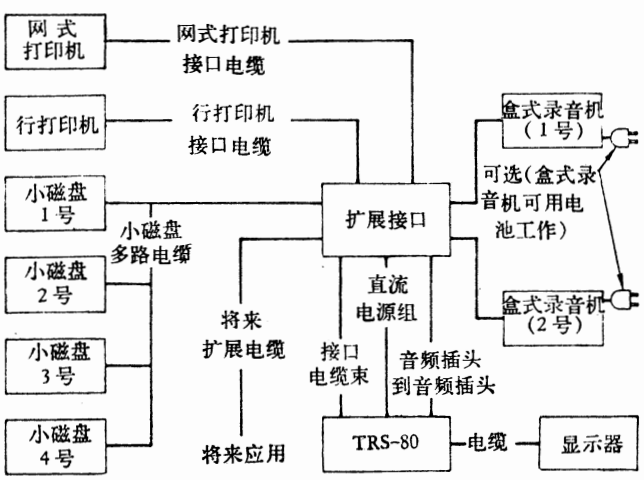
要使用 TRS-80 的扩展接口、行式打印机及小磁盘等设备，必须具有Ⅱ级 BASIC 的 TRS-80 微计算机。如把你只有Ⅰ级 BASIC 的机器，则必须改装成能用Ⅱ级程序。只有网式打印机才可以用在按Ⅰ级 BASIC 工作的 TRS-80 微计算机上。

我们正在不断地改进我们的 TRS-80 微计算机系统，你将从消息报导（你已列在邮寄地址单上）、补遗和手册的修订本中了解到。

完整的电气连接方块图见图 6 。



不带扩展接口的TRS—80 微计算机系统



不带扩展接口的TRS—80微计算机（最大系统）

图6　电气连接方块图

零　件　表

扩　展　接　口

小型磁盘的门	1400212
打印机的门	1400212
网式打印机的门	1400216

将来的扩展板的门	1400216
小型磁盘的罩	1400213
打印机的罩	1400213
网式打印机的罩	1400218
将来的扩展板的罩	1400218
TRS-80微计算机系统的罩	1400214

TRS-80 微计算机系统

门	140083
罩	1400217

（沈国锷译　周宝兴校）

第三篇　TRS-80扩展接口硬件手册

TRS-80 扩展接口大大提高了 TRS-80 Ⅰ级计算机的功能和用途。有了它，就可以连接多种外部设备（“外围”）——打印机、大容量存储设备、通信设备、声音分析器、語言識别设备、常规的 I/O设备等等到计算机上。

同样重要的是，扩展接口可以让你增加随机存储器。这样，系统就可处理更长的程序和更多的数据。一旦你已在TRS-80 中装满了 RAM(16K)，你可以在扩展接口中再加 16K 或 32K，这样总存储量可达 32K或 48K（1K＝1024 存储单元或字节）。

没有RAM 的扩展单元的目录号为 26-1140，16K RAM 的为 26-1141，32K RAM 的为 26-1142。

接口提供下列接线.

- 两台录音机的 DIN 插座，以便你从一台录音机上读出数据而写到另一台上。
- 行式打印机接头，以便接到 Radio Shack 行印机或其它合适的并行接口的打印机上。
- 小型磁盘接头，以便接到至多四台 Radio Shack 的小型磁盘驱动器。
- 扩展线路板的接头，用于增加RS-232C 接口（或其它常规设计的PCB）时，把它接到内附的接口插座中。
- 总线印刷板接头，它可以把 TRS-80 印刷板插头上的各种信号引出来。

除了上述这些连接外，扩展接口还提供一个对计算机每过 25 毫秒中断一次的节拍信号，这可以用来作实时钟程序的“节拍”。

注意：不要把扩展接口连接到Ⅰ级 TRS-80 上，这两者是不能相容的。

一、安　　装

安装电源

扩展接口有一个可放两个电源的空间，一个电源用于接口，另一个用于 TRS-80 计算机。把电源放在这个空间里可节省地方并使整个系统更易于“安排”些。

1）整个系统应处于断路 (OFF) 状态，两个电源都不要插入交流插座。

2）把电源空间盖板上的三个十字螺丝拧下（见图 1）。

3）观察此电源空间并找出在印刷电路板上的 5 腿 DIN 圆插座（见图 1），把一个电源处来的 DIN 插头接到此插座，这个刚接上的电源现在已可以向扩展接口供电。

4）把扩展接口的电源放进空间，如图 1 所示。

5）再把 TRS-80 的电源按图示放入空间。

6）把三条未连接的电线（两个 AC 电源线及一个 DIN 电缆）引出外壳，注意要把它

从外壳的缺口处往外引。

7） 把外盖装回原处，螺丝不能拧得太紧，否则会损坏外壳。

8） 把电源 DIN 插头联到 TRS-80 的 POWER 插座。但在没有把所有外围设备连接好之前先别插入交流电源线。

电缆、板插头及插座

图 2 （略，与452页图 1 同）表示扩展接口及附带的各种连接电缆。

同时你还收到各种板插头的罩子，包括 TRS-80 左后侧所用的一个。请你一定要用电缆接头的罩子，它们可用作为定位锁以免插错。

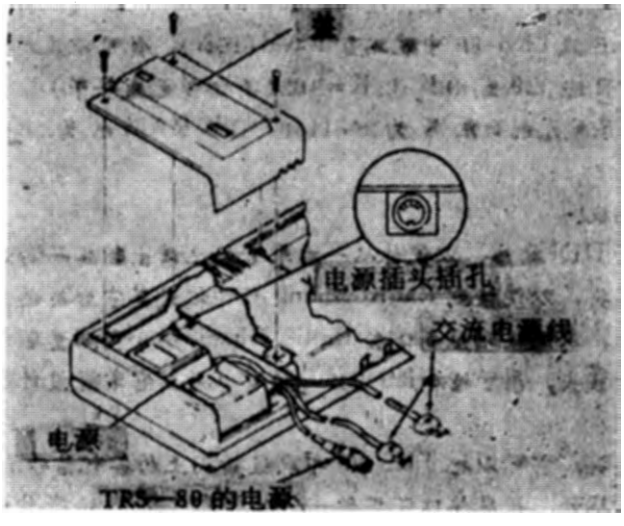


图 1 电源安装

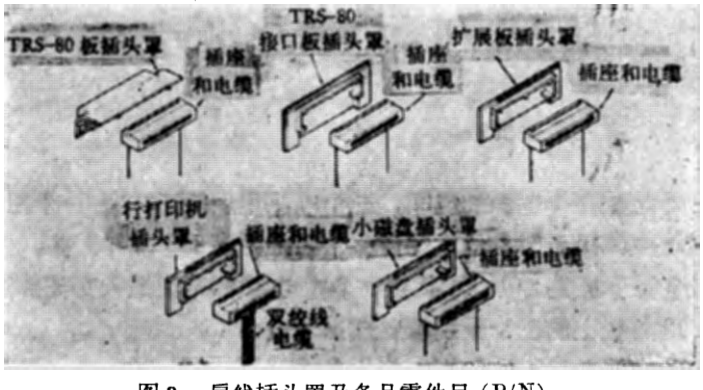


图 3 扁线插头罩及备品零件号 (P/N)

图 4 说明了各个板插头和插座的位置，注意板插头是在保护盖板后边遮蔽着的。不用时请勿将盖板去掉。

要取走盖板时，先按它的右侧使它稍有倾斜，然后拿住板的左侧并向外拉。

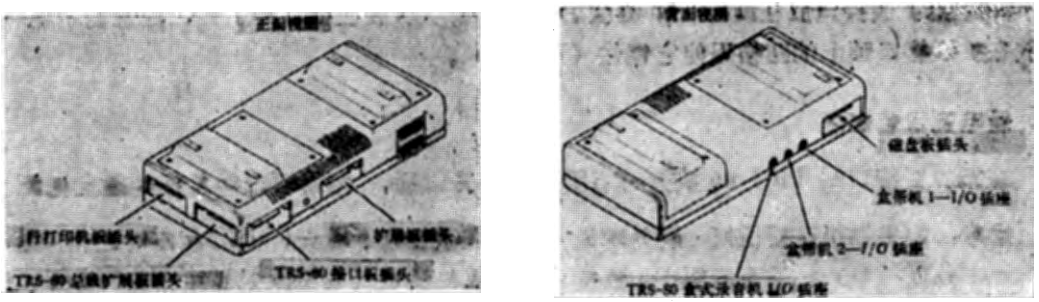


图 4　板插头和插座的位置

与 TRS-80 的联接

特别注意：带缓冲的接口电缆

某些装有存储器（16K 或32K RAM）的扩展接口单元有时会不按规定工作。例如存储内容消失及（或）磁盘自动再起动（在具有 TRS-80 磁盘操作系统时）。这种情况只有在1978年 11 月以前的产品上会发生。

如果你的接口有这种问题，请退回到 Radio Shack 商店或修理中心以便修改。

在这个修改中包括一个带缓冲的 TRS-80 扩展接口电缆，缓冲线路位于带状电缆的中部，它包含各种零件，如掉到地上或受到过热或过湿就会损坏（要避免此种情况）。

这种电缆必须用在改进后的扩展接口上。如果仍用原来的电缆，则会损坏电源。这个缓冲电缆的线路图可见本手册的最后部分。

在 1978 年 10 月份以后供应的接口或者提供缓冲电缆，或者内部经过了修改。除非是原来提供的或工厂在专门修改时所提供的，不要用缓冲电缆。缓冲电缆和非缓冲电缆是不可互换的。

1）　把 TRS-80 如图放置（图 5 后视图）。把盖住扩展接口扁线的门往上抬，稍往右推，再把它取下（注意不要把塑料小片弄碎）。

2）　把 TRS-80 扩展接口电缆的一端与板接头相连，电缆要从接头的底部引出。（注意：有些单元所提供的缓冲电缆有箭头画在缓冲盒上以表示那个接头是通往 TRS-80，哪个是通往扩展接口的。如果用的是无缓冲的电缆，则哪一端都可以接到 TRS-80 上）。

3）　把弯的 TRS-80 接头罩接到键盘外壳，带状电缆应从罩的下面引出（见图 5）。

4）　把平的罩接到电缆的另一端，电缆也从罩的底下引出（图 6）。

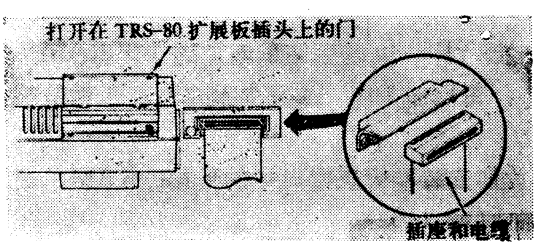


图 5　TRS-80的连线

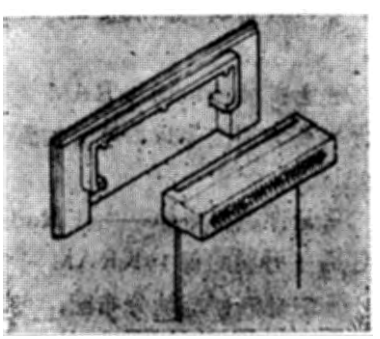


图 6　把罩放在接头上，电缆从下面引出

5） 把扩展接口放在 TRS-80 键盘后面，把显示器放在接口的上面，注意显示器“脚”要放在扩展接口顶上的凹槽里使它锁住（图 7 略）。

通电及注意事项

在把扩展接口与 TRS-80 按上一节所述连接好以后， 你就可以对各部件通上电源了。

注意：如果你想连接任何一种外部设备，要在系统处于断开(OFF)状态时进行（见外设一节）。千万不要在扩展接口或计算机通电状态下改变通往它们的接线!

1） 在插上交流电源线之前把一切连线接好。

2） 把你要用的外设电源都接通。

3） 接通扩展接口的电源，即按下电源按纽直到它“卡哒”一声固定在缩进的位置里，这可用一个铅笔的橡皮头或其它类似物件来做。（在通电状态时，按纽是缩进去的，所以不会因偶然机会把电源切断）。

4） 接通 TRS-80 的电源。

注意：如果你没有小型磁盘机接在系统上则在把计算机通电时要按下BREAK键，如果系统中有磁盘，则按照磁盘机的操作手册所规定的顺序操作。对某些行印机，在装入和操作 TRSDOS 时，必须打开行印机电源。但是，关闭扩展接口以前，要先关闭所有的外设。

要把系统断电时，其顺序正好相反，即首先切断 TRS-80 的电源。

交流电的注意事项

尽管 TRS-80 微计算机系统使用了最新的高效率低功耗的电 子 器 件，但 请 不要用家庭用的延长电源线、 多插头座等。 要把 TRS-80 尽量靠近标准的交流电源插座。不 然 的话，从小磁盘、行式打印机或其它外设来的线路干扰会通过共用的电源线传送到扩展接口或 TRS-80中去，引起内存丢失或偶然的清除。

如果你不能把系统中每个部件都直接插到交流电源插座，则可用一个高质量多头插座，这可从当地 Radio Shack 的商店购得。

还有一点值得注意的是，不要把交流电源线靠近系统 的 输 入 输 出 电 缆（例如带状电缆），这可以减少从 I/O 电缆感应干扰信号的可能性。

在扩展接口中增加RAM

如果你定购的扩展接口是没有 RAM 的(26-1140)，则接上接口不会改变系统的总 存 储量。

在给扩展接口增加 RAM 之前，必须先把 TRS-80 的存储量增加到 16KRAM。然后，当你再要增加内存时，就得加到扩展接口中，每批加 16K，共可加到32K，也就是整个系统达48K。

保留你的保修单——让 Radio Shack 为你安装和检查接口中的附加的RAM。安装费用已包括在另加的每 16KRAM 一套零件价格中(26-1102)。

下边是内存地址分布图，它表明各种可能方案的最高地址。

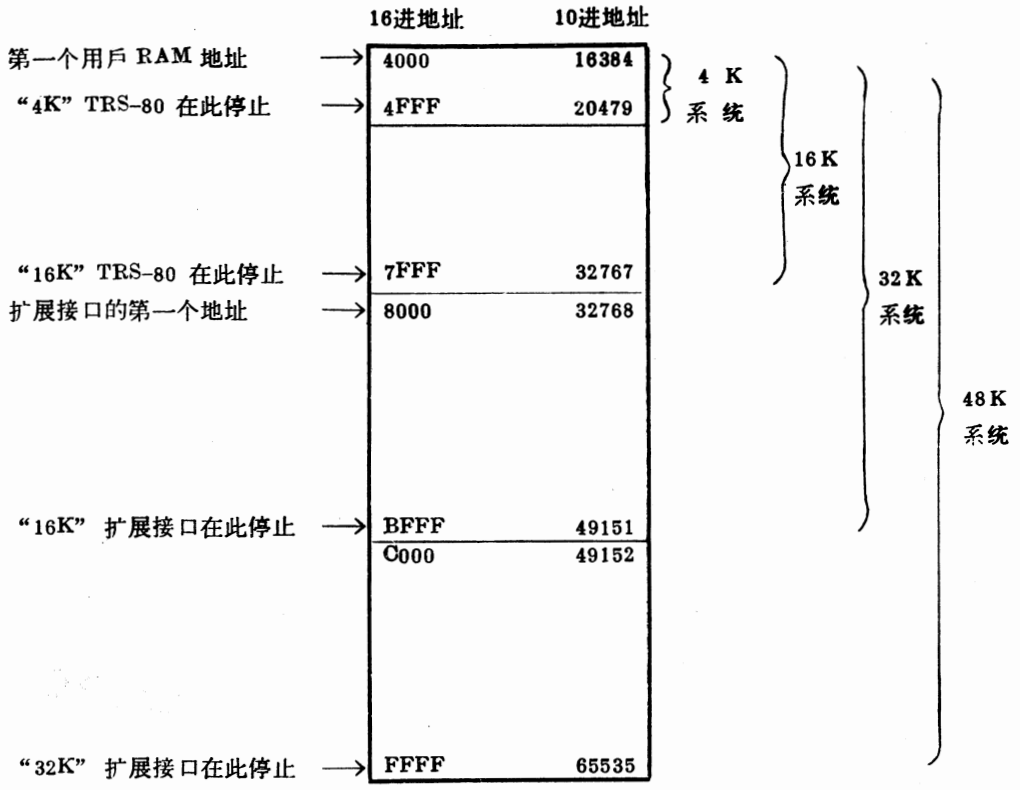


图 8　各种 TRS-80/扩展接口组合的RAM地址

二、外 部 设 备

（在连接各种外设时参考图2，3，4）

双盒式磁带机

在你的系统中增加第二台录音机，你就可以加快你的磁带输入/输出操作。例如，你可以从一台录音机读出程序或数据，编辑程序或修改数据，然后写到另一台上——这样就没有必要倒换磁带，返绕或重新按下放音、录音键等操作！

注意：如果你只有一台录音机，则直接把它联到 TRS-80 的 CASSETTE 插座上。把单个录音机通过扩展接口再连接是没有任何益处的！

1） 找出盒带机接口电缆。它的两端都有一个 DIN 插头。

2） 把一头插入 TRS-80 的 CASSETTE 插座，另一头插入扩展接口的录音机插座（电源线边上的 DIN 插座）。

3） 你现在有两个盒式录音机电缆——一个是 TRS-80 带来的，另一个是扩展接口带来的，把其中之一插到 1 号 Cassette DIN 插座，另一个插到 2 号插座。

4） 把两组录音机插头按下列方式插到两台录音机上：

（1）黑色插头插到 EAR 插座，

(2) 大的灰色插座插到 AUX 插座，

(3) 小的灰色插座插到 REM 插座。

5) 接到 1 号 Cassette 插座的录音机是 1 号盒带驱动器，2 号 Cassette 的是 2 号驱动器。

注意：根据你使用的录音机的不同，连接也可以不同，一般来说，要使用你在单录音机中使用的，同样的三插座方式。

双磁带机的操作

Ⅱ级 BASIC

用Ⅱ级 BASIC 选择盒式录音机时，要用下列语句（详情请见Ⅱ级BASIC使用手册）：

CLOAD#-1，文件名　　从 1 号录音机输入程序

CLOAD#-2，文件名　　从 2 号录音机输入程序

注意：文件名是任意的——如要输入的是磁带上首先遇到的程序，则可省略此文件名。

CLOAD#-1，？文件名　　内存与 1 号录音机所存程序相比较

CLOAD#-2，？文件名　　内存与 2 号录音机上所存程序相比较

注意：文件名也是任意的。

PRINT#-1，数据　　把数据存入 1 号录音机

PRINT#-2，数据　　把数据存入 2 号录音机

注意：数据是标准的打印清单。

INPUT#-1，变量名　　从 1 号录音机输入数据

INPUT#-2，变量名　　从 2 号录音机输入数据

注意：变量名是一个标准的输入清单，它必须与写入数据的 PRINT 清单相对应。

在 SYSTEM 方式时，总是选择 1 号录音机。

汇 编 语 言

要用汇编语言的 I/O 子程序选用 1 号盒式带时，在 16 进制地址 37 E 4 中存入零，要选用 2 号盒带时则在37E4中存入 1 。关于更详细的情况，请参考 TRS-80 的编辑汇编语言手册，目录号26-2002。

行式打印机

打印机板接头提供了并行的数据接口，它与 Radio Chack 的行印机是相容的。连接电缆和说明书是随行印机提供的。

使用其它打印机

某些别的打印机也可以连接到打印机接口上，一般说来，打印机必须：

1) 要有 34 芯的插座以便与扩展接口的板接头相配。

2) 能接受并行格式的 7 位或 8 位的 ASCII 数据。

3) 向计算机提供下列的状态信号：

BUSY（忙）（低＝不忙，可以送信息；高＝忙，不能送）

注意：其余的打印机状态信号是任选的，如下：

OUT OF PAPER (纸尽)——如果打印机不能提供此信号，则可把此输入端接到系统的公共地（以后可查技术说明书以找到是那条腿。并不一定要用 Radio Shack 的打印机)。

SELECT 和 FAULT（选择和故障）——如果打印机不提供这些信号，接口的外加电阻会自动地把这些输入端抬高以保证打印机的正常输出。

关于在打印机接头上各个信号的详细情况可在以后查阅技术手册。

行式打印机的输出

Ⅱ 级 BASIC

有两个语句用于输出信息到打印机，LPRINT 和 LLIST。详细情况见Ⅱ级 BASIC 参考手册。

注意：如果你没有接上行打印机，LPRINT和 LLIST 会使计算机锁住，这时得按复位键（在按复位键时要按 BREAK 键）。在接有扩展接口时，使计算机复位会使你在内存中的BASIC 程序都丢失。

汇 编 语 言

16 进制地址 37E8 是按内存划分给行式打印机接头的，它用作输入/输出端口。

在给端口送一个字节之前，检查允许使用状态位：

位	如置位则状态为
7	忙
6	纸尽
5	设备未选中
4	无故障

注意：查看你的行印机手册以查明用了哪些状态位。

当行印机处于READY(准备就绪)时，把字节(要打印的字符ASCII码)存在地址 37E8。

例如，如果ASCII码字符是存在寄存器C中：

```
        ;   SUBROUTINE TO OUTPUT
        ;   A BYTE TO A LINE PRINTER
        ;
        ;   LOAD ASCII—CODED CHARACTER INTO C—REGISTER,
            THEN CALL
        ;   PRTDVR
        ;
PRTDVR  LD A, (37E8H)       ; CHK STATUS
        BIT  7, A           ; BUSY BIT SET?
        JP NZ,  PRTDVR      ; LOOP WHILE BUSY
        LD A, C             ; GET CHARACTER
        LD (37E8H), A       ; SEND IT TO LP
        RET                 ; RETURN
```

小型磁盘

连接一台 26-1160 小型磁盘驱动器和最多三台另加的 26-1161 小磁盘驱动器，见小磁盘操作手册。连接电缆是和 26-1160 一起提供的。

把一个扁的接头罩装在小型磁盘接头上，然后把它装到扩展接口的插头上。带状电缆在罩的下面引出。

总线相容的设备

这个板接头提供与 TRS-80 的扩展板接头上相同的信息编排。因此任何可以直接接到 TRS-80 上的设备都可以接到这里，利用这个接头，就没有必要只让一个设备占用了全部 TRS-80 的接口。例如，你可以把 Radio Shack 的语音合成器（目录号26—1180）通过此接头接到扩展接口。

扩展板

扩展接口包含一个专门的扩展板空间，它可以安装另一块任选的印刷电路板以扩充你的系统的功能。

例如，你可以在此空间安装一块 Radio Shack RS—232C 串行接口（目录号 26—1145）。这可使你的 TRS-80 和别的 RS—232C 设备（如电话接口，串行行式打印机等等）相联。

如果你有足够的数字电路方面的经验，可以自己设计线路并安装于此。

当你安装上这样的扩展板后你可以通过扩展板接头来与它相联。

扩展板的安装

为露出此扩展空间，把四个十字头螺丝从盖板上拧下并取走盖板（图 9）。注意插座在里边，它用于连接附加的板。

图10展示了安装在扩展空间并插在扩展插座上的自制板的尺寸。

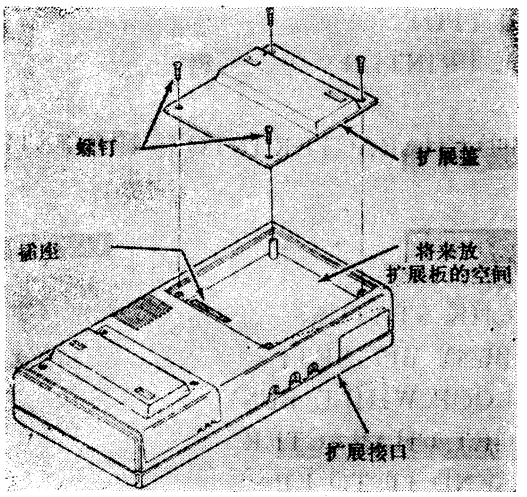


图 9　取走盖板以露出扩展板空间

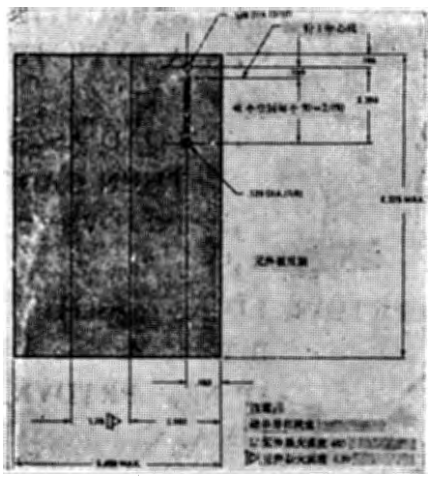


图10　扩展板所需尺寸

三、技术资料

25毫秒的节拍中断

扩展接口中有一个时钟线路，它给 Z-80 提供每秒钟 40 次的中断信号。当 Z-80 收到此中断信号时，它就从 ROM 中得到一个指令，告诉它到 RAM 的某一特定地址取下一个指令。一般情况下，这下一个指令只是叫 Z-80 退出中断顺序而回到它的被中断点继续执行。

如果你对Z80的指令系统有足够的经验（同时对一般的编程技术也有经验），那末，你可以在它回到被中断的程序控制之前，让它执行一个中断服务子程序以便完成某些简单的前台工作。

例如，你可以安装一个能守时的实时钟而不管当前执行的是什么程序——BASIC 命令方式也好，BASIC 执行方式也好，或者是机器语言程序也好，如此等等。具体做法如下。

1） 写一个中断服务子程序以便：

（1）使中断失效，

（2）读 FDC 状态寄存器 37ECH 以清除它，

（3）读 37E0H 中的中断状态，

（4）软件时钟计数器增 1，

（5）再一次读 37E0H，以便使中断锁存器复位，

（6）起动中断，

（7）回到被中断的程序，

2） 使中断失效，

3） 用一个转服务子程序的指令来代替原来的自动中断转移指令（在 4012H—4014H 单元），

4） 起动中断使时钟工作。

下面是一个 BASIC 程序，它能把所需的代码放到地址 32680—32767 中，注意，在使用此程序之前，把 TRS-80 复位并回答 MEMORY SIZE（内存容量）为 32679。

注意：此程序仅用于Ⅱ级BASIC，不要把它用于 DISK BASIC（此时要用 TRS DOS 时钟！）。

程序清单：

```
 50  REM……PROGRAM TO LOAD CLOCK CODE INTO RAM
100  FOR I%=1 TO 88
110  READ D%: POKE 32679+I%, D%
120  NEXT
130  POKE  16526, 168 'LSB OF ISR START ADDRESS
140  POKE  16527, 127 'MSB OF ISR START ADDRESS
150  DATA 243, 205, 127, 10, 175, 181, 40, 14
160  DATA 237, 86, 62, 195, 50, 18, 64, 33
170  DATA 193, 127, 34, 19, 64, 251, 195, 154
180  DATA 10, 229, 245, 58, 224, 55, 203, 119
```

```
190 DATA 32, 49, 203, 127, 40, 38, 33, 94
200 DATA 64, 52, 126, 254, 40, 56, 29, 175
210 DATA 119, 35, 52, 126, 254, 60, 56, 20
220 DATA 175, 119, 35, 52, 126, 254, 60, 56
230 DATA 11, 175, 119, 35, 52, 126, 254, 24
240 DATA 56, 2, 175, 119, 58, 224, 55, 241
250 DATA 225, 251, 201, 58, 236, 55, 24, 244
```

在你执行上述程序后，运行下面的程序以执行上述代码，接通并使用实时钟。

```
10 'INITIALIZE CLOCK
20 CX=USR(0) 'MAKE SURE CLOCK IS OFF
30 POKE 16478, 0 'SET TICKER TO ZERO
40 INPUT "ENTER THE TIME (HR, MIN, SEC)"; CH%, CM%, CS%
50 POKE 16481, CH%: POKE 16480, CM%: POKE 16479, CS%
60 CX=USR(1) 'TURN ON CLOCK
100 'YOUR PROGRAM GOES HERE
110 'WHEN YOU WANT THE TIME, INSERT "GOSUB 10000"
120 'TIME WILL BE RETURNED IN CT$, FOR EXAMPLE……
130 GOSUB 10000
140 PRINT@ 56, CT$
150 GOTO 130
10000 'READ THE CLOCK AND FORM CT$
10010 CH%=PEEK(16481): CM%=PEEK(16480): CS%=PEEK(16479)
10020 CC$=STR$(CH%): GOSUB 10100: CH$=CC$ '2—DIGIT HOUR
10030 CC$=STR$(CM%): GOSUBT10100: CM$=CC$ '2—DIGIT MINUTES
10040 CC$=STR$(CS%): GOSUB 10100: CS$=CC$ '2—DIGIT SECONDS
10050 CT$=CH$+":"+CM$+":"+CS$
10060 RETURN 'TO MAIN PROGRAM WITH CT$
10100 'MAKE INTO 2—DIGIT STRING
10110 IF LEN(CC$)=3 THEN CC$=RIGHT$(CC$, 2): RETURN 'W/STRING
10120 CC$="0"+RIGHT$(CC$, 1): RETURN 'WITH 2—DIGIT STRING
```

注意：要关掉时钟，可执行语句，CX=USR(○)。在对盒式带作输入/输出操作时，一定要关掉时钟！

汇编语言的程序员可用下列清单来重新确定实时钟代码。

```
        ORG 7FA8H
ISR     EQU 7FC1H                ; SET TO ISR START ADDR
        ; ENTRY VIA BASIC USR(N)
USR     DI                       ; DISABLE INTERRUPT
        CALL 0A7FH               ; GET (HL)
        XOR  A
        OR      L                ; IS USR ARG=0?
        JR      Z, EXI           ; IF YES THEN EXIT
        IM      1                ; ELSE LINK TO ISR
```

```
        LD      A, 0C3H         ; PUT IN JUMP TO ISR
        LD      (4012H), A
        LD      HL, ISR         ; START ADDR OF ISR
        LD      (4013H), HL
        EI                      ; ENABLE INTERRUPT
EXI     JP      0A9AH
        ; REAL TIME CLOCK CODE
CLK     EQU     405EH           ; TICKS STORED HERE
SEC     EQU     405FH           ; SECONDS HERE
MIN     EQU     4060H           ; MINUTES HERE
HOURS   EQU     4061H
        PUSH    HL              ; SAVE REGISTERS
        PUSH    AF
        LD      A, (37E0H)      ; GET INTERRUPT STATUS
        BIT     6, A
        JR      NZ, FDC         ; IF FDC MAKING RQST
        BIT     7, A
        JR      Z, XIT          ; EXIT IF INVALLID INTRPT
        LD      HL, CLK         ; HL=>TICKS COUNTER
        INC     (HL)            ; UPDATE"TICK"
        LD      A, (HL)
        CP      40              ; 40 TICKS PER SECOND
        JR      C, XIT          ; IF NO CARRY INTO SECS
        XOR     A
        LD      (HL), A         ; RESET TIKCNT
        INC     HL              ; POINT TO SECONDS—COUNT
        INC     (HL)            ;       AND UPDATE
        LD      A, (HL)         ; GET SECONDS COUNT
        CP      60
        JR      C, XIT          ; DONE IF NO CARRY TO HRS
        XOR     A               ;       ELSE RESET SECONDS
        LD      (HL), A

        INC     HL              ; POINT TO MINUTES AND
        INC     (HL)            ;       INCREMENT
        LD      A, (HL)         ; GET MINUTES COUNT
        CP      60
        JR      C, XIT          ; DONE IF NO CARRY
        XOR     A               ;       ELSE RESET MINUTES
        LD      (HL), A
        INC     HL              ; POINT TO HOURS AND
        INC     (HL)            ;       UPDATE
        LD      A, (HL)         ; GET HOURS
```

```
        CP      24              ; 24—HOUR CLOCK
        JR      C, XIT          ; DONE IF NO CARRY
        XOR     A               ;     ELSE RESET HOURS
        LD      (HL), A
XIT     LD      A, (37E0H)      ; TO RESET LATCH
        POP     AF              ; RESTORE REGS AND EXIT
        POP     HL
        EI
        RET                     ; ENABLE INTERRUPTS
FDC     LD      A, (37ECH)      ; RESET FDC IRQ RQST
        JR      XIT
        END     USR
```

软磁盘控制器集成电路

在扩展接口中的软磁盘控制器（FDC）直接控制驱动马达、磁道选择和定向、写数据和选通等。它还输入磁盘索引道上的信息、找零道、写入保护和数据/计时钟等——所有这些都在扩展接口的小磁盘板接头上。FDC寄存器(CMD/STATUS, TSACK, SECTOR, DATA即CMD/状态，道，扇区，数据）相应地是TRS-80内存的37ECH、37EDH、37EEH和37EFH各单元。

驱动选择通过Z36进行。每次只选中一个驱动器，每当一个驱动器被选中/再选中时，一台"时间到"计时回路（Z29）被启动/再启动，这样在程序有错时可以保护磁盘驱动器，因为驱动器的设计没有按马达长时间工作进行设计。经过2或3秒后MOTOR ON(马达工作)线（J5，腿16）就会停止工作（变高），除非Z29被驱动器选中/再选中信号所触发。当给出磁头加载命令时（FDC状态=READY），门Z35的6腿给FDC提供一个信号（腿23及32）。

TRSDOS(磁盘操作系统）考虑到磁盘驱动马达要经过1秒钟才能达到规定速度，磁头负载要80毫秒才能稳定。在FDC操作的结尾，产生一个中断(FDC的39腿变高)，而它又通过Z35（腿12和13）使Z28的9腿置位。中断请求后，先读FDC状态寄存器（地址37ECH)——这使FDC的腿39变低——然后再读37E0H——这使Z28的9腿复位——然后中断结束。

由于软盘操作的复杂性，Radio—Shack并不鼓励用户们不通过TRS-80磁盘操作系统而直接进行磁盘输入/输出，我们也不能回答用户们关于这方面的问题，用户们如果要作这方面的使用，请先阅读下列文件：

- Shugart SA400 OEM和维修手册
- Western Digital FD 1771B—01数据资料。

地址译码方案

地址译码逻辑包括Z42，Z43，Z32和一个反相器Z23。

Z43是一个二至四线的双多路器，这个组件的一半选择RAM的某一个16K，这一部分的输入信号是RAS*，A14及A15。RAS*用作正在工作的内存地址信号，逻辑1表示地址已

稳定，表 1 表示这些 I/O 组合。

表 1

输入			输出				
RAS*	A15	A14	Z 43 4 腿	Z 43 5 腿	Z 43 6 腿	Z 43 7 腿	选择的地址范围
1	×	×	1	1	1	1	无
0	0	0	0	1	1	1	0000—3FFF
0	0	1	1	0	1	1	4000—7FFF
0	1	0	1	1	0	1	8000—BFFF
0	1	1	1	1	1	0	C000—FFFF

注.×＝无所谓

腿 6 和腿 7 分别选择 32K 和 48K RAM 的行。4 腿的信号返回 Z43 的第二个一半，在那里它和与非门 Z42 的输出结合，当 A11，A14，A15 和 RAS* 有逻辑 0 及 A5，A6，A7，A8，A9，A10，A12，A13 为逻辑 1 时，它在 12 腿上给出逻辑 0，在其它情况下，12 腿都给出逻辑 1，腿 5 没用，在表 1 上也列出它，只是为了I/O组合的连续性。

从 Z43 腿 12 来的信号与 A2，A3，WR* 和反相的RD*相结合以产生表 2 所列的信号。

表 2

输入					输出									
Z 43 腿12	RD*	WR*	A 3	A 2	Z 32 腿 7	Z 32 腿 6	Z 32 腿 5	Z 32 腿 4	Z 32 腿 9	Z 32 腿10	Z 32 腿11	Z 32 腿12	产生的信号	信 号 到
1	×	×	×	×	1	1	1	1	1	1	1	1	无	—
0	0	1	0	0	0	1	1	1	1	1	1	1	37E0 READ	中断逻辑
0	0	1	0	1	1	0	1	1	1	1	1	1	37E4 READ	—
0	0	1	1	0	1	1	0	1	1	1	1	1	37E8 READ	打印机逻辑
0	0	1	1	1	1	1	1	0	1	1	1	1	37EC READ	磁盘控制器
0	1	0	0	0	1	1	1	1	0	1	1	1	37E0 WRITE	驱动器选择
0	1	0	0	1	1	1	1	1	1	0	1	1	CSW	盒带机继电器
0	1	0	1	0	1	1	1	1	1	1	0	1	37E8 WRITE	打印机逻辑
0	1	0	1	1	1	1	1	1	1	1	1	0	37ECWRITE	磁盘控制器

注意：×＝无所谓

盒式机插座信号

这些插座可以通过扩展接口连接或控制一对盒式录音机，三个插座的引出腿都是相同的。

表 3　盒式机插座信号

腿	信 号 名 称	说　　明
1	马达通/断	允许 TRS-80 控制磁带走动
2	地	信号地
3	马达通/断	允许 TRS-80 控制磁带走动
4	CASSIN（入）	允许磁带上的程序进入TRS-80
5	CASSOUT（出）	允许 TRS-80 的程序存在磁带上

Z-80的端口 255 用于盒式机的控制和I/O，如下：

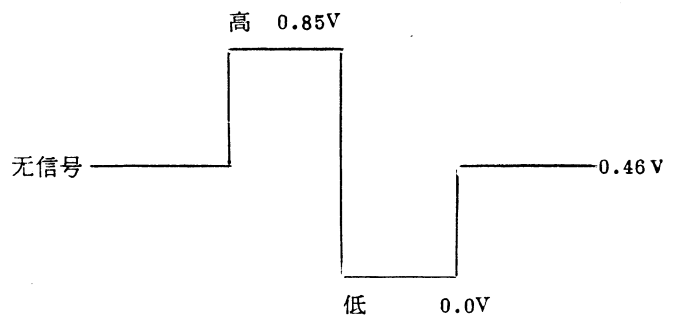
表 4　Z-80 I/O端口 255 的信号

位	功　　能	
	输　　入	输　　出
6	不用	CASSIN 锁存
7	不用	显示方式状态
5	不用	不用
4	不用	不用
3	显示方式选择　0＝32　1＝64　字符/行	不用
2	盒式机马达继电器 0 ＝断　1 ＝通	不用
1	CASSOUTB	不用
0	CASSOUTA	不用

注意：CASSOUT A 和 B 两者都用于产生盒式机的音频信号，如表 5 。

表 5　CASSOUT A，B如何产生三态盒带音频（高、低、无）

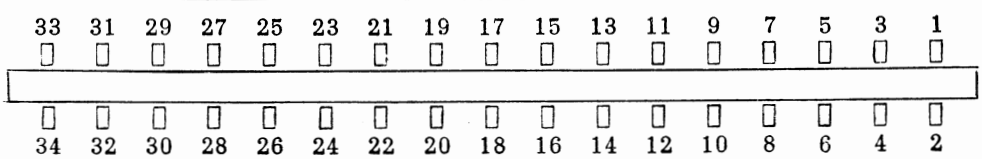
CASSOUT		音频脉冲
A	B	
0	0	无信号
1	1	脉冲低
0	1	脉冲低
1	0	脉冲高



打印机板接头信号表

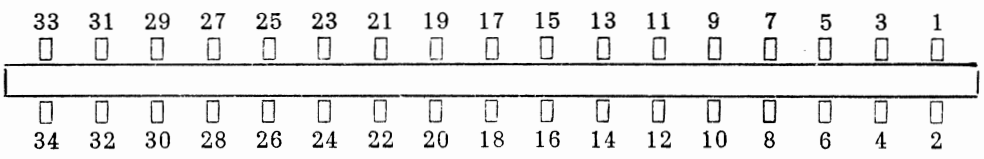
腿	信 号 名	说 明
1	DATA STROBE*	一个 1.0 微秒的脉冲用作把数据从处理器送到打印机逻辑的时钟信号
2	GND	信号地
3	D1	输入数据电平，高表示二进制的 1，低表示 0。所有可打印的字符（即DATA6和DATA7为1）存于打印机缓冲中。控制字符（即 DATA6 和 DATA7 都为 0 的代码）用于规定各种控制功能。这些代码不存入打印机缓冲，除非它们规定了打印命令并且该行中至少有一个可打印的字符在它们前面。
4	GND	
5	D2	
6	GND	
7	D3	
8	GND	
9	D4	
10	GND	
11	D5	
12	GND	
13	D6	
14	GND	
15	D7	
16	GND	
17	D8	
18	GND	
19	NC	不接
20	GND	信号地
21	BUSY	表示打印机不能接收数据的电平
22	GND	信号地
23	OUT OF PAPER PE	表示打印机纸尽的电平。
24	GND	信号地
25	UNIT SELECT SLCT	表示打印机被选中的电平
26	PRIME*	清除打印机缓冲并使逻辑置初态的电平
27	GND	信号地
28	FAULT*	表示打印机有故障，如纸尽、未选中等情况。
29	NC	不接
30	NC	不接
31	GND	信号地
32	NC	不接
33	GND	信号地
34	GND	信号地

注意：所有 GND 信号都是共同的



小磁盘板接头信号表

腿	信　号　名	说　　明
1	GND	信号地
2	NC	不接
3	GND	信号地
4	NC	不接
5	GND	信号地
6	NC	不接
7	GND	信号地
8	INDEX PULSE*	道的物理开头
9	GND	信号地
10	DS1*	有效时，在小磁盘上锁住 R/W磁头（1号驱动器）
11	GND	信号地
12	DS2*	有效时，在小磁盘上锁住R/W磁头（2号驱动器）
13	GND	信号地
14	DS3*	有效时，在小磁盘上锁住R/W磁头（3号驱动器）
15	GND	信号地
16	MOTOR ON	接通所有驱动马达
17	GND	信号地
18	DIRECTION SEL*	当STEP线有脉冲时确定R/W磁头运动方向
19	GND	信号地
20	STEP*	根据 DIRECTION SEL 规定的方向使 R/W 磁头运动
21	GND	信号地
22	WRITE DATA*	提供写盘的数据
23	GND	信号地
24	WRITE GATE*	使数据写在盘上
25	GND	信号地
26	TRACK ZERO*	0状态表示R/W磁头停在0道上
27	GND	信号地
28	WRITE PROTECT*	向用戶提供信号表示磁盘是禁止写入的
29	GND	信号地
30	READ DATA*	提供原始数据（时钟和数据）由驱动器电路检测
31	GND	信号地
32	DS4*	有效时，在小磁盘上锁住R/W磁头（4号驱动器）
33	GND	信号地
34	NC	不接



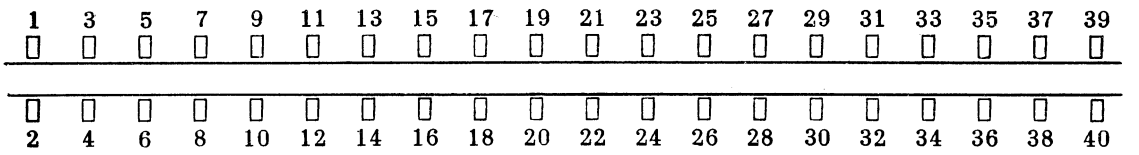
总线板插头信号表

这个插头复现了 TRS-80 板插头上的信号。

腿	信号名	说明
1	RAS*	16腿动态RAM的行地址选通输出
2	SYSRES*	系统复位输出，电源打开或按下复位键时为低
3	CAS*	16腿动态RAM的列地址选通输出
4	A10	地址输出
5	A12	地址输出
6	A13	地址输出
7	A15	地址输出
8	GND	信号地
9	A11	地址输出
10	A14	地址输出
11	A 8	地址输出
12	OUT*	外设写选通输出
13	WR*	内存写选通输出
14	INTAK*	中断认可输出
15	RD*	内存读选通输出
16	MUX	16腿动态RAM的多路器控制输出
17	A 9	地址输出
18	D 4	双向数据总线
19	IN*	外设读选通输出
20	D 7	双向数据总线
21	INT*	中断输入（可屏蔽）
22	D1	双向数据总线
23	TEST*	TEST*输入为"0"时，使A0—A15；D0—D7；WR*；RD*；IN*；OUT*；RAS*；CAS*；MUX*成为三态
24	D6	双向数据总线
25	A0	地址输出
26	D3	双向数据总线
27	A1	地址输出
28	D5	双向数据总线
29	GND	信号地
30	D0	双向数据总线
31	A4	地址总线
32	D2	双向数据总线
33	WAIT*	处理器WAIT输入，使内存变慢
34	A3	地址输出
35	A5	地址输出
36	A7	地址输出
37	GND	信号地
38	A6	地址输出

39	GND	信号地
40	A2	地址输出

注意：* 表示负（逻辑“0”）逻辑输入或输出。



扩展板接头信号表

如果你在扩展空间安装一块扩展板，这个接头可以提供方便的信号接线，例如，安装了 Radio Shack RS—232C 串行接口，你就可以把外加的 RS-232C 设备接到此接头上。

此接头连接到扩展空间内部的接头上，也接到某些 TRS-80 的信号上：

TRS—80 数据线（D0—D7）；TRS-80 的某些地址线（A0—A2）；I/O 选通（IN* 和 OUT*）；复位线 SYSRES*；+5V；地线；中断线 INT* 以及一个已经译码的信号 E8*。E8* 在 A3，A5，A6 及 A7 为逻辑“1”和 A4 为逻辑“0”时，为逻辑“0”。

板接头本身并不接到任何 TRS-80 的信号上。应该由你的扩展板向你的外部设备提供所需的信号，详细情况见线路图。

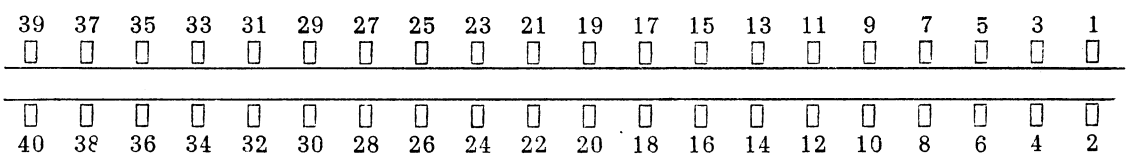
下表表示在安装了选购件 RS—232C 接口板后，板插头的输出腿。

扩展板插头信号表（RS—232C 已装上）

腿	信号名*	说明
1	GND	信号地
2	NC	
3	GND	信号地
4	NC	
5	GND	信号地
6	NC	
7	GND	信号地
8	NC	
9	GND	信号地
10	—	内部扩展接头 16腿（不用）
11	GND	信号地
12	—	内部扩展接头 15腿（不用）
13	GND	信号地
14	—	内部扩展接头 14腿（不用）
15	GND	信号地
16	PGND	保护地
17	GND	信号地

18	TD	传送数据—此线路的信号送到远距设备
19	GND	信号地
20	SGND	数据通信设备来的信号地
21	GND	信号地
22	RD	此线路的信号是从数据通信（远距）设备接收来的
23	GND	信号地
24		内部扩展接头 9 腿（不用）
25	GND	信号地
26	—	内部扩展接头 8 腿（不用）
27	SIG GND	
28	—	内部扩展接头 7 腿（不用）
29	GND	信号地
30	CD	载波检测（接收线路信号检测器）表示数据设备通过通信线收到从远距数据设备来的一个字符
31	GND	信号地
32	CTS	数据通信设备产生的允许发送信号，这表示数据设备（调制解调器）是否准备好发送数据
33	GND	信号地
34	DTR	这个数据终端就绪信号是送到数据通信设备去的，用于控制将设备接到通信通道上去
35	GND	信号地
36	RTS	这个信号用于控制数据传输方向
37	GND	信号地
38	RI	此信号（从数据通信设备来的振铃信号）表示数据设备已被询问，表示对方需要通信
39	GND	信号地
40	DSR	数据设备准备好，表示本地的数据设备的状态

* 此表中所用信号都是和Radio Shack的接口有关的。



四、故障检查

如果你的系统有什么问题，请先再三检查所有的连接。（连接不合适会使系统看来有故障。如果你不能解决问题，请把设备送回当地的 Radio shack 商店或修理站。我们将尽快送回！）

注意：下列资料是为在数字电子学方面相当有经验的人员提供的。请记住，打开扩展接口的外壳（不是电源部分）就会使本手册后附的保证书失效！

警告！ 拆开Radio Shack计算机设备就使保证失效！

在本设备中所包含的固态组件对你所感觉不到的静电是极其敏感的，技术人员是在带有接地线的特殊的防静电的条件和设备下对它们进行工作的，这些条件对一般用户是不具备的。除非你是一个受过训练的计算机技术人员并且愿意失去你的保证，否则还是把所有修理工作交 Radio Shack 商店或修理站。

参考资料

在开始对扩展接口作故障检查之前，一个很好的参考材料是Radio Shack的《TRS-80微计算机技术参考手册。》或《硬件手册》此手册包括了 TRS-80 微计算机的详细操作和故障检查说明。由于扩展接口是 TRS-80 的一个扩充，你所遇到的某些问题可能直接与 TRS-80有关，

扩展接口的某些诊断软件可向 Radio Shack 的国内零件中心索取，这些软件要另加费用。

拆卸

要拆卸扩展接口时可参考图11。把该设备颠倒过来，卸掉外壳的上部和底部相联接的六个螺丝，当你把底半部取走时，你就可以看到印刷电路板的有元件的那一面。

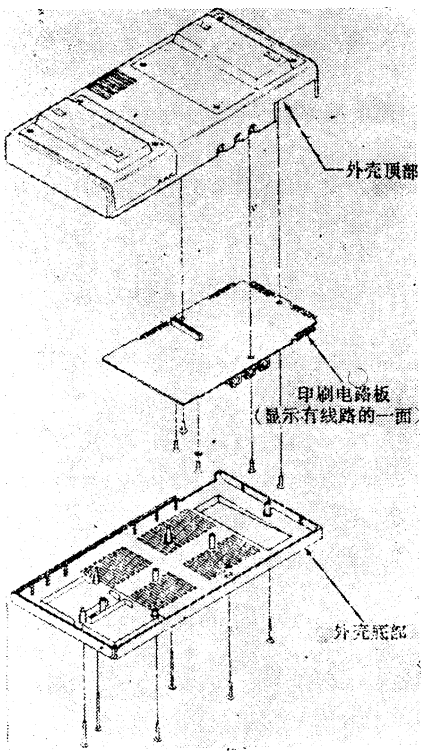


图11 拆卸扩展接口

电源检查和调整

把设备的塑料外壳打开并把板子放到试验台上后，安上DIN电源插头。

注意：此时扩展接口的板子相对于在外壳中的正常情况来说，是反过来放着的。请注意DIN电源插头一定要接在电源插座 J9 中而不是在盒带机插座(J6，J7，J8)中，电源插座最靠近大的散热片。

接通扩展接口线路板的电源并测量电源电压（见图12）。

1） 用一个数字电压表或类似设备，共同端（－）接到电容 C5 的顶部，这是板上最大的一个电容。

2） 12V电源：电压表量程放在＋20V 直流上，把红(＋)端接到大电阻 R8 的底端（底端是指最靠近 C5 的一端）。电压指示应为 12.0V ±5%(12.6 到 11.4V)。如果电压不在此范围内，调节电阻 R27 以得到正确电压（R27 在板上的左上边）。

3） ＋5V电源：电压表量程放在＋10V直流上，把红(＋)端接到 C51 的左侧（C51 位于大散热板的右边）。电压读数应为 5.0V ±5%(5.25 到 4.75 V)，如果电压不在此范围内，调节电阻 R26 以得到正确电压（R26 就在 C5 的上边）。

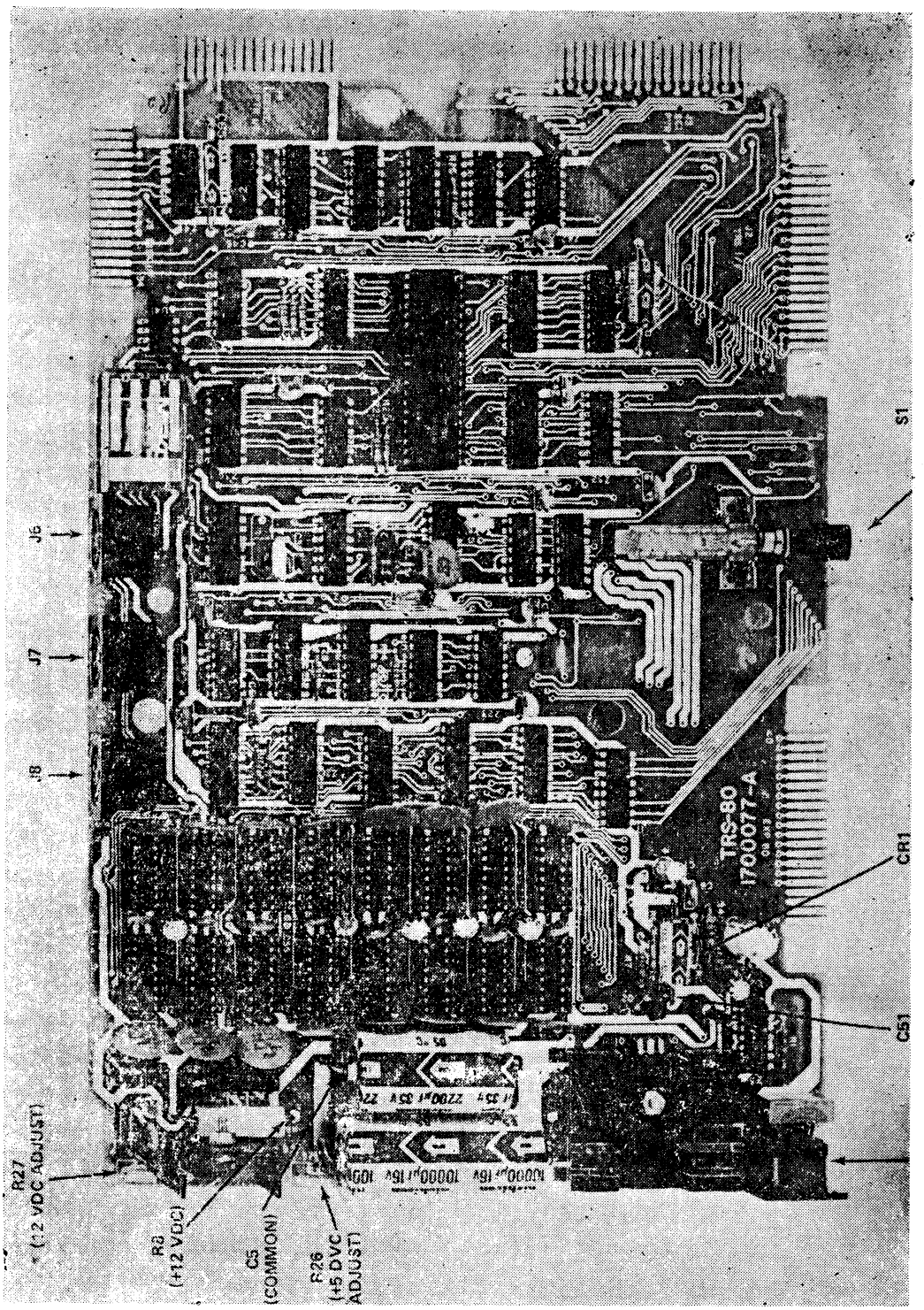


图12　电路检查和调整

注意：在12V电源未检查或未调整到允许范围之前，不要调整＋5V电源。

4）　－5V电源：电压表改为－10V直流量程，把红（＋）端接到CR1的阳极(CR1就在C51的下面），电压读数应为－5V（±5%），－5V电源没有可调整之处。如果此电源不在

规定的范围内，则一定是有损坏的器件。

原理图（简图）

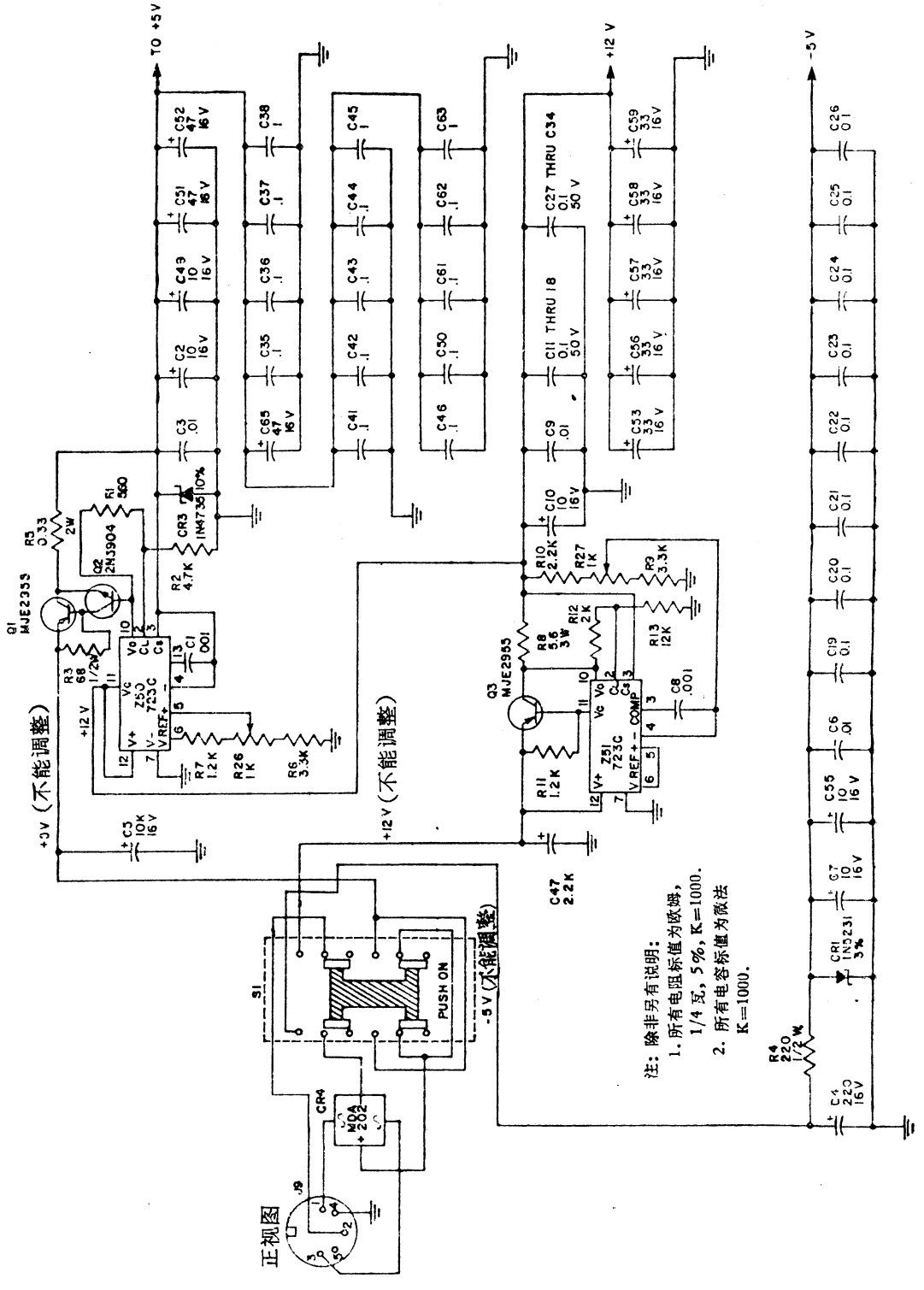


图13　扩展接口线路图(1)

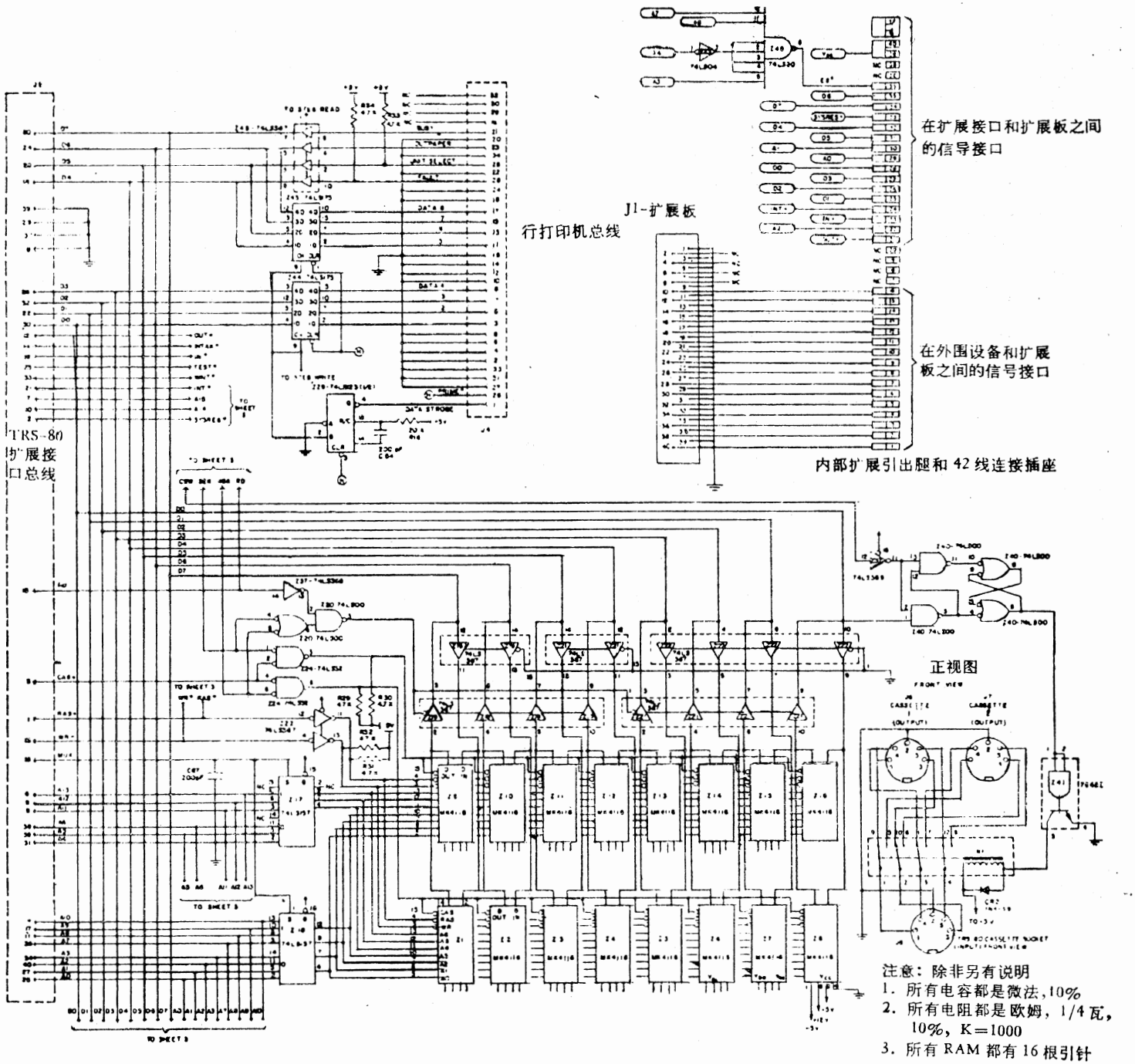


图13　扩展接口原理图（2）

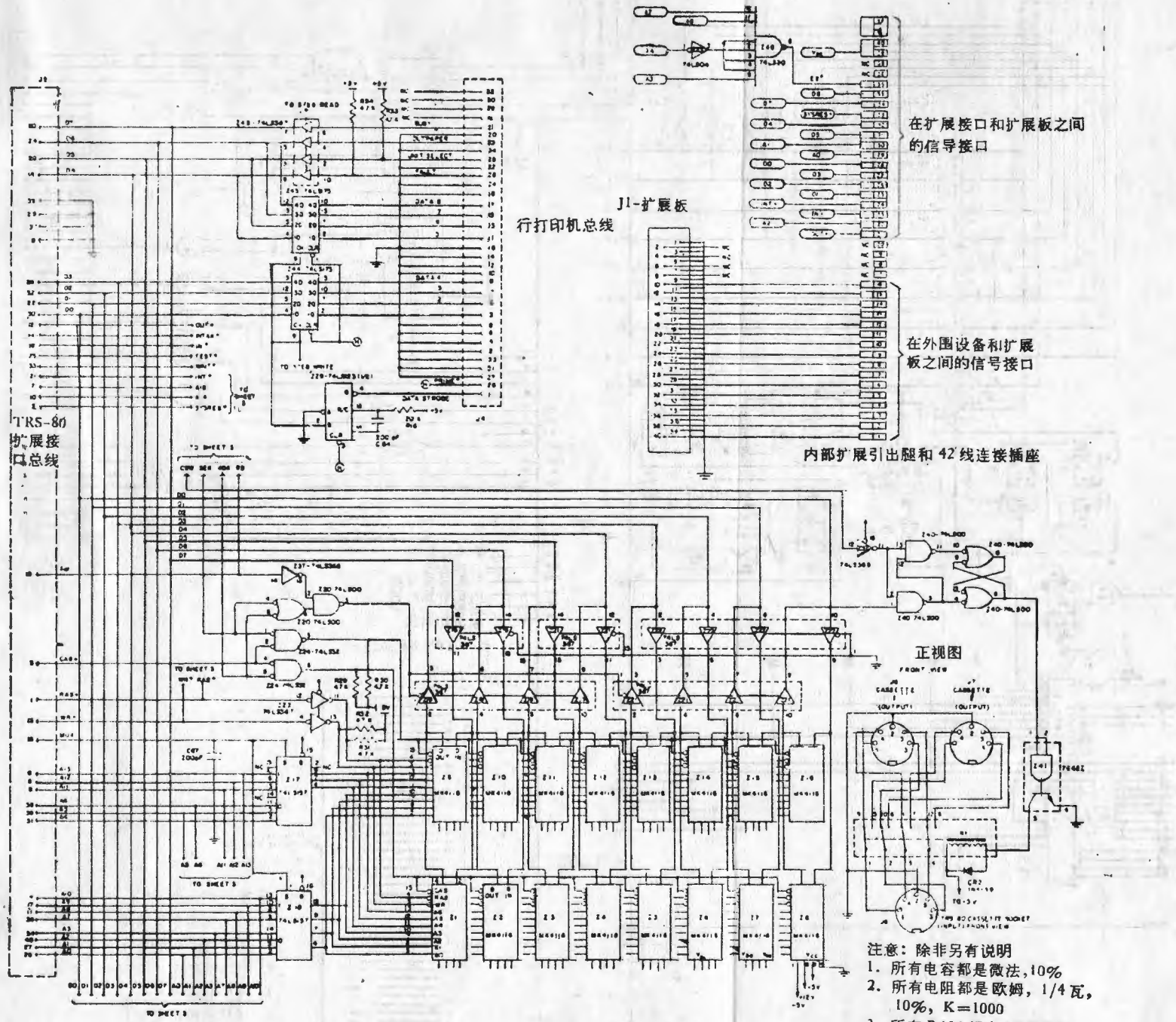


图13 扩展接口原理图（2）

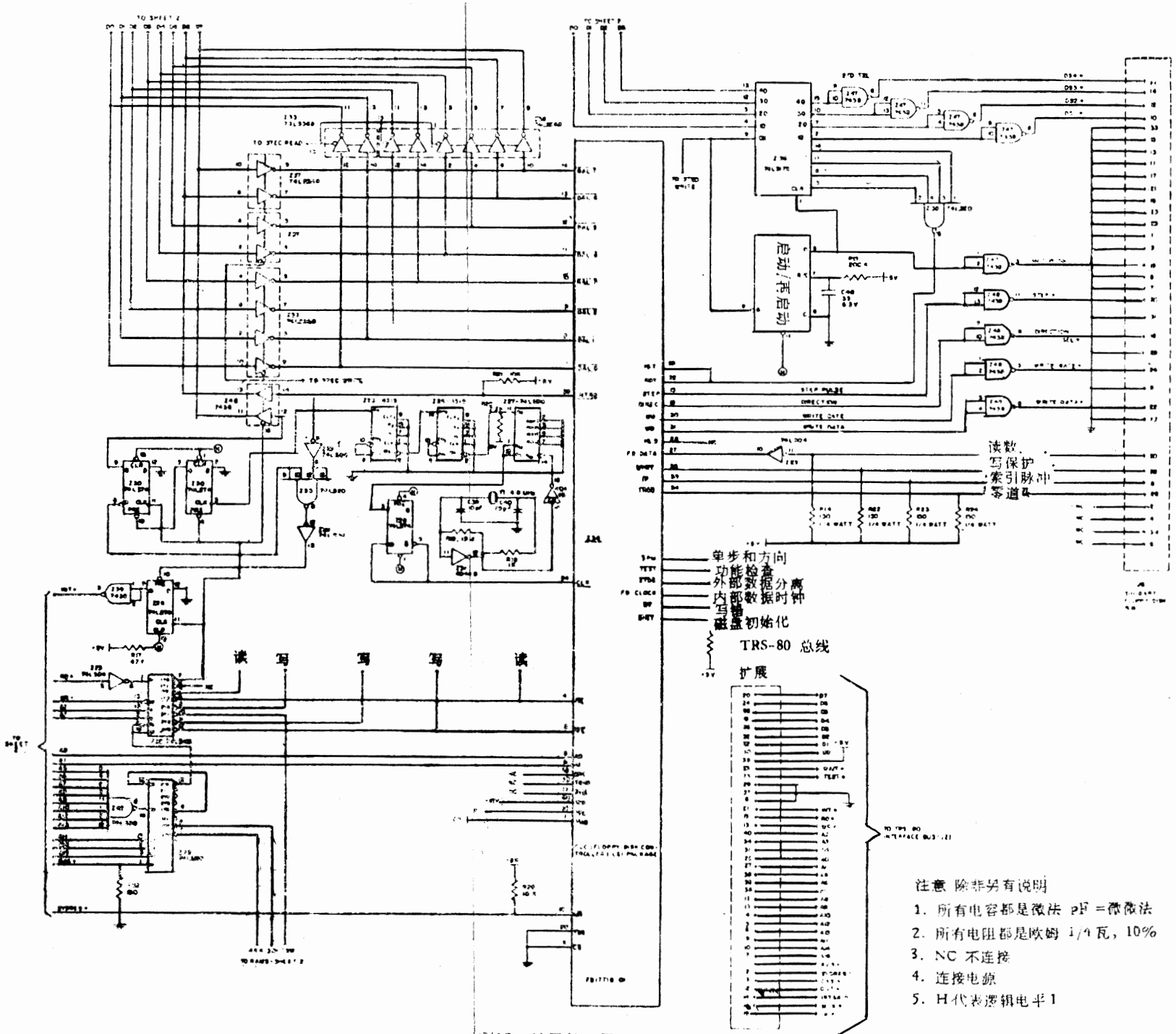


图13 扩展接口原理图(3)

(沈国锋译，周宝兴校)

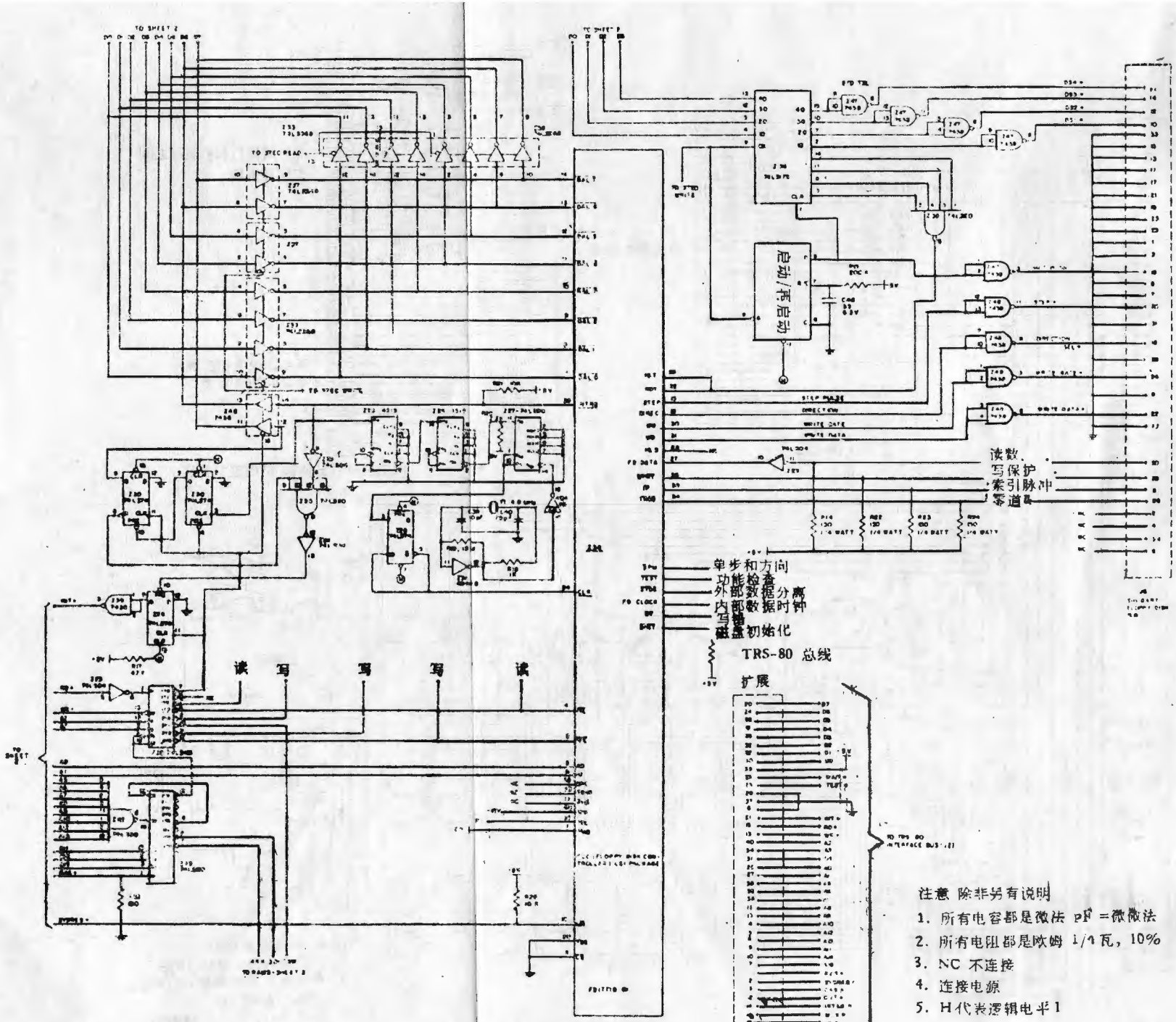


图13 扩展接口原理图(3)

(沈国铎译、周宝兴校)

第四篇 TRS-80 RS-232-C 接 口

“接口”是什么？

这是在各位拥有的TRS-80和某些外部设备间进行通讯时最广泛使用的设备。该接口在数据识别、数据传送速度、发送/接收顺序、故障检查等方面符合有关的规程，但并不包含欲利用某个特定的TRS-80外部设备系统所必需的程序设计。

例如，即使配置了这种接口，也不能做到从某个TRS-80向其他TRS-80自动地传送BASIC程序，或者将BASIC程序输出到一台行式打印机。必须有“驱动程序”，才能这样使用；这种程序必须单独设计以适合于预定使用的设备。

本手册中，附有两种驱动程序的文本。其中之一给各位的TRS-80附加了作为终端的功能；另一种驱动程序给TRS-80提供了向普通的串行行式打印机（DECWRITER）输出的能力。

Radio Shack公司的RS—232—C接口的设计虽然符合EIA（电子工业协会）规格，但是，并不是任何所谓“和RS—232—C兼容”的设备都能完全地、很好地与其连接和通讯。在这种情况下，或者在关于各位用户自己的使用方法方面，这里难以提供硬件和软件的支持。

不过，本接口设备规定能和装用Tandy Radio Shack公司的RS—232—C任何机器相联接。

一、引 言

“RS—232—C”是表示EIA的特定规格的术语。在该规格中，定义了数据终端设备和数据传送设备连接时，一般的使用方法。本RS—232—C接口作为数据处理装置连接用的标准设备，已蒙各方面广泛使用。特别是在大多数图象终端设备，调制解调器，卡片输入机，行式打印机，小型机、微型机等方面，被用于设备间的数据交换。

当TRS-80扩展接口中安装上RS—232—C时，在兼容性方面就开辟了一个全新的领域。本设备是为安装在TRS-80扩展接口的内部而设计的。串行行式打印机，调制解调器，图象终端设备等能用作外部设备。总之，和RS—232—C有兼容性的设备，各位用户都能和其连接。

RS—232—C接口的一个最有用的应用程序，就是利用电话接口，能把各位的TRS-80和分时计算机系统连接起来。这个名为TERM的程序以磁带形式附于RS—232—C接口。

TRS-80的许多用户希望用LEVEL Ⅱ BASIC的LPRINT命令来操作自己的串行打印机，那么，只要把这种RS—232—C接口安装在扩展接口中，就能实现之。本手册中附有实现这种应用的程序示例和文本。

数字数据的传送

在比较远距离的传送数字数据时，一般采用串行式的数据发送技术。此时，可用一根双

绞线连接发送和接收设备。通常，在所谓“非同步式”和“同步式”两种传送技术中，选择使用一种。在Radio Shack的系统中，使用非同步的位串行式技术。关于同步式技术，请允许不在这里涉及。在非同步传送时，没有必要和数据一起发送同步时钟，字符也没有必要是连续的。因此，对各字符间的间隙长度没有限制。

如图1所示，数据字符（通常是5～8位长）和同步用的起始码以及停止码一起构成了一个字组。起始码用逻辑0表示，位于各字组的头部；停止码用逻辑1表示，位于各字组的尾部。停止码在后续字组的起始码被传送以前，一直保持着。停止码的长度没有上限，可是有下限，这取决于系统的特性。典型的数值是1.0，1.42，2.0数据位间隔（但是，在最新的系统中，采用1.0或2.0停止位）。起始码往负向变位，表示传送中的字组内的数据位的位置。接收设备的时钟源根据这种变位被复位。用这个时钟源来决定每个数据位的中心位置。

采用非同步式数据传送系统的理由如下：由于没有必要和数据一起发送时钟信号。所以设备比较简单。另外，由于字符也可在一段时间内不发送，所以能根据需要来传送。这种特性，在操作员通过手工操作从必要的输入设备（例如键盘等）传送数据的情况下，特别有效。非同步式传送的最大缺点在于起始码和停止码占有相当长的通信带区。使用这种起始/停止码的方式称为“起停式”。

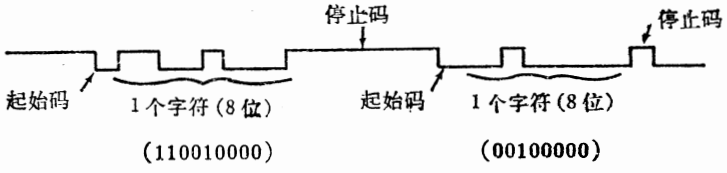


图1　非同步式数据

非同步式数据的传送速度被定义为“波德率”。所谓波德率，即传送最短信号码所需时间的倒数。一般情况下，这表示1个数据位间隔，在使用一个停止位时，波德率等于位速度，在使用一个以上的停止位时，波德率不等于位速度。

在非同步式传送时，只须用一根双绞线就可以达到相当高的波德率（10K波德或10K以上，因导线长度、驱动器种类而异）。在利用电话网传送时，通常被限制为约2K波德。另外，为了要使数据脉冲变换为能通过那种电话网的音调，就必须有调制解调器，而RADIO SHACK的电话接口即是用于RS—232—C的理想的调制解调器。

关于讯号的规定

在EIA的RS—232—C电气系统规格中，定义了与设备间传送的数据和控制信息有关的电平以及通信上的逻辑规定。在数据交换时，认为在接口部位测得的电平低于－3V（相对于信号的地电平）的信号为“标记状态”；高于＋3V的信号为“空状态”。“标记状态”用逻辑1表示；“空状态”用逻辑0表示。

另一方面，在定时和控制交换电路中，认为交换电路的电平高于＋3V（相对于信号的地电平）时，它的功能处于“接通”状态；而在低于－3V时，处于“断开”状态。“接通”状态用逻辑0表示，“断开”状态用逻辑1表示。概括上述内容，有下表：

项　　目	传　送　电　平	
	负	正
二进制表示	1	0
信号的状态	标　记	空
功　能	断　开	接　通

表 1　接通/断开状态

插针和信号的对应关系

在RS—232—C的机械系统规格中，必须要有一种 25 个插针的连接器（称为DB—25）。表 2 说明了 Tandy Radio Shack 的RS—232—C接口的插针和信号的对应关系。

插 针 号	缩 写 形 式	信　　号
1	PGND	保护用的接地
2	TD	发送数据
3	RD	接收数据
4	RTS	发送要求
5	CTS	允许发送
6	DSR	数据设备准备好
7	SGND	信号用的接地
8	CD	载波检测
20	DTR	终端设备就绪
22	RI	振铃指示器

表 2　插针和信号的对应

保护用的接地：它和机架或设备的框架相连，也可和“信号用的接地”连接。

发送数据：给数据通信设备的指示。本线路上的信号由把数据传送给远程设备的数据终端设备发送。在字符间的空白部分或不传送数据时，该信号必须保持为“标记状态”。

接收数据：来自数据通信设备的指示。本线路上的信号是由传送数据给终端设备的远程设备所发送。在字符间的空白部分或没有发送来数据时，该信号必须保持为“标记状态”。

发送要求：给数据通信设备的指示。该信号由终端设备要求，用于控制数据通信设备的数据传送方向。单向或双通道处于“接通状态”时，数据通信设备保持传送模式。在“断

开”状态时，数据通信设备则为非传送模式。

半双通道为“接通”状态时，数据通信设备保持传送模式，但接收模式被禁止。“断开”状态时，数据通信设备转为接收模式。

允许发送：来自数据通信设备的指示。该信号由数据通信设备发送，指出“数据设备”（调制解调器）是否处于传送的就绪状态。该线路的“接通”状态指示数据终端设备：“数据设备”的传送数据线路处于可接收数据的状态，而“断开”状态则表示数据终端设备不得向“数据设备”传送数据。

数据设备准备好：来自数据通信设备的指示。该信号给数据终端设备指示“本地数据设备”的状态。该线路的“接通”状态表示：数据通信设备的当前模式不是测试、会话、拨号模式，就是完全结束了由呼号设定的必要的定时功能(回答声等)。除此以外则都成为“断开”状态，它表示数据终端设备接收的只是振铃指示器信号，其他信号（交换线路上的所有其他信号）完全不予考虑。

终端设备就绪：给数据通信设备的指示。该信号用于控制数据通信设备和通信通道间的连接的切换。“接通”状态对应数据通信设备连向通信通道，处于“接通”状态期间，表示继续保持这种连接。一转为“断开”状态，在传送中的数据结束传送后，数据通信设备和通信通道切断。

振铃指示器：来自通信设备的指示。该线路的“接通”状态表示振铃讯号在通信通道上。这一般意味着某个设备向数据设备请求进行数据通信，在振铃周期的断开段期间以及不接收讯号时，通常保持“断开”状态。

载波检测（接收线路信号检测）：来自数据通信设备的指示。该信号为“接通”时，表示“数据设备”通过通信通道，从远程“数据设备”接收载波。另外，在不接收载波，或信号的特性不适于数据的解调时，变为“断开”状态。

二、 RS—232—C的安装方法

在实际安装以前，请注意下列事项：

TRS-80 扩展接口设备中，有时没有配置 42 线连接器（安装 RS—232—C 时所必需的）因此，万一您的扩展接口中没有安装这种连接器时，由于不能和 RS—232—C 恰当地连接，请退回本公司。此外，若自己随意安装的话，则敬请谅解今后不能兑现所作的保证。

扩展接口中配有必要的 42 线连接器时，请按下列步骤安装RS—232—C：

1） 如图 2 所示，放置好 TRS-80 扩展接口，卸下扩展用的盖板。

2） 从 42 线连接器上卸下两个螺钉和垫片。

3） 如图所示准备好 RS—232—C 接口使 42 线连接器上的螺孔和 RS—232—C 接口的印刷板上的螺孔对准，然后，用两个螺钉和垫片将其紧固。

4） 按图 3 所示，放置好RS—232—C，卸下扩展接口中央的盖板。然后，把RS—232—C所带的电缆的40线连接器插牢在露出的直插连接器上。扁形电缆往下方伸展，电缆的另一端接在要用的调制解调器、行式打印机等设备上。

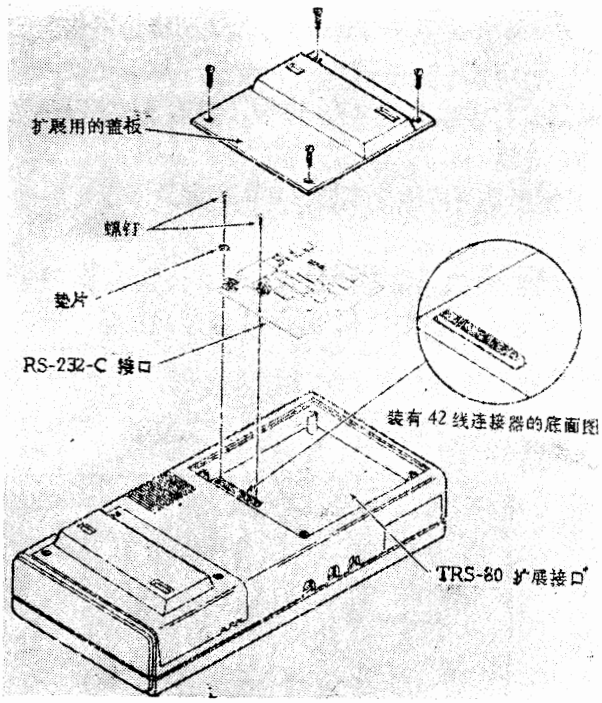


图2　RS-232-C安装法

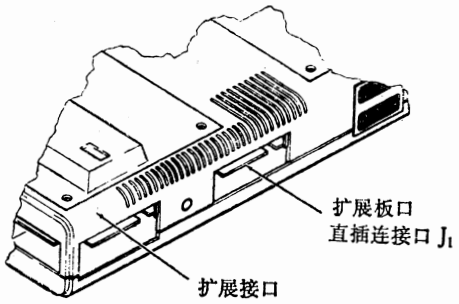


图3　扩展接口的正面图

检　　验

1）　如下例所示，为了设定要用的波德率，奇偶校验等状态，按下表设置 RS—232—C 的转换开关（变换操作列表于表 3）：

开　　关	S_8	S_7	S_6	S_5	S_4	S_3	S_2	S_1
波德率＝300	闭	闭	开					
1 个停止位，有奇偶校验				闭	闭			
7 位字长的数据和 1 个奇偶位						闭	开	
偶　校　验								开

然后，把 TEMR/COMM 开关置于 TERM（和调制解调器相连的情况）。

2） 使用 LEVEL Ⅱ BASIC “SYSTEM” 命令，从所带的磁带中取出“TERM”（TERM是文件名)。然后，打印出斜线（/）记号，按下“ENTER”键，启动程序〔规定存贮单元地址为 5000（16 进制)〕。

3） 根据视频显示器屏幕是否清除来判断 TERM 带是否正常启动。在视频显示器屏幕的左上方应该出现光标。

注：若将 RS—232—C 电缆的25 线连接器的插针 2 和 3 连接，则键盘的输出就返回呈现在视频显示器屏幕上，由此可判断接口和 TERM 程序的正常工作。

4） 安装好扩展用的盖板。

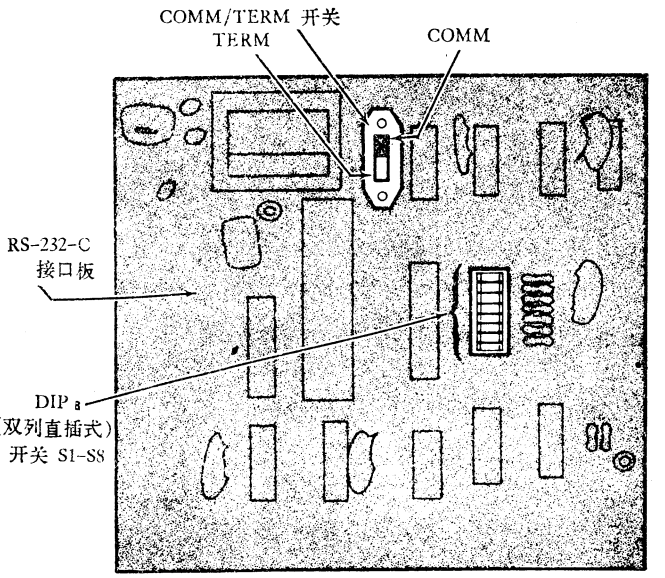


图4 RS—232—C的各种开关

三、工 作 原 理

由于 Radio Shack的RS—232—C 接口是一种灵活的、可编程序的接口，所以几乎所有和 RS—232—C兼容的设备都能和其连接。本来，在 RS—232—C 设备中，有所谓数据终端设备和数据通信设备两种不同类型，但是，Radio Shack的 RS—232—C 接口则是从数据终端的观点来构造的，用来和数据通信设备相连。

接口的两种结构的切换控制既可由转换开关指定，也可用软件方法规定。

通用异步接收 / 发送器

图 5 是 RS—232—C 接口的框图，接口的中心部分是 UART（通用异步接收/发送器）串行数据的接收/发送所必要的硬件几乎都组装在这种 MOS 大规模集成电路（TR1602A）中。UART是通用的，且符合工业规格。它是可编程序的设备，在把非同步式串行数据通道和具有并行数据结构的微处理器连接时使用这种 UART。UART 的发送部分把微处理器送

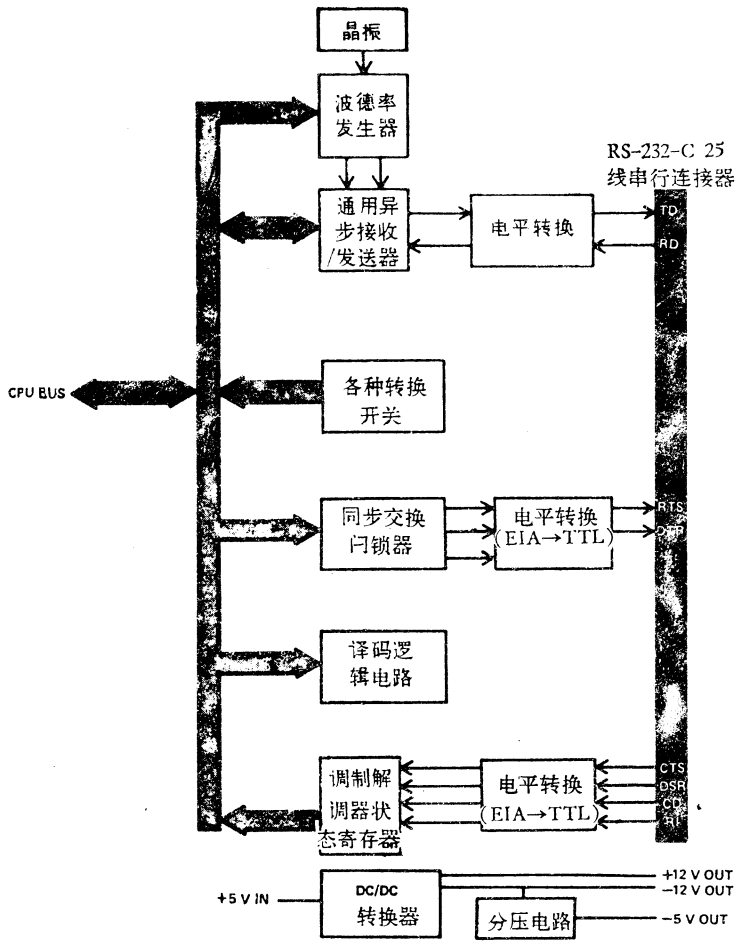


图5　RS-232-C　框　图

出的并行的数据变换成由起始、奇偶、停止和数据各位组成的串行字。而在接收部分中，与此相反，把串行字（由起始、数据、奇偶、停止各位组成）变换为并行数据，而且验证数据送传是否正确地进行（对接收到的奇偶和有效的停止位进行检验）。

UART 能根据用下述方法编制的程序来操 作：字长可在 5，6，7，8 位中选择使用；奇偶校验码的形成和校验能禁止；可以规定奇偶校验为奇校验还是偶校验；停止位的个数可以使用 1 个或 2 个，但是，在传送 5 位码时，停止位则为 1～1.5 位。

由于本UART 中备有若干个内部寄存器，所以可以进行串行数据通道 的构成、UART 的状态的监视、传送用的数据的取入、串行数据通道上的数据的读入等。关于这些寄存器的详情，将在后面叙述。

波德率发生器

图 5 中，UART 的正上方框内是 BRG（波德率发生器）。这部分输出两个时钟信号，这些信号是UART 正确地工作所不可缺少的条件，即由这两个信号 [TRANSMIT FREQ（发送频率）和 RECEIVE FREQ（接收频率）] 来决定串行通道的发送/接收波德率。作

为 BRG 的定时基准器，采用 5.0688MC 的晶体振荡器。在这个 BRG 中，用程序可以把定时基准器的频率分频成 UART 用的各种频率。UART 在发送/接收时，都要求时钟频率为所需要波德率的 16 倍。由于 BRG 的发送/接收用的频率输出可以用把适当的常数取入内部寄存器方法编程，所以也可能用不同的波德率进行数据的发送和接收。

转换开关

图 5 的 UART 的正下方框内是转换开关。这部分由 8 个单刀单掷开关（S1～S8）组成。这些开关用于波德率、位/字、奇偶校验形式（奇校验或偶校验）、停止位个数（1，1～1.5，2）等的选择以及奇偶校验产生机构的起动和停止。另外，如有必要，可不用这些开关，用软件控制也可直接构成非基准波德率或使用不同的发送/接收频率的接口。表 3 汇总了转换开关的作用。

波德率	S6	S7	S8	波德率	S6	S7	S8
110	闭	闭	闭	1200	闭	开	闭
150	闭	闭	开	2400	闭	开	开
300	开	闭	闭	4800	开	开	闭
600	开	闭	开	9600	开	开	开

奇偶校验功能	S4	停止位个数	S5	奇偶校验形式	S1
有	闭	1	闭	奇校验	闭
无	开	2	开	偶校验	开

字长（除奇偶位）	S2	S3	字长（除奇偶位）	S2	S3
5 位长	闭	闭	7 位长	开	闭
6 位长	闭	开	8 位长	开	开

表 3 转换开关的作用

同步交换闩锁器

Tandy Radio Shack 的 RS—232—C 接口能够（使用适当的软件）控制内部的两个控制信号（发送要求和终端设备就绪）的逻辑状态。为了实行这种功能，必须把适当的位模式取入同步交换闩锁器中，如图 5 所示，CPU 读出调制解调器状态寄存器的内容，最多能检测到四个接口信号（允许发送，数据设备准备好，载波检测，振铃指示器）。如使用适当的软件，同步交换闩锁器和调制解调器状态寄存器能在 CPU 和 RS—232—C 与相连的设备间进行同步交换式对话。

逻辑规定

RS—232—C 的内部逻辑根据 TTL 电平工作（3.5V 以上＝逻辑值 1，0.8V 以下＝逻辑值 0）。另一方面，在两个 RS—232—C 设备相互连接时的逻辑规定中，使用 EIA 电平（－3V 以下＝逻辑值 1，＋3V 以上＝逻辑值 0）。因此，为了连接不同的 RS—232—C 设备，逻辑电平必须相互转换。这种转换工作由图 5 中写有 EIA→TTL，TTL→EIA 的框来进行。

口地址

RS—232—C接口和 CPU 通信时，相当于 CPU 的I/O设备，它使用四个口地址，（E8H，E9H，EAH，EBH）。Z80系统的 I/O设备用低位地址位（A_0—A_7）和IN*，OUT* 信号一起来寻址既定的口。图 6 是和这种地址指定机构有关的时序图。

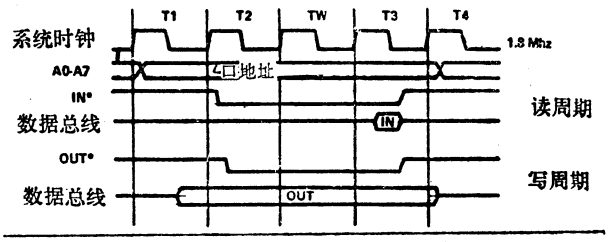


图6　I/O 时　序　图

译码逻辑电路

图 5 中写有“译码逻辑电路”的框产生 8 个负选通，供 RS—232—C 使用，根据这些选通，CPU 可选定 RS—232—C 接口的任意寄存器和闩锁器，进行 I/O 操作。表 4 汇总了使用RS—232—C 接口的 I/O 口地址分配和它的功能。

地　　址	输　　出（I/O口写）	输　入（I/O 口读）
E8H	主复位（全数据）	调制解调器状态寄存器
E9H	波德率选择	结构转换开关
EAH	UART控制寄存器和同步交换闩锁器	UART状态寄存器
EBH	发送数据寄存器	接收数据寄存器

表4　I/O 的存贮器分配

上表中所列的寄存器各自执行 RS—232—C 接口的特定功能。当向主复位口（E8H）输出时，就把 UAPT 复位到一定的状态。这种操作在装入或读取UART 寄存器的任何内容之前，必须进行一次，这可以由下列 Z-80 指令的执行来实现：

OUT　（0E8H），A；输出到主复位单元

此时，A的内容并不重要，对操作结果毫无关系。当从这个口地址（E8H）进行输入时，

调制调解器状态寄存器的内容就被取入CPU的寄存器。该操作可以通过执行下列Z80的指令来实现：

IN A，(0E8H)；读调制解调器状态寄存器

当执行该指令时，调制解调器状态寄存器的内容取入A中。取入A中的信息是和接口相连的外部设备的状态有关的信息。

当在口E9H输出时，某常数就被取入波德率发生器中。由该常数确定串行通道的接收/发送波德率。此操作可通过执行下列Z80的指令来实现：

LD A，22H ；把常数取入A中

OUT (0E9H)，A[1]；把常数取入BRG中

一执行上述程序，就使得串行通道以110波德的速率进行数据的发送/接收。在表5中概述了取入BRG中的常数和由此选定的波德率之间的关系。

取入BRG中的常数的高位部分（D_7～D_4）决定了串行通道的发送波德率，低位部分（D_3～D_0）决定接收波德率。

取入的半字节	发送/接收波德率	10×时钟频率	误码率（%）	能否由开关选定
0H	50	0.8KC	0	NO
1H	75	1.2KC	0	NO
2H	110	1.76KC	0	Yes
3H	134.5	2.1523KC	0.016	NO
4H	150	2.4KC	0	Yes
5H	300[2]	4.8KC	0	Yes
6H	600	9.6KC	0	Yes
7H	1200	19.2KC	0	Yes
8H	1800	28.8KC	0	NO
9H	2000	32.081KC	0.253	NO
AH	2400	38.4KC	0	Yes
BH	3600	57.6KC	0	NO
CH	4800	76.8KC	0	Yes
DH	7200	115.2KC	0	NO
EH	9600	153.6KC	0	Yes
FH	19,200	316.8KC	3.125	NO

表5 BRG用的程序编制概要

注[1]：原文误为H——校译者

注[2]：原文误为130——译者

BRG 的程序编制

表 5 开关选定栏中印有“Yes”的波德率由Radio Shack 的软件支持。非标准的波德率，根据取入 BRG 的适当的常数来编程，从口 E9H 的输入操作把由结构转换开关程序化了的位模式取入 CPU 的寄存器中。如前所述，图 5 的转换开关框由 8 个单刀单掷开关组成。因此，开关为“开”状态时，结构为 8 进制表示的三态缓冲器的输入被提高（达＋5V）。相反地，在开关为“闭”状态时，输入被降低。

译码逻辑电路响应从口 E9H 的输入操作，产生负选通。该选通和三态缓冲器的“允许”输入端相连。为向 CPU 存数，缓冲器利用这个负选通，把其输入逻辑状态给出在数据总线上。此操作根据下列指令的执行来实现：

IN A，(0E9H)；取“开关选定”

执行该指令后，和设定的开关相对应的位模式就被存入 CPU 的 A 寄存器中。被存入的信息立刻由软件和 CPU 发出的一系列命令加以翻译，以构成接口。

当在口地址 EAH 输出时，UART 控制寄存器和同步交换闩锁器就被取入。这两个寄存器的功能彼此不同，控制时使用的寄存器位也不同。该口地址的高 5 位（D_7～D_3）表示串行通道的字长、停止位和奇偶规则等；而低 3 位（D_2～D_0）则表示被闩锁的控制输出，在这个输出中，其中一位在低电平时禁止数据传送，另两位控制“终端设备就绪”和“发送要求”的。在这个口地址上输出时，欲不产生干扰，必须给于一定的注意。表 6 是 RS—232—C 的各寄存器和位的分配的概要。

数据位	调制解调器的状态寄存器	结构转换开关	UART控制寄存器和同步交换闩锁器	UART 的状态寄存器
D_7	允许发送 插针5，DB-25	偶 校 验 1＝偶数，0＝奇数	偶 校 验 1＝偶数，0＝奇数	数据收到 1＝条件成立
D_6	数据数传机准备好 插针6，DB-25	字长选定 1	字长选定 1	发送保持寄存器为空 1＝条件成立
D_5	载波检测 插针8，DB-25	字长选定 2	字长选定 2	溢出故障的状态 1＝条件成立
D_4	振铃指示器 插针22，DB-25	停止位个数选定 1＝2位，0＝1位	停止位个数选定 1＝2位，0＝1位	成帧误差的状态 1＝条件成立
D_3	不用	禁止奇偶校验 1＝不进行奇偶校验	禁止奇偶校验 1＝不进行奇偶校验	奇偶校验故障的状态 1＝条件成立
D_2	不用	波德率 3	发送中断 0＝数据不能传送	不 用
D_1	接收设备输入 UART P20	波德率 1	发送要求 插针4，BD-25	不 用

D_0	不用	波德率2	终端设备就绪 插针20，DB-25	不　用
	输入0E8H	输入0E9H	输出0EAH	输入0EAH

表6　各寄存器和闩锁器的位分配

用Z80汇编语言书写的下述程序示例的内容是：读取转换开关并和既定的闩锁位组合在一起，取入UART控制寄存器和同步交换闩锁器，为了以后在必要时进行修改，把这些控制位的现状保存起来。

```
IN A, (0E9H)      ; 读转换开关
AND 0F8H          ; 使最低三位为0
OR  05H           ; 置位“发送要求”，复位“终端设备就绪”并建立断点
OUT (0EAH)        ; 取入UART控制寄存器和同步交换闩锁器
LD (IMAGE),A      ; 保存位模式以备修改
```

上例最后一行中的IMAGE表示存贮单元。该存贮单元用于保存控制寄存器和同步交换闩锁器的现状。假如欲改变发送要求的逻辑状态而不改变UART控制寄存器和同步交换闩锁器其他位的情况，请看下例。

```
LD  A, (IMAGE) ; 取UART控制寄存器和同步交换闩锁器的现行状态
RES 0, A       ; 复位A中的位0（发送要求）
OUT(0EAH), A   ; 存入新的位模式
LD (IMAGE), A  ; 保存新的位模式以备修改
```

在口地址EAH输入时，UART状态寄存器的内容就被读取。位D_7～D_3传递了有关UART现状的信息（输入时的接收数据的准备情况，发送保持寄存器是否为空的情况，有无溢出故障，有无成帧差错，有无奇偶错）。其他的D_2～D_0位不用。这种寄存器的读取可通过执行下述汇编指令来实现：

```
IN  A,  (0EAH)  ;  读UART状态寄存器
```

当执行此指令时，UART状态寄存器的内容就被取入A寄存器。若使用附加指令，就能译出该状态信息，并不验证新字符能否传送、是否接收了新字符、接收了的新字符中有无错误。

当向口地址EBH输出时，将被UART发送的新字符被取入UART发送保持寄存器中。但是，在没有判明保持寄存器为空状态（即前面的字节已传送到准备传送的发送设备的寄存器中）以前，不能进行此操作。在实际进行操作时，要读出UART状态寄存器，验证对应位为逻辑1。下述汇编指令组就是这种方法：

```
                    ; 把字符输出到UART以便发送
                    ; A中应含有要被发送的字符
        PUSH  AF    ; 保存要被发送的字符
STATIN  IN  A,(0EAH); 取UART状态寄存器
        BIT  6, A   ; 测试发送保持寄存器是否为高电平
```

```
        JR Z, STATIN     ; 如果不是，重新测试
        POP  AF          ; 恢复要被发送的字符
        OUT (0EAH),A     ; 把字符取入保持寄存器
```

当从口地址 EBH 输入时，UART 的接收保持寄存器的内容被读取，UART 的状态寄存器的接收数据位被复位。在 UART 状态寄存器内的接收数据位被置位（即一字符被接收并被传送到接收保持寄存器）以前，此操作不能进行。下述汇编指令组就是这种方法：

```
                    ;  从 UART 输入一字符
                    ;  接收到的字符将在A寄存器中
INCHAR IN A,(0EAH)  ;  取 UART 状态
       BIT 7, A     ;  测试接收到的数据是否为高电平
       JR Z, INCHAR ;  如果不是，重新测试
       IN A, (0EBH) ;  取接收到的字符
```

执行上述程序后，接收到的数据被取入寄存器 A 中，UART 控制寄存器的接收数据位被复位。

本手册的开始部分说过：在 RS—232—C 设备中，有数据终端设备和数据通讯设备两种不同类型。它们不同点在于：在发送数据和接收数据的功能中，利用 DB—25 连接器的不同扦针。数据终端设备在插针 2 发送数据，在插针 3 接收数据。因此，和数据终端设备相连的设备，在插针 2 接收数据，在插针 3 发送数据(请参阅表 2)。换言之，根据接口设备的特定的要求，有时也有必要使这两种设备的信号操作反演。考虑到这种情况，在 Tandy Radio Shack 的 RS—232—C 接口中安装一个双掷开关（请参阅接口板的略图），开关的一边写有“TERM”（数据终端设备），另一边写有“COMM”（通信设备）。

RS—232—C 接口中，必须供给四种电压(+5V，-5V，+12V-12V)。扩展接口供给+5V电压，但是，使用 DC—DC 转换器，可从这个电压产生更高的电压。在Tandy Radio Shack 接口 RS—232—C 中使用的 DC—DC 转换器，只要输入 +5V，就能输出 +12V 或 -12V。如对 -12V 使用简单的电阻分压电路，还能供给 -5V电压（请参阅图 5 的略图）。由于 DC—DC 转换器本来是一种会发出令人讨厌的噪声的部件，所以必须和输入源进行适当的隔离，而在输出端要加滤波器。这种隔离和滤波功能由 L_1，L_2，C_1，C_{10}，C_{11}，C_3 和 C_9 来实现。

终端设备的程序

下列设备是必要的

*带 16K RAM 的 TRS-80 LEVEL Ⅱ

*配备 RS—232—C的扩展接口

TRS-80/RS—232--C 由于“TERM”程序而能获得作为全双工终端设备的功能。这种所谓全双工终端设备可以显示从 RS—232—C 口输入的文本（变换成 ASCII 码），也可以把从键盘来的输入内容输出到 RS—232—C 口。(TERM 程序的完整文本刊载在本手册的后边）。

RS—232—C向显示设备的输入

程序一被启动，屏幕被清除，光标置于屏幕的左上部。文本在输入时将被显示，同时光标向右移。一行满后，跳向下一行（不舍去溢出字符）。画面满幅时，后续的文本把该画面向上卷起，再在屏幕的下方显示出来。

“退格”（16进制数08）和“回行”（16进制数0D）由显示程序识别和执行，但是，不考虑其他ASCII控制码。另外，以较高波德率（1200，2400等）回车时，必须有让画面卷起的延迟时间[空白（16进制数00）]，如表7所示。

波德率	空白的个数	波德率	空白的个数
110	无	1200	2
150	无	2400	4
300	无	4800	8
600	无	9600	16

表7　回车时必需的空白

小写字母的ASCII码将用大写字母来显示。（在TRS-80中，要显示小写字母必须对硬件进行修改，但是，在Tandy Radio Shack的接口中，没有这种功能）。

键盘向RS—232—C的输出

为了检测新的键入信号，连续地扫描键盘。新的键选通被变换成ASCII码以备传送，并向RS—232—C接口输出。

键盘输出时，若RS—232—C的输出不被返回到RS—232—C口的输入部分，其内容不显示。这种功能称为全双工，即FDX。

若把RS—232—C连接器的插针2和3短接，终端设备就工作在“本地”方式，键盘的动作可直接在CRT上显示。还应注意到，键盘的信息并不存入内存用于以后的传送，因为某些比较贵的CRT终端，有时会对键盘信息加以选择。

若在Radio Shack电话调制解调器中安装有一个开关，将其置于“半双工”（即HDX）位置，则可使用电话线进行通信，也能够利用回波功能（限于调制解调器的耦合器内）。如使用这种回波功能，也能在半双工状态下使用TERM程序。就是在一般的低速（300波德）调制解调器中，大多数情况下，这也是可能的。

这里特殊规定的是允许从键盘输出四种7位码（由用户指定）。请参阅表8。

使用鍵	地址内的代码	缩写码
shift↓，A	50　B9H	03H＝EOT
shift↓，B	50　BAH	1BH＝ESC
shift↓，C	50　BBH	FCH＝1
shift↓，D	50　BCH	FFH＝DEL

表8　来自键盘的输出码

欲形成这些代码，需进行以下操作：

(1) 按下“SHIFT”键

(2) 按下“↓”键

(3) 按下A，B，C，D中的任一个键

这些键可以任何次序来释放。

同样地，也可形成其他的标准控制码。例如，按下E～O键时，形成05H～0FH，而按下P～Z键时，则形成10H～1AH。（要验证这点时，可按下SHIFT ↓，H键形成08H的退格）。

当这些控制码不适当时，可利用BASIC的SYSTEM命令（请按下复位按纽）取入TBUG LEVEL Ⅱ，并把所要的代码存入50B9～50BCH中。使用TBUG转移到地址5000H，重新输入TERM。因为TERM保存在地址5000H～50BCH，因此TERM和TBUG能同时保留在RAM中。

注意事项

1) 为了测试程序和硬件，可把RS—232—C连接器的插针2和3短接。

2) 在数据通信时，错误地建立奇偶性、奇偶校验功能、波德率、终端设备/数据设备（对插针2和3的I/O分配）时，会立刻发生问题。

3) TERM程序不进行RS—232—C硬件用的奇偶错、溢出等各种故障状态的标志检查。若是对程序设计有经验的用户，就一定注意到使用TBUG，能在各种应用中利用这个被忽视的信息。

TERM程序文本

TERM程序文本（在磁带上）被修改后，含有如下的“补充”（Patch）。由于有了该“补充”，所以在发生下列任一串行输入故障时，通常在显示屏幕上显示竖条：

1) 奇偶故障

2) 溢出（前面的字符还没有被TERM程序读取时，UART又接收了别的字符，等等）

3) 成帧错误（在理应遇到停止位的部分，却由UART检测到空白状态）

另外，若不要这种“补充”，要进行如下的改变。

如文本的示例那样，地址5018，5019，501A（16进制）的C3，C0，50分别被E6，7E，FE所置换。

欲要“补充”，必须有下述步骤：

1) 把C3，C0，50输入到地址5018，5019，501A中；

2) 在地址50C0（16进制数）插入：

```
50C0    F5          PATCH   PUSH    AF
50C1    3A6650              LD      A,(5066H)     ; 取得 UART 状态
50C4    E638                AND     38            ; 来自设备控制块 UCB
50C6    2805                JR      Z,NOFLT       ; 判断溢出、成帧、奇偶?
50C8    3EAA                LD      A,0AAH
```

```
50CA    CD 3300          CALL  DSP$     ; 显示竖条
50CD    F1       NOFLT   POP   AF
50CE    E67F             AND   7FH
50D0    FE60             CP    60H
50D2    C31C50           JP    501CH ; OUT—OF—PATCH
```

下面是完整的 TERM 程序文本：

```
00001                    ; 被系统用 RS—232—C 接口取入 LEVEL Ⅱ TRS-80 的磁带程序（它使系
00002                    ;   统起 CRT 终端的作用）
00003                    ;
00005                    ; RKUBALA 编制  1978年 7 月20日修订
00007          0033      DSP$    EQU   0033H
00008          002B      KBD$    EQU   002BH
00009          0046      CIO$    EQU   0046H
00011          5000 >           ORG   5000H
00013  5000    3E1C      TERM    LD  A, 1CH          ; 给光标定位
00014  5002    CD3300            CALL  DSP$
00015  5005    3E1F              LD  A,1FH           ; 清除屏幕
00016  5007    CD3300            CALL  DSP$
00017  500A    3E0E              LD  A,  0EH         ; 接通光标
00018  500C    CD3300            CALL  DSP$
00019  500F    CD5850 >          CALL  MRUART
00021                    ; 主程序——连接RS—232—C，鍵盘及 CRT
00023  5012    CD4F50>   INS     CALL  SIN           ; 检查是否是新的UART输入
00024  5015    B7                OR  A
00025  5016    2812              JR  Z,  OUTS        ; 如不是，则扫描鍵盘
00026  5018    E67F              AND  7FH            ; 从 ASCII 中消除奇偶
00027  501A    FE60              CP  60H
00028  501C    FA2150>           JP  M, $+5
00029  501F    E65F              AND  5FH            ; 小写字母转换为大写字母
00030  5021    FE0A              CP  0AH
00031  5023    28ED              JR  Z, INS          ; 忽略"LF"
00032  5025    CD3300            CALL  DSP$          ; 在 CRT 上显示可显示字符
00033  5028    18E8              JR  INS
00035  502A    CD2B00    OUTS    CALL  KBD$
00036  502D    B7                OR  A
00037  502E    28E2              JR  Z,  INS
00038  5030    FE05              CP  05H
00039  5032    F23D50 >          JP  P, NOSPEC
00040  5035    21B850 >          LD  HL, SPECTB—1    ; 特殊代码表
00041  5038    4F                LD  C, A
00042  5039    0600              LD  B, 0
00043  503B    09                ADD  HL,  BC        ; HL→ 被选的特殊码
```

```
00044  503C  7E              LD   A, (HL)          ; 获得特殊码
00045  503D  FE1A    NOSPEC  CP   1AH
00046  503F  28D1            JR   Z, INS          ;不考虑 SHIFT ↓ 键
00047  5041  CD4650  >        CALL SOUT            ; 把键盘输入输出到RS—232—C
00048  5044  18CC              JR   INS
00050                         ; RS—232—C 驱动程序调用序列
00052  5046  116150  > SOUT    LD   DE, SUCB
00053  5049  C5                PUSH BC
00054  504A  0620              LD   B, 20H          ;输出字节
00055  504C  C34600            JP   CIO$
00057  504F  116150  > SIN     LD   DE, SUCB
00058  5052  C5                PUSH BC
00059  5053  0640              LD   B, 40H          如果有，就输入字节
00060  5055  C34600            JP   CIO$
00062  5058  116150  > MRUART LD   DE, SUCB
00063  505B  C5                PUSH BC
00064  505C  0680              LD   B, 80H          ; 复位UART
00065  505E  C34600            JP   CIO$
00067                    ; RS—232—C 设备控制块 (UCB)
00069  5061  E0        SUCB    BYTE 0E0H            ; 功能屏蔽
00070  5062  6950  >            WORD RS232           ; 驱动程序地址
00071  5064  00                BYTE 0               ; TERM. 开关结构
00072  5065  00                BYTE 0               ; 波德率码
00073  5066  00                BYTE 0               ; UART 状态
00074  5067  00                BYTE 0               ; 调制解调器状态
00075  5068  00                BYTE 0               ; 控制寄存器映象
00077                    ; RS—232—C 驱动程序
00078                    ; 入口：IX→UCB+0
00079                    ;       C=参数
00080                    ;       B=FCT. 码
00081                    ;
00082                    ; 出口：A=状态或数据
00083                    ;
00085  5069  78        RS232   LD   A, B            ; 检查 FCT. 請求
00086  506A  17                RLA
00087  506B  3821              JR   C, IUART
00088  506D  17                RLA
00089  5C6E  3805              JR   C, RSRD
00090  5070  17                RLA
00091  5071  380E              JR   C, RSWR
00092  5073  AF        RSX     XOR  A
00093  5074  C9                RET
```

```
00095  5075  DBEA      RSRD   IN A, (CTRL)        ; 获得 UART 状态寄存器
00096  5077  DD7705           LD (IX+5), A        ; 映象到 UCB
00097  507A  CB7F             BIT 7, A            ; 被接收的字节是否有效
00098  507C  28F5             JR Z, RSX           ; 如果无效
00099  507E  DBEB             IN A, (DATA)        ; 取得数据，不消除奇偶位
00100  5080  C9               RET
00102  5081  DBEA      RSWR   IN A, (CTRL)        ; 获得UART状态寄存器内容
00103  5083  DD7705           LD (IX+5), A        ; 映象到 UCB
00104  5086  CB77             BIT 6, A            ; 发送保持寄存器是否空
00105  5088  28F7             JR Z, RSWR          ; 如果不空，则等待
00106  508A  79               LD A, C
00107  508B  D3EB             OUT (DATA), A       输出字节
00108  508D  C9               RET
00110        00E8      MR     EQU 0E8H
00111        00E8      MODEM  EQU 0E8H
00112        00E9      CONFIG EQU 0E9H
00113        00EA      CTRL   EQU 0EAH
00114        00EB      DATA   EQU 0EBH
00116                  ; 用结构开关预置RS—232—C硬件
00118  508E  D3E8      IUART  OUT (MR), A         ; 通过输出数据来复位UART
00119  5090  DBE9             IN A, (CONFIG)      ; 获得TERM结构开关信息
00120  5092  DD7703           LD (IX+3), A        ; 保存映象
00121  5095  E6F8             AND 0F8H            ; 屏蔽波德率信息
00122  5097  F605             OR 05H              ; 置位BRK,复位 DTR, 置位 RTS
00123  5099  DD7707           LD (IX+7) , A       ; 保存控制寄存器的映象
00124  509C  D3EA             OUT (CTRL), A       ; 并放入控制寄存器中
00126  509E  DBE9      IBRG   IN A, (CONFIG)      ; 获得波德率开关状态
00127  50A0  E607             AND 07H             ; 只要波德率位
00128  50A2  21B150    >      LD HL, BAUDTB
00129  50A5  0600             LD B, 0
00130  50A7  4F               LD C, A
00131  50A8  09               ADD HL, BC          ; HL→波德率码
00132  50A9  7E               LD A, (HL)          ; 获得波德率码
00133  50AA  DD7704           LD (IX+4), A        ; 把映象保存在 UCB中
00134  50AD  D3E9             OUT (CONFIG), A     ; 存入BRG
00135  50AF  AF               XOR A
00136  50B0  C9               RET
00138                  ; 波德率代码表
00140  50B1  22       BAUDTB  BYTE 22H           ; 110波德
00141  50B2  44               BYTE 44H            ; 150波德
00142  50B3  55               BYTE 55H            ; 300波德
00143  50B4  66               BYTE 66H            ; 600波德
```

```
00144   50B5   77              BYTE  77H       ;  1200
00145   50B6   AA              BYTE  0AAH      ;  2400
00146   50B7   CC              BYTE  0CCH      ;  4800
00147   50B8   EE              BYTE  0EEH      ;  9600
00149                  ; 特殊代码表
00151   50B9   03      SPECTB  BYTE  03H       ;  空缺：EOT —— CTRL "A"
                                                  (还有"ATTN")
00152   50BA   1B              BYTE  1BH       ;  空缺：ESC —— CTRL "B"
00153   50BB   7C              BYTE  7CH       ;  空缺：竖条—— CTRL "C"
00154   50BC   7F              BYTE  7FH       ;  空缺：DEL —— CTRL "D"
```

标量

CIO$——0046　　CONFIG——00E9　　CTRL——00EA

DATA——00EB　　KBD$——002B　　MODEM——00E8

MR——00E8

%至（空缺）区（50BD）

BAUDTB——50B1　　IBRG——509E　　INS——5012

IUART——508E　　NOSPEC——503D　　OUTS——502A

RS232——5069　　RSRD——5075　　RSX——5073

SIN——504F　　SOUT——5046　　SPECTB——50B9

TERM——5000

156　源行　　156汇编行

行式打印机用程序示例

为了在行式打印机上输出串行数据，Tandy Radio Shack 的 RS—232—C 也许是使用最多的一种设备。

在叙述有关软件的详细说明以前，先大致介绍一下关于 LEVEL Ⅱ BASIC 的 I/O 驱动程序的结构。LEVEL Ⅱ BASIC 一上电就立刻把 I/O 驱动程序有关的一系列地址写入 RAM 的低端部分。BASIC 软件被适当的 I/O 驱动程序分配到这些地址。(LPRINT, PRINT, CSAVE, LIST, LLIST等)

计算机上电以后，这些地址就一直保持在 RAM 的低端部分，除非用新的信息覆盖它们。一般说来，只要操作正常，这些地址也不会被分配和覆盖，因为这些地址在内存中的区域是不会被 LEVEL Ⅱ BASIC 占用的，但是，也可有意识地改写这些地址，以便将 BASIC 引导到自行设计的 I/O驱动程序中。上面谈到的地址以及其它的有关 I/O 的讯息称为"设备控制块"，即 DCB。

行式打印机用的 DCB 从存贮器地址 4025 开始，按如下形式组成：

存贮器地址	内　　容
4025H	DCB 类型（I/O操作）
4026H	驱动程序地址（字节的低位部分）
4027H	驱动程序地址（字节的高位部分）

一般，DCB 中的行式打印机驱动程序地址被放在 LEVEL Ⅱ 的 ROM 中，这是考虑到要使用 Centronix 公司的并行接口。如果要使用串行行式打印机并利用行式打印机用的 LEVEL Ⅱ 命令（LPRINT，LLIST），那末应有意识地改变 DCB 以指向 RAM 的地址，该地址放着自行设计置的 I/O 驱动程序。

例如，我们考虑一个把 LEVEL Ⅱ TRS-80 和以 300 波德工作的串行行式打印机连接的情况。我们采用这样的设计：在行式打印机中使用 7 位字长，1 位停止位，偶校验。设定使用一行输出信息来检查打印机是否处在可接收数据的状态。该行为低时，打印机为空闲状态，可从接口接收数据。另一方面，该行为高时，打印机为忙状态，在此状态被解除以前，不能接收数据。

若把从打印机输出的这种状态信息和 RS—232—C 中能使用的 4 种输出（CTS，DSR，DC，RI）中的任一种结合起来，就可用 TRS—80CPU 测试。唯一需要的修改就是把打印机连接器（假定在打印机中使用规定的 DB—25）的插针 6 和打印机的“状态输出”连接起来的跨接线。

至此，如果编制适当的软件使两者连接起来，那么，打印的准备工作就算结束了。在实际数据输出以前，打印机和RS—232—C 首先必须进行和打印要求相适应的波德率、字长、奇偶形式、停止位个数等的预置工作。这是应当由驱动程序的软件处理的任务之一。502～504 页上的汇编程序文本就是为驱动 DECMRITER 而编制的。文本中的 180—440 行这一段是用作驱动程序的预置的。这部分程序检查存贮器中的标志，检查接口的预置是否完成。若是，则跳过予置程序的其余部分转到程序的实际输出部分（500行～620 行）。预置是 BASIC 在一开始调用驱动程序时进行的。这时，标志被设置，表示进入预置阶段。这部分程序在以后调用驱动程序时被旁路。

汇编语言 DECWRITER 驱动程序

```
00E8    00100   RESURT   EQU   0E8H  ; 对本单元的“输出”复位 UART，“输入”则读
                                       RS—232的控制位
00E9    00110   SWITCH   EQU   0E9H  ; 对本单元的“输出”存入BRG，“输入”则读转换
                                       开关
00EA    00120   CNTREG   EQU   0EAH  ; 对本单元的“输出”存入 UART 控制寄存器，“输
                                       入”则读 UART 状态寄存器
00EB    00130   DTAREG   EQU   0EBH  ; 对本单元的“输出”存入 UART 发送保持寄存器，
                                       “输入”则读被接收的数据
7F00    00140            ORG   7F00H ; 驱动程序的起始地址规定为 32512D
                00150; 以 LEVEL Ⅱ BASIC 的 LPRINT 命令来使用 RS—232—C
                       的驱动程序。该驱动程序被放置在存贮器的高端部分
                00160; (16K 的机器)，并且可改变设备控制块以使 LPRINT 命
                       令指向带有一个短的 BASIC 程序的驱动程序
                00170;
7F00    E5      00180 INIT   PUSH  HL       ; 保存使用的寄存器
7F01    C5      00190        PUSH  BC
7F02    F5      00200        PUSH  AF
```

```
                00210;   这部分代码用于预置RS—232—C接口以适合由接口中的转换开关指定
                00220;   的选择（波德率，停止位，位/字符等）
7F03  3A487F  00230        LD A, (FLAG)          ; 检查标志，验证UART和BRG是否已被
                                                   预置
7F06  FE01    00240        CP 01H
7F08  2820    00250        JR Z, RESTOR          ; 如果已被预置，则恢复寄存器和输出字符
7F0A  3E01    00260        LD A, 01H
7F0C  32487F  00270        LD (FLAG), A          ; 设置指示预置的标志
7F0F  D3E8    00280        OUT (RESURT), A       ; 读37DCH以复位UART
7F11  DBE9    00290        IN A,(SWITCH)         ; 读转换开关
7F13  E6F8    00300        AND 0F8H              ; 去除低三位
7F15  F604    00310        OR  04H               ; 复位RTS，复位DTR，建立同步交换闩
                                                   锁器中的BRK
7F17  32477F  00320        LD (SWTIMG), A        ; 取闩锁器的W/映象
7F1A  D3EA    00330        OUT (CNTREG), A       ; 取UART W/开关映象
7F1C  DBE9    00340 BAUDST IN A, (SWITCH)        ; 根据开关选择设置波德率
7F1E  E607    00350        AND 07H               ; 除去高5位
7F20  213F7F  00360        LD HL, BDTABL         ; 指向波德率代码表中的第一单元
7F23  0600    00370        LD B, 00H             ; 清零B寄存器
7F25  4F      00380        LD C, A               ; 把“偏置”置于C寄存器
7F26  09      00390        ADD HL, BC            ; 把“偏置”加到HL寄存器
7F27  7E      00400        LD A, (HL)            ; 取所指示的值
7F28  D3E9    00410        OUT (SWITCH), A       ; 取 BRG W/ 表值
7F2A  F1      00420 RESTOR POP  AF
7F2B  C1      00430        POP  BC
7F2C  E1      00440        POP  HL               ;恢复REG
                00450;   这部分程序把字符真正输出到UART以备串行发送。它首先检查UART，
                00452;   验证其保持寄存器是否空，若不空，则循环直到保持寄存器为空。然后，
                00453;   把要传送的字符取入保持寄存器。如果这字符是“回车”，则再输出一
                00454;   “换行”符
7F2D  DBEA    00490 STATIN IN A, (CNTREG)        ;  取UART状态
7F2F  CB77    00500        BIT 6, A              ;  测试THRE是否为高电平
7F31  28FA    00510        JR Z, STATIN          ;  如不为高电平，则循环
7F33  79      00520        LD A, C               ;  取要被输出的W/字符
7F34  D3EB    00530        OUT (DTAREG), A       ;  取保持寄存器W/字符
7F36  FE0D    00540        CP  0DH               ;  该字符是否为“回车”符
7F38  2004    00550        JR NZ, RETRN          ;  若不是，则返回
7F3A  0E0A    00560        LD C, 0AH             ;  若是，则输出“换行”符
7F3C  18EF    00570        JR STAIN              ;  输出到UART
7F3E  C9      00580 RETRN  RET                   ;  返回
                00590;   下表定义了由接口中的转换开关所选择的波德率
7F3F  22      00600 BDTABL DEFB  22H             ;  110 波德
```

```
7F40    44      00610           DEFB    44H     ;  150 波德
7F41    55      00620           DEFB    55H     ;  300 波德
7F42    66      00630           DEFB    66H     ;  600 波德
7F43    77      00640           DEFB    77H     ; 1200 波德
7F44    AA      00650           DEFB    0AAH    ; 2400 波德
7F45    CC      00660           DEFB    0CCH    ; 4800 波德
7F46    EE      00670           DEFB    0EEH    ; 9600 波德
7F47    00      00680 SWTIMG    DEFB    00H     ; 同步交换闩锁器的映象
7F48    00      00690 FLAG      DEFB    00H     ; 指示预置的标志
0000            00700           END
00000 TOTAL ERRORS
RETRN           7F3E
STATIN          7F2D
BDTABL          7F3F
BAUDST          7F1C
SWTIMG          7F47
RESTOR          7F2A
FLAG            7F48
INIT            7F00
DTAREG          00EB
CNTREG          00EA
SWITCH          00E9
RESURT          00E8
```

DECWRITER 用的 BASIC 程序

```
10    REM** POKE NEW DCB TYPE AND ADDRESS IN RAM (4025H) **
20    POKE 16421, 2: POKE 16422, 0: POKE 16423, 127
30    REM**POKE RS—232—C I/O DRIVER INTO HIGH MEM (7F00H) **
40    FOR X=32512 TO 32584
50    READ Y
60    POKE X, Y
70    NEXT X
75    END
80    DATA 229, 197, 245, 58, 72, 127, 254, 1, 40
90    DATA 32, 62, 1, 50, 72, 127, 211, 232, 219, 233
100   DATA 230, 248, 246, 4, 50, 71, 127, 211, 234
110   DATA 219, 233, 230, 7, 33, 63, 127, 6, 0, 79, 9
120   DATA 126, 211, 233, 241, 193, 225, 219, 234
130   DATA 203, 119, 40, 250, 121, 211, 235, 254
140   DATA 13, 32, 4, 14, 10, 24, 239, 201, 34, 68
150   DATA 85, 102, 119, 170, 204, 238, 0, 0
```

必须由预置代码进行的操作：

• 复位 UART

• 读取结构转换开关

• 把由开关（停止位、字长、奇偶形式）指定的控制信息取入UART

• 把由开关指定的波德率代码正确地取入 BRG

• 在存贮器中设置予置标志

500行～620行的这一部分程序执行向打印机的数据输出。软件在一开始就检查“数据设备准备好”是否为负，若不为负，则进行循环测试。当测试得知“数据设备准备好”为负时，软件就检查UART的发送保持寄存器的状态是否为正（“正”表示UART能接收传送中的下个数据）。如不为正，则循环；如为正，则把后续字符取入UART保持寄存器中。字符取入后就检查是否“回车”(0 DH)。如果是“回车”，则“换行”码(0AH) 也一起被送出。这种“换行”功能，根据相连的各种打印机型式，可要可不要（这部分代码也可删除）。在这时，控制返回BASIC。

504 页上刊载的、接在驱动程序的汇编程序后面的那一部分是 BASIC 程序文本。该 BASIC 程序处理两个任务：向行式打印机的设备控制块 DCB 写入新的地址（第 20 行）；向存贮器的高端部分写入打印机驱动程序的实际代码（40～120 行）。此外，80～150 行的数据语句是用 10 进制表示的驱动程序目的代码的各字节。该代码就是在 502～504 页的汇编文本中的第二列内记载着的。

欲使用这个程序。请参阅 505 页。

假定接口本身的要求和本例相同，那么在这里要完成的工作就是：打入 504 页上的 BASIC 程序，并使其运行。这样，就用 BASIC完成了使用串行行式打印机的准备工作。下面写出每个步骤，以表示使用顺序：

1） 请给 TRS—80 和扩展接口上电。

在图象监控器上出现下述显示：

MEMORY SIZE?

对此，作如下回答：

32511 （7EFFH的十进制数值）

然后，按下 ENTER 键。

根据这个回答，预定了打印机驱动程序用的16K TRS—80 的最高 256 字节。

2） 根据需要，为了进行规定的波德率、停止位、字长、奇偶状态的设定，设置好结构开关。

3） 在进行修改以满足特殊应用后，打入 504 页上的 BASIC 程序，或者从磁带上取入以前准备好了的上述程序。

4） 使 BASIC 程序运行。

5） 使用附带的电缆（或者，专门修改过的电缆），把行式打印机接入RS—232—C。

6） 以上完成了准备。从这里开始可用 LEVEL Ⅱ BASIC LPRINT 和 LLIST 命令来使用串行行式打印机。

任何使用这种方法的 BASIC 程序中，必定含有 504 页上的 BASIC 程序作为某一部分。而这部分代码在串行行式打印机使用前必须执行。如果不想把行式打印机驱动程序用的目的码放在地址 7F00H中时（如同本例），那么，使用 TRS—80 的程序汇编程序就可以重新汇编 502 页上起点不同（第 140 行）的源码，并修改 BASIC 程序使它和新的地址对应。

对于询问 MEMORY SIZE 的回答，也必须修改成用十进制表示的、此行式打印机驱动程序的目的码的开始地址减 1 的地址。504 页的 BASIC 程序的数据语句有必要改变，以完全反映根据再定位和修改而生成的目的码的变化。504 页的第 20 行的最后两个 POKE语句必须改正，以和驱动程序的目的码的新的开始地址的十进制表示相对应。（第二个 POKE 语句的变元相当于新地址的低位字节部分，第三个 POKE 语句的变元相当于新地址的高位字节部分）。

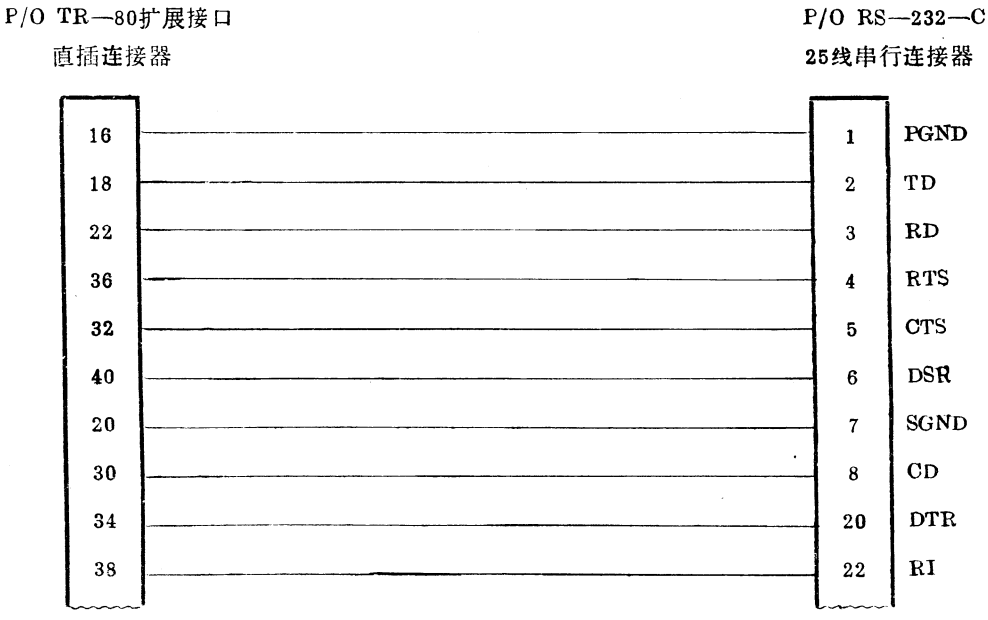
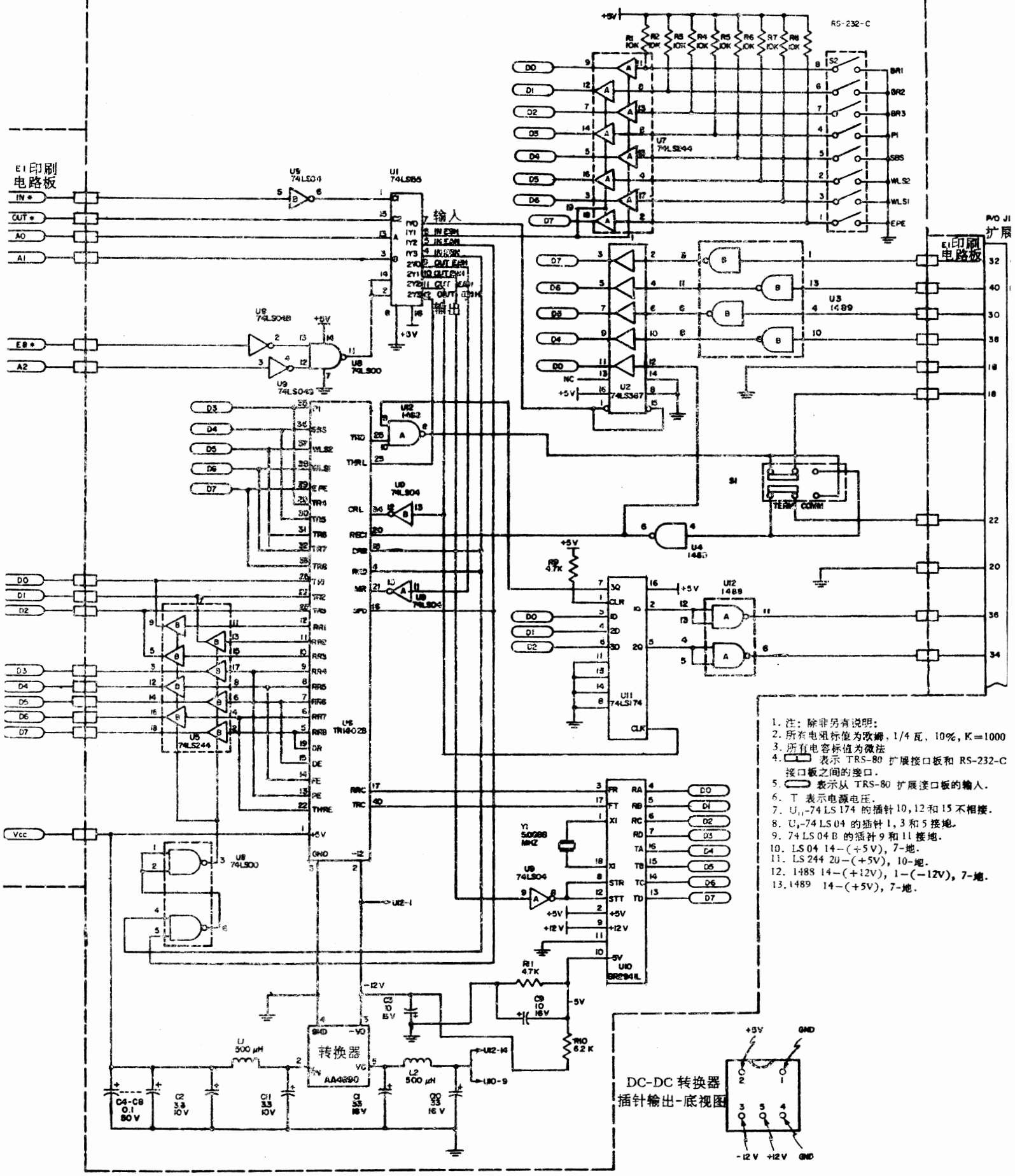


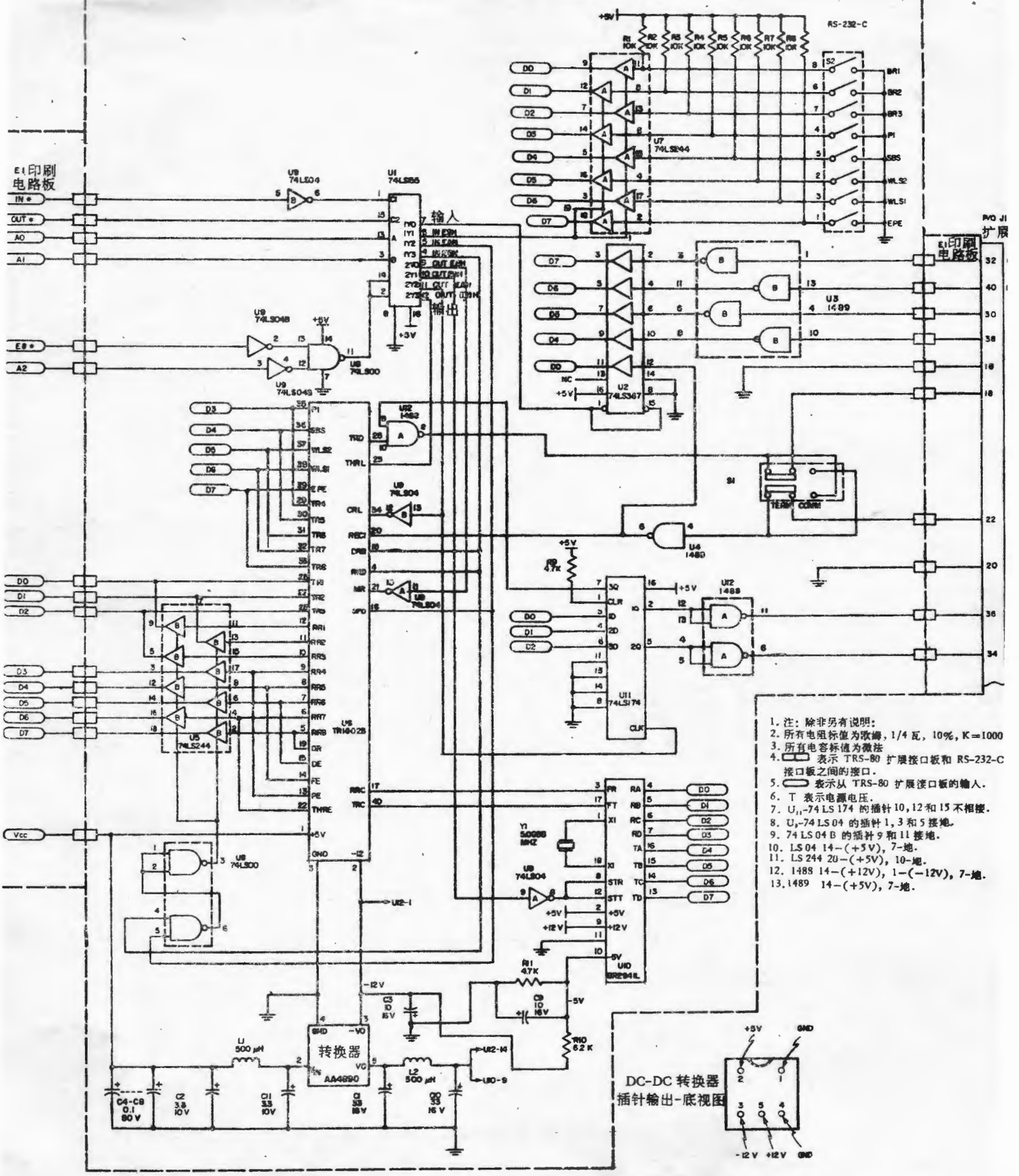
图 7　RS—232—C 电缆（接在 40 线和 25 线连接器之间）

（莫渭浓译　王玉铃　张乃尧　周宝兴校）



1. 注：除非另有说明：
2. 所有电阻标值为欧姆，1/4 瓦，10%，K=1000
3. 所有电容标值为微法
4. ▭▭ 表示 TRS-80 扩展接口板和 RS-232-C 接口板之间的接口。
5. ⬭ 表示从 TRS-80 扩展接口板的输入。
6. ⊤ 表示电源电压。
7. U_{11}-74 LS 174 的插针 10，12 和 15 不相接。
8. U_{13}-74 LS 04 的插针 1，3 和 5 接地。
9. 74 LS 04 B 的插针 9 和 11 接地。
10. LS 04 14－(+5V)，7-地。
11. LS 244 20－(+5V)，10-地。
12. 1488 14－(+12V)，1－(−12V)，7-地。
13. 1489 14－(+5V)，7-地。

图 8 RS-232-C 原理图


RS-232-C
输入
输出
E1印刷电路板
P/O J1 扩展
DC-DC 转换器
插针输出-底视图
转换器
AA4890
1. 注：除非另有说明：
2. 所有电阻标值为欧姆，1/4 瓦，10%，K=1000
3. 所有电容标值为微法
4. 表示 TRS-80 扩展接口板和 RS-232-C 接口板之间的接口。
5. 表示从 TRS-80 扩展接口板的输入。
6. T 表示电源电压。
7. U11-74 LS 174 的插针 10，12 和 15 不相接。
8. U9-74 LS 04 的插针 1，3 和 5 接地。
9. 74 LS 04 B 的插针 9 和 11 接地。
10. LS 04 14-(+5V)，7-地。
11. LS 244 20-(+5V)，10-地。
12. 1488 14-(+12V)，1-(-12V)，7-地。
13. 1489 14-(+5V)，7-地。