TP86A
8086单板计算机使用手册

北 京 工 业 大 学

微型计算机研究开发应用中心

前　言

本书是TP-86A单板计算机的使用手册，共分五章及附录。第一章概述TP-86A单板计算机的主要技术特性及功能；第二章主要介绍了TP-86A基本电路功能以及与使用有关的一些硬件问题；第三章和第四章分别说明了键盘监控程序和串行监控程序命令的使用；第五章扼要介绍了PL/M-86高级语言、ASM-86汇编语言及TP-86A监控程序。为了用户使用和维修方便，在附录中我们编入了TP-86A键盘监控程序和串行监控程序PL/M-86源程序、TP-86A电原理线路图和所用主要集成电路引脚图。

本手册连同《MCS-86　宏汇编语言参考手册》和《MCS-86汇编程序袖珍手册》一起可作为TP-86A的实用教材。

TP-86A单板计算机由北京工业大学无线电电子学系研制，参加研制工作的人员主要有：王开西、李慧文、何德祥、徐立和张黎明。硬件总体逻辑设计由王开西完成，软件配置及文件资料由李慧文负责组织。此外，李礼贤与谈根林参加了方案讨论，并为该机翻译了MCS-86宏汇编语言参考手册部分内容及PL/M-86语言程序设计手册；何筱筱同志设计绘制了该机的印刷电路板图。本手册由王开西、李慧文、徐立、张黎明编写。书中错误和不当之处，恳请读者予以指正。

北京工业大学微型计算机研究开发应用中心

1982年9月

目　　录

第一章　概述 …… (1)
1.1　引言 …… (1)
1.2　TP-86A简介 …… (1)
1.3　TP-86A主要技术指标 …… (1)
1.4　注意事项 …… (3)
第二章　TP-86A 基本电路 功能 …… (4)
2.1　引言 …… (4)
2.2　时钟发生器电路 …… (4)
2.3　等待状态发生器电路 …… (5)
2.4　中央处理器 8086CPU …… (6)
2.5　可编程只读存储器 PROM …… (6)
2.6　随机存储器 RAM …… (8)
2.7　输入输出I/O译码电路 …… (9)
2.8　键盘显示器电路 …… (11)
2.9　串行输入输出接口电路 …… (12)
2.10 并行输入输出接口电路 …… (16)
2.11 计数计时电路 …… (18)
2.12 离板译码电路 …… (19)
2.13 总线扩展电路 …… (19)
第三章　TP-86A 键盘监控程序命令的使用 …… (22)
3.1　引言 …… (22)
3.2　键盘 …… (22)
3.3　显示 …… (25)
3.4　总操作过程 …… (25)
3.5　监控程序命令 …… (26)
3.6　命令EB（检测字节）和 EW（检 测字） …… (28)
3.7　命令ER（检测字 寄 存 器） …… (31)
3.8　命 令 IB（输入 字 节）和IW（输 入字） …… (33)
3.9　命令OB（输出字节)和 OW(输出 字) …… (35)
3.10 命令GO（转移） …… (37)
3.11 命令MV（传送） …… (39)
3.12 命令ST（单步） …… (41)
3.13 命令CD（转录或写磁带） …… (42)

3.14 命令CL（转储或读磁带）……（44）
3.15 命令PE（EPROM写入）……（46）
第四章 TP-86A 串行监控程序命令的使用……（49）
4.1 引言……（49）
4.2 串行监控的命令格式和命令集……（50）
4.3 命令S[W]：更换存储器命令……（52）
4.4 命令X：检测/修改寄存器的命令……（53）
4.5 命令D[W]:显示存储器命令……（54）
4.6 命令M：移动数据块命令……（55）
4.7 命令I[W]：端口输入命令……（56）
4.8 命令O[W]：端口输出命令……（57）
4.9 命令G：转走（或转移）……（58）
4.10 命令N：单步命令……（59）
4.11 命令R：读十六进制文件命令……（60）
4.12 命令W：写十六进制文件命令……（61）
第五章 PL/M-86高级语言、ASM-86汇编语言及TP-86A监控程序简介……（63）
5.1 PL/M-86高级语言简介……（63）
5.2 ASM-86 汇编语言简介……（83）
5.3 TP-86A 监控程序简介……（124）
附录Ⅰ：TP-86A 键盘监控程序PL/M-86源程序……（139）
附录Ⅱ：TP-86A 所用主要集成电路引脚图……（177）
附录Ⅲ：TP-86A 电原理线路图……（189）

第一章 概 述

1.1 引 言

近几年，微型计算机的发展十分迅速。在国内，微机系统已深入到工业、农业、商业和国防各个领域，在科学计算、自动控制、企业管理和文化教育等各个方面取得了不少可喜的成果。

TP-86A 单板计算机是北京工业大学无线电电子学系为满足目前我国微机应用发展的需要，在引进国外先进技术加以吸收提高的基础上，而研制的十六位微型计算机。

TP-86A 具有体积小、结构简单、功能强、速度快、容量大、易于操作和维护等特点。它从性能上完全兼容国外同类型产品 SDK-86 单板计算机，并做了进一步的扩充完善工作，从而更适合于我国目前微机应用的需要。TP-86A 不仅可以作为"智能"部件，用于生产过程控制、各种仪器仪表或机械的单机控制、数据处理等，它还能够解决许多八位机所不能解决的要求处理速度快及信息量大的课题。TP-86A 具有较强的应用开发功能，并可以在原有基础上扩充，发展成为功能更强的十六位微型机系统。

1.2 TP-86A简介

TP-86A是十六位单板计算机，采用INTEL8086 微处理器作为中央处理单元（CPU）。8086 具有一兆字节的内存寻址能力。它有较高的工作时钟，其 CPU 内部采用了先行取指令的工作方式，从而运行速度与 8080A 比较大约快 7—12 倍，与 Z-80 比较大约快 2—4 倍。8086 与 8080 在汇编语言一级向上兼容，除包括 8080的全部指令系统外，还有功能更强的十六位指令系统。它有丰富的寻址方式，可以处理多种数据类型，并可使用浮动程序。

与一般单板计算机比较，TP-86A 板上具有较大的存储区和多种功能的输入输出接口，并可以直接使用终端键盘屏幕显示器 CRT、电传打印机 TTY、并行打印机、绘图机和音频盒式磁带机等外围设备，方便的 EPROM 编程逻辑可成为开发应用的有利工具。另外，总线扩展电路可使 TP-86A不断完善，发展成一个完整的微型机系统。

TP-86A 监控程序（键盘监控程序 VA.1 和串行监控程序 VA.2）提供了格式严谨、使用方便、操作灵活的监控命令，为软件研制和程序调试工作创造了有利条件。容量较大的只读存储区为固化软件的研制打下了基础。

1.3 主要技术指标

1. 中央处理器（CPU）：8086。

2. 工作时钟频率：5MHz

3. 指令周期：800ns

4. 基本指令：133条

5. 只读存储器（芯片类型2716/2732）：32K字节，可立即扩展为64K字节。

6. 随机存储器（芯片类型6116）：16K 字节，可立即扩展为32K字节。

7. 外部中断：可屏蔽中断

非屏蔽中断NMI（中断矢量2用于非屏蔽中断）。

8. 内部中断：中断矢量1（单步），中断矢量3（断点）。

9. EPROM 编程：可编程2716和2732。

10. 串行输入输出接口：一块8251A(20ma电流环或RS232,波特率为75—9600九档)

11. 并行输入输出接口：两块8255A。

12. 键盘显示控制器：一块8279。

13. 计数计时器：一块8253。

14. 串行外部设备：可带终端键盘屏幕显示器CRT、电传打印机TTY、盒式音频磁带机还可与INTEL开发系统MDS相连接。

15. 并行外部设备：Centronics接口标准（如μ-80打印机、WX4675绘图机)。

16. 板上键盘显示：24键键盘，8个七段显示灯，还具有键盘显示扩展接口（最多可带128键、16个七段显示灯）。

17. 总线扩展：全部系统控制总线、地址总线和数据总线；48线并行扩展接口（均与TTL电平兼容）

18. 键盘监控命令：

- EB键——检查、修改内存字节
- EW键——检查、修改内存字
- ER键——检查、修改寄存器
- GO键——由监控程序转向用户程序
- ST键——单步执行
- IB键——I/O口输入字节
- IW键——I/O口输入字
- OB键——I/O口输出字节
- OW键——I/O口输出字
- MV键——内存区传送
- CD键——磁带转录
- CL键——磁带转储
- PE键——EPROM编程

19. 串行监控命令：

- S[W]——显示、修改内存字节或字

X——显示、修改寄存器

D——显示内存区

M ——内存区传送

I[W] ——I/O口输入字节或字

O[W] ——I/O口输出字节或字

G ——由监控程序转向用户程序

N ——单步执行

R ——由纸带机或盒式音频磁带机输入十六进制格式文件

W ——将内存区内容以十六进制格式文件输出给纸带机或盒式音频磁带机。

20. 总线扩展布线区：5.2×12.7厘米

21. 直流电源：＋5伏（±5％），3安培

1.4 注意事项

1. TP-86A出厂前板上只装有4K字节RAM(U24、U25)和8K字节ROM（U38、U39；TP-86A BUG Ⅱ和Ⅰ）。

2. 初次使用TP-86A时，应先阅读本手册第二章及以后有关章节，检查板上各短路开关位置是否正确，是否已连接好所选外围设备。

3. TP-86A板上方中央处有三个电源接线柱：GND（地线）、＋5V、－12V。单板机5V工作电源由＋5和GND接线柱接入。－12V接线柱只有当接电传打印机TTY和CRT时才需接入－12V(大多数CRT连接时只需将－12V接线柱与GND接线柱相接)。

4. 单板机接通工作电源后应显示系统提示符，否则应断电，参照本手册进行检查。

5. TP-86A板上COUT插孔经转录线与盒式音频磁带机MIC插孔连接；CIN插孔经转录线与盒式音频磁带机EAR插孔连接。

6. 切勿用手直接触摸集成电路，切勿对本机直接使用电源电压为交流220V的电烙铁，以防损坏芯片。必须使用电烙铁时，应将电源断开，利用烙铁的余热进行焊接。

第二章　TP-86A 基本电路功能

2.1 引　言

本章介绍组成 TP-86A 单板计算机的基本电路功能。同时给出 TP-86A 系统框图和与 TP-86A 使用有关的说明。

在本章中介绍的基本电路有：

- 时钟发生器电路
- 等待状态发生器电路
- 中央处理器8086
- 可编程只读存储器PROM
- 随机存储器 RAM
- 输入输出I/O译码器电路
- 键盘显示器电路
- 串行输入输出接口电路
- 并行输入输出接口电路
- 计数计时电路
- 离板译码电路
- 总线扩展电路

以上各部分电路介绍，请参看插页 TP-86A 系统框图和附录 Ⅳ TP-86A 电原理图。

2.2 时钟发生器电路

TP-86A 采用 INTEL 8284 时钟发生驱动器作为整个系统的时钟发生器电路。8284（U12）接收来自晶体振荡器（Y）的输入。晶振的基频为 14.7456MHz，它不仅提供 8086（中央处理器）稳定的工作频率，并且还是串行接口波特率选择的整倍数。8284将晶振基频三分频，得到8086（U15）要求的5MHz时钟信号（CLK）。此外，8284还将这个时钟信号进一步二分频，得到外围时钟信号（PCLK），它是除 8086 外系统所用的时钟信号。

8284提供两个与5MHz时钟信号（CLK）内部同步的输出控制信号：RDY（就绪）和RST（复位）。当8284输入信号$\overline{RES}$为低电平（系统第一次加电或按下系统复位SYSTEM RESET键）时，RST信号将TP-86A复位到初始状态。当从“等待状态发生器”（见2.3节）输入的 RDY1 有效（逻辑高电平）时，RDY输出有效（逻辑高电平），8086处于正常工作状态；否则，它将停在RDY 信号下降沿状态上，直到 RDY 恢复高电

平才继续正常工作。

短路开关W62是为克服外部设备启动或关闭造成的电路干扰设置的。在启动或关闭外部设备前，需先跨接W62或按着RESET键：启动或关闭后，再将W62拔去或放开 RESET键。跨接W62时，系统不给出任何显示，一旦不用后应立即拔掉。

短路开关W39、W40可选择 8086 工作于 2.5MHz（2.45MHz）和 5MHz（4.9MHz）两种工作频率（表2.1）。当跨接 W39 时，CPU 工作于2.5MHz 时钟频率，PCLK 信号代替 CLK信号，板上存储器和I/O运行时不用插入等待状态。这种工作频率一般是为系统组装调试而用。跨接 W40时，CPU工作于 5MHz 时钟频率。若 TP-86A 上安装的8086最高输入频率为 4MHz，则整个系统只能工作在2.5MHz时钟频率。（具有最高输入频率4MHz的8086，在器件顶部上印有“－4”的尾缀，即8086－4）。

表 2.1　　8086(CPU)工作频率选择

短路开关位置	CPU 工作频率
W39	2.5MHz(2.45MHz)
W40	5MHz(4.9MHz)

2.3 等待状态发生器电路

等待状态发生器电路主要由一块74LS164（U11）移位寄存器组成，它与辅助电路配合可往 8086 总线周期内插入等待状态周期，能使8086与慢速的外围 I/O 电路或存储器配合工作。当 8086工作在2.5MHz 时钟频率时，TP-86A 板上电路不需要插入等待状态（等待状态零，W31）；当8086工作在5MHz时钟频率时，要求插入一个等待状态（等待状态1，W32）。当系统运行要求多于 1 个等待状态时，可根据所需求的等待状态数，按表 2.2 跨接相应的短路开关。

表 2.2　　等待状态选择

短路开关位置	等待状态数	短路开关位置	等待状态数
W31*	0	W35	4
W32	1	W36	5
W33	2	W37	6
W34	3	W38	7

* 不跨接任何等待短路开关时，等效于零等待状态

通过等待状态发生器辅助电路，在每个读、写或中断周期后都要清零 74LS164等待状态发生器。当清除信号 CLR 输入变为无效（逻辑低电平）后，相继的下一个需要插入等待状态的读、写或中断周期开始时，使等待状态发生器处于允许状态。此时，8284 RDY1和 RDY 均无效（逻辑低电平），8086停在 RDY 信号下降沿状态上。处于允许状

态的等待状态发生器将一个“1”（逻辑高电平）在寄存器中移位，当移到所选择的等待状态数位置时，被跨接的短路开关输出变为逻辑高电平，使RDY1有效（逻辑高电平），从而使输出到8086的RDY信号有效（逻辑高电平），CPU继续正常工作。

当跨接短路开关W30时，只要是读、写或中断周期，均插入所选择的等待状态周期。当不跨接W30时，只有I/O操作及离板存储器操作，才插入所选择的等待状态周期。板上使用低速PROM或RAM时，则需要跨接W30。另外，EPROM编程写入要求插入等待状况，关于这一点见2.5节。

2.4 中央处理器8086

关于8086的运行、功能和指令系统等请参看有关资料，这里只说明与TP-86A使用有关的问题。

1．TP-86A中，8086用于最小系统方式（MN/$\overline{MX}$引脚置逻辑高电平）。

2．在TP-86A组装后，已将W41、W42、W43跨接了短路开关，禁止了INTR、TEST和HOLD到8086的输入。当一个外围电路连接到总线扩展逻辑，并要使用这些信号时，就要将相应信号的短路开关去掉。

3．当TP-86A复位（或第一次加电）时，8086执行FFFF0H单元的指令。

4．当INTR键按下时，使8086NMI（非屏蔽中断）输入信号为逻辑高电平，8086把当前的系统状态保存起来（在清除IF和TF标志后，把IP、CS和FL寄存器内容进栈），并按内存08H—0BH单元内容（中断矢量2）执行一条间接的长转移指令。08—0BH单元被监控程序在复位时进行初始化，放有监控程序的中断程序入口地址。

5．TP-86A没有使用可屏蔽中断（8086 INTR引脚置逻辑低电平），但通过总线扩展逻辑可由外部电路使用。为了使用可屏蔽中断，当$\overline{INTA}$有效（逻辑低电平）时，必须在数据总线上提供一个中断矢量指示字。当有一个以上中断源时，要提供中断优先权电路（如使用INTEL 8259A中断控制器）。

2.5 可编程只讀存儲器PROM

TP-86A具有64K字节的PROM区，板上有8个插座（U32—U39），使用2732(4K×8）EPROM芯片，共32K字节；另外32K字节可通过总线扩展逻辑加以扩充，也可在插座U32、U33上插上PROM扩充板在板上扩充。这时要将短路开关W44拔去（平时要跨接W44）。U32、U33为28脚低插拔力插座，其1—12脚和17—28脚构成24脚PROM插座，13—16脚只有当板上扩充PROM时才使用。

PROM片选译码由八中选一译码器74LS138（U22）完成，它给出了板上PROM区的4个片选信号和PROM扩充的4个片选信号。区址位A13—A15为输入信号，当M/$\overline{IO}$为逻辑高电平，且地址位A16—A19全为“1”时，$\overline{RD}$信号选通U22。A0地址位和$\overline{BHE}$信号直接接到EPROM芯片的$\overline{CE}$端来决定字节或字（双字节）的寻址。U22片选输出信号接到EPROM芯片的$\overline{OE}$端，每个片选输出接两块EPROM芯片。EPROM芯片本身寻址由

地址位 A1—A12 决定。其余4个 PROM 片选输出信号接到总线扩展插座 J2上，并且还接到插座U32、U33的 13—16 脚上。

表 2.3 和 2.4 分别给出板上 PROM 区和 RROM 扩充的使用情况。

表 2.3　　　　板上 PROM 区

地址范围	插座位置		备注
	偶地址	奇地址	
F8000H—F9FFFH	U32	U33	用户区,EPROM编程插座,PROM扩充插座
FA000H—FBFFFH	U34	U35	用户区
FC000H—FDFFFH	U36	U37	用户区
FE000H—FFFFFH	U38	U39	监控程序区(VA.1,VA.2)

表 2.4　　　　PROM 扩充

地址范围	符号标记	总线扩充	板上扩充	备注
F0000H—F1FFFH	$\overline{CSY0}$	J2—21	U32—13,U33—13	板上通过U32,U33扩充时，拔去W44，平时要跨接W44
F2000H—F3FFFH	$\overline{CSY1}$	J2—23	U32—14,U33—14	
F4000H—F5FFFH	$\overline{CSY2}$	J2—25	U32—15,U33—15	
F6000H—F7FFFH	$\overline{CSY3}$	J2—27	U32—16,U33—16	

在FE000H—FFFFFH（U33、U39）PROM 区中，驻存两个监控程序：键盘监控程序（VA.1）和串行监控程序（VA.2），各占4K字节。当选择跨接短路开关W56、W57时，可使两个监控程序都具有直接系统复位（或第一次加电）进入的功能。

表 2.5　　　　监控程序选择

短路开关位置	功能
W56	系统复位，进键盘监控程序
W57	系统复位，进串行监控程序

板上PROM区F8000H—F9FFEH（U32、U33），除可使用2732EPROM 芯片外，还可使用 2716EPROM芯片，这时其地址范围是 F9000—F9FFFH。2716 的使用要通过W46—W53短路开关的选择来实现。

TP-86A EPROM编程电路可通过插座U32、U33编程 2716/2732 两种EPROM芯片。2716/2732 EPROM 的编程选择也同PROM 区使用一样，见表2.6。编程短路开关 W54平时跨接，当需要对 EPROM 编程时，将 W54 拔掉，跨接 W55，此时编程指示灯亮，提示可使用键盘监控程序的编程命令进行 EPROM 编程。编程结束后，要拔掉 W55，跨接W54，，编程指示灯灭。

TP-86A EPROM 编程所需高压（+25V）是通过 TL497（U44）开关电源由+5V

变换而成的，所以不需要再为 EPROM 编程外加 25V 电源。

表 2.6　　在U32、U33上2716/2732使用选择

短路开关位置	芯片类型	备　　注
W46、W48、W50、W52	2716	地址范围F9000H—F9FFFH
V47、W49、W51、W53	2732	地址范围F8000H—F9FFFH

2726/2732EPORM芯片编程时的 50 毫秒等待是通过等待状态发生器电路插入的。当跨接W55后，$\overline{WR}$写信号也选通U22（PROM片选译码器）。EPROM编程时读（$\overline{RD}$）、写（$\overline{WR}$）周期都插入等待状态。当编程写入时，除插入所选择的等待状态外，还通过等待状态发生器插入由 8253(U7) 计数计时器零通道产生的 50 毫秒等待周期。注意：EPROM 编程时，一定要跨接等待状态选择短路开关W31—W38之一，否则 50 毫秒等待周期将不能插入。

2.6 随机存储器 RAM

TP-86A 具有 32K 字节的 RAM 区，板上有 8 个插座（U24—U31），使用6116（2K×8）静态RAM芯片，可得到16K字节；另外 16K字节可通过总线扩展逻辑加以扩充，也可在插座 U30、U31 上利用 RAM 扩充板在板上扩充，这时要将短路开关 W45 拔掉（平时要跨接W45）。U30、U31为28脚插座，其 1—12 脚和 17—28 脚构成 24 脚RAM插座，地址范围是 03000H—03FFFH；13—16 脚只有当板上扩充 RAM时才使用。

RAM 片选译码由两块八中选一译码器 74LS138（U20、U21）完成，它给出了板上RAM区的 8 个片选信号和RAM扩充的 8 个片选信号。地址位 A12、A13和 A14 作为输入信号，当M/$\overline{IO}$为逻辑高电平（即 $\overline{M/IO}$ 为低电平），且地址位 A15—A19 全为“0”时，A0地址位和$\overline{BHE}$信号分别选通 U20 和U21，共同决定字节或字（双字节）的寻址。U20为偶地址译码，它分别选择 U24、U26、U28 和 U30；U21 为奇地址译码，它分别选择U25、U27、U29 和 U31。U20 和 U21 共有奇偶 8 对 16个片选输出，除板上使用 4 对 8 个片选输出外，其余 4 对接到总线扩展插座 J2 上，并且还接到 U30、U31插座的13—16脚上。系统读（$\overline{RD}$）、写（$\overline{WR}$）信号直接接到 RAM 芯片的$\overline{OE}$、$\overline{WE}$端，以控制数据的流通方向。RAM芯片本身寻址由地址位A1—A11决定。

表2.7和2.8分别给出板上RAM区和RAM扩充的使用情况。

RAM区中 00000H—000FFH 的 256 字节保留给监控程序使用。图2.1给出监控程序在此区域中的实际分配情况。

表 2.7　　板上 RAM 区

地　址　范　围	插座位置		备　注
	偶地址	奇地址	
00000H—00FFFH	U24	U25	00H—FFH 为监控程序保留区，其余为用户区
01000H—01FFFH	U26	U27	用户区
02000H—02FFFH	U28	U29	用户区
03000H—03FFFH	U30	U31	用户区，RAM扩充插座

表 2.8　　RAM 扩充

地址范围	符号标记		总线扩充		板上扩充		备　注
	偶地址	奇地址	偶地址	奇地址	偶地址	奇地址	
04000H—04FFFH	$\overline{CSX0}$	$\overline{CSX1}$	J2-7	J2-17	U30-13	U31-13	板上通过 U30
05000H—05FFFH	$\overline{CSX2}$	$\overline{CSX3}$	J2-9	J2-19	U30-14	U31-14	U31 扩充时，
06000H—06FFFH	$\overline{CSX4}$	$\overline{CSX5}$	J2-5	J2-15	U30-15	U31-15	拔掉W45，平
07000H—07FFFH	$\overline{CSX6}$	$\overline{CSX7}$	J2-3	J2-13	U30-16	U31-16	时要跨接W45

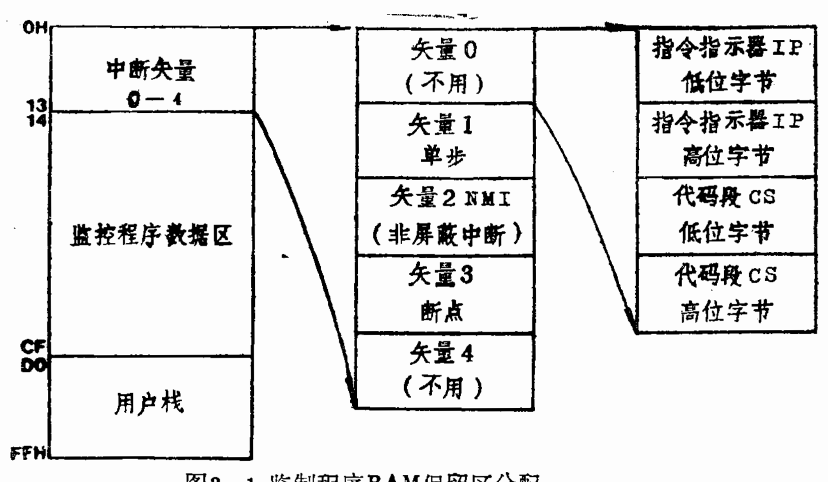


图2.1 监制程序RAM保留区分配

2.7 輸入輸出I/O譯碼电路

TP-86A I/O 译码由双路四中选一译码器 74LS155（U23）完成。地址位A3、A4作为输入信号、输出信号为8个端口选择信号，其中每个包含4个地址。当M/IO为逻辑低电平（$\overline{M/IO}$ 即为逻辑高电平）时，地址位A5—A15全为“1”时才使U23有效。地址位A_0和$\overline{BHE}$信号用来决定奇偶I/O端口地址。

表2.9和2.10分别给出I/O译码情况和I/O端口地址的功能分配。

表 2.9　　　　　　　　I/O译码

符号标记	端口地址	备注
$\overline{\text{XIOSEL0}}$	FFE0H, FFE2H, FFE4H, FFE6H	总线扩展I/O译码偶地址J2-1
$\overline{\text{XIOSEL1}}$	FFE1H, FFE3H, FFE5H, FFE7H	总线扩展I/O 译码奇地址，J2-11
$\overline{\text{KDSEL}}$	FFE8H, FFEAH, FFECH, FFEEH	8279键盘显示器选择
$\overline{\text{CTSEL}}$	FFE9H, FFEBH, FFEDH, FFEFH	8253计数计时器选择
$\overline{\text{USARTSEL}}$	FFF0H, FFF2H, FFF4H, FFF6H	8251串行接口选择
$\overline{\text{PPSSEL}}$	FFF1H, FFF3H, FFF5H, FFF7H	并行外设接口状态选择
$\overline{\text{LPSEL}}$	FFF8H, FFFAH, FFFCH, FFFEH	8255并行接口低位选择
$\overline{\text{HPSEL}}$	FFF9H, FFFBH, FFFDH, FFFFH	8255并行接口高位选择

表 2.10　　　　　　　　I/O 端口地址分配

端口地址	端口功能
0000H—FFDFH	未用
FFE0H	总线扩展 I/O留给用户 (FFE0H—E7)
E1	
E2	
E3	
E4	
E5	
E6	
E7	
E8	读/写 8279 显示 RAM 或读 8279 FIFO(先进先出栈)
E9	选择 8253 计数计时器通道0
EA	读 8279 状态或写 8279 命令
EB	选择 8253 计数计时器通道1
EC	保留
ED	选择 8253 计数计时器通道2
EE	保留
EF	选择 8253 控制寄存器
FFF0H	读/写 8251 串行口数据
F1	读并行外设接口状态
F2	读 8251 串行口状态或写 8251 串行口命令
F3	与 FFF1 相同
F4	保留
F5	与 FFF1 相同
F6	保留
F7	与 FFF_1相同
F8	读/写 8255 并行口低位A通道P2A
F9	读/写 8255 并行口高位A通道P1A
FA	读/写 8255 并行口低位B通道P2B
FB	读/写 8255 并行口高位B通道P1B
FC	读/写 8255 并行口低位C通道P2C
FD	读/写 8255 并行口高位C通道P1C
FE	写 8255 并行口低位 P2 命令
FFFFH	写 8255 并行口高位 P1 命令

2.8 键盘显示器电路

以 8279 键盘显示控制器为中心，组成了 TP-86A 的键盘显示逻辑。8279(U55) 完成对键进行消振、键盘矩阵编码和显示元件的刷新。

在 2.7 节中指出，8279 占用板上 I/O 地址空间的两个端口，并且由于 8279 与数据总线低 8 位(D0—D7)相接口，所以两个端口地址均为偶地址（FFE8H和FFEAH)。8279 端口的各种作用由A1地址位和$\overline{RD}$（读）、$\overline{WR}$（写）信号选定，见表2.11

表 2.11　　键盘显示 I/O端口

8279 输入			端口地址	端口功能
A_1	$\overline{RD}$	$\overline{WR}$		
0	0	1	FFE8H	读显示 RAM 或键盘 FIFO
0	1	0	FFE8H	写显示 RAM
1	0	1	FFEAH	读状态
1	1	0	FFEAH	写命令

由于 A2 地址位不参加I/O 译码，所以端口地址FFECH 和 FFEEH 分别与端口地址FFE8H和FFEAH 的译码一样，如表 2.9 中所示，这两个地址被保留起来。

键盘监控程序通过写命令端口，对8279作如下安排：

• 8 个 8 段数字显示，左移进入，编码扫描键盘—2 键锁定输出（键盘显示方式置命令字00H)

• 时钟倍乘因子 25，以提供 5 毫秒键盘显示扫描时间和 10 毫秒的消振时间程序时钟置命令字39H)。

由于指定了编码扫描为键盘工作方式，所以 SL0—SL3 输出为一个二进制计数。7445（BCD 到 10 线的译码器，U57）将这二进制计数变换成8个单线的输出（有两个输出未用），分别去接通 8 个显示元件。此外 SL0—SL2 的输出引到 74TS156（3 到 8 线译码器，U58）以提供给“键盘开关矩阵”“三行扫描”的输入信号，由开关矩阵的 8 个输出，即 8279 的输入 RL0—RL7，代表 8 个开关列，当一个键按下时，在扫描到这一行时，对应的列就被选通。8279利用这个被选通列的位置及它的行扫描值(SL0—SL2)去产生一个对应被按下键的代码。这个代码存入 FIFO（先进先出栈）中，CPU 通过读键盘 FIFO 的 I/O 端口可以取得这个代码。

8279 的 A0—A3 和 B0—B3形成 8 位并行输出，经过显示段驱动电路，每位分别驱动一个显示段。表 2.12 说明了 8 位并行输出与 8 段显示之间的相互关系(输出逻辑高电平对应显示段接通)

为了充分利用 8279 的键盘显示功能，TP-86A 提供了一个 34 线的 8279 扩展接口 J7。通过这个扩展接口可直接使用 8279 来完成新的键盘显示需要，连接一个与本机分离的键盘显示单元。但这时要取消板上相应电路，重新组织如编译码电路，键盘开关矩

阵以及显示驱动电路等。表2 .13 给出了 8279 键盘显示扩展接口 J7 的引线定义。

2 2.12　　显示段定义

	8279输出位	选 通 段	8279输出位	选 通 段
0	B0	0	A0	4
5 6 1	B1	1	A1	5
2	B2	2	A2	6
4 3 1 7	B3	3	A3	7

表 2.13　　8279 键盘显示扩展接口 J7

J7 引 脚	8279 信 号	J7 引 脚	8279 信 号
1	RL0	2	SL0
3	RL1	4	地
5	RL2	6	SL1
7	RL3	8	地
9	RL4	10	SL3
11	RL5	12	地
13	RL6	14	SL4
15	RL7	16	地
17	A0	18	地
19	A1	20	地
21	A2	22	地
23	A3	24	地
25	B0	26	地
27	B1	28	地
29	B2	30	地
31	B3	32	地
33	+5V	34	地

2.9 串行輸入輸出接口电路

TP-86A 的串行接口电路是建立在 8251A USART（通过同步异步接收发送器）基础上的，将它置于异步工作方式。8251A（U53）在板上 I/O 地址空间占用两个 I/O 端口。由于 8251A 数据线与数据总线的低位（D0—D7）相接口，所以两个端口为偶数地址（FFF0H 和 FFF2H），A1 地址位和$\overline{RD}$（读）、$\overline{WR}$（写）信号决定 8251A的端口功能。

由于 A2 地址位不参加 I/O译码，因此端口地址 FFF4H 和 FFF6H 分别与端口地址 FFF0H和 FFF2H 的译码一样，如表 2.9 中所示，这两个地址被保留起来。

表 2.14　　8251A I/O端口

8251A 输入			端口地址	端口功能
A1	RD	WR		
0	0	1	FFF0H	读 8251A 数据
0	1	0	FFF0H	写 8251A 数据
1	0	1	FFF2H	读 8251A 状态
1	1	0	FFF2H	写 8251A 命令

串行监控程序向 8251A 控制端口写入 CFH 作为工作方式控制命令，8251A 工作方式如下：

- 每字符 8 个数据位
- 禁止奇偶校验
- 2 个停止位
- 64×波特率倍乘因子异步方式

TP-86A 串行口有一个相关的波特率发生器，跨接 W21—W29 相应的短路开关，可选择 75—9600 波特范围内的 9 个波特率之一。由于 8251A 工作在 64×方式所以这些频率是对应波特率的64倍。表2.15定义了波特率的选择和对应波特率发生器的输出频率。

表 2.15　　串行口波特率选择

波特率	短路开关位置	输出频率
9600	W21	614.4 KHz
4800	W22	307.2 KHz
2400	W23	153.6 KHz
1200	W24	76.8 KHz
600	W25	38.4 KHz
300	W26	19.2 KHz
150	W27	9.6 KHz
110	W28	6.98 KHz
75	W29	4.8 KHz

波特率发生器由两块双四位二进制计数器 74LS393（U48、U52）构成，其输入为 PCLK/2 时钟信号(1228.8KHz)，它除提供以上可被选择的九个串行口波特率外，还提供了 300Hz，600Hz，1200Hz 和 2400Hz 四个计数计时时钟信号 CTCLK（见 2.11节）。此外，1200Hz 和 2400Hz 时钟信号还作为外存音频盒式磁带调制信号使用。

通过串行接口短路开关 W1—W20 的选择，可经 J6（DB-25S，标准的 25 线 EIA 接口）与电传打印机TTY（20mA电流环输出）或CRT终端（RS232 输出)相连。其中不论是 TTY 还是 CRT，又分为独立工作方式（与独立的 TTY、CRT 相连）和从属于 MDS

（微型机开发系统）工作方式（与 MDS 开发系统 TTY、CRT 相连）。波特率的选择要与所连接的 TTY 或 CRT 一致。当与 CRT 相连，工作于 300 波特率时，通过 CIN、COUT 插孔（盒式录音机转储、转录插孔）可使用盒式录音机作为外存设备。串行监控程序的 R 命令，[读带命令]和 W（WX）命令[写带命令]将音频调制的 8086（或 8080）十六进制文件转储于内存或转录于盒式录音磁带。

表 2.16 给出了串行接口插座 J6 的引脚定义，图 2.2 给出了 TP-86A 与 INTEL 微型机开发系统连接的几例。

TP-86A 键盘监控程序可使用音频盒式录音机作为外存设备，其磁带音频调制解调电路采用了“肯萨斯城标准”（简称 KC 标准），其规定如下：

· 逻辑高电平用 8 个周期的 2400Hz 频率脉冲表示。

· 逻辑低电平用 4 个周期的 1200Hz 频率脉冲表示。

表 2.16　　　　**串行接口插座 J6**

引　脚	短路开关 W1—W19 选择			
	TTY（W9—W18）	CRT（W1—W6）	MDS—TTY（W15—W20）	MDS—CRT（W3—W8）
1				
2		数据接收		数据发送
3		数据发送		数据接收
4				
5				
6				
7	地	地	地	地
8				
9				
10				
11				
12	数据接收		数据发送	
13	数据发送		数据接收	
14				
15				
16	输入机控制			
17				
18				
19				
20				
21	输入机控制回送			
22				
23				
24	接收数据回送			
25	发送数据回送			

• 数据记录格式采用异步通讯格式，每字符8个数据位，2个停止位。

• 文件内容记录以前，有30秒以上的高电平引导，文件内容记录完后，有5秒以上的高电平结尾。

当通过键盘监控程序命令使用盒式录音机作为外存设备时，要跨接串行接口波特率选择短路开关W26。CIN为转储插孔，COUT为转录插孔，二者均为3.5mm录音机插孔。工作于键盘监控磁带转储、转录功能的8251A必须选择300波特率（W26），否则不能正常工作。另外，键盘监控磁带接口具有简单实用的文件编码功能。

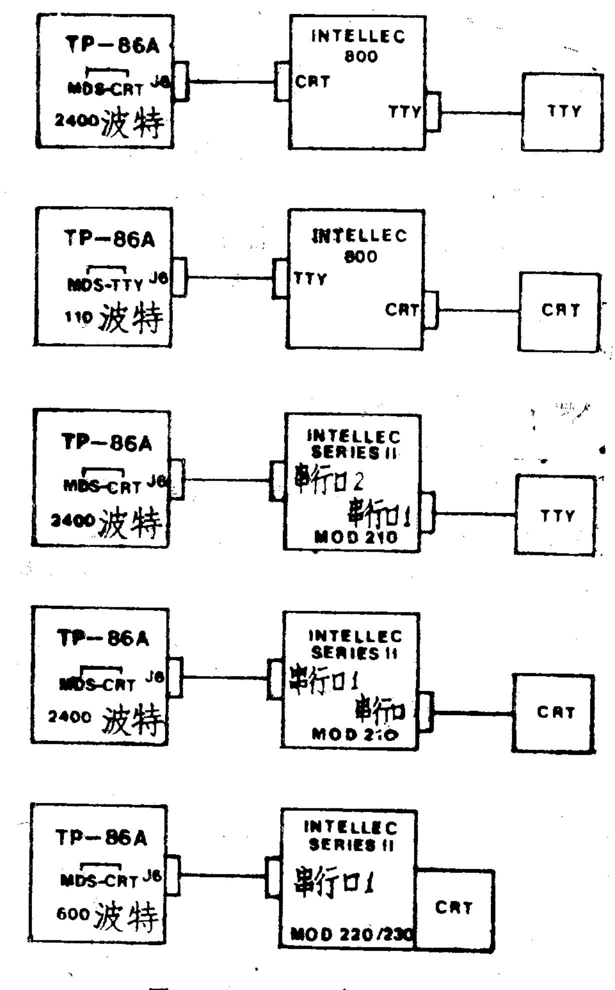


图 1.2 TP-86A 与 MDS 接口

工作于键盘监控磁带转储、转录的8251A置有工作方式控制字FFH其，工作方式如下：

• 64×波特率倍乘因子异步方式
• 每字符8个数据位
• 偶校验
• 2个停止位

表2.17给出串行口短路开关矩阵表

表 2.17 串行口短路开关矩阵

接口安排	短路开关位置	板面标记	接口插座
独立的CRT终端	W1—W6	[CRT]	J6
独立的电传打印机	W9—W18	[TTY]	J6
智能从属CRT终端	W3—W5W7W8W17	MDS-[CRT]	J6
智能从属电传打印机	W15—W20	MDS-[TTY]	J6

2.10 并行輸入輸出接口电路

TP-86A板上有两块并行接口芯片 8255A，它们共占用 8 个 I/O 端口地址，分别与数据总线的高八位和低八位相接口（U40，P2与D0—D7 相接口；U41，P1 与 D8—D15 相接口），因而除提供 8 位并行口电路外，还可以方便地提供 8 位以上直到 16 位的并行口电路。两块 8255A 共有 6 个 8 位独立口（P2A、P2B、P2C；P1A，P1B，P1C），48条引线均可由并行扩展接口 J3 引出使用。表 2.18 和表 2.19 分别给出 8255A 并行 I/O 端口功能和并行扩展接口定义。

TP-86A 可由 34 线并行外围设备接口 J4 配用并行外设（如可接μ-80行式打印机和WX4675XY绘图机）。J4并行外设接口使用了 8255A 的 P2B口 P2B0—P2B7 8 线作为数据输出，P2C 口的 P2C0 作为数据选通信号。所以当经并行外设接口 J4使用并行外设时，并行扩展接口 J3 中相应引脚就不能再被使用，并且对 8255A 编程也要注意不影响已在 J3 上连接运行的并行外设。表 2.20 给出了并行外围设备接口 J4的定义。

表 2.18　　8255 A I/O 端口

8255A（P_1和P_2）输入						端口地址	端口功能
HSEL (P1)	LSEL (P2)	A1	A2	$\overline{RD}$	$\overline{WR}$		
1	0	0	0	0	1	FFF8H	读 8255A P2A口数据
1	0	0	0	1	0	FFF8H	写 8255A P2A口数据
1	0	1	0	0	1	FFFAH	读 8255A P2B口数据
1	0	1	0	1	0	FFFAH	写 8255A P2B口数据
1	0	0	1	0	1	FFFCH	读 8255A P2C口数据
1	0	0	1	1	0	FFFCH	写 8255A P2C口数据
1	0	1	1	1	0	FFFEH	写 8255A P2口命令
0	1	0	0	0	1	FFF9H	读 8255A P1A口数据
0	1	0	0	1	0	FFF9H	写 8255A P1A口数据
0	1	1	0	0	1	FFFBH	读 8255A P1B口数据
0	1	1	0	1	0	FFFBH	写 8255A P1B口数据
0	1	0	1	0	1	FFFDH	读 8255A P1C口数据
0	1	0	1	1	0	FFFDH	写 8255A P1C口数据
0	1	1	1	1	0	FFFFH	写 8255A P1C口命令
0	0	0	0	0	1	FFF8H	读 8255A PA口数据(16位)
0	0	0	0	1	0	FFF8H	写 8255A PA口数据(16位)
0	0	1	0	0	1	FFFAH	读 8255A PB口数据(16位)
0	0	1	0	1	0	FFFAH	写 8255A PB口数据(16位)
0	0	0	1	0	1	FFFCH	读 8255A PC口数据(16位)
0	0	0	1	1	0	FFFCH	写 8255A PC口数据(16位)
0	0	1	1	1	0	FFFFH	写 8255AP1和 P2 命令(16位)

表 2.19　　并行扩展接口 J3

引　　脚	功　　能	引　　脚	功　　能
33	P2A0	36	P1A0
37	P2A1	40	P1A1
41	P2A2	44	P1A2
45	P2A3	48	P1A3
47	P2A4	50	P1A4
43	P2A5	46	P1A5
39	P2A6	42	P1A6
35	P2A7	38	P1A7
9	P2B0	16	P1B0
21	P2B1	12	P1B1
17	P2B2	8	P1B2
13	P2B3	4	P1B3
15	P2B4	6	P1B4
19	P2B5	10	P1B5
11	P2B6	14	P1B6
7	P2B7	18	P1B7
23	P2C0	26	P1C0
1	P2C1	24	P1C1
3	P2C2	22	P1C2
5	P2C3	20	P1C3
25	P2C4	28	P1C4
27	P2C5	30	P1C5
29	P2C6	32	P1C6
31	P2C4	34	P1C7

表 2.20　　并行外围设备接口 J4

引　　脚	功　　能	引　　脚	功　　能
1	数据选通输出 P2C0	2	地
3	数据输出 P2B0	4	地
5	数据输出 P2B1	6	地
7	数据输出 P2B2	8	地
9	数据输出 P2B3	10	地
11	数据输出 P2B4	12	地
13	数据输出 P2B5	14	地
15	数据输出 P2B6	16	地
17	数据输出 P2B7	18	
19		20	地
21	状态输入 PS0	22	地
23	状态输入 PS1	24	地
25	状态输入 PS2	26	
27	地	28	状态输入 PS3
29		30	
31	地	32	
33	地	34	

为了不对并行外设接口 J4 的使用提出更多的限制，并有利于保护 8255A 并行接口电路，并行外设的状态 I/O（PS）输入是通过 U46(74LS367A,三态缓冲器)进入数据总线 D8—D11 位的。由 I/O 译码得到的 $\overline{PPSSEL}$ 包括 4 个 I/O 端口地址(FFF1H,FFF3H,FFF5H，FFF7H），它与 $\overline{RD}$（读）信号选通输入并行外设接口状态。并行外设接口状态可由 I/O 端口地址 FFF1H 读入，其余三个地址 FFF3H、FFF5H 和 FFF7H 不能再做其它使用。表 2.21 给出了并行外设接口状态PS 与数据总线之间的接口关系。

表 2.21　　并行外设接口状态

PS	PS0	PS1	PS2	PS3
数据总线	D8	D9	D10	D11
J_4 引脚	21	23	25	28

2.11 計数計时电路

TP-86A 板上装有一块 8253 计数计时器电路(U_7)。它占有4个I/O个端口地址，与数据总线高位（D8—D15）相接。地址位 A1 和 A2 及 $\overline{RD}$（读）、$\overline{WR}$（写）信号共同决定8253端口功能。

表 2.22　　8253 I/O端口

8253 输入				端口地址	端口功能
A_1	A_2	$\overline{RD}$	$\overline{WR}$		
0	0	1	0	FFE9H	写 8253 通道 0 计数值
0	0	0	1	FFE9H	读 8253 通道 0 当前计数值
1	0	1	0	FFEBH	写 8253 通道 1 计数值
1	0	0	1	FFEBH	读 8253 通道 1 当前计数值
0	1	1	0	FFEDH	写 8253 通道 2 计数值
0	1	0	1	FFEDH	读 8253 通道 2 当前计数值
1	1	1	0	FFEFH	写 8253 命令

8253包含 3 个计数计时通道（通道 0 、通道 1 和通道 2 ）。每个通道分别有时钟信号输入（C），控制门信号（G），计数计时输出（O）。通道 0 已被板上 EPROM 编程占用，其余两个通道由总线扩展插座 J1 提供给外围电路使用。此外，还在 J1 的 43脚设有计数计时时钟信号 CTCLK,通过选择短路开关 W58—W61 可得到 300Hz，600Hz，1200Hz 和 2400Hz 时钟信号。

表 2.23　　J1-43 计数计时时钟信号选择

短路开关位置	W58	W59	W60	W61
时钟信号频率	300Hz	600Hz	1200Hz	2400Hz

表 2.24　　8253 计数计时通道总线扩展输出

J_1 引脚	功能	J_1 引脚	功能
31	通道 1 计数计时输出 O1	37	通道 2 控制门信号 G2
33	通道 1 控制门信号 G1	39	通道 2 计数计时输出 O2
35	通道 1 时钟输入信号 C1	41	通道 2 时钟输入信号 C2

2.12 离板譯碼电路

TP-86A 离板译码电路用来产生 $\overline{\text{OFF BOARD}}$ 离板信号。等待状态发生器利用这个信号强迫离板存储器操作插入所选的等待状态周期。总线扩展逻辑在离板 I/O 操作和离板存储器操作时，利用这个信号产生 $\overline{\text{BUFFER ON}}$ 信号，来向离板外围电路打开数据总线。表 2.25 给出了 TP-86A 离板译码的逻辑关系。

表 2.25　　离板译码的逻辑关系

M/$\overline{\text{IO}}$	地址范围	$\overline{\text{OFF BOARD}}$	备注
1	00000H—03FFFH	1	板上16K字节RAM区
1	04000H—07FFFH	1	16K字节RAM扩充区，拔去W45在板上扩充
1	04000H—07FFFH	0	16K字节RAM扩充区，跨接W45离板扩充
1	08000H—EFFFFH	0	离板存储区
1	F0000H—F7FFFH	1	32K字节PROM扩充区，拔去W44在板上扩充
1	F0000H—F7FFFH	0	32K字节PROM扩充区，跨接W44离板扩充
1	F8000H—FFFFFH	1	板上32K字节PROM区
0	0000H—FFDFH	0	离板I/O空间
0	FFE0H—FFE7H	0	离板I/O扩充
0	FFE8H—FFFFH	1	板上I/O空间

* $\overline{\text{OFF BOARD}}$ = 0　为离板有效

2.13 总綫扩展电路

TP-86A 的总线扩展电路提供与外围电路的接口，由总线扩展接口插座 J_1 和 J_2 与外围电路相连接。总线扩展逻辑包括四部分：控制信号双向驱动电路（一块 8286 双向驱动

器 U1 和一块 74LS244 三态锁存器 U9)，数据总线双向驱动电路（两块 8286 双向驱动器 U2 和 U3），地址锁存电路（三块 74LS373三态锁存器 U4、U5 和 U6)，以及一些立即扩充信号。

控制信号双向驱动电路确定5个双向控制信号（M/$\overline{IO}$、$\overline{RD}$、$\overline{WR}$、$\overline{DEN}$和DT/$\overline{R}$）的源信号（U1)。当板上 8086 将总线控制权交给外围“主”电路时，HLDA有效（逻辑低电平），使 U1 双向控制输入端（T）为逻辑低电平，接收来自扩展总线的相应控制信号。这时 HLDA 为逻辑高电平，U9关闭扩展总线，相应信号输出为高阻状态；相应地，当板上 8086 具有总线控制权时，控制信号由它发出，HLDA无效（逻辑高电平），经 U1 向扩展总线发送控制信号，同时 U9 开放，板上 8086 控制和状态信号（HLDA，ALE，$\overline{INTA}$，A16/S3，A17/S4 和 A18/S5）经 U9 可进入扩展总线。

表 2.26 总线扩展接口 J₁

J1功能	引脚	J1引脚	功能
1	GND	2	BD0
3	〃	4	BD1
5	〃	6	BD2
7	〃	8	BD3
9	〃	10	BD4
11	〃	12	BD5
13	〃	14	BD6
15	〃	16	BD7
17	〃	18	BD8
19	〃	20	BD9
21	〃	22	BD10
23	〃	24	BD11
25	〃	26	BD12
27	〃	28	BD13
29	〃	30	BD14
31	O1	32	BD15
33	G1	34	RESET OUT
35	C1	36	PCLK
37	G2	38	INTR
39	O2	40	$\overline{TEST}$
41	C2	42	HOLD
43	CTCLK	44	BHLDA
45	BS3	46	BDEN/
47	BS4	48	BDT/$\overline{R}$
49	BS5	50	BALE

表 2.27 总线扩展接口 J₂

J2引脚	功能	J2引脚	功能
1	$\overline{XIO\ SEL0}$	2	$\overline{BHE}$
3	$\overline{CSX6}$	4	A0
5	$\overline{CSX4}$	6	A1
7	$\overline{CSX0}$	8	A2
9	$\overline{CSX2}$	10	A3
11	$\overline{XIOSEL1}$	12	A4
13	$\overline{CSX7}$	14	A5
15	$\overline{CSX5}$	16	A6
17	$\overline{CSX1}$	18	A7
19	$\overline{CSX3}$	20	A8
21	$\overline{CSY0}$	22	A9
23	$\overline{CSY1}$	24	A10
25	$\overline{CSY2}$	26	A11
27	$\overline{CSY3}$	28	A12
29	GND	30	A13
31	GND	32	A14
33	GND	34	A15
35	GND	36	A16
37	GND	38	A17
39	GND	40	A18
41	GND	42	A19
43	GND	44	BM/IO
45	GND	46	BRD/
47	GND	48	BWR/
49	GND	50	BINTA/

当处于离板存储器和离板 I/O 操作以及中断响应周期时，数据总线双向驱动电路（U_2 和 U_3）被$\overline{\text{BUFFER ON}}$信号开放。$\overline{\text{BUFFER ON}}$信号只有当离板存储器和离板 I/O 操作（$\overline{\text{OFF BOARD}}$有效，逻辑低电平）或中断响应周期（$\overline{\text{INTA}}$有效，逻辑低电平），并且$\overline{\text{DEN}}$有效（指令周期的第 2 到第 4 周期，逻辑低电平）时才有效（逻辑低电平），接在 U2 和 U3 双向驱动控制输入端（T）的 $DT/\overline{R}$ 信号的状态决定数据总线的传输方向。

当板上 8086 控制总线时（HLDA 无效，逻辑低电平），扩展总线的地址线及 $\overline{\text{BHE}}$ 信号直接由地址锁存器（U_4、U_5和U_6）输出。当总线控制权交给外国电路时（HLDA 有效，高电平），地址锁存器输出为高阻状态，扩展总线的地址有效。

TP-86A 经总线扩展插座 J1 和 J2，还向外围电路提供了一些立即离板扩充可使用的信号，这些信号在前面各节已经详细叙述。它们包括 PROM 扩充片选择码信号（$\overline{\text{CSY0}}$—$\overline{\text{CSY3}}$），RAM 扩充片选译码信号（$\overline{\text{CSX0}}$—$\overline{\text{CSX7}}$）和 I/O 端口扩充译码信号（$\overline{\text{XIOSEL0}}$ 和 $\overline{\text{XSOSEL1}}$）。这些信号使用中注意的问题请参阅有关各节。另外还有两个计数计时通道的 6 个信号（C1，G1，O1，C2，G2，O2）和计数计时时钟信号（CTCLK）。

表2.26和2.27表分别给出总线扩展接口 J1 和 J2 的定义。

第三章　TP-86A键盘监控程序命令的使用

3.1 引　言

这章说明用户如何通过键盘监控程序与TP-86A进行相互作用及通迅。键盘监控程序固化在4K字节的二片2732EPROM中。

在本机中，当W56接通时：键盘监控程序起始地址在FF000H；串行监控程序起始地址在FE000H。而当W57接通时：键盘监控程序起始地址在FE000H；串行监控程序起始地址在FF000H。

一经接通电源或按下“SYSTEM RESET”(系统复位)键，TP-86A就从FF000H为起始的程序开始执行。所以当W56接通时，TP-86A就会自动进入键盘监控程序，而当W57接通时，TP-86A就会自动进入串行监控程序。

在键盘控制下，可以由键盘发出一条GO命令转到串行监控程序起始地址（FE000H或FF000H分别对应接通W56或W57），于是就在串行监控程序控制之下了。

同样，在串行监控程序控制下，也可以由外围控制台发出一条GO命令转到键盘监控程序起始地址（FE000H或FF000H分别对应接通W57或W56），这样就在键盘监控程序控制之下了。

如在串行监控程序控制下，一经接通电源或“SYSTEM RESET”键被按下，则键盘显示器上就会显示出：

8　6	A.　2

由于本章主要介绍键盘监控程序，所以有关串行监控程序的细节请参看第四章。

在使用键盘控制时，首先要接通W56。这时只要接通电源或按下“SYSTEM RESET”键，监控程序就被初始化，用户就可以通过键盘和显示完成下列操作：

- 检测和修改8086微处理器内的寄存器内容。
- 检测和修改存储器单元内容。
- 输入和开始执行用户的程序或子程序。
- 通过监控程序的单步和断点功能来鉴定用户程序的执行（调试）。
- 将所选的存储器块由一个单元传送到另一个单元中去。
- 往I/O端口写数据或从I/O端口读数据。

3.2 键　盘

在键盘监控程序控制下，用户可以通过按下键盘上的键来输入命令和数据（通过显示来进行用户和监控程序之间的通讯）。如图3.1，键盘分为两个逻辑组：右边16个为十

六进制数字键，左边为 8 个功能键。

16个十六进制数字键中多数是复合功能键，其功能符号印在键上，即在十六进制数字下面的英文字母是监控命令和 8086 寄存器名的首字母缩写（斜线左边的缩写 名 为监控命令名，斜线右边为 8086 寄存器名）。无论何时，一个十六进制键的功能总是依赖于监控程序的当前状态，以及期望输入的是什么监控程序。指 3.1 中给出了这些十六进制键的功能，表 3.2 中给出 8 个功能键的功能。

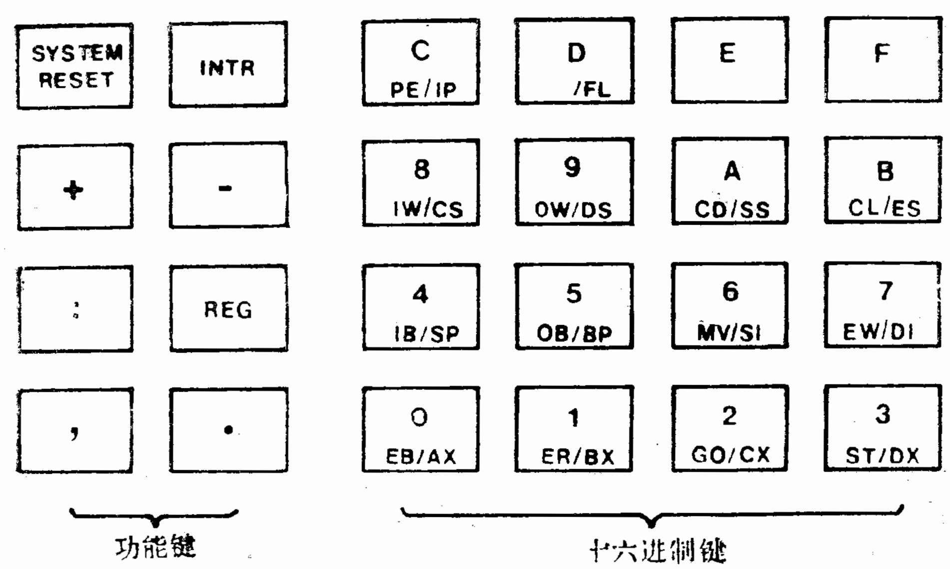


图 3.1 键盘安排

表 3.1 十六进制键上所印字符的说明

十六进制键	命令		寄存器	
	缩写符	名称	缩写符	名称
0 EB/AX	EB	Examine Byte	AX	Accumulator
1 ER/BX	ER	Examine Regester	BX	BASE
2 GO/CX	GO	GO	CX	Count
3 ST/DX	ST	(Single) Step	DX	Data
4 IB/SP	IB	Input Byte	SP	Stack Pointer
5 OB/BP	OB	Output Bvte	BP	Base Pointer
6 MV/SI	MV	Move	SI	Source Index

7 EW/DI	EW	Examine Word	DI	Destinatin Index
8 IW/CS	IW	Input Wore	CS	Code Segment
9 OW/DS	OW	Ontput Word	DS	Data Segment
A CD/SS	CD	Cassette DUMP	SS	Stack Segment
B CL/ES	CL	Cassette LOAD	ES	Extra Segment
C PE/IP	PE	EPROM Programming	IP	Instrction Pointer
D /FL	none	N/A	FL	Flag
E	none	N/A	none	N/A
F	none	N/A	none	N/A

表 3.2　　功 能 键 的 操 作

功能键	操作
SYSTEM RESET	SYSTEM RESET（系统复位）键允许用户终止任何当前的活动，并将 TP-86A 返回到初始化状态。当按下此键时，显示 TP-86A 的提示符，并使控制程序就绪，等待用户输入命令。
INTR	INTR（中断）键用来产生一个立即的、非屏蔽类型 2 的中断（NMI）。在加电或系统复位时，NMI的中断矢量就被初始化，以指向监控程序中的一个例行程序，这个例行程序将保护所有 8086 的寄存器，控制返回到监控程序并等待用户输入命令。
+	+（正）键允许用户作两个十六进制数的加法。这个功能键允许用户容易地计算相对于一个基地址的地址，从而简化了相对寻址的方法。
−	−（负）键允许用户从一个十六进制数中减去一个十六进制数。
:	:（冒号）键用来将要输入的地址分成两部分：段值和偏移量。还用来在 EB/EW 命令中作地址减量（−1/−2）。
REG	RED 键允许用户使用任意一个 8086 寄存器中的内容作为一个地址或数据项。
,	,（逗号）键用来分隔盘项目，并用来将地址段增量，以指向相邻的下一个存储单元。
.	·（句号）键是命令的终止符。当按下此键时，当前的命令就被执行。注意：使用 GO 命令时，按下此键就开始执行指定地址处的程序。

3.3 显　示

TP-86 A 通过8个数据显示器与用户进行通讯。依赖于监控程序的当前状态，被显示的信息将是：

- 一个寄存器或存储单元的当前内容
- 一个十六进制键输入的响应
- 监控程序的“提示符”
- 一个信息或状态信息

8个显示器按四个字符被分成两组，左边的一组为“地址段”，右边的一组为“数据段”。所有的显示都用十六进制给出。

3.4 总操作过程

在使用键盘监控程序时，TP-86 A 是通过“提示符”来要求输入命令的。只要监控程序要求输入命令时，在“地址段”的最左边的显示器上会出现“小横”。当“小横”出现时就意味着命令输入，可按任意一个十六进制的命令键（0—C键）。当命令键按下时“小横”就消失了，并在地址段中最后一个显示器上显示出小数点。这就意味着下面要接着输地址。

注意：依赖于不同的命令，在地址段或数据段内可能会显示出字符，从这点开始。由实际输入的命令确定监控程序的操作。命令格式及操作请参看3 6节到3.15节。

只要一加电或按下“SYSTEM RESET”键时，监控程序就被初终化了，并显示出监控程序的提示符（在显示器的地址段的后两位出现“86”，数据段后两位上显示出监控程序的版本号A.1），并同时在地址段的最高位显示器上出现监控程序提示符“小横”。当初始化完成时，监控程序就将8086的寄存器置成如表3.3所示的值。

表 3.3　寄存器的初始化

寄　存　器	值
CS　（代码段）	0H
DS　（数据段）	0H
ES　（附加段）	0H
SS　（堆栈段）	0H
IP　（指令指示器）	0H
FL　（标志）	0H
SR　（堆栈指示器）	0100H

注意：上表及本手册的其余部分中，字母“H”用来表示十六进制数。

图 3.2 示出了监控程序的存储器保留区和用户的栈区，（RAM中第一个用户可以用的存储单元是 0100H）。

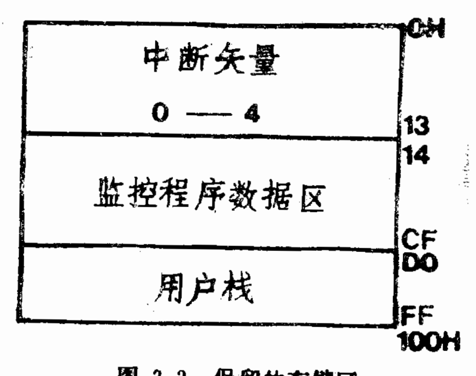


图 3.2 保留的存储区

只要 TP-86A 加电或按下“系统复位”键时，监控程序立即终止它当前的操作并转到它的初始化程序去。这个程序对中断矢量作如下的初始化：

中断 1：单步：使用单步命令

中断 2：NMI（非屏蔽中断）：监控程序 INTR 键

中断 3：断点：使用 GO 命令

当单步、NMI 或断点中断的结果重入监控程序将8086寄存器的内容保存进用户栈时，并且在输入命令之前相继地将栈中的内容弹出到寄存器。SP 寄存器的初值为 0100H（栈底），为用户栈保留着用来在上述任一中不产生时保存寄存器的内容。

注意：在下面的命令说明中，监控程序总是用一个 16 位的段地址值和一个 16 位的偏移地址值来算出一个20位的物理存储器地址。如果只输入一个地址(必须略去冒号:)，监控程序把它解释成一个偏移地址段，并且段地址默认为代码段寄存器（CS）的当前内容。CS寄存器内容和输入的偏移地址结合起来便产生一 20 位的物理存储器地址，如图3.3所示。

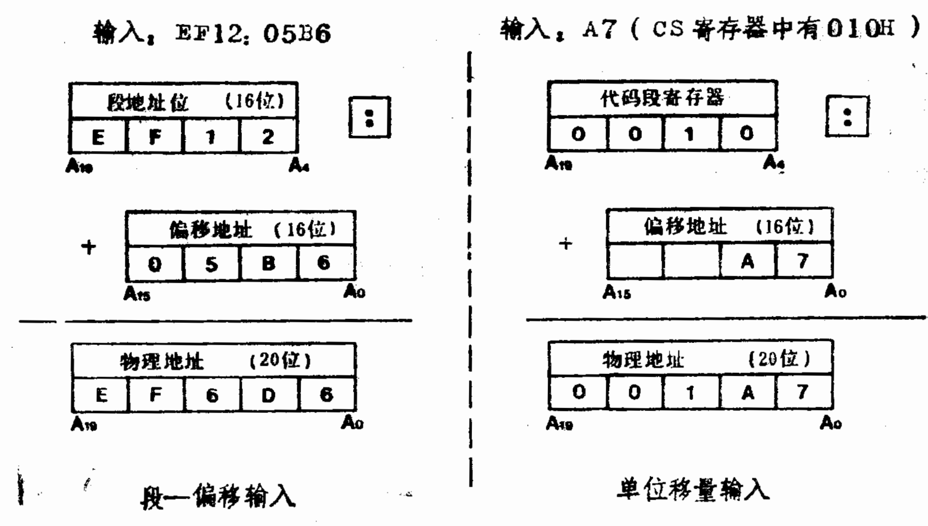


图 3. 3 物理地址的计算

3.5 监控程序命令

键盘监控程序可以执行如表 3.4 中列出的 13 个命令，本章的以后各节给出这些命令的详细说明。在命令表及说明中所用的命令格试如下：

×—指出一个键盘键。

[A]—指出“A”是任选的。

[A]*—指出“A”是任选的，并可以是一个或多个。

[B] —指出“B”是变量。

表 3.4　　　　监控程序命令汇总

命　　令	功　能　及　格　式	注　　释
Examine Byte 检测字节	显示/修改存储器字节单元 EB (addr) , [[(data)] , / :]* . ;	, 地址加 1 : 址址减 1
Examine Word 检测字	显示/修改存储器字单元 EW (addr) , [[(data)] , / :]* . ;	, 地址加 2 : 地址减 2
Examine Register 检测寄存器	显示/修改 8086 寄存器内容 ER (reg key)[[(data)] ,]* [.]	
Input Byte 输入字节	显示输入端口的数据字节 IB (port addr) , [,]* .	
Input Word 输入字	显示在输入端口的数据字 IW (port addr) , [,]* .	
Output Byte 输出字节	输出数据字节到输出端口 OB (port addr) , (data) [, (data)]* .	
Output Word 输出字	输出数据字到输出端口 OW (port addr) , (data) [, (data)]* .	
GO 转移	将控制从监控程序转给用户程序 GO [(addr)] [, (break point addr)] .	
MV 传送	在存储器内部传送数据块 MV (start addr) , (end addr) , (destination addr) .	
ST 单步	执行单条用户程序 ST [(Start addr)] , [[(Start addr)] ,]* .	
CD 转录	将内存 RAM 的信息转录在录音机磁带上 CD [(文件名)] , [(首地址)] , [(末地址)] .	
CL 转储	将录音机磁带上的信息输入到内存 RAM 中 CL [(文件名)] , [(内存地址)] .	
PE EPROM 写入	将 RAM 中的数据写入到 EPROM 中去 PE [(编程类型)] , [(源文件首地址)] , [(源文件末地址)], [(EPROM相对地址)]。 .	

3.6命　令EB（检测字节）和EW（检测字）

功能：检测字节EB和检测字EW命令用来检测所指定的存储器单元的内容。若需要修改内存单元（即RAM中的一个单元），则可用另外的内容去修改它。

格式：

EB(addr) , [[(data)] ,]* . 地址加1
EB(addr) , [[(data)] :]* . 地址减1
EW(addr) , [[(data)] ,]* . 地址加2
EW(addr) , [[(data)] :]* . 地址减2

注：addr——地址
data——数据

操作过程：

当命令提示符(－)显示出后，为使用这两个命令中的一个可按EB键（检测字节）或EW（检测字）键。当按下上述两键中的任意一个时，地址段右边的小数点就亮了（其余的显示器为空白），这就表示从键盘的输入应直接到地址段。从键盘上输入要检测的字节或字的存储器地址时，首先输入最高位的字符。注意，所有存储器地址由段值和偏移值两部分组成，当没有指定段值时，缺省的段值是代码段寄存器CS中的当前值；当指定了段值时，第一个地址项是段值，输入的冒号是分隔符，输入的第二个地址是偏移值。一个地址段的输入限制为4个字符。如果输入多于4个字符，那么，只有输入的最后4个字符是有效的。输入地址后，按下逗号“，”键，这时地址所指定的单元中的内容（数据字节或字）就显示在数据段的显示器上，并且在数据段的右边出现一个小数点，这就指示可再用十六进制键输入数据并送到数据段。

注意：当使用测试命令EW时，存储单元的字节将出现在数据段最右边的两个显示器上，并且下一个相郊的存储单元的字节内容出现在数据段最高的两个显示器上。

如果只是要检查一下地址指定的存储单元的内容，可按下句号“.”键，以结束这条命令。

如果还要检查相郊的下一个单元（地址比当前地址大的）相邻存储单元的内容（检测字），那么就按逗号“，”键。如果在按完“，”键后，用户还想检查相邻往上（地址比当前地址要小）的内存单元的内容，则可按“:”键。对于检查字节命令EB地址就会减1，并显出该单元的内容。对于检查字命令EW地址就会减2，并显示该地址单元及下一相郊单元两单元的内容。地址减量的好处是便于修正已输入的错误数据。若要修正地址指定的存储单元的内容，就还要过十六进制键输入新的数据。

注意：在检测字节时；数据段限制为两个字符；在检测字时，数据段限制为四个字符。如果输入了比限制更多的字符，那么只有当前显示出来的字符才是有效的。

在没有按下“·”键、“，”键或“:”键之前，不执行用显示的数据去修改存储单元中的老数据。只有按下“·”键、“，”键或“:”键时，才修改数据。如果按下的是“·”键，那么命令就结束了，并在地址段上显示出等待输入下一个命令

的提示符。如果按下的是"，"键，那么就显示出下一个相邻单元（地址比原来的大）的"偏移地址"和数据内容。如果按下的是"："键，那么就显示出上一个（地址比原来的小）的相邻单元的"偏移地址"。

出错条件： 试图去修正一个"不存在"的或只读（如ROM或PROM）存储单元。注意：在没有按下"，"键或"•"键或"："键时，是不能显示出错的。当产生错误时，在地址段中显示出提示符"—"和出错等"Err"。

例 3.1 检查相对于CS寄存器的一系列存储器字节单元（地址增量）的内容。

按键	地址段显示	数据段显示	注释
SYSTEM RESET	-　8　6	A.　1	系统复位
0 EB/AX	.		检测字节命令EB
1 ER/BX	1.		要检测第一个存储单元
E	1　E.		要检测第一个存储单元
，	1　E	×　×.	存储器数据内容
，	1　F	×　×.	下一个存储单元地址和数据
，	2　0.	×　×.	下一个存储单元地址和数据
，	2　1.	×　×.	下一个存储单元地址和数据
，	2　2.	×　×.	下一个存储单元地址和数据
.	-		命令结束／提示符

例 3.2 显示相对于CS寄存器一系列存储单元（地址减量）的内容：

按键	地址段显示	数据段显示	注释
SYSTEM RESET	-　8　6	A.　1	系统复位
0 EB/AX	.		检测字节命令EB
1 ER/BX	1.		要检测的第一个存储单元
E	1　E.		要检测的第一个存储单元
，	1　E	×　×.	存储器数据内容
：	1　D	×　×.	前一个存储单元地址和数据

:	1 C	× × .	前一个存储单元地址和数据
:	1 B	× × .	一个存储单元地址和数
.	-		命令结束／提示符

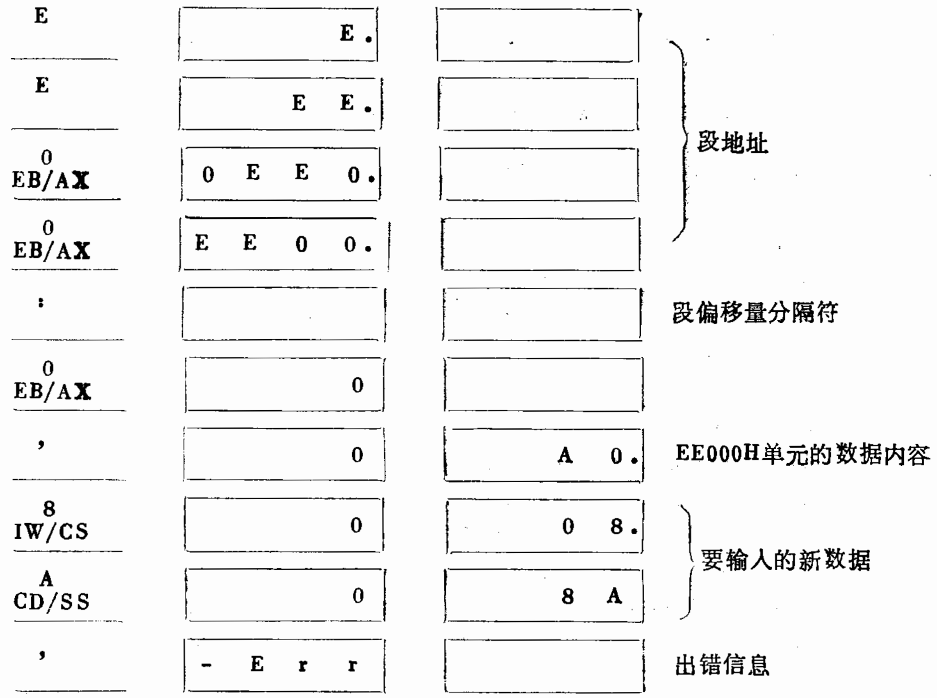
例 3.3 检测和修改相对于 DS 寄存器的存储单元。

按　键	地址段显示	数据段显示	注　　释
SYSTLM RESET	- 8 6	A. 1	系统复位
7 EW/DI	.		检测字命令EW
REG	r .		寄存器输入
8 OW/DS	0 .		DS 寄存器
:	0 .		段／偏移量分隔符
1 ER/BX	1 .		偏移地址
F	1 F .	.	偏移地址
,	1 F .	× × × × .	存储器数据内容
8 IW/CS	1 F	0 0 0 8 .	要输入的新数据
0 EB/AX	1 E	0 0 8 0 .	要输入的新数据
E	1 F	0 8 0 E .	要输入的新数据
D /FL	1 F	8 0 E D .	要输入的新数据
.	-		修正数据，命令束结／提示符

为了检查修改数据是否成功，按 EW 键并输入存储器地址（DS：1FH）；按“，”键，并注意到“80ED”显示在数据段内。

例 3.4 试图修改 PROM

按　键	地址段显示	数据段显示	注　　释
SYSTEM RESET	- 8 6	A. 1	系统复位
0 EB/AX	.		检测字节命令EB



由于只读存储器是不能修改的，只能读出，所以出现错误信息。可再重复打出EB命令系列看出EE000H单元中的A0H内容没有被修改过来。

3.7 命令ER（检测寄存器）

功能：检测寄存器命令ER用来检查和修改8086的寄存器内容。

格式：

ER (rge key)[[(data)] ,]*[·]

注：reg key ——寄存器键

data——数据

操作过程：为了测试一个寄存器的内容，当提示符出现后，按ER键，地址段的最右边出现一个小数点。和检测字节命令不同，后继的输入的十六进制键被解释成寄存器名（键面上斜线右边的缩写名），而不是它的十六进制值。当十六进制键按下后，寄存器的缩写符在地址段中显示出来，寄存器中16位的内容在数据段中显示出来，并且最右边的小数点也亮了。

表3.5中给出了8086寄存器名，十六进制键盘缩写名和显示的缩写符。

表 3.5　寄　存　器

寄存器名		键面缩写名	显示缩写名
Accumulator	累加器	AX	A
Base	基地址寄存器	BX	b
Count	计数寄存器	CX	c
Data	数据寄存器	DX	d
Stack Pointer	栈指针	SP	SP
Base Pointer	基指针	BP	bP
Source Index	源变址	SI	SI
Destination Index	目的变址	DI	dI
Code Segment	代码段寄存器	CS	CS
Data Segment	数据段寄存器	DS	dS
Stack Segment	堆栈段寄存器	SS	SS
Extra Segment	附加段寄存器	ES	ES
Instruction Pointer	指令指针	IP	IP
Flag	标志寄存器	FL	FL

当寄存器内容显示出来后(即当数据段最右边的小数点亮了以后)，寄存器的内容就能被修改了。从十六进制键上输入的数值在数据段显示器上得到“响应”，这时再按下“·”键或“，”键，就用显示出的数据去修改寄存器中原来的数值。若按下“·”键，命令被终止并显示命令提示符。若按下“，”键，下一个寄存器的缩写名和内容就被显示出来，下一个被打开的寄存器是按表 3.5 中的顺序进行的。

注意：顺序是不能循环的。若按下“，”键显示的是 FL 寄存器时，就终止测试寄存器命令，并返回到命令的提示符。

例 3.5　检查和修改一个寄存器。

按键	地址段显示	数据段显示	注释
SYSTEM RESET	-　8　6	A.　1	系统复位
1 ER/BX	.		检测寄存器命令ER
A CD/SS	S　S	0　0　0　0.	堆栈段寄存器内容
E	S　S	0　0　0　E.	新寄存器内容
F	S　S	0　0　E　F.	新寄存器内容
·	-		寄存器被修改，命令结束/提示符

例 3.6 检查一串寄存器。

按 键	地址段显示	数据段显示	注 释
SYSTEM RESET	- 8 6	A. 1	系统复位
1 ER/BX	.		检测寄存器命令ER
0 EB/AX	A X	0 0 0 0.	AX 寄存器内容
,	B X	0 0 0 0.	BX 寄存器内容
,	C X	0 0 0 0.	CX 寄存器内容
,	D X	0 0 0 0.	DX 寄存器内容
,	S P	0 0 0 0.	堆栈寄存器内容
.	-		命令结束／提示符

3.8 命令 IB（输入字节）和 IW（输入字）

功能： 输入字节 IB 和输入字 IW 命令用来从一个输入端口输入（接收）一个 8 位的字节或 16 位的字。

格式：

IB (port addr) , [,]* .

IW (port addr) , [.]* .

注：port addr——端口地址

操作过程： 为了使用输入字节或输入字命令，当输入命令的提示符出现时，按下相应的十六进制键。在 IB 或 IW 键按下后，地址段右边的小数点就亮了，以表示要求输入“端口地址”，要使用十六进制键来输入要读入端口的地址。注意：由于 I/O 端口编址最多为 64K（最大地址为FFFFH），因而对于端口地址不允许用段值。

输入端口地址后，按“，”键，这时地址指定的端口中的输入字节或字在数据段上显示出来；再次按“，”键，就会修正数据段的显示，即显示地址指定的输入端口的当前数据字节或字。按“.”键，结束命令并出现输入命令的提示符。

TP-86A 包括有两个 8255A的并行I/O 端口电路，可用输入字节和输入字命令从外部设备输入数据。这两个端口用 P1 和 P2 来指定，每个电路由三个端口 A、B、C组成，如图 3.4 所示。当字节操作时，每个端口独立地工作；当字操作时，一对端口（如P1A 和 P2A）共同建立一个 16 位字长的数据字，P2 的端口对应低 8 位的字节。

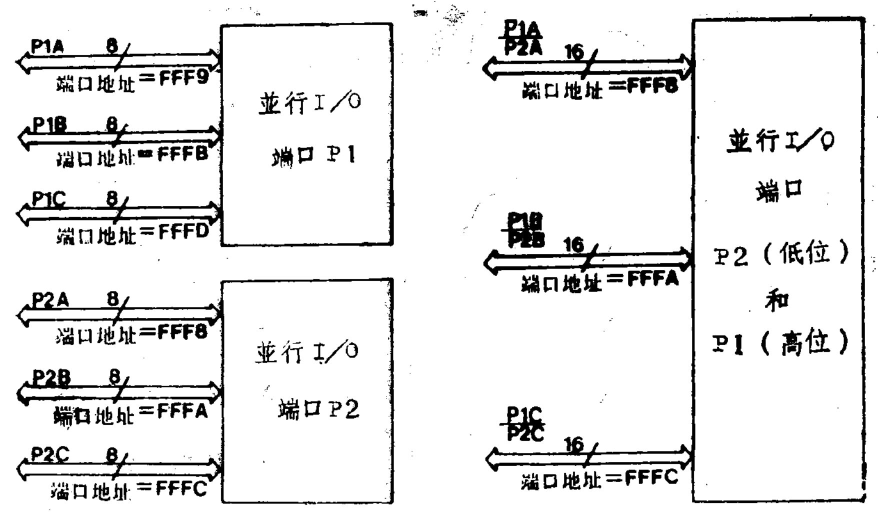


图 3.4 并行端口結构

表 3.6 定义各个端口地址。注意：当字操作时，输入的是低位（P2）端口地址（对应的高位地址将自动寻址）。

并行 I/O 端口电路偏移用于加电或按下系统复位键时的输入。如果电路事先已编程用于输出，那么按下“SYSTEM RESET”键时，(在按命令键以前，可参考表 3.7) 输出合适的字节或字到电路的控制端口以便将端口编程用于输入。

表 3.6 并行 I/O 端口地址

端口	地址
P2A	FFF8
P1A	FFF9
P2B	FFFA
P1B	FFFB
P2C	FFFC
P1C	FFFD

例 3.7 从端口 0EEH* 输入单字节。

按键	地址段显示	数据段显示	注释
SYSTEM RESET	- 8 6	A. 1	系统复位
4 IB/SP	.		输入字节命令IB
E	E.		端口地址
E	E E.		端口地址
,	E E	× ×	输入数据字节
.	-		命令结束／提示符

* 在 TP-86A 上没有提供 0EEH 端口

例 3.8 从并行I/O端口1A和2A输入多个字。

按键	地址段显示	数据段显示	注释
SYSTEM RESET	- 8 6	A. 1	系统复位(将端口初始化为输入)
8 IW/CS	.		输入字命令 IW
F	F.		端口 A 地址 FFF8H
E	F F.		
F	F F F.		
8	F F F 8.		
,	F F F 8	× × × ×	从端口输入数据字
,	F F F 8	× × × ×	再输入数据字
,	F F F 8	× × × ×	再输入数据字
.	-		命令结束／提示符

3.9命令OB（输出字节）和OW（输出字）

功能：输出字节 OB 和输出字 OW 命令用来把一个字或字节送到一个输 出端口。

格式：

OB (prot addr) , (data) [, (data)]* .

OW (port addr) , (data) [, (data)]* .

注：prot addr ——端口地址

data ——数据

操作过程：当输入命令提示符出现时，就可以按下所需要的这两个命令中的任何一个。在按下 OB 或 OW 命令键后，地址段右边的小数点就亮了，表示要求输入一个端口地址。和输入字节及输入字命令一样，I/O地址最多 64K 个，不允许有段地址。当输入端口地址后，按“，”键，数据段右边的小数点亮了，表示现在可以输入要输出的数据字节和字了，要使用十六进制数据键输入到输出的字节或字。在数据 输入以 后，按“.”键，就将要输出的字节或字送到输出端口并结束这 个命令。若还有 数 据 要输出，则按“，”键。正如前面已提到过，能用输出字节和输出字 命令来 对 8255A 并 行 I/O 端口进行编程，使它们能用于输入和输出以及输出数据到单个的端口。在数据能输出到有关的端口中去之前，当加电或系统复位时，I/O 端口编程为输出，首先送到 控 制 端口一个控制

字，使之用于输出。(用送往电路的控制端口，输出合适的数据字节或字来达到此目的)表 3.7 给出了控制端口的地址及要输出到控制端口去的有关数据字节或字。

表 3.7 控制端口地址

端口号	端口地址	数据字节或字	
		输入	输出
P2	FFFEH	9BH	80H
P1	FFFFH	9BH	80H
P2/P1	FFFEH	9B9BH	8080H

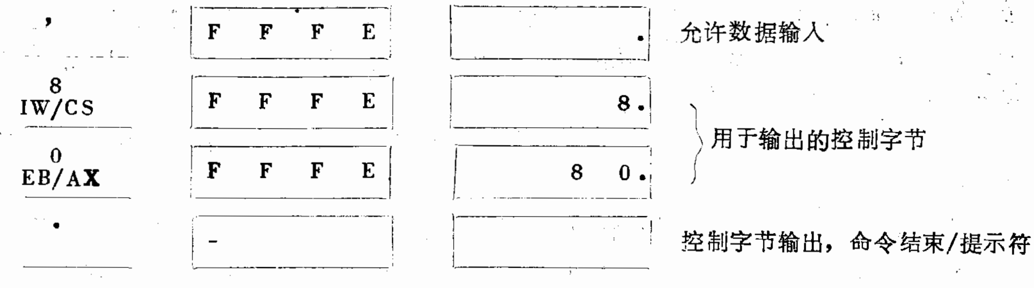
例 3.9 将 DI 寄存器内容输出到输出端口 0D6H*。

按键	地址段显示	数据段显示	注释
SYSTEM RESET	- 8 6	A. 1	系统复位
9 OW/DS			输出字命令 OW
D /PL	D.		输出端口地址
6 MV/SI	D 6.		
,	D 6.	.	允许数据输入
REG	D 6	r	寄存器输入
9 EW/DI	D 6	× × × ×.	DI 寄存器内容
.	-		数据输出，命令结束／提示符

* 在 TP-86A 中没有提供端口 0D6H

例 3.10 将 P2端口编程为输出端口。

按键	地址段显示	数据段显示	注释
SYSTEM RESET	- 8 6	A. 1	系统复位(将端口初始化为输入)
5 OB/BP	.		输出字节命令
F	F.		P2 控制端口地址
F	F F.		
F	F F F.		
E	F F F E.		



3.10 命令GO（轉移）

功能：GO 命令用来将 TP-86A 的控制由键盘监控程序转到存储器中的用户程序。

格式

GO [(addr)] [, (break point addr)] .

注：addr——地址

break point addr——断点地址

操作过程：为了使用 GO 命令，当出现输入命令提示符时，按下 GO 键。当按下该键时，在地址段上就显示出当前 IP（程序指针）的内容，同时 IP 指示的存储单元内容显示在数据段上，这时地址段的小数点亮了，表示能输入一个起动地址了。如果要求输入起动地址的话，从键盘上输入此地址。（当输入一个地址时，数据段的显示是空的）。为使程序开始执行（在当前指令开始执行或从改变了程序地址处开始执行），要按下"."键。当按下该键时，在将控制转给用户程序之前，监控程序在地址段的最高位显示一个"E"。

为了说明 GO 命令的操作，可将下面的实例程序用检查字节命令 EB 输入到存储器中去。这个程序模拟一个骰子（小方块）游戏，你可以往其中投一个骰子，在程序输入进去以后，并在按下 GO 命令使控制转向该程序后，按键盘上任意一个键（注意：不能按 SYSTEM RESET 或 INTR），就可启动这个骰子滚动；再按任注键，就可以停止这个骰子并显示它的值（在地址段最左边显示出 1—6 之间的数字）。

程序实例

存储单元	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0100	8C	C9	8E	D9	BA	EA	FF	B0	D3	EE	BA	EA	FF	EC	24	0F
0110	74	FB	E8	28	00	BB	00	00	43	80	FB	07	74	F7	8B	FB
0120	8A	4D	47	90	BA	EA	FF	B0	87	EE	BA	E8	FF	8A	C1	EE
0130	BA	EA	FF	EC	24	0F	74	E0	E8	02	00	EB	CD	BA	EA	FF
0140	B0	40	FE	BA	E8	FF	EC	C3	06	5B	4F	66	6D	7D		

实例程序占用 4E 个十六进制单元，其起始地址在 0100H。在输入了所有程序数据后，按"."键以结束 EB（检查字节）命令，并使监控程序再请求输入下一个命令。当出现提示符时，按下 GO 键，使用一个段值为 10 和偏移值为 0（10:0）作为起始地

址，输入该地址，按下“.”键。并注意显示是空白的，现在就可以按任意一个键来启动骰子滚动了。然后再按任意键就使骰子停止滚动。意意：由于实例程序用了地址段的最高位数字，所以“E”就被复盖上了。

（注）：程序输入到存储器中以后，它将一直保留在存储器中，直到关闭电路。系统复位不影响存储器中的程序。

要想从运行的用户程序将控制返回到监控程序时，可以按系统复位键或 INTR 键中的任意一个。如果按下系统复位键，则重入监控程序并且所有的8006寄存器被保护起来，还出现输入命令的提示符。当用 INTR 键重入监控程序之后，按 GO 键后，就会将当前 IP 寄存器值和该地址单元中的字节内容在显示器上显示出来。当按过 INTR 键后，IP 中的内容为被执行程序下条指令的偏移地址。地址单元是由 IP 和 CS 寄存器共同确定的单元。按“.”键可将控制从监控程序转回到这条指令所在处的程序，并继续执行程序。

GO 命令允许任意地输入一个“断点地址”。当一个程序在执行时，断点地址的作用和按 INTR 键的作用一样。为了输入一个断点地址，在输入起始地址和输入断点地址以后，按“，”键。注意：指定一个断点地址时，缺省的段值要么是起始地址的段值（若指定了的话），要么是当前 CS 寄存器内容（如果在指定起始地址时没有指定段值）。此外，在被断点地址指定的单元中，必须包含有指令的第一个字节（操作码或前缀）。当按“.”键时，监控程序用中断指令来代替在断点地址处的指令，并且在将控制转到用户程序去以前，把“被断点的”指令保存起来。当程序执行到断点地址时，控制将返回监控程序，程序中“被断点的”指令被恢复，所有寄存器被保护起来，并且出现监控程序的输入命令提示符，以便允许用户检查任何一个寄存器内容。

注意：由于当控制返回到监控程序时，“被断点的”指令恢复了，因此在程序带断点执行时，每次都要指定一个断点地址。

出错条件：企图在只读存储器的程序中设断点。

例 3.11 将控制传给实例程序。

按键	地址段显示	数据段显示	注释
SYSTEM RESET	- 8 6	A. 1	系统复位
2 GO/CX	0.	× ×	GO 命令(IP 寄存器偏移地址和数据内容)
1 ER/BX	1.		段(CS寄存器)地址
0 EB/AX	1 0.		段(CS寄存器)地址
:	1 0.		段／偏移分隔符
0 EB/AX	0.		偏移地址
.			控制传送到 0100H

例 3.12 在实例程序中引入和执行一个断点

按 键	地址段显示	数据段显示	注 释
SYSTEM RESET	- 8 6	A. 1	系统复位
1 GO/CX	0.	× ×	GO 命令
1 ER/BX	1.		段(CS寄存器)地址
0 EB/AX	1 0.		
,	1 0.		段/偏移分隔符
0 EB/AX	0.		偏移地址
,			
2 GO/CX	2.	2	断点偏移地址
0 EB/AX	2 0.		
.			转移控制
×	-	b r	按下任何键，断点达到。命令提示符

3.11 命令MV(传送)

功能：MV 命令可以实现在存储器中的一个数据块的移动。

格式：

MV (Start addr) , (end addr) , (destinetion addr) .

注：Stast addr ——起始地址

end addr ——终止地址

destinetion addr 目的地址

操作过程：MV 命令的格式是唯一的。命令中的三项全在地址段中产生。为了使用MV 命令，当输入命令提示符出现时，按 MV 键，这时在地段中有三个小数点亮起来，它表示要求输入三个项目，每一个项目输入完后，最左边的小数点消失，余下的小数点表示还要求输入地址。MV 命令所要求输入的项目要按以下顺序输入。

一、要传送数据块的存储器起始地址

二、要传送数据块的存储器终止地址

三、数据块要传送到的目的地址

注意：终止地址不允许有段地址，并且传送的数据块大小限制在64KB之内。

当按下“·”键时，就进行数据的传送并且命令提示符显示出来。注意：当数据块传送后，在源存储单元中包含的存储块不变（除非目的块的存区与源块存区复盖部分，由于传送进来的数把原有数冲掉了）。

由于在传送数据块时每次只传送一个字节，因此MV传送命令可将一个常数填入存储块，这时只要将目的地址指定的比起始地址大1就行了。从起始地址到终止地址加1的存储单元块中，将被起始地址单元中的内容填满。（检查字节命令可用来指定起始地址中的常数）。

出错条件：试图在只读存储器或不存在的存区中传送数据块。

例 3.13 将实例程序传送到0300H开始的存区

按键	地址段显示	数据段显示	注释
SYSTEM RESET	- 8 6	A. 1	系统复位
6 MV/SI	. . .		传送命令 MV
1 ER/BX	. . 1 .		起始地址(0100H)
0 EB/AX	. 1 . 0 .		
:	. 1 . 0 .		
0 EB/AX	. . 0 .		
,	. .		
4 IB/SP	. 4 .		终止偏移地址(04DH)
D /FL	4 . d		
,			
3 ST/DX	3 .		目的地址(0300H)
0 EB/AX	3 0 .		
:	3 0 .		
0 EB/AX	0 .		
·	-		程序传送，命令提示符

3.12 命令ST（单步）

功能： 单步 ST 命令允许存储器中程序的每条指令单独执行，每执行一条指令，控制将从程序返回到监控程序。

格式：

ST [（Start addr）], [[（Start addr],]* .

注：Start addr——起始地址

操作过程： 为使用 ST 命令，当输入命令提示符出现后，按下 ST 键，若起始地址和显示出的地址不同，那么就要输入所希望的起始地址。当按下“，”键后，地址指定单元中的指令开始执行，并且将下条要执行的指令的偏移地址在地址段显示器上显示出来，而这个地址中的指令字节显示在数据段显示器上，再按下“，”键时，执行当前这条指令并步进到下一条要执行的指令去。

在下面给出的例子中，使用单步命令来执行滚骰子程序中的头几条指令。下面的表中给出了该程序开始部分的清单。（完整的滚骰子程序清单在本章未尾给出）。

单元	内容	符号	注释
00	8CC9	MOV CX,CS	;
02	8ED9	MOV DX,CX	;
04	BAE AFF	MOV DX,0FFEAH	;
07	B0D3	MOV AL,0D3H	;
09	EE	OVT DX	;
		READKEY:	;
0A	BAEAFF	MOV DX,0FFEAH	;
0D	EC	IN DX	;
0E	240F	AND AL,0FH	;
10	74FB	JZ READKEY+3	;
12	E82800	CALL READATA	;

注意： 当程序从 10H 处的指令开始步进时，在0DH 处的指令要再次执行并且在 12H 处的指令不执行，这是由于没有键能按到“滚骰”程序中去（监控程序是在键盘控制中）。因此在 10H 处的 JZ（零跳转）指令转回到 0DH 处的指令，继续按“，”键将重复这三条指令序列（0DH，0EH，10H）。

限制：

（一）如果在完成一条单步指令前产生一个中断，或者如果一条单步指令产生一个中断，当重入监控程序时，则 CS 和 IP寄存器中将包含有中断服务的地址；随后，由于类型 3（断点）中断指令（0CCH 或 0CD）的执行将步进进入监控程序，所以类型 3（断点）中断指令将不被单步执行。

（二）指令序列中指令若是堆栈段之间的开关指令（即改变 SS和SP寄存器内容），则不能被单步执行。

（三）修改段寄存器的MOV或POP指令不能被单步执行，在下条指令（跟在MOV或POP指令后的一条指令）执行完后，控制返回监控程序。

例 3.14　程序单步执行。

按　键	地址段显示	数据段显示	注　释
3 ST/DX	0.	× ×	单步命令ST
1 ER/BX	1.		程序的起始地址
0 EB/AX	1 0.		
:	1 0.		
0 EB/AX	0.		
,	2.	8 E	下一条指令
,	7.	b 0	
,	9.	E E	
,	A.	b A	
,	d.	E C	
,	E.	2 4	
,	1 0.	7 4	
,	d	E C	

3.13 命令CD（轉录或写磁带）

功能： CD命令用来将内存RAM的信息转录到盒式录音机磁带上。磁带作为TP-86A的外存，以保存暂时不用的程序数据。

格式：

CD[（文件名）]，[（首地址）]，[（末地址）]·

操作过程： 在转录过程中，使用美国“肯萨斯城（Kansas）标准”。传送信息的速率为300波特（Baud）。即以8个2400赫音频脉冲信号表示“1”，以4个1200赫音频脉冲信号表示“0”。在一个文件转录过程中，开头有30秒钟“全1”的引导信号，末尾有5秒钟“全1”的结尾信号。

转录过程如下：

首先把磁带装入录音机；

用转录线将 TP-86A 的 CDOUT 端与录音机的 MIC 输入端相连接；

这时在键盘上按下“SYSTEM RESET”键，使控制转到监控程序初始化状态，并显示命令提示符，等待用户输入命令。这时可按下“CD”转录命令键。在按下“CD”键之后，显示器上马上出现三个小数点，小数点亮的顺序是：地址段最右边两个小数字亮：数据段最右边的一个小数点亮；小数点亮表示要求输入“文件名”和要转录的用户程序首地址和末地址。

这时首先输入“文件名”。注意“文件名”必须是十六进制码的组合，最好不用功能键，否则在某些情况下会显示出错信息“Err”。键入的四个十六进制码组合的文件名则按键入的顺序显示在数据段显示器上。如果输入的“文件名”多于四个，则显示最后四个作为有效的“文件名”，如不足四个则高位补“0”输入完“文件名”后，按下“，”键，表示“文件名”输入完了。同时，数据段最右边的小数点消失，等待输入地址，但这时“文件名”仍然保留在数据段上。

在“文件名”输入后，按下“，”键，就可以输入要转录的用户文件首地址。作为首地址可以用段地址，并且这个首地址显示在地址段显示器上。输入完首地址，按下“，”键，表示输入完首地址；同时，地址段左边的小数点和首地址消失，这时只有一个小数亮着，等待输入末地址。

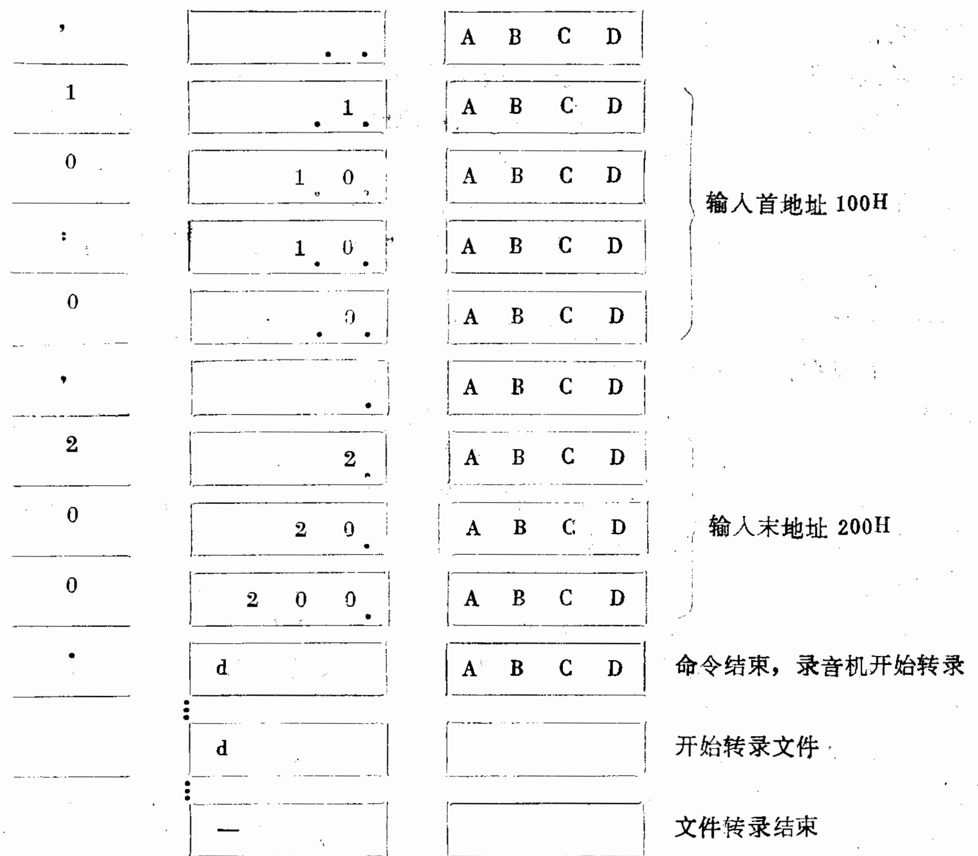
在文件首地址输入完并按下“，”键以后，就可以输入要转录的文件末地址了。作为末地址不允许用段地址，并且也显示在地址段显示器上，输入完末地址后可按下“·”键，这时在地址段最左边显示出“d”，它表示可以开始转录用户文件了。同时，地址段最右边的小数点和末地址消失。而“文件名”却一直保留到发完文件长度时才消失。

当按下“·”键后，立即把录音机置成录音方式，开始正式转录内存中的信息。此时指示灯变亮。

在转录时不需要进行音量调节，因为录音机内有 AGC 自动增益控制。当 d 消失时，表示文件转录结束，延迟 5 秒钟后，当转录完成时指示灯灭，同时显示命令提示符“—”。此时按下录音机的 STOP 键，停止走带。整个过程最少要35秒钟。

例 3.15 转录文件名为 ABCD 的文件，该文件存在 100H 到 200H 的存储器中。

按键	地址段显示	数据段显示	注释
SYSTEM RESET	- 8 6	A. 1	系统复位
A CD/SS	. .	.	CD 命令
A	. .	A .	输入文件名
B	. .	A B .	
C	. .	A B C .	
D	. .	A B C D .	



3.14 命令CL(轉儲或讀磁帶)

功能： CL 命令用来将录音机磁带中的信息即文件输入到内存 RAM中。

格式：

CL[（文件名）]，_[（内存地址）]_·

操作过程：

用转录线将录音机的 ERA 输出端与 TP-86A 的 CLIN 端相连接；

将磁带走到要转储的文件的相应位置上；

将录音机的高音控制和音量控制开到最大；

按下“SYSTEM RESET”键，使控制转到监控程序初始化状态，并显示命令提示符，等待用户输入命令。这时就可按下“CL”键。按下“CL”命令键后，立即显示两个小数点。小数点亮的顺序是：地址段最右边的亮；数据段最右边的亮；它表示可以输入“文件名”了。在输入“文件名”时有两种情况：

知道要转储的“文件”：在此情况下，就可以在十六进制键上按下相应的键，即已知的文件名。

不知道要转储的“文件名”：在此情况下，则在键盘上按下“全F”，即FFFF。这时，当监控程序收到“全F”文件名时，则不查找原文件名，而只把文件长度相同的文

件给转储到内存中去。

然而，不论哪一种“文件名”输入完后，“文件名”都显示在数据段上。注意，“文件名”系由四个十六进制码组合而成（最好不用功能键，以免显示出错信息），这时按下“，”键，表示“文件名”已经输入完，此时数据段最右边的小数点消失，而保留“文件名”。

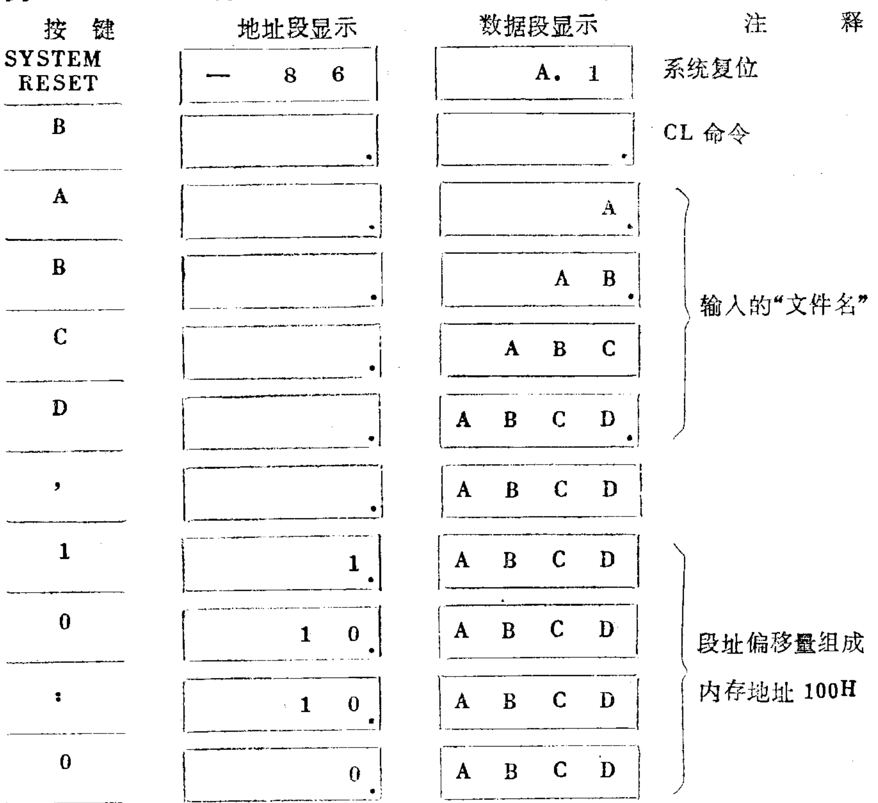
在“文件名”输入后，就可以输入内存地址了。用户在输入内存地址时可以用段地址。内存地址显示在地址段显示器上。输入完内存地址后接着按下“·”键，这时在地址段最左边显示出“L”提示符，它表示用户可以开始转储；同时，地址段的内存地址和小数点消失，而“文件名”一直保留。

当按下“·”键之后就可以将录音机置成放音状态，开始转储，同时指示灯亮。注意，显示器上的“文件名”只有接收到正确组合的“文件名”后才被清除掉。当“文件名”消失后，表示开始接收文件长度和文件内容。在这当中，只要有奇偶校验错误及溢出重叠错误就会显示出错信息“-Err”，从而坏掉全部转储。指示灯在整个转储过程中都保持发亮。

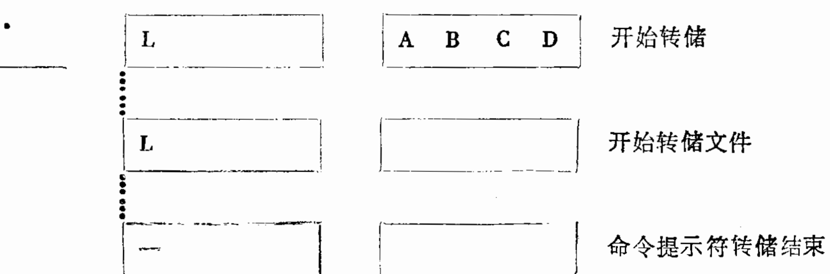
如果转储过程获得成功，则“L”提示符消失，表示转储结束，但还有5秒钟的延迟；接着显示命令提示符“—”。在这个过程中就可以关闭录音机。

注意：在同一条磁带上可以纪录好几个文件，用户可以使用语言标志或录音机上的走带计数器对文件进行识别。

例 3.16 ABCD 将文件转储到起始地址为 100H 的内存中去



按键	地址段显示	数据段显示	注释
SYSTEM RESET	— 8 6	A. 1	系统复位
B	.	.	CL 命令
A	.	A.	输入的“文件名”
B	.	A B.	
C	.	A B C	
D	.	A B C D.	
,	.	A B C D	
1	1.	A B C D	段址偏移量组成内存地址 100H
0	1 0.	A B C D	
:	1 0.	A B C D	
0	0.	A B C D	



3.15 命令PE(EPROM)写入

功能： 在本机中，可以将RAM中的数据写入插在U32(偶)、U33(奇)插座中的2716或2732型的EPROM中，对EPROM进行写入。

格式：

PE [(编程类型)] ， [(源文件首地址)] ， [(源文件末地址)] ， [(EPROM相对地址 ·

操作过程：

首先将需要写入的EPROM用紫外线穿过器件窗口进行照射，以擦去其中原有的内容。如果是未用过的新器件，则可以免去此步。

所用的紫外线波长为2537A，照射强度为12000μW/cm²，照射能量为15W—sec/cm²，普通情况下可使用医用的紫外线灯（15—30W），照射距离为25～5.0cm，照射时间为20～40分钟，便可擦去EPROM的原内容。

在对EPROM进行写入前，首先要把开关置到正确位置上。如果是2732型的片子，则把W47、W49、W51、W53接通；若是2716型的片子，则把W46、W48、W50、W52接通，并且当写入时一定要通W55，同时指示灯亮。一般情况下，则一定要接通W54。

加电或按下“系统复位”键，在显示器上出现命令提示符，等待输入命令。这时就可以按下PE命令键。当按下PE命令键后，在显示器上马上会出现四个小数点。在数据段最右边的小数点发亮，表示要求输入编程类型。在地址段显示器上从最右边向左依次三个小数点亮，第一个小数点（从右边开始）表示要求输入RAM中源文件首地址；第二个小数点表示要求输入RAM中源文件末地址；第三个小数点表示要求输入EPROM中的相对地址。

在这四个小数点亮了以后，就可以输入编程类型。编程类型的种类是根据所用片子的型号而确定的。一种是2716，它的地址范围是从0000H—0FFFH。另一种是2732，它的地址范围是从0000H—1FFFH。如在插座上插入的是2716，则在四个小数点亮后就要输入“2716”；如在插座上插入的是2732，则在四个小数点亮后一定要输入“2732”；如输入的不是这两种类型符号，则在执行时显示出错信息“Err”。也就是说编程类型要和所选的片子相同。并且编程类型显示在数据段显示器上。当编程类型输入完后，就按下“，”键表示编程类型已经输入完毕，并且在数据段最右边的小数点消失，编程类型也同时消失。

当输入完编程类型后，就可以输入RAM中的源文件首地址。作为首地址可以用段地址，并且这个首地址显示在地址段显示器上。输入完首地址便按下“，”键，表示输入完首地址；同时地址段最左边的一个小数点和首地址消失，而只有两个小数点亮着，等待输入下一个内容。

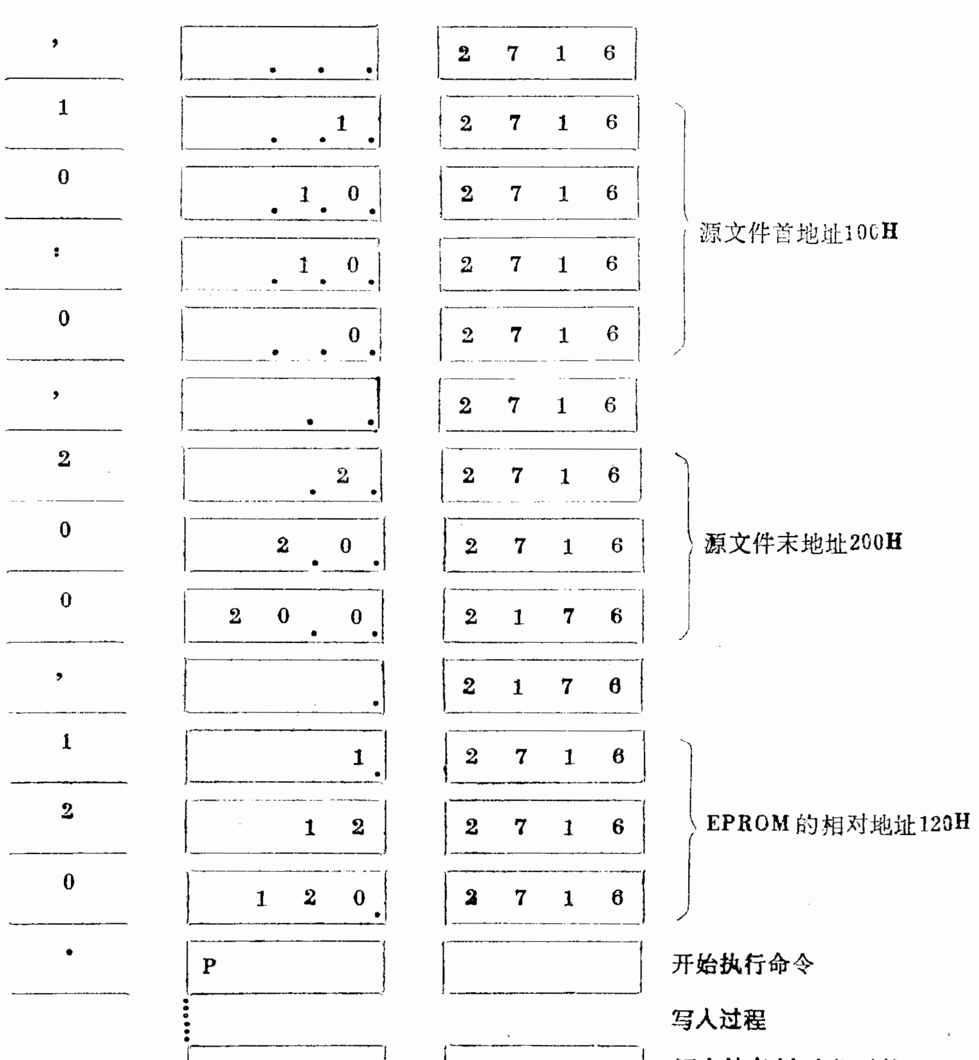
在源文件首地址输入完后，就可以输入源文件末地址了，而末地址是不允许用段地址的；并且这个末地址显示在地址段灯上。当输入完末地址时，按下“，”键，表示输入完末地址，同时地址段最左边的一个小数点和源文件末地址消失，而只剩下一个小数点亮着，等待输入最后一个内容。

在末地址输入完并按下“，”键以后，就可以输入 EPROM 的相对地址。注意，这个相对地址是从 0000H 开始的。它的地址范围要由插入的片子类型所决定。如是 2716，地址范围则 0000H—0FFFH；如是 2732，地址范围则是 0000H—1FFFH；所以相对地址可在这个范围内任选，也就是说源文件可以从任何一个地址开始写入。但是这个相对地址与最大地址之差必须大于或等于源文件的地址长度，即必须保证源文件一个不漏地写入到 EPROM 中去，否则在执行时就会显示出错信息“—Err”。输入的相对地址显示在地址段显示器上。当输入完相对地址时就按下“·”键，表示开始执行 EPROM 写入命令，同时在地址段最右边的小数点和相对地址同时消失。

这时可能有两种情况。一个是在地址段显示器上显示出错情息“—Err”，它表示源文件不能完整地写入到 EPROM 中去，这时用户就要查一下输入的相对地址到最大地址的长度是否合适，修改相对地址，重新开始输入命令；另一个是在地址段显示器上显示“P”提示符，它表示命令格式输入正确，并且正式开始把 RAM 中的数据写入到 EPROM 中去了。在写入的过程中，如果哪一个单元有错误，比如原内容未擦掉等，在显示器上就会显示出错误的相对地址，并且显示在地址段上；同时在数据段上显示“—bad”提示符，表示某个单元是“坏”的。这时不继续写入了，而是停在这个单元并等待用户检查。此时按下“·”键则转回到键盘监控程序。待查出错误后，重新输入命令，直到正确写入提示符“P”消失，接着显示命令提示符“—”。

例3.17 将源文件写入到 2716 中去。源文件首地址为 100H，末地址为 200H，相对地址为 120H。

按键	地址段显示	数据段显示	注释
SYSTEM RESET	— .8 6	A.1	系统复位
PE	. . .	.	输入 PE 命令
2	. . .	2.	编程类型
7	. . .	2 7.	
1	. . .	2 7 1.	
6	. . .	2 7 1 6.	



源文件首地址100H
源文件末地址200H
EPROM的相对地址120H
开始执行命令
写入过程
写入结束/命令提示符

第四章　TP-86A串行监控程序命令的使用

4.1 引　言

本章说明在串行监控程序中设置的各种控制、操作命令以及如何通过外围设备字符显示器（CRT）或电传打字机（TTY）使用这些命令。

串行监控程序命令包括了键盘监控程序中前八条命令，同时又另外设置了纸带输入输出命令。

外围设备 CRT 或TTY 是通过串行输入输出接口插座 J7 联接的（关对其引线布置请参看第一章外围接口）。

串行监控程序固化在二片 2732 EPROM 之中，各占有 2K 字节。在本机中跳接插头 W57接通时：

串行监控程序起始地址在 FF000H

键盘监控程序起始地址在 FE000H

当跳接插头W56 接通时：

串行监控程序起始地址在 FE000H

键盘监控程序起始地址在 FF000H

一经接通电源或按下“SYSTEM RESET”（系统复位）键，TP-86A 就从 FF000H 为起始的程序开始执行，所以当 W57 接通，TP-86 A 就自动进入了串行监控；而当 W56 接通，TP86A 就自动进入了键盘监控。在键盘控制下，可以由键盘发出一条‘GO命令’转到串行监控程序起始地址（FE000H 或 FF000H 分别对应接通 W57 或W56），于是就在串行监控程序控制之下了。

同样在串行监控程序控制之下，也可以由外围控制台上发出一条‘GO 命令’，转到键盘监控程序起始地址（FF000H 或 FE000H 分别对应接通 W57 或 W56），这样就在键盘监控程序控制之下了。

无论是由于接通电源或按下“SYSTEM RESET”键进入串行监控程序（W57 接通串行监控程序起始地址在 FF000H），或者是由键盘控制下经‘GO 命令’转入到串行监控程序，都会在键盘的显示器上显示出：

□□86　□□A.2

同时在 CBT 上显示出：

TP86-A MONITOR VA. 2.

在下一行开始处显示出句点“·”称之为提示符、以表示监控程序正在等待接收命令进来。此时键盘除了按下“SYSTEM RESET”键（系统复位）或 INTR（中断）键以外，按下其他键都没有任何作用了。

从最低位开始的256个存储器地址（OH～OFFH）留作监控器和用户堆栈，如图4.1所示。（在RAM中用户可用的地址最低位是0100H）

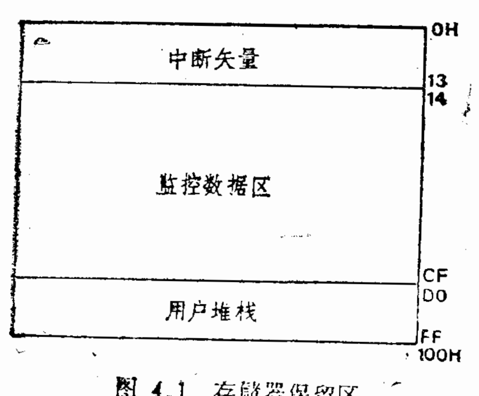


图 4.1 存储器保留区

本机一经接通电源或按下“SYSTEM RESET”键，监控器都立即结束它的操作作用，并且跳转到它的初始化子程序，这子程序对中断矢量1到3进行如下的初始化：

中断1：单步，用于单步命令。

中断2：NMI（非屏蔽中断），用于监控器INTR键。

中断3：断点，用于GO命令。

子程序还将8086的CS，DS，SS，IP和FL寄存器初始化为0，将SP寄存器初始化为0100H（栈底）。

一旦由于单步中断，NMI中断或断点中断使控制又回到监控，监控就将8086寄存器的内容保存（PUSH指令）到用户堆栈中去。而在它送出提示符准备接收命令进入之前，又依次送回（POP指令）寄存器内容。因为SP（堆栈）寄存器初始化到0100H（栈底），经这样初始化的堆栈就为用户保留了48个字节（DO-OFFH单元）。而其中还须保留26个字节，为了在上述中断之一的中断发生时用来保存寄存器的内容。

4.2 串行监控的命令格式和命令集

一、串行监控命令格式

当监控准备接收命令时就在显示器新的一行开始处送出提示符句点“·”，这一行就称之为命令行。在这行中，先送入一个或两个命令助记符，以表示是何种命令。此后就是一个到三个命令参数或是自变量。（在命令助记符与第一自变量之间允许插入空格）。若用一个以上的自变量，则在两个自变量之间要用一个逗点“，”分开，作为分界符。命令参量说明操作的是字还是字节。自变量则说明或是存储器的地址、或是某个寄存器、或者是要更改的新数据。按照命令本身来决定是用回车还是用逗点来结束这一命令行。回车表示执行一次就结束，下一次重新写入新命令；逗点则表示执行一次，并且下一次仍为该命令的继续，而不需重新写入命令。

除去将8086的寄存器用英文字母缩写表示以外，其他自变量都用十六进制数表示。对应输入字节是二位十六进制数从00到FF，对应输入字是四位十六进制数从0000到FFFF。开始0可以不写，若是在输入字节时输入了多于两位数字，则仅最后两位有效。同样输入字时输入多于四位，则仅最后四位有效。地址自变量包括段地址和偏移地址，若不输入段地址，就自动以当时CS（代码段）寄存器的内容作为缺省的段地址，否则就得另有命令进行指定。当需要输入段地址和偏移地址两个自变量时，第一个是段地址（可以是段地址的数值，也可以是段寄存器的缩写），第二个是偏移地址。在这两个自变量当中要插入一个冒号“:”作为分隔。

输入'命令结尾'（回车或逗点），命令才被执行。因此在输入结尾之前，随便什么时候送入了不合法的字符，都可以勾消这个命令、当勾消了这个命令，监控在命令行就输出一个数号"#"并且进行回车、换行，再在新的一行送出提示符"·"，准备接收新的命令。

二、串行监控的命令集

串行监控能够执行十种命令，如表4.1所列。在以后几节中分别详述各个命令，并且附以例子。在表4.1和以后几节中描述命令时用下列符号：

[A] 说明A是任意值

[A]* 说明A是一次或多次出现的任意值

〈B〉 说明B是自变量

〈cr〉 说明送入回车

注意，上列符号仅为了说明命令格式中各参量的性质，它们是既不送到控制设备（CRT、TTY）中去，也不从控制设备送出来。命令格式如表4.1所示：

表 4.1 监控器命令一览表

命令	功能及格式
S（更换存储器内容） （Substitute Memory）	显示/修改存储器单元 S[W]〈addr〉, [[〈new contents〉],]* 〈cr〉
X（检测/修改寄存器） （Examine/Modify Register）	显示/修改 8086 寄存器 X [〈reg〉] [[〈new contents〉],]*〈cr〉
D（显示存储器） （Display Memory）	显示存储器数据块 D ([W] 〈start addr〉 [, 〈end addr〉] 〈cr〉
M（传送） （Move）	传送存储器数据块 M 〈start addr〉, 〈end addr〉, 〈destination addr〉 〈cr〉
I（口输入） （Port Input）	接收并显示输入数据 I [W] 〈port addr〉, [,]*〈cr〉]
O（口输出） （Port Ouput）	输出数据到输出口 O [W] 〈port addr〉, 〈data〉 [, 〈data〉]* 〈cr〉
G（转走） （Go）	对8086的控制由监控转到用户程序 G [〈start addr〉] [, 〈breakpoint addr〉] 〈cr〉
N（单步） （Single Step）	单步执行用户程序 N [〈start addr〉], [[〈start addr〉],]* 〈cr〉
R（读十六进制文件） （Rdad Hex File）	从纸带读十六进制目的文件到存储器 [R〈bias number〉] 〈cr〉
W（写十六进制文件） （Write Hex File）	输出存储器数据块到穿孔纸带 W [X] 〈start abbr〉, 〈end addr〉 [, 〈exec addr〉] 〈cr〉

注意，以后各节中为了对命令加以说明，举出了应用例子，为了区别是从控制显示设备输出的内容还是输入的内容，在输出的内容下面加了横线，而实际显示器上是没有横线的。

4.3 命令S〔W〕：更换存储器命令

功能： 命令S或SW是用来检测选定的存储器单元中的字节(S)或字(SW)。

若存储单元的内容可以修改（例如RAM单元），还可以从控制设备输入新数据值用来对内容进行修改。

格式： S[W]<addr>, [[<new contents>],]*<cr>

addr—被检测单元的地址

new contents—修改的新数据内容

操作过程：监控在显示器的新一行开始处输出提示符“·”时，便可送命令符S或SW，然后送入要进行检测的存储地址。

在地址<addr>中先送入段地址。若段地址用的是某一个段寄存器的内容，也可以送这个寄存器的缩写。然后送入冒号“：”，其后送入偏移地址。若段地址不指定（即缺省的段地址，监控就使用当时的CS（代段码）寄存器的内容。

在地址进入之后送入逗点“，”，这时监控就会把该地址存储单元的内容显示出来，并随后加一小横作为“数据提示符”，等待新的修改数据到来。注意，当应用了“SW命令”时，监控首先将下一个相邻地址（addr+1）单元中的字节内容显示出来，随后才将指定地址（addr）单元中的字节内容显示出来。与此相仿，用SW来修改存储器内容，输入的第一个字节（前两个十六进制数）是对下一个相邻地址（addr+1）存储单元进行修改的内容。输入的第二个字节则是对指定地址（addr）单元进行修改的内容。若仅检测一个数据，不送“，”，送入回车<cr>结束这条命令。回车送入后监控按新数据进行修改，（若未曾输入新数据，则保持原来内容），并且在新的一行开始处给出提示符等待新命令。若不仅检测一个数据，而要检测连续一系列的数，则不送回车，而再一次送入逗点，则监控仍按新内容修改以外，还会在显示器的下一行将下一个存储器单元的内容显示出来（命令S），或将下两个单元内容显示出来（命令SW）。同样，若须修改，则送入新数据；不须修改，则不必送数据。此后再输入逗点，则又检测下一个单元或下两个单元（命令S或命令SW）的内容。如此可以一直连续检测下去，直到输入回车结束这个命令。

例4.1： 检测ROM中FE000单元。

```
.S  FE00:0, 90 -<cr>
.
```

例 4.2： 检测RAM以DS（数据段）寄存器内容为段地址的010H单元，并修改012H单元为OC9H。

```
.S DS:10, EE -,
0011  74 -
```

0012 C4-C9〈cr〉

.

例 4.3：检测和修改栈顶。

. SWSS:SP， 574A-574B〈cr〉

.

4.4 命令X：检测修改寄存器的命令

功能：命令 X 用来检测 8086 各个寄存器的内容，并且如果需要，还可以将它们加以修改，或者用来显示出8086的全部寄存器的内容。

格式：X[〈reg〉][[〈new contents〉],]*〈cr〉

reg—寄存器的缩写

new contents—修改的新数据内容

操作过程：当监控在显示器新一行开始处输出一提示符"·"时，便可送命令符 X。如果仅需要显示当时的全部寄存器内容，就送入回车，这样十四个寄存器的内容全部输出。如果需要检测和有选择地修改某个寄存器内容，可按照表 4.2 送进寄存器缩写。

表 4.2 寄 存 器 缩 写

寄存器名称		缩写
累加器	Accumulator	AX
基地址	Base	BX
计　数	Count	CX
数　据	Data	DX
栈指针	Stack Pointer	SP
基指针	Base Pointer	BP
源变址	Source Index	SI
目的变址	Destination Index	DI
代码段	Code Segment	CS
数据段	Data Segment	DS
栈　段	Stack Segment	SS
附加段	Extra Segment	ES
指令指针	Instruction Pionter	IP
标　志	Flag	FL

当送入寄存器缩写时，监控在显示器输出一个等号"="及现行寄存器的内容。此后输出"数据提示符"一小横"—"和一小空格，表示等待新的数据输入。如果希望改变寄存器的内容，就将新数据内容送入，随后送入逗点，则继续检测和修改下一个寄存器的内容（按表 4.2 的次序）。随数据之后若送入回车，则结束了这条命令，即经修改内容后，监控在下一行输出提示符，并等待下一条命令进入。

注意寄存器的次序须按照表 4.2，但并不是循环的。也就是在最后一个 Elag（标

志）寄存器内容输出之后，送修改的新数据（或不修改就不送数），再按下逗点，命令被执行，此后监控在下一行输出提示符，处于等待命令状态。

例4.4：检测8086的寄存器。

```
. X<cr>
AX=89D3   BX=0002   CX=0010   DX=F450   SP=0200
BP=38AE   SI=4765   DI=0700   CS=0020   DS=0010
SS=0000   ES=0010   IP=0231   FL=F046
```

例 4.5：　修改 CS 寄存器，检测下两个寄存器内容。

```
.X CS=0000-20,
   DS=0010-,
   SS=0000-<cr>
   .
```

4.5 命令D[W]：显示存储器命令

功能：命令 D[W]用来以字节的形式（命令D）或以字的形式（命令 DW）在显示器上显示出一块存储区

格式：D[W]<start addr>[, <end addr>]<cr>

start addr—起始地址

end addr—结束地址

操作过程：当监控在显示器新一行开始处输出提示符句点时，便可送命令符D（显示字节）或 DW（显示字）。然后送入需要显示的存储区的起始、终止地址(终止地址内容也显示）。最后送回车，则监控就将这个存储区的内容从显示器命令行的下一行开始显示，每行十六个字节或是八个字连续输出显示，直到该区全部内容都输出完为止。

起始地址包括段地址和偏移地址，所以送起始地址是先输入起始地址的段地址。若段地址使用的是某一个段地址寄存器的内容，那么也可以送入这个寄存器的缩写，随后加冒号，再输入偏移地址。在这个过程中若不送段地址，则按 CS（代码）寄存器的当时内容作为段地址。

终止地址不允许有段地址，所以一条指令显示的数目限定在 64K 字节或 32K 字之内。若无终止地址，则监控仅在下一行显示出一个字节或一个字来。

显示存储器命令可以在任何时候送一个控制字符 C，立即结束这个命令，监控在下一行显示出提示符，准备接收下一条命令。显示存储器命令也可以在任何时候送一个停止字符 S，暂时仃止输出显示，但并未结束这个命令，再送字符 Q 则又恢复了仃下来的令，继续输出显示。不送字符 Q 而送字符 C，则结束了这个命令。但在送过字符 S 后，除字符 Q 和字符 C外，不能送其他字符。

出错条件：终止地址小于起始地址。

例4.6：显示以 DS（数码段）寄存器内容为段地址，从 1AH 到 03CH 的内容。

```
.D DS：1A，3C<cr>
```

```
001A FB 07 74 F7 BB FB
0020 8A 4D 47 90 BA EA FF B0 87 EE BA E8 FF 8A C1 EE
0030 BA EA FF EC 24 0F 74 E0 E8 02 00 EB CD
.
```

例 4.7：显示以CS（代码段）寄存器内容为段地址的0450H的内容。

```
.D 450<cr>
0450 7C
.
```

例 4.8：显示FE00AH到FE022H的字内容。

```
.DW FE00:A, 22<cr>
000A 2029 3931 3837
0010 4920 544E 4C45 4320 524F 4050 0000 4000
0020 7F00 007D
```

4.6 命令M：移动数据块命令

功能：命令 M 用来在存储器中移动数据块，还可以用来在一块存储区内填入同一常数。

格式：M <start addr>, <end addr>, <destination addr> <cr>

start addr—起始地址

end addr—终止地址

destination addr—目的地址

cr—回车

操作过程：当监控在显示器新一行开始处输出提示符时，便可送命令符 M。然后送入需要移动的存储区起始和终止地址（终止地址内容也移动）。此后还须送入'目的地址'，最后送入回车，则监控就将这个存储区的内容移到从目的地址为起始的存储区间去。

起始地址和目的地址二者包括段地址和偏移地址，所以送起始地址或目的地址时先送段地址，加入冒号再送入偏移地址。段地址可送段寄存器缩写，或实际段地址值；不送入段地址，监控就以 CS（代码）寄存器的当时内容使为段地址。

终止地址不允许有段地址，所以每一个命令移动的数据块的字节数在64K之内。

用 M 移动命令在一块存储区内填入某同一常数的方法，是先利用 S命令将某常数写入此区起始地址单元中，然后送入命令符 M，而命令的起始地址为此区的起始地址，命令中的终止地址为此区终止地址减 1，命令的目的地址为起始地址加 1，以回车起动该命令后，即在该区内填入了某常数。

操作错误：1. 移动数据到只读存储器（譬如 ROM 或 PROM）或不存在的存储单元。

2. 指定的终止地址小于起始地址的偏移地址。

3．目的地址大于起始地址而小于终止地址

例 4.9：以CS（代码）段寄存器内容此段地址，从0200H单元至0250H单元的存储区移到0300H为起始的存储区中去。

.M200，250，300〈cr〉

例 4.10：将存储区以DS（数据段）寄存器内容为段地址的0300H单元到0470H单元，移到以ES（额外段）寄存器内容+20为段地址，以03A0H为起始的存储区中去。

.M DS：300，470，ES+20:3A0〈cr〉

.

例 4.11：在CS（代码段）地址内的400到700单元都填入90。

.S 400，73—90〈cr〉

.M 400，6FF，401〈cr〉

.

4.7 命令I[W]：端口输入命令

功能：命令I（或IW）用来将指定输入口的字节（或字）显示到显示器上。

格式：I[W]：〈port abbr〉，[，]*〈cr〉

port addr—指定的输入口地址

操作过程：当监控在显示器新一行开始处输出提示符"·"时，便可送命令符I或命令符IW。I/O地址在64K地址之内，口地址不允许用段地址。在口地址进入后，用一个逗点就可以使监控将在这个指定口中的数据（字节或字）显示到显示器的下一行上去。连续送入逗点，监控就连续地在下一行显示出这个口的数据内容，直到送回车，监控除传送数据外，在下一行送出提示符，并等待新命令入内。

若从并行输入/输出口8225A输入数据，可按照表4.3中的口地址。

表 4.3 8225并行输入/输出地址表

口名称	口数据地址	口控制字地址
P2A	FFF8H	FFFEH
P1A	FFF9H	FFFFH
P2B	FFFAH	FFFEH
P1B	FFFBH	FFFFH
P2C	FFFCH	FFFEH
P1C	FFFDH	FFFEH

注意：这个口在开机或复位后已按输入工作方式初始化，若最后的编程是按输出工作方式编程的，则必须再重新按输入编程（见4.9节口输出命令）。

若是用命令IW输入'字'内容，则口地址应该用低字节地址，在8255中为P2的地址。

例 4.12：从P1B和P2B输入字内容。

```
. IW FFFA
    47C4〈cr〉
.
```

例 4.13：从口P2A输入多字节。

```
. I FFF8
  01,
  02,
  02,
  B4,
  3F, 〈cr〉
.
```

4.8命令O〔W〕：端口输出命令

功能： 命令'O'或OW用来输出（写入）字节或字到指定的输出口。

格式： O[W]〈port addr〉,〈data〉[,〈data〉]*〈cr〉

port addr—指定的输出口地址

data—写入输出口的数据

操作过程： 当监控在显示器输出了提示符“·”，可送此命令符O或OW，再送入输出口地址（无段地址，参见端口输入命令）及逗点。此后若连续送入数据、逗点，则监控连续将数据送出，直到送入回车，监控送出数据后，结束这命令。也可以送入数据后就送入回车，则监控只送一个数据出去，就回到等待命令状态，并送出提示符“·”。

注意： 正如上节所述，接通电源或按下系统复位（System Reset）键，监控对接口电路按照输入工作方式进行初始化，因此要使口能输出数据，必须先按输出工作方式编程，表4.4中给出并行输入/输出口8255的一种编程控制字。

表 4.4　**口控制地址及控制字**

口　名　称	口控制地址	控制字	
		输入方式	输出方式
P2	FFFEH	9BH	80H
P1	FFFFH	9BH	80H
P2/P1	FFFEH	9B9BH	8080H

有关8255的其他编程资料，请查阅8255的使用说明。

例 4.14　将并行输入/输出口P2A按输出方式进行编程，并输出一个01字节。

```
. O FFFE, 80〈cr〉
```

.0 FFF8, 01 <cr>

.

例 4.15 连续输出多个字节到 FFFD口

.0 FFFD, B0,

.C3,

.74,

.E8,

.07<cr>

.

4.9命令G：轉走(或轉移)命令

功能：命令 G 用来将对 8086 的控制由监控转到指定的起始地址中，从而进入这个程序控制中去。并且可以在这程序中再指定一个断点地址。

格式：G[<start addr>] [, <breakpoint addr>]<cr>

start addr—起始地址

breakpoint addr—断点地址

操作过程：当监控在显示器新的一行开始处输出提示符句点时，就可以送入命令字符 G。命令符 G 进入后，监控就将当时的 IP（指令指针）寄存器内容（即下一条将执行的指令地址）以及该地址存储单元的字节内容（即下一条将执行的指令第一个字节），中间插一数字提示符（小横），一同显示出来。若要改变起始地址（即改变下一条将执行的指令地址），则送入新的起始段地址（可以是段地址值，也可以是某一段寄存器的缩写，如果不送入，则按 CS 寄存器内容）和偏移地址，再送回车，则对8086 的控制由监控传送到所执行的程序，并且连续执行这程序。

若在键盘上按下“系统复位”(System Reset)键或按下“中断”(INTR)键，都可以从执行的程序中退出来，回到监控控制中。而按下“系统复位”(System Reset)键，控制就转回到以 FF000 为起始地址的监控中去（参阅 4.1 节），并且对 8086 作相应的初始化。而按下“中断”(INTR）键，程序被中断，控制就重新转回到串行监控，对8086的所有寄存器进行保护，在显示器上输出提示符之前还输出了下列信息：

@aaaa：bbbb

aaaa—当时的 CS 寄存器内容

bbbb—当时的 IP 寄存器内容

这两个寄存器内容组合成一条指令地址。这指令正是在按下“中断”(INTR) 键时所执行程序的下条指令。若再送入“命令G”的命令符 G，则监控再次送出 IP 寄存器的内容及该地址（按 CS 和 IP 寄存器内容组成的地址）单元的字节内容。若送入回车，则对8086的控制又转入原来的程序中，重新从当时的指令地址开始继续执行程序。

命令 G 中允许送入断点地址，当程序执行到断点地址时，与按“中断”(INTR) 键有着同样的效果。请注意“断点地址”中无段地址（与起始地址用同样的段地址）。

另外“断点地址”一定是指令第一个字节（操作码或前缀）的地址。当指定了“断点地址”，监控就保护了“断点地址”的原来指令，又以中断指令改换了这个地址的原来指令内容。当程序执行到该断点地址就回到了监控，监控保护了8086所有寄存器的内容，又重新恢复了断点的原来指令，并输出下列信息

BR @ aaaa:bbbb

aaaa—当时的CS寄存器内容

bbbb—当时的IP寄存器内容

表明断点已发生，并显示出下一条将执行的指令地址（由CS和IP两寄存器内容组成），也就是“断点地址”。在此之后，监控在下一行输出提示符，等待命令入内，允许对任何寄存器，存储器进行检测、修改，以便对程序进行考查。若再用命令G，就可以重新从断点开始执行断点的原来指令，而进入原来的程序。

注意：因为控制回到了监控，断点指令已被恢复，在程序中每设置一次断点，必须每次送入“断点地址”。

操作错误：试图在只读存储器指令中设置断点。

例 4.16 将控制转动到段地址为CS寄存器内容、以04A0H单元为起始地址的程序中去。

.G 020F-EE 4A0⟨cr⟩

例 4.17 将控制转移到地址为20:0H的程序中，并在20:0CH处设置一断点。

.G 0210-BA 20:0，0C⟨cr⟩

BR @ 0020:000C

4.10 命令N：单步命令

功能：命令N用来单个地执行用户程序指令，每执行一条指令，控制就回到监控，并且可以考查指令执行的情况。

格式：N[⟨start addr⟩], [[⟨start addr⟩],]*⟨cr⟩

start addr—起始地址

操作过程：当监控在显示器新的一行开始处输出了提示符句点，就可以送入命令符N。这时监控器就会将当时的IP（指令指针）寄存器内容（也就是下一条将要执行的指令偏移地址）显示出来，并且在数据提示符一小横之后将这地址单元的字节内容（按CS和IP寄存器内容组成指令地址中的内容）也显示出来。假若此时要想单步执行另外一条指令，则要送入这另外一条指令的起始地址（这个起始地址可以包括有段地址），再送入逗点，则监控将CS（代码段）寄存器和IP寄存器两个都按新地址进行修改（若不送段地址，则仅修改IP内容），送入逗点则按新地址执行这条指令，此后再转回监控。监控保护了所有寄存器内容，在显示器的下一行上又将IP寄存器内容（下一个将要执行的指令地址）以及这一条指令的第一个字节内容显示出来。每送入一个逗点就按照地址执行一条指令。此后在显示器的下一行又显示出来下一次要执行的指令地址及指

令的第一个字节内容。

因此用‘单步命令’单步地通过一个程序，只须送入一个新地址，而不须重复送命令入内。每当送入一个逗点，就执行这选定的指令，执行后又将下一条指令的地址和内容显示出来，直到回车才结束这个命令。

不允许的情况：假如单步执行一条指令，在其完成之前有中断产生，或者假如单步执行一条指令中产生了中断，此时控制回到了监控，CS 和 IP 两个寄存器的内容就是中断服务子程序的起始地址了。因此，“中断 3”（断点中断）指令（OCCH 或OCDH)不能单步执行，如执行就会进入监控。

若指令序列在堆栈中间转向[也就是 SS（栈段）寄存器和 SP（栈指针）寄存器内容改变了]，这序列的一部份指令不能单步执行。

因为 MOV（移动）指令和 POP（推入）指令修改了段寄存器内容，所以也不能用单步执行。若执行，则在执行了紧随着 MOV 或 POP 指令的下一条指令之后控制就传回到监控了。

例 4.18 单步执行段地址按 CS 寄存器内容从 0200H 开始的指令序列。

```
. N 0000-00 200,
  0202-8E,
  0207-BO ,
  0206-EE ,
  020A-BA <cr>
 .
```

例 4.19 单步执行两条指令，再单步重复第二条指令。

```
. N 0218-03,
   021A-40,
   021B-50 21A,
   021B-50<cr>
   .
```

4.11 命令R：讀十六进制文件命令

功能：命令 R用来从纸(磁)带读出十六进制目的文件，并且将文件中读出的数据存入存储器中去。

格式：R[<bias>]<cr>

bias—偏置数

操作过程：当监控在显示器新的一行开始处输出了提示符句点，就可以送入命令符R。若纸(磁)带机上装上了纸(磁)带，并且准备就绪，就可以送入回车。监控从文件中读出数据，又按照每一个记录中送来的地址作为开始地址，将数据存入存储器。若文件按8086格式，其中包括了执行地址的记录，CS(代码段)和 IP（指令指针)寄存器也就按记录中的指定执行地址进行修改，若文件按 8080格式，其中包括了EOF（end-of-file）记

录，IP 寄存器就按照 EOF 记录中指定的地址进行修改。CS 寄存器内容不变。注意：8080文件中不用段地址，读出的数据写入存储器中，其单元相对应的段地址为 0，当指定了执行地址，CS 寄存器内容不变。

若选用了偏置数，这个给定的数就加到每个记录送来的地址上，使得在存储器中可以调整文件位置。

操作错误：

1．“纸（磁）带校验”错误。

2．试图往不存在的存储区中传送数据。

3．试图往只读存储区中传送数据。

例 4.20 读文件并且将其数据传送到存储器，存储器地址按照文件指定的传送地址再加上 200H字节。

.R 200 〈cr〉

.

4.12 命令W：写十六进制文件命令

功能： 命令 W 用来按照 8086 或按照 8080 十六进制文件格式，将一块存储区内容输出到纸带（磁带）机中去。

格式： W[x]〈start addr〉,〈end addr〉[,〈exec addr〉]〈cr〉

start addr—起始地址

end addr—终止地址

exec addr—执行地址

操作过程： 当监控在显示器新的一行开始处输出了提示符句点，就可以送入命令符 W（按 8086 文件格式）或送入命令符 WX（按 8080 文件格式），此后送入要输出的存储器块起始地址和终止地址。注意，终止地址不允许使用段地址（按起始地址的段地址），而起始地址的段地址若不指定，则按当时的 CS（代码段）地址内容。几个地址都存入后，可送回车，监控往带上按下列信息输出：

60个零字符引导

扩展地址记录（仅8086格式）

起始地址和终止地址之间的存储器数据内容（终止地址内容也包括在内）。

EOF 记录（文件结尾记录）

60个零字符结尾

用8086格式（命令 W）将起始地址的段地址值（若不指定段地址值就使用CS寄存器的内容）输出在扩展地址记录中，将起始地址偏移值输出在第一个数据记录的传送地址场。用 8080 格式(命令 WX)仅将起始地址偏移值输出在第一个数据记录的传送地址场，起始地址的段地址值只用来选定存储区，并不输出。

在送入回车之前，可以不指定也可以指定出执行地址。在用命令 R读纸(磁)带时，这个地址传送到 CS 和 IP(指令指针)地址寄存器中（在 8080 文件格式中仅送到 IP 寄存

器)。指定的执行地址在选用的不同格式中以不同的方式传送。在8086文件格式中执行地址包括在执行起始地址记录之中，紧接着纸(磁)带引导之后将这记录输出在纸(磁)带上。在8080格式中将执行地址的偏移地址记在EOF记录中（执行地址不允许使用段地址值）。

写十六进制文件命令在任何时候都可以从显示器上送入控制字符将其消去或将其停止。控制字符C消去了命令，并输出新的提示符，等待新命令入内。控制字符S停止了输出，但不取消这个命令，控制字符Q恢复已经停下来的输出。在控制字符S之后仅能用控制字符Q或控制字符C。

错误的情况：指定的终止地址小于起始地址

例 4.21 以当时的CS值为段地址值，将04H到06DDH之间的一块存储区按8086文件格式输出到纸(磁)带上，并附有执行地址CS：040H。

.W 4，6DD，40<cr>

.

例 4.22 将FF200H与FF2FFH之间的存储区按8080文件格式输出到纸带上，并给定了起始传送地址0100H和执行地址011A。

.W FF10:100，1FF，11A<cr>

注意：输出至盒式磁带机时，为符合“肯萨斯城”标准

1．波特率选用300波特（跨接短路插头W21—W29中的W26）

2．在送入回车之前，先启动磁带录音等待约30秒，在该命令执行完后，等待约5秒关闭磁带录音。

第五章 PL/M-86高级语言、ASM-86汇编语言简介及TP-86A监控程序简介

TP-86A 监控程序的源程序基本上是用 PL/M-86 高级语言编写的，有部分过程是用 ASM-86 汇编语言编写的。

为了帮助用户阅读 TP-86A 监控源程序，在这一章中将对 PL/M-86 高级语言及 ASM-86 宏汇编语言作一简单的介绍。

注意：本章的重点在于阅读监控源程序，而不是对 PL/M-86 及ASM-86 语言本身及程序设计作专门介绍。因此，对于想深入了解这些语言的用户，请参阅 PL/M-86 程序设计手册及 MCS-86™ 宏汇编语言参考手册。

5.1 PA/M-86高級語言簡介

一、引 言

PL/M 语言是世界上第一种专门为微型计算机设计的高级语言。它的第一个文本是英特尔（Intel）公司于 1973 年首次公布的。在随后几年里，又作了若干次修改。先有 PL/M-80 语言，它用于 Intel 8080/8085 系列微型计算机，由于 Intel 公司是世界上最大的微型计算机公司，所以 PL/M 语言的用户相当普通。据调查，在日本已使用的三种编译程序语言(FORTRAN，PL/M和 BASIC)在微型机中共占约80%(其中FORTRAN 列为 35%，PL/M 为 23%，BASIC 为21%）。把 PL/M-80 稍加改动和扩充，就产生了适用于新产品 Intel 8086/8088 16 位微型机的 PL/M-86 语言。此外，作为交叉软件使用的全 PL/M 语言编译程序，可编译出 Z-80 微型机的目标代码。

我国从1974年开始进行微型计算机的试制，1977年确定生产两个系列—050 和 060。其中 050 是对应于 Intel 8080 系列的。此后，国内许多单位为 050 系列机配上了 PL/M 语言。随着 050 系列的成批生产和 8080 系列在我国的应用和发展，PL/M语言也势必在我国被广泛采用。

二、PL/M-86的语句和注释

一个 PL/M 源程序是 PL/M语句的一个序列。PL/M 程序是用自由格式书写的，即与程序的各输入行是无关的；在程序的各元素之间可以自由地嵌入空格。程序行文之间可以书写解释性的注释。

PL/M 语句分为两种类型：

1. **说明语句**：即不可执行的语句

DECLARE 语句和 PROCEDURE 语句为说明语句。

DECLARE 语句最重要的用途是说明变量。变量可以是标量（纯量），或者是数组，或者是结构。

过程说明总是以一个 PROCEDURE 语句开头，并以一个与其相匹配(呼应）的END 语句结尾，可以把过程看作为一个“子程序”，当在程序中另一处调用它时，就执行这个“子程序”。

2. **可执行语句**，PL/M语句中除了 DECLARE 语句和 PROCEDURE 语句之外，都是可执行语句，而绝大多数可执行语句能产生机器代码。可执行语句有赋值语句、程序流程控制语句及过程调用语句。

PL/M-86 源程序由所需的不同类型的语句所组成。这些语句的末尾总是用分号（；）来结束。为了改善程序的可读性，可在任何地方自由地使用空格，如在语句后面，或在语句当中插入空格。一条语句可占用多行。为了使程序易于阅读并提出程序文件，在程序行文之中可写入解释性的注释。一个 PL/M 注释要夹在符号/*和*/之间。可用任何能打印的 ASCII 字符来写注释。注释中可以包含间隔、回车、换行和标号等字符，只是不能把注释嵌入到一个字符串常数中间去。除此之外，PL/M 注释可嵌入任何地方，也就是说注释可以自由地分布在 PL/M 源程序内的任何地方。

例 5.1 PL/M-86 语句和注释：

```
/*TRAFFIC DATA RECORDER CONTROL PROGRAM*
 *VERSION 2.2, RELEASE 5, 23 APR 79.*
 *THIS RELEASE FIXES THREE BUGS*
 *DOCUMENTED IN PROBLEM REPORT 16.*/
/*COMPUTE TOTAL PAYMENT DUE*/
TOTAL = PRINCIPAL + INTEREST;
IF TERMINAL$READY
    THEN CALL FIIL$BUFFER;
    ELSE CALL WAIT (50); /*WAIT 50 MS FOR RESPONSE*/
```

三、数据的定义及DECLARE说明语句

对于绝大多数 PL/M-86 源程序来说，在其开始处都要用 DECLARE 语句来定义程序中要引用的数据元素（变量和常量）。

变量是常数的对象，它的值在程序执行过程中可以是变化的，并由标识符引用。常数有固定的值并可直接引用。例如：表达式；

APPROX* (OFFSET-3/SCALE;

中包括了变量 APPROX，OFFSET，SCALE 和常数 3。

单个的PL/86 数据元素称为一个标量(或纯量，英文为SCALAR)。标量变量SCALAR VARIABLE）是这样的一种变量，它的值在编译期间内不必知道，而且在执行程序期间

可以改变，因此是由标识符（或称名字 NAME）来引用它的。标量变量（简称标量）总是表示一个具有单一数值的变量。

标识符用来命名变量、过程、宏说明和语句标号。一个标识符最长可由31个字符组成。第一个字符必须是字母，其余字符可以是字母，也可以是数字。标识符是由程序员提供的，为改善可读性，PL/M-86 允许程序员在标识符中间任何地方插入美元符号 $。编译程序对嵌入行文中的美元符号不作任何处理。一个包含美元符号的标识符与把美元符号从其中删去之后的标识符完全相同。

例如：KB$EXAM$MEMORY 与

KBEXAMMEMORY 是恒等的，可互换的标识符。

1．标量的类型

PL/M-86 的标量具有下列五种类型之一，表 5.1 中列出了这五种类型及其特点。

表 5.1　PL/M-86数据类型

类　　型	字节数	范　围	使　　用
字　　节	1	0to255	无符号的整数，字符
字	2	0to65,535	无符号的整数
整　　型	2	−32,768to +32,767	有符号的整数
实　　型	4	$1\times10^{-38}$to $3.37\times10^{+38}$	浮点数
指引元或指示器	2/4	N/A	地址处理

2．简单的说明语句

一个变量在被它的标识符引用之前必须先加以说明，这个说明任务是由DECLARE语句完成的。DECLARE语句的格式如下：

DECLARE scalar-name type;

这里 scalar-name 为标量名，用任选的标识符来表示。Type 就是表5-1中给出的五种类型之一。

例如：

```
DECLARE UNKNOWN BYTE;
DECLARE PTR POINTER;
DECLARE (WIDTH, LENGTH, HEIGTH) REAL;
```

上面的第一个说明语句中引入了一个标识符 UNKNOWN，并指出它是一个类型为BYTE(字节)的值。第二个说明语句引入标识符 PTR，并指出它是一个类型为POINTER（指引元或指示器）的值。第三个说明语句为“因子的说明”语句，它等效于下面的三个说明语句：

```
DECLARE WIDTH REAL;
DECLARE LENGTH REAL;
DECLARE HEIGHT REAL;
```

在说明语句中用括号括起来的变量表中的变量要连续地存放。但在邻接的说明语句中宣布的变量不必连续地存放。

表 5.1 中给出的数据类型的概念不仅适在用于变量，而且还适用于被 PL/M-86 程序处理的每一个值，包括由“过程”返回的值及表达式计算出的值。

PL/M-86的数据元素不仅可以是变量（SCALAR VARIABLES），而且可以是常数（SCALAR CONSTANTS）。见例5.2。

例 5.2 PL/M-86 常数

```
          10 /*DECIMAL NUMBER*/                十进制数
         0AH /*HEXADECIMAL NUMBER*/            十六进制数
         12Q /*OCTAL NUMBER*/                  八进制数
   00001010B /*BINARY NUMBER*/                 二进制数
        10.0 /*FLOATING POINT NUMBER*/         浮点数
       1.0E1 /*FLOATING POINT NUMBER*/         浮数点
         'A' /*CHARACTER*/                     字符
/*CONSTANTS MAY BE GIVEN NAMES*/ 常数可以给出名字
DECLARE STATUS$PORT LITERALLY'0FFEH';
DECLARE THRESHOLD LITERALLY'98.6';
```

在例 5.2 中我们可以看到说明语句的另一种格式：

DECLAKE 标量名 LITERALLY‘串’；

这里标量名可为任选的标识符。LITERALLY 是一个保留字，中文可以译成“字面上”或“文字的”。

LITERALLY 说明为宏说明。若在一个说明中使用了保留字 LITERALLY，则这个说明是为了在编译时进行扩充而定义的无参数的宏说明。一个标识符（即标量名）被说明为表示一个字符串，则在以后的行文中标识符的每一次出现都由这个串所代替。说明语句中的‘串’是 PL/M 字符集中任意字符的序列，序列的字符数不超过 255。

在 PL/M 语言中允许把许多个标量（SCALARS）聚集成为有名字的数据集合体。例如，数组（ARRAYS）和结构（STRUCTURES）就是这种数据集合体。

数组是同一类型标量的集合。如数组中的元素可以全为整数或者全为实数等等。用数组来表示具有重复特性的数据是很方便的。

例如：一年中每月降雨量的采样值可用一个具有同一类型的 12 个元素的数组来表示。

DECLARE RAINFALL (12) REAL;

在上面这条说明语句中，RAINFALL 为降雨量的标识符括’在括号中的 12 是该数组的“维数”，语句未的 REAL 指的是数组的 12 个元素全是实型的。数组是利用“维数说明符”来说明的。维数说明符是一个括在园括弧内的常数，而常数的值说明了该数组的元素的数量。上面这条语句引出了与 12 个数组元素相联系的标识符 RAINFALL，其中每个元素都是 REAL（实数）型的。

可用一个称为下标变量（SUBSCRIPT）的数来引用数组中的单个元素。下标变量

（可简称下标）表示的是元素在数组中的相对位置。在PL/M-86中规定数组中第一个元素的下标为0，第2个元素的下标为1，…………依此类推，于是：

RAINFALL（11）表示12月的降雨量。

下标不一定必须是常数'变量和表达式也能用来作下标。以后数组中元素的顺序号就按下标号来称谓了。这里就改称为第0个元素，第一个元素…………第十一个元素

结构（STRUCTURE）是一些数据元素的集合。这些数据元素可具有不同的类型。下面给出一个简单结构的例子：

```
DECLARE BRIDGE STRUCTURE   注：  BRIDGE—桥
        (SPAN   WORD               SPAN—跨度
        YR$BUILT  BYTE,            YR$BUILT—建造时间
        AVG$TRAFFIC REAL);         AVG$TRAFFIC—平均交通量
```

在上面DECLARE语句中，BRIDGE为结构名的标识符。结构元素用园括号括起来，其中的SPAN、YR$UILT、AUG$TRAFFIC为结构元素的标识符；SPAN为WORD（字）类型，YR$BUILT为BYTE（字节）类型，AVG$TRAFFIC为REAL（实数）类型。假设我们要引用上面结构中的第2个元素，可写BRIDGE.YR$BUILT，这里可以说是用结构名BRIDGE和·来限制结构元素名YR$BUILT。这样就允许在不同的结构中使用相同的元素名，而引用该元素时不会混乱；如在另一结构HIGHWAY中也有YR$BUILT元素，那么引用此元素时需写HIGHWAY.YR$BUILT。（HIGHWAY中文意思为“公路”）

利用数组和结构还可以组合成许多更复杂的集合。PL/M-86语言中允许有：

- 结构的数组—结构作为数组的成分（元素）
- 内部有数组的结构—数组可作为结构的成分（元素）
- 内部有数组结构的数组

例 5.3 PL/M-86数据定义

```
    /****SCALARS****/
DECLARE SWITCH         BYTE;
DECLARE COUNT          WORD,                  /*1SCALAR*/
        INDEX         INTEGER;                /*1SCALAR*/
DECLARE (NET, GROSS, TOTAL) REAL;             /*3SCALARS*/
    /****ARRAYS****/
DECLARE MONTH(12)      BYTE,
DECLARE TERMINAL_LINE(80)     BYTE;
    /****STRUCTURE..../
DECLARE EMPLOYEE STRUCTURE
        (ID_NUMBER             WORD,
        DEPARTMENT             BYTE
        RATE                   REAL);
/****ARRAY OF STRUCTURES****/
```

```
DECLARE INVENTORY__ITEM(100)  STRUCTURE
        PART__NUMBER              WORD,
        ON__HAND                  WORD,
        RE__ORDER                 BYTE);
    /****ARRAY WITHIN STRUCTURE****/
DECLARE COUNTY__DATA          STRUCTURE
 (NAME(20)                    BYTE,
TEN__YR__RAINFALL(10)         BYTE,
PER CAPITA__INCOME            REAL);
```

例5.3中给出了五个例子：第一个为定义标量的例子；第二个为定义数组的例子；第三个为结构的例子；第四个为“结构的数组，”它说明了与数组标识符INVENTORY__ITEM（物质清单项目）相联系的一百个结构。这一百个结构由从0到99的下标加以区分，其中每个结构由三部分组成：一个是WORD型标量PART__NUMBER（零件号），一个是WORD型标量ON__HAND（现存）及一个是BYTE型标量RE__ORDER（再定货）。因此，在存储器中要为二百个WORD型标量和一百个BYTE型标量分配地址。譬如，当需要引用INVENTORY__ITEM数组中第5个结构的ON__HAND元素时，可写：

```
INVENTORY(5).ON__HAND
```

第五个例子为“内部有数组的结构”。数组可以作为结构的成分。这个结构名称为COUNTY__DATA（县的记录）。这个结构由三个元素组成：第一个元素为数组NAME（名称），它是BYTE类型的；第二个元素为数组TEN__YR__RAINFALL（十年的降雨量），它是BYTE类型的；最后一个元素为标量PER CAPITA__INCOME（平均每人的收入），它是REAL型的。若要引用该数组中第二个元素中的第四个分量，则写：

```
COUNTY__DATA.TEN__YR__RAINFALL(4)
```

我们再举一个“内部含有数组结构的数组”的例子如下：

```
DECLARE PAYROLL(100) STRUGTURE(
    LAST$NAME(15) BYTE,
    FIRST$NAME(15) BYTE,
    MI$NAME BYTE, DOLLARS WORD, CENTS WORD);
```

这样，我们得到了包含有一百个结构的数组，其中每一个结构在执行程序期间可以用来存放一个职工的姓，教名，中间名，元，分。在每个结构中又用两个含有15元素的BYTE型数组LAST$NAME和FIRST$NAME来存放名字的字符串。在打印输出工资名单时经常要用到这样的数据定义，譬如，若要引用第N个职工的教名中的第K个字母，可写PAYROLL(N).FIRST$NAME(K)，当前N和K是在前面已定义过的变量。

3．指引元(或指示器)的间接引用：有基变量

在编制程序过程中经常遇到许多数据元的存储地址不能事先确定，而要到程序运行时经过计算才能确定的情况。对于这种情况，直接引用这些PL/M数据元是不可能的。

对于这些数据元，在编制PL/M源程序时，就必须书写PL/M代码来控制数据元的地址，而不是控制数据元本身。为了实现这种控制方式，PL/M使用了有基变量。用另一个叫做“基地址”的变量指向的变量称为有基变量。编译程序不给有基变量分配存储器，在程序运行期间的不同时刻，它在存储器中的地址实际上是可变的，这是因为程序可以改变它的基地址。有基变量的说明方式是先说明它的基地址，基地址必须是WORD或POINTER型的；然后再说明有基变量本身。例如，

```
DECLARE ITEM$PTR POINTER;
DECLARE ITEM BASED ITEM$PTR BYTE;
```

上面给出的这些说明表示，对ITEM的引用实际上是引用由当前的的ITEM$PTR的值所指向的BYTE类型的值。这就表示，序列

```
ITEM$PTR=34AH;
ITEM=77H
```

是把BYTE型的值77H存入存储器中地址为34AH的那个单元中去。

上面第二句中的BASED为保留字。

对基地址有如下限制：

- 基地址必须是POINTER或WORD型的。在PL/M-86语言中总是用POINTER，WORD是与PL/M-80语言兼容而采用的。
- 基地址不可以带下标，即不能是数组元素。
- 基地址本身不可以是有基变量。

保留字BASED在说明语句中必须紧跟在有基变量的名字（标识符）之后，如下列

```
DECLARE (AGE$PTR,INCOME$PTR,RATING$PTR,CATEGOR$PTR)
POINTER;
DECLARE AGE BASED AGE$PTR BYTE;
DECLARE (INCOME BASED INCOME$PSR, RATING BASED RATING$
PTR) WORD;
DECLARE (CATEGORY BASED CATEGORY$PTR) (100) WORD;
```

第一个DECLARE语句用来将变量AGE$PTR、INCOME$PTR，RATING$PT和CATEGOR$PTR定义为POINTER（指引元）型的，以供后继的三个$DECLARE语句用它们来作为基地址。第二个DECLARE语句定义了一个叫做AGE的BYTE型变量，AGE是基址变量，当运行中的程序要引用AGE基值变量时，总是到AGE$PTR当前值指定的那个存储单元中去取。这里AGE是BYTF型的，AGE$PTR是POINER型的。第三个语句定义了两个WORD型的有基变量INCOME和RATING。第四个语句定义了一个叫做CATEGOR的WORD型数组，这个数组含有100个元素；它的基址是CATECOR$PTR。这就是说，当CATEGOR数组中的任一元素在程序运行期间被引用时，CATEGOR$PTR的当前值指向CATEGOR数组中下标为0元素的地址，而其余元素顺次接下去。括在CATEGOR BASED CATEGOR$PTR之外的园括号是为了使该语句易于阅读而加上的，也可以略去。例如：CATEGOR（5）是引用CATEGOR数组中下标为5的那个元素。CATEGOR（5）元素存放在CATEGOR$PTR当前值（基地址）加10的

地址单元中。因为在CATEGOR（5）元素前面还有5个WORD型的数组元素，每个WORD型的数组元素要占用两个相邻的字节单元，故要加上10才能形成CATEGOR(5)元素所在的地址。

4. 改进的说明语句

在改进的说明语句中，除去标量名（标识符）和类型外，还根据不同情况引入有关的一信息。在前面例5.2中已给出了一种带有宏说明信息LITERALLY的说明语句。为了能使用户顺利地阅读TB-86A监制源程序，下面再给出两种信息。（至于另一些信息这里就不给出了，感兴趣的用户可参看PL/M-86语言程序设计手册）。

（一）AT属性（THE AT ATTRIBUTE）

（1）存储单元引用

一个POINTR变量的值是一个8086的存储单元地址。POINTER变量的主要用途是作为有基变量的基地址。

①@运算符：

一个“存储单元的引用”（以后简称单元引用）是用@运算符来建立的。一个单元引用有一个POINTER类型的值，即一个单元地址。

单元引用的基本形式是：

@Variable-ref

这里“Variable-ref”指的是对某一个变量的引用。这个单元地址是在该变量运行时实际的单元地址，于是任一单元引用总是POINTER类型的。

“Variable-ref”可以是一个数组或一个结构，而这时它就是数组或结构第一个元素的单元地址。

例如：假设我们写了如下的说明语句。

若
```
DECLABE RESULT REAL;
DECLARE XNUM(100)BYTE;
DECLARE RECORD STRUCTURE   (KEY BYTE,
                            INFO (25) BTTE,
                            HEAD POINTER) ;
DECLARE LIST (128) STRUCTURE (KEY BYTE,
                              INFO (25) BYTE,
                              HEAD POINTER) ;
```

则@RESULT是REAL型标量RESULT的单元地址，而@XNUM（5）是数组XNUM第六个元素的单位地址，@XNUM是该数组的起始地址——即第一个元素（下标为0的元素）的单元地址。

同样，@RECORD. HEAD是POINTER型标量RECORD. HEAD的单元地址，而@RECORD是BYTE型标量RECORD. KEY的单元地址；@RECORD.INFO是有25个元素的BYTE型数组RECORD. INFO的第一个元素（下标为0的元素）的单元地址，而RECORD. INFO（7）是同一个数组的第八个元素（下标为7的元素）的单元地址。

LIST是结构的数组。存储单元引用@LIST（5）.KEY是标量LST（5）. KEY的

单元地址。注意：@LIST．KEY 是不合法的，由于它不能指定一个唯一的单元地址。

单元引用@LIST（0）．INFO（6）是标量 LIST（0）.INFO（6）的单元地址。同样 LIST（0）．INFO 是同一个数组的第一个元素（下标为 0 的元素）的单元地址。

当说明语句中的标识符是一个过程的名字时，这个过程必须在该程序模块的外层中说明（即在外层中定义，参看 PL/M-86 语言程序设计手册第十一章）。即使在过程说明中含有形式参数，也不能给出任何实在参数。这时，存储单元引用的值是这个过程的入口点的地址。

②“·”点运算符：

其作用与@运算符相似，这是为使 PL/M-86 与PL/M-80 相兼容而提供的。

（2）AT 属性的格式

AT(location)

这里“location”可以是由@运算符建立的单元引用，又可称为限制表达式，也可以是范围在 0 到 1048575 中的一个常数。若“location”为一个单元引用，那么它必须是一个无基变量且已被定义的变量。如果在“location”处有一个表达式，那么该表达式只能是含有＋，－算符的常数表达式。若“location”处为一个常数值，则表示的是8086存储单元的绝对地址。

下面给出的都是合法的 AT 属性：

```
AT (4096)
AT (@BUFFER)
AT (@BUFFER (128))
AT (@NAMES (INDEX+1))
```

在最后一个例子中，INDEX 是事先用“LITERALLY”定义好的一个常数。在编译过程中，编译程序用定义好的 INDEX 值去取代 INDEX，故此例是满足前面的约定的，即表达式为常数表达式。

AT 属性的作用是把一个变量分配到某个存储单元去，这个存储单元的地址是由“location”（单元引用或限制表达式）指定的。被分配到该存储单元中去的变量是说明语句中的第一个标量。若同一个说明语句中有多个标量，则依次接下去存放。例如：

```
DECLARE (CHAR$A, CHAR$B, CHAR$C) BYTE AT (@BUFFER);
```

该说明语句把 BYTE 型的变量 CHAR$A 分配在数组 BUFFER 的地址单元中。变量 CHAR$B 和 CHAR$C 依次存放在 CHAR$A 所在的后两个字节单元中。

再看语句

```
DECLARE T (10) STRUCURE (X(3) BYTE,
                         Y(3) BYTE,
                         Z(3) BYTE)AT (@DATA$BUFFER)
```

该语句给结构 T 的第一个元素—即标量 T（0）．X（0）分配存储单元，该单元的地址与一个预先说明过的变量 DATA$BUFFER的地址相同。组成这个结构的其它标量按照逻辑顺序接在这个地址之后存放。这些标量是 T（0）．X（1）；T（0）．X（2），………直到 T（9）．Z（2）．最后这个标量，即 T（9）.Z（2）存放在 DATA$BUFFER 的地址之后的第 89 个字节单元中。

（二）DATA 初始化

DATA 初始化的格式是：

DATA) Value list)

此处"Value list"是"值表"。"值表"可以是一个或多个值的序列；各个值之间用逗号隔开；每次从"值表"中取一个值，并用它来给各个要被说明的标量初始化。例如，

DECLARE EVEN(5)BYTE DATA（2，4，6，8，10）；

此语句说明了 BYTE 型数组 EVEN，并把它的五个标量元素分别初始化为2、4、6、8、10。

AT 属性不能与 DATA 初始化一起使用，因为 AT 属性强行把一个变量存入指定的地址，这就会破坏 DATA 初始化的目的。

四、赋值语句

已经被定义好的数据可由 PL/M-86 的可执行语句来对它们进行处理，最基本的执行语句是赋值语句，其格式如下：

Variable-name = expressin；　　（变量名 = 表达式）

该语句的意思是："对表达式求值，并将结果送到变量名指定的变量中去"。PL/M-86语言中有三种基本类型的表达式，它们是算术表达式、关系表达式和逻辑表达式。见表5.2 及例 5.4。

表 5.2　　PL/M-86 表达式的特性

表达式	运算符	结果
算术表达式	+,- *,/,MOD	NUMBER(数)
关系表达式	>,<,=,>=,<=	"TRUE"-FFH(真) "FALSE"-0H(假)
逻辑表达式	AND,OR,XOR,NOT	8/16-BITS TRING(位串)

例 5.4　PL/M-86赋值语句

```
/*ARITHMETIC*/
A = 2;  B = 3;
B = B + 1;                          /*B CONTAINS4*/
C = (A*B) - 2;                      /*C CONTAINS6*/
C = ((A*B) + 3)MOD3;                /*C CONTAINS2*/
/*RELATIONAL*/
A = 2;  B = 3
C = B>A;                            /C CONTAINS 0FFH*/
C = B<>A                            /C CONTAINS 0FFH*/
C = B = (A + 1);                    /C CONTAINS 0FFH*/
/*LOGICAL*/
A = 0011$0001B;                     /*$IS FOR READABILITY*/
B = 1000$0001B;
```

```
C=NOT B;                          /*C CONTAINS 0111$1110B*/
C=A AND B;                        /*C CONTAINS 0000$0001B*/
C=A OR B;                         /*C CONTAINS 1011$0001B*/
C=B XOR A;                        /*C CONTAINS 1001$0000B*/
C=(A AND B) OR 0F0H               /*C CONTAINS 1111$0001B*/
```

五、程序流程控制语句

简单的PL/M-86程序可用前面介绍过的数据说明语句（DECLARE）及赋值语句（ASSIGNMENT）写成。PL/M-86还提供了程序流程控制语句，以使用户能编制各种循环结构及分枝结构的较复杂的程序。下面对这些程序流程控制语句作一简单的介绍。

1. IF语句

IF语句的功能是：根据一个关系表达式所求的值去选择执行两个语句（或DO程序块）中的一个。

IF语句的格式为：

IF 关系表达式
  THEN 语句1.（或DO程序块）
  ELSE 语句2（或DO程序块）

如果关系表达式所求的值为“真”，则执行语句1，而语句2被跳过；如果关系表达式所求的值为“假”，则语句1被跳过，执行语句2。在确定表达式结果为“真”或“假”时，IF语句先去检测结果的低位（为“1”，则为“真”），因此，算术和逻辑表达式也能用在IF语句中。例5.5给出了一些PL/M86语句的实例。IF语句的流程图示于图5.1中。

例5.5 PL/M-86IF语句实例

```
A=3B=5
IF A<B
  THEN MINIMUM=1;/*EXECUTED*/
  ELSE MINIMUM=2;/*SKIPPED*/
MORE—DATA=0FFH;
IF NOT MORE—DATA
  THEN DONE=1;     /*SKIPPED*/
  ELSE DONE=0      /*EXECUTED*/
/*NESTED IF STATEMENTS*/
CLOCK—ON=1;HOUR=24;ALARM=OFF;
IF CLOCK—ON
  THEN IF HOUR=24
    THEN IF ALARM=OFF
      THEN HOUR=0;/*EXECUTED*/
```

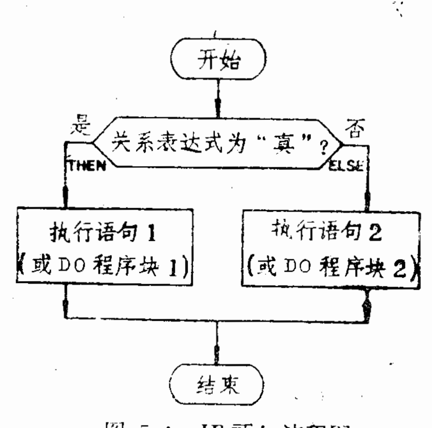


图5.1 IF语句流程图

在 IF 语句的 THEN 部分和 ELSE 部分使用DO 程序块可以增强 IF 语句的能力，因为在可以使用单个语句的地方都可以使用 DO 程序块，所以 IF 语句中的两个语句都可以是一个 DO程序块。

```
IF  A = B THEN
  DO;
    EQUAL$EVENTS = EQUAL$EVENTS + 1;
    PAIR$VALUE = A
    BOTTOM = B
  END
ELSE
  DO;
    UNEQUAL$EVENTS = UNEQUAL$EVENTS + 1;
    TOP = A;
    BOTTOM = B;
END;
```

嵌套在 IF 语句中的 DZ 程序块还可继续嵌套 DO 程序块、IF语句、变量和过程说明等等。

可以把一个 IF 语句（如果有 ELSE 部分，也可包括在内）看成单个的PL/M 语句，因此 IF语句可以互相嵌套。当一个 IF 语句嵌套在外层的 IF 语句之中时，只有一个限制，这个限制就是：如果一个 IF 语句是嵌套在外层的 IF 语句的 THEN 部分，则这个外层的 IF 语句不可以带有 ELSE 部分。换言之，象

```
IF    条件 1   THEN
IF    条件 2   THEN语句 1
ELSE语句 2
```

这样的构造如果没有这个限定，将会产生二义性，即产生 ELSE 部分是属于哪一个 IF语句的疑问。根据上述限制，这个 ELSE 部分不能属于外层的 IF 语句，它只能属于内层的 IF 语句。上面的构造等价于下列语句：

```
IF 条件 1  THEN
  DO;
  IF  条件 2 THEN 语句 1;
  ELSE 语句 2 ;
END;
```

应注意的是，用这种方式书写更易于阅读，且减少了出错的机会。

如果企图使 ELSE 部分属于外层的 IF 语句，则必须利用一个DO程序块进行嵌套：

```
IF 条件 1 THEN
  DO;
    IF 条件 2 THEN语句 1;
  END;
ELSE 话句2;
```

2．DO语句和END语句：DO程序块

DO 和 END 语句的作用象括弧一样用来构成“DO程序块”。在 PL/M 程序中，凡是能放置可执行语句的地方，均可以放 DO 程序块。

DO 语句一共有四种：

- 简单的 DO 语句
- DO CASE（情况）语句
- 交互的（或重复的）DO 语句
- DO WHILE（当型）语句

END 语句的形式是：END [标号]；在此语句中如果使用标号的话，则所用的标号必须是这个 DO 程序块开始的 DO 语句的标号。如果 DO语句的标号不止一个，则 END语句中的标号必须与这个标号中的最后一个，即最右边的标号相符。END 语句中的标号对程序不起作用。它只作为使程序易于理解的工具和一个调试程序的辅助手段。编译程序可以利用它找出错误的标号并用此提醒程序员，在他的程序中有错误。

（一）简单的 DO程序块：

简单的 DO 程序块以一个简单的 DO 语句开头，形如：

```
DO;
  语句1；
  语句2；
   ⋮
   ⋮
  语句n;
END;
```

例 5.6 简单 DO 语句的实例

```
    /*SIMPLE  DO*/

    A=5; B=9;
    IF (A+2) <B THEN DO;
                    X=X-1;        /*EXECUTED*/
                    Y(X) =0;      /*EXECUTED*/
                    END;
               ELSE  DO;
                    X=X+1;        /*SKIPPED*/
                    Y(X) =1;      /*SKIPPED*/
                    END;
```

简单的 DO 程序块内部的每一个语句可以是任意的 PL/M 语句，既可以是执行语句，也可以是说明语句。但有一点例外，即在这个 DO 程序块外层中的所有说明，必须在这个外层中第一个可执行语句之前。

DO 程序块可以嵌套。如下所示。

```
DO;
    语句1;
    语句2;
DO;
    语句a;
    语句b;
    语句c;
END;
    语句3;
    语句4;
END
```

简单的DO程序块中还允许使用一个简单的IF语句去引出多重语句的执行。

（二）DO CASE程序块：

DO CASE程序块以一个DO CASE语句开头，并有选择地执行块中的某个语句。这个语句是由算术表达式的值选定的。

DO CASE程序块的形式是：

```
DO CASE 算术表达式;
  语句1;
  语句2;
   ⋮
   ⋮
  语句n;
END;
```

其流程图示于图5.2中

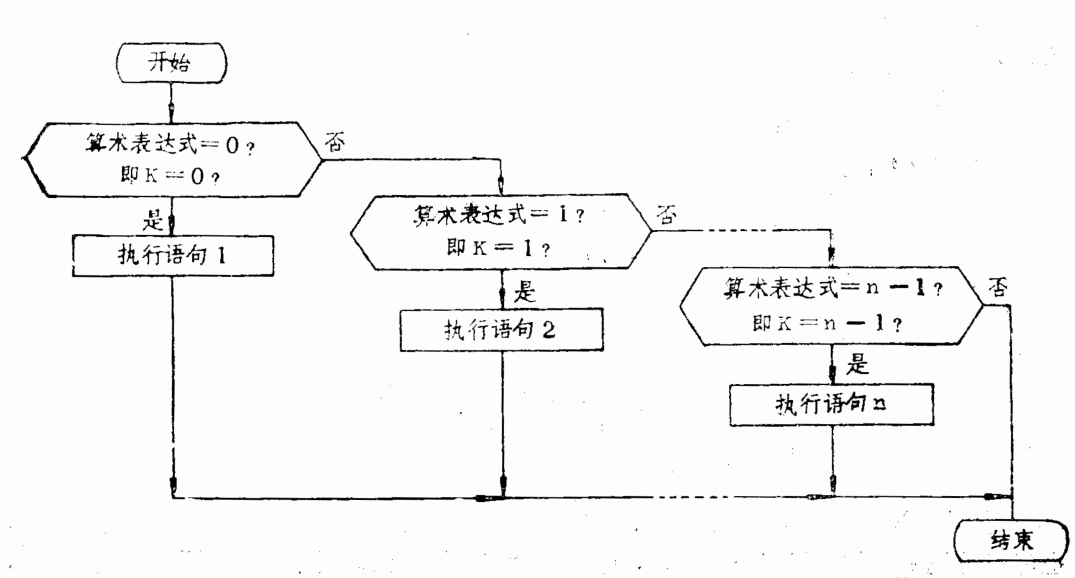


图5.2 DO CASE程序块流程图

首先计算DO CASE语句中的表达式的值，计算的结果必须是一个在0到n-1之间的值，并把这个值叫K，用K来选择DO CASE程序中n个语句中的一个语句，然后执行之。第一种情形（语句1）对应于K=0，第二种情形（语句2）对应于K=1，依此类推。只从程序块中选出一个语句，且这个语句只执行一次。执行之后，控制就转向接在这个DO CASE程序块的END语句后面的语句。如果在运行期间DO CASE语句中的表达式的值大于DO CASE程序块中的情形语句的数量，则DO CASE语句的作用无定义。

例5.7 DO CASE 语句实例

```
/*DO CASE*/
A = 2;
DO CASE(A);
    X = X + 1;        /*SKIPPED*/ 对应A=0，跳过
    X = X + 2;        /*SKIPPED*/ 对应A=1，跳过
    X = X + 3;        /*EXECUTED*/对应A=2，执行
    X = X + 4;        /*SKIPPED*/对应A=3，跳过
    END;
```

（三）重复的（或交互的）DO程序块：

重复的DO语句的格式为：

```
DO index = start-expr TO stop-expr BY step-expr;
```

用中文可写成：

```
DO 索引 = “起始表达式”到“终止表达式”按照“步长表达式”；
  语句1；
  语句2；
   ⋮
   ⋮
  语句n;
END;
```

重复的DO程序块以一个重复的DO语句开头，并且以如上述的方式重复地执行程序块内的语句。重复的DO程序块的流程图如图5.3所示。

例5.8 可用下面的重复DO语句将一个含有10个元素数组的元素全清0。

```
DO I = 0 TO 9;
    ARRAY(I) = 0;
END;
```

例5.9 可用下面的重复DO语句来计算前N个奇整数之积。

```
PROD = 1;
DO I = 1 TO (2*N-1) BY 2;
    PROD = PROD*I;
END;
```

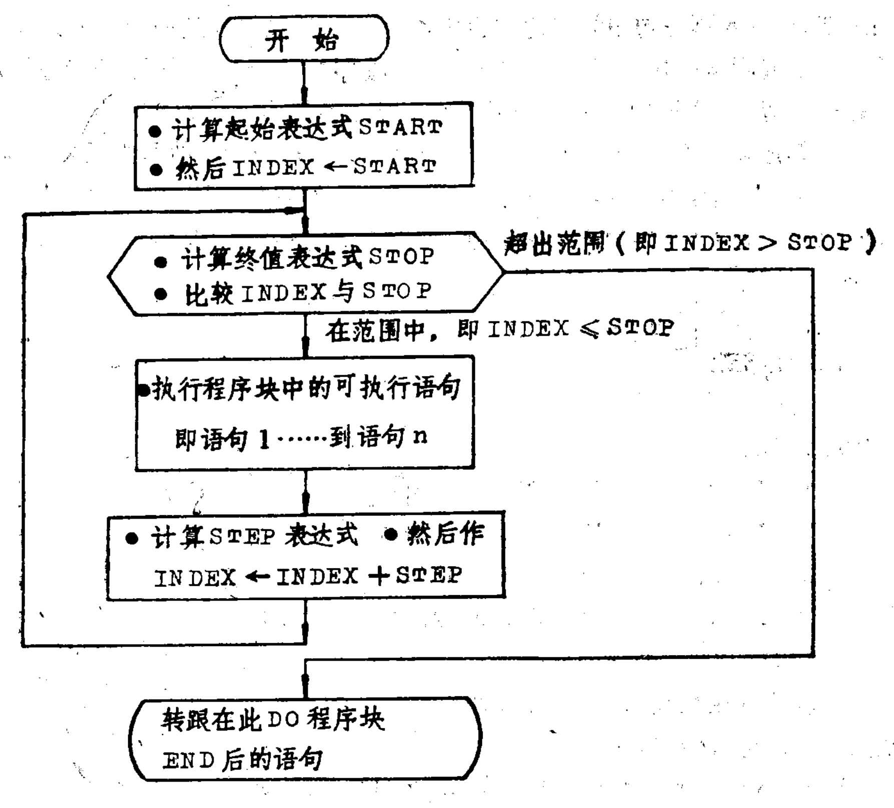


图 5.3 重复DO程序块的流程图

在重复DO语句中，不存在负的步长。譬如，若步长表达式为-5，则与步长为251的表达式相同，即负数用其二制进的补码表示。在PL/M中，“向下”步进到终止表达式的值是不可能的，即终止表达式的值不可能小于起始表达式的值。这是因为，如果下标变量的值大于终止表达式的值，则重复的DO程序块停止块行。

与简单的DO程序块一样，可以把一个重复的DO程序块看成单个的PL/M语句。但是、与简单的DO程序块不同的是，在重复的DO程序块的最外层不能含有说明。重复的DO语句中可嵌套简单的DO程序块，而在这个简单的DO程序块中是可以含有说明的。

（四）DO WHILE程序块：

在叙述DO WHILE语句之前，对逻辑运算符和DO WHILE语句之前的关系要解释一下。这些解释也适用于IF语句。在PL/M中规定关系运算的结果以FFH表示“真”，以00H表示“假”。用这样的“真”，“假”值来控制DO WHILE语句或IF语句。但要注意一点，在DO WHILE语句和IF语句中只检查表达式所求出值的最低有效位，因此不要求表达式的值必须为00H，或FFH。它(表达式)可以具有任意的BYTE型或ADDRESS型的值。如果值是奇数（最低位=1），就可把它看成“真”；如果是偶数（最低位=0），就把它看成“假”；

DO WHILE程序块以一个DO WHILE语句开头，其格式为：

```
DO WHILE 表达式;
     语句1;
     语句2;
       ⋮
       ⋮
     语句 n;
END;
```

DO WHILE 程序块的流程图示于图 5.4 中。

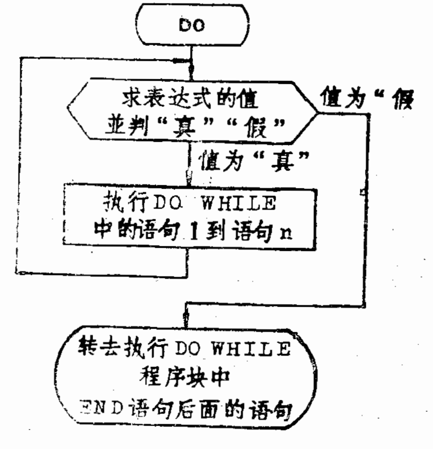


图 5.4 DO WHILE程序块流程图

从流程图中后以看出这个语句的作用是:

- 首先计算跟在保留字 WHILE 后面的表达式,如果计算的结果是一个最右边一位为 1 的量(奇数),则执行语句 1 到语句 n(即执行直到 END 为止的语句序列)。
- 到达 END之后,再计算 WHILE 表达式,仅当表达式的值最右边一位为1(奇数)时,再块行这个语句序列。
- 一遍又一遍地执行这个语句序列(程序块),直到表达式的值最右边一位是 0(偶数)为止,这时就跳过程序块中的语句序列,程序控制就指向接在 END 语句后面的语句。

注意:在上面所说的程序块(语句序列)若包含有任何程序流程控制语句,例如 GOTO(转移)语句,就可以不管 DO WHILE 语句中的条件而把控制传送到 DO WHILE 程序块之外去。还有,与简单的 DO 程序块不同,在一个 DO WHILE 程序块的最外层不能含有说明。但是,在 DO WHILE 程序块中可嵌套简单的 DO 程序块,而在简单的DO 程序块中可以含有说明。

例 5.10

```
AMOUNT = 1;
DO WHILE  AMOUNT <= 3;
     AMOUNT = AMOUNT + 1;
END;
```

在上例中,语句 AMOUNT = AMOUNT + 1 执行了 3 次。当程序的控制转出 DO WHILE 程序块时,AMOUNT 的值为 4。

3. GoTo语句

GOTO 语句的格式为:

```
     GOTO 标号;
```

GOTO语句通过把控制直接转向一个带标号的语句来改变程序的执行顺序。在GOTO 语句中引用了这个带标号语句的标号。接收控制的语句要写成如下的形式:

标号：语句；

此处“标号”是识别该语句的标号。

使用GOTO语句要注意下例几点：

- 标号的作用域影响到某些语句的合法性；
- 若使用GOTO语句来从一个过程中退出，则GOTO语句中的标号必须是该主程序模块的最外层中的一个语句的标号；
- 在必须使用GOTO语句时再使用它。对于绝大多数需要转移控制的场合，交换使用DO，DO WHILE，DO CASE，IF或过程调用会更好一些。在一个程序中滥用GOTO语句会使程序难以理解，纠错和维护。

4．CALL语句和RETURN语句：

CALL指令的格式为：

CALL 过程名（参数表）；

CALL指令的功能是用来调用前面已定义过的“过程”。列在语句“参数表”中的为实际参数。CALL语句将“参数表”中的实际参数传送给“过程名”指定的“过程”，然后执行此“过程”，最后再将控制返回到CALL语句后面的语句上去。与GOTO语句不同，CALL语句要将控制反回到程序断开的地方。

RETURN语句在过程体内部使用，它使程序返回到这个过程被调用的那一点处。

六、过　程

一个“过程”是一段PL/M代码，在书写这段代码的地方只对它作说明而并不执行，然后再由程序中的其它部分“调用”它。这种调用“启动”该过程，使得离开正常的顺序而去执行这段过程代码：即程序控制从调用点转移到这段过程代码的开头，接着执行这段代码，然后再从这个过程中退出；同时程序控制恰好转回到调用点的下一步。

过程的使用构成了模块式程序设计的基础，促进了程序库的形成和使用。过程提供了一种将大的、复杂的程序分成小的程序来设计的机构，并使得为一个问题编制的“过程”可为其它多个程序所共享。

下面从两方面来简单介绍一下PL/M-86的过程。

1．过程说明：

过程与变量一样，必须对它加以说明。一个过程说明由三部分组成：一个PROCEDURE语句。一个构成“过程体”的语句序列和一个END语句。这三部分的格式如下：

```
名字：PROCEDURE[(参数表)][类型][表征]；
    语句1；
    语句2；
    ……
    语句n；
END [名字]；
```

在名字和PROCEDURE之间可插入对此过程的注释。注意：注释要用/*…*/号括起来。

例 5.11 TP-86A监控程序中用来显示地址，数据的过程：

```
KB$DISPLAY:PROCEDURE (PTR, FIELD, PROMPTS) ;
        DECLARE PTR POINTER, (FIELD, PROMPTS, T) BYTE,
          DISPLAY BASED PTR (1) BYTE;
        IF FIELD=ADDR$FIELD THEN
          OUTPUT (KB$STAT$PORT) =94H; /*地址字段*/
        ELSE
            OUTPUT (KB$STAT$PORT) =90H; /*数据字段*/
        DO I=0 TO 3;
            T=DISPLAY (3-I) ; /*倒着显示*/
            IF PROMPTS>I THEN T=T OR 80H;
            OUTPUT (KB$DATA$PORT) =T;
        END;
      END;
```

请用户自已分析一下这个过程中各语句是什么语句。

（一）参　数：

形式参数是在过程说明中加以说明的无基标量变量，它们的标识符是在PROCEDURE语句的参数表中给定的。表中各个标识符之间短号隔开，并用园括号括起来，见前面的KB$DISPLAY过程中的（PTR，FIELD,PROMPTS)。在参数表中不允许出现下标或成分标识符。如果一个过程没有形式参数，则把参数表（连同园括弧）从PROCEDURE语句中省略掉。

每个形式参数都必须在一个DECLARE语句中被说明为是一个无基标量变量，而这个DECLARE语句位于过程体的第一个可执行语句之前。见前面KB$DISPLAY过程中的DECLARE语句。

调用一个带有形式参数的过程时，在CALL语句或函数引用中含有一个实际参数的表。每个实际参数是一个表达式，并且在过程开始执行之前先把实际参数的值赋给过程中与之相对应的形式参数。

（二）有类型的过程和无类型的过程，

PL/M-86提供两类过程：有类型的过程和无类型的过程。

上面给出的KB$DISPLAY过程是一个无类型的过程。因为在PROCEDURE语句中没有给出任何类型，它不回送值。对于无类型过程的调用，只要在CALL语句中写上它的名字即可。如调用上面的KB$DISPLAY时写CALL KB$DISPLAY就行了。无类型的过程能接收参数，但不能反回一个值。

有类型的过程在它的PROCEDURE语句中有一个类型，这个类型或者是BYTE，或者是ADDRESS，而且它送回一个具有与PROCEDURE语句中相同类型的值。对它的调用是通过在一个表达式中使用它的名字而实现的。这是一种特殊的变量引用，叫做“函数引用”。函数引用可以是一个表达式的操作数。

当在运行期间处理这个表达式时，这个函数引用的出现使该过程被执行。而函数引用本身由该过程回送的值代替，然后计算这个表达式并以正常的次序继续执行程序。与

无类型的过程一样，有类型的过程可以具有参数，对它们的处理方式与无类型的方式相同。

（三）过程出口：

对一个过程的执行有下列三种结束执行的方法：

- 通过执行过程体内部的一个 RETURN 语句，一个有类型的过程必须包含一个带有表达式的 RETURN 语句。
- 到达该过程说明末尾的 END 语句。
- 通过执行 GOTO 语句转到该过程体之外的某个语句。这个 GOTO 语句的目标必须位于主程序模块的外层，应尽量少地使用这种方式。

RETURN 语句有两种格式：

```
RETURN;
RETURN 表达式。
```

第一种格式用于无类型的过程；第二种格式用于有类型的过程。表达式的值就是该过程回送的值。在计算时，这个值的类型是 PROCEDURE语句中给定的类型——意即如果得到一个 BYTE 型的结果，而过程是 ADDRESS 型的，则给这个值加上 8 个高位的“0”（零）以构成一个 ADDRESS 型的值。如果得到一个 ADDRESS 型的结果，而过程是 BYTE 型的，则把结果的高 8 位丢掉使之成为一个 BYTE 型的值。

（四）过程体：

过程体内的语句可以是任何正确的 PL/M 语句，也可以是调用语句和嵌套的过程说明。

例 5.12 一个有类型的过程说明：

```
AVG: PROCEDURE (X,Y) ADDRESS;
    DECLARE (X,Y) ADDRESS;
    RETURN (X+Y)/2;
END AVG;
```

这个过程可以象下面这样调用：

```
LOW=3;
HIGH=4;
MEAN=AVG (LOW, HIGH),
```

它的作用是把值 3 赋给 MEAN（因为被二除的过程体中得到一个整数结果）。

例5.13 一个没有类型的过程：

```
AOUT: PROCEDURE (ITEM);
    DECLARE ITEM ADDRESS;
    IF ITEM>=0FFH THEN COUNTER=COUNTER+1;
    RETURN;
END AOUT;
```

其中 COUNTER 是一个在本过程之外被说明的变量——即一个“全程”变量。这个过程可以象下面这样来调用：

```
CALL AOUT (UNKNOWN) ;
```

如果变量UNKNOWN的值大于或等于0FFH，则COUNTER的值加1。

例5.14 有基变量的一个重要用途：

```
SUM$ARRAY: PROCECURE (PTR,N)BYTE;
    DECLARE PTR ADDRESS;
        ARRAY BASED PTR (I)BYTE,
        (N, SUM, I) BYTE;
    SUM=0;
    DO I=0 TO N;
        SUM=SUM+ARRAY (I) ;
    END;
    RETURN SUM;
END SUM$ARRAY;
```

这个过程回送的是一个 BYTE 型数组的前 N+1 个元素（从第 0 个到第N个）之和，这个数组是由 PTR 指示的。

如果要利用这个过程对名叫 PRICE 的有 100 个元素的 BYTE 型数组元素求和，并把所得的和数赋给变量TOTAL，就写：

```
TOTAL=SUM$ARRAY(@ PRICE, 99);
```

这里 @PRICE 是该数组的起始地址，即第一个元素（下标为 0 的元素）的单元 地 址。

5.2 ASM-86 汇編語言簡介

INTEL 公司为 8086 提供的 ASM-86 及 MCS-86 这两个汇编程序是兼容的，后者是增加了很强的宏功能的汇编程序。我们 为 8086 的汇编语言专门提供了 一 本 资 料。叫 MCS-86 宏汇编语言参考手册，这里简单介绍一下 ASM-86 的 目的只是为用户阅读监控程序中部分汇编语言子程序用的，以避免翻阅内容庞多的 MCS-86 宏汇 编 语 言参考手册。如果用户要在 TP-86A 上用汇编语言开发自已的应用软件，本章的内容是远远不够的，那时可参考 MCS-宏汇编语言参考手册。其中，我们对 8086 指令系 统 作了尽可能完整的介绍。

对于熟悉 8086CPU 结构的高级程序员来说，最好是 用 ASM-89 来编制程序，这样可以充分利用 CPU 的特性。 PL/M-86 高级语言是不能利用 CPU 的这些特性的。对于 8086 CPU 来说，在 ASM-86 中可利用的列性有：软件中断，WAIT和ESC指令及段寄存器的控制。

用ASM-86 写的程序比用 PL/M-86里的程序占用的存储空间小且执行速度快，这是由于编译程序对所有情况产生的指令序列相同；而用汇编语言可针对具体情况编出最佳的指令序列。

例如：把一个数组中的元素相加起来并把结果置入存储器的变量中。从分析PL/M-86 编译程序产生的目标看出，PL/M编译程序的作法是：把下一数组元 素 移 到 一个寄

存器中去，然后再把这个寄存器的元素值加到存储器的变量中去。若程序员用汇编语言ASM-86来编程序，则大可不必这样做。通常是，选用一个寄存器来存和，再把数组元素加到此寄存器中去，等数组元素加完后，再把此寄存器中的结果“数组元素和”存入存储器的变量中去。这样，对于每一个数组元素可节约一条指令。

ASM-86除去具有一般汇编语言的特点外，它还有自已的独到之处，即具有高级语言的数据结构。这就是使得用ASM-86汇编语言写程序比其它汇编语言容易，并且允许在一个程序中使用PL/M-86编的过程和ASM-86编的过程。

从程序员的角度来看，ASM-86汇编语言简化了机器指令系统。例如：传送指令MOV的机器语言格式虽然有28种不同的类型，程序员可以只写一种形式，即：

MOV destination, source

其中destination为目的操作数；source为源操作数。汇编程序将要按照源和目的操作数的属性来产生正确的机器指令。ASM-86汇编程序还要广泛地检查操作数定义的兼容性，并且用户在指令中用的操作数进行比较，以找出许多共同的错误。

一、语　句

例5.15　ASM-86语句举例

```
; THIS STATEMENT CONTAINS A COMMENT ONLY
MOV     AX, [BX+3]                          ; 典型的指令。
    MOVAX,           [BX+3]                 ; 空格无效。
MOV     AX,
&       [BX+3]                              ; 延续的指令。
ZERO    EQU     0                           ; 简单的ASM-86定向语句。
CUR_PROJ EQU  PROJECT [BX] [SI]             ; 较复杂的定向语句。
THE_STACK_STARTS_HERE SEGMENT               ; 长标识符
TIGHT_LOOP: JMP TIGHT_LOOP                  ; 标号语句。
MOV ES: DATA_STRING [SI], AL                ; 段取代前缀
WAIT: LOCK XCHG AX, SEMAPHORE               ; 标号和LOCK前缀
```

例5.15中给出了ASM-86语句的格式，可以看出它们是松散的语句，而这有助于程序员提高程序的易读性，从而减少错误。

变量和标号名最多可达31个符号长，并不限制为字母和数字，特别是可以用“—”横杠来改善长名字的可读性。在标识符间可以自由地插入空格。一条汇编语句可分布在许多行中。

ASM-86语句可分为指令语句和定向语句两种。定向语句就是在其它汇编语言中常称为伪指令的语句。汇编语言中的指令与机器语言中的指令是有区别的。汇编程序把程序员写的指令译成机器指令。每条ASM-86指令产生一条机器指令，但随着在ASM-86指令中所写操作数不同而产生不同的机器指令。

例如：MOV BL, 1指令将产生一个使一字节的立即数传送到寄存器去的机器指令。

MOV TERMINAL-NO, BX 指令将 BX 寄存器中的字传送到标号 TERMINAL 指定的相邻两存储单元中去。对于程序员来说，上面的两条传送指令都是把源中的操作数传送到目的中去。

ASM-86 汇编语言的指令具有如下的格式，

(label:) (prefix) mnemonic (operand(s)) (;comment)

这里 Label 为该语句的标号，括号表示根据需要标号可有可无；prefix为前缀，括号亦表示根据需要可有可无；mnemonic 助记符指的是指令操作码的助 记 符，而 (operand (s))操作数对于不同的指令操作数可有一个或两个，或根本没 有。comment 注释为程序员自已加的，可有可无。“:”和“;”为分界符，只能用它们，而且不可互换位置，也不可用其它符号代替。程序员在编程时不写出括号，语句格式中的括号只表示括号中的内容为任选项。操作数之间要用逗号分开。

ASM-86汇编程序中大约有20个“定向语句”（伪指令），其格式如下：

(name) mnemonic (operand(s)) (; comment)

name 为名字标号。有些定向语句前面必须有名字，而另一些定向语句前面 则 禁止写名字。mnemonic 为定向语句的助记符，汇编程序靠它来识别不同功能的定向语句，其作用与关键字差不多。Operand(s) 为定向语句中要求的操作数，若有两个操 作 数，则必须用括号分开。定向语句中最后一个字段为注释，它的前面必须有分界符“；”(分号）。

在这些定向语句中常用的有：

- 定义过程的定向语句，其助记符为PROC；
- 给变量分配存储空间的定向语句DB，DW，DD；
- 给一个说明名或表达式赋值的定向语句EQU；
- 定义段边界的定向语句SEGNENT和ENDS；
- 强迫指令和数据定位在字边界处的定向语句EVEN。

二、常　数：

在 ASM-86 汇编语句中可使用二进制，十进制，八进制和十六进制的常数，如

例 5.16　ASM-86常数

```
MOV         STRING [SI], 'A'     ; 字符
MOV         STRING [SI], 41H     ; 与字符 A 等效的十六进数。
ADD         AX, 0C4H             ; 16进制数必须由数字打头。
OCTAL_8     EQU 100              ; 八进制数
OCTAL_9     EQU 10Q              ; 八进制数的另一种表示法。
ALL_ONES    EQU11111111B         ; 二进制数。
MINUS_5     EQU -5               ; 十进制数。
MINUS_6     EQU -6D              ; 十进制数的另一种表示法。
```

ASM-86 汇编程序还可以对这些常数进行基本的算术运数，但要求所有的数 必 须 为 整数，并必须用16位（二进制位）来表示，负数用补码来表示，字符常数要用单引号括起

来，当用它来初始化存储区时。字符最多达255个。当用字符常数来作为立即操作数时，以便与目的操作数的长度相适应，字符常数的长度可为一字节或两字节。

二、数据的定义

对于绝大多数ASM-86程序来说，在程序的开头部分总是先对该程序中要使用的变量进行定义工作。在ASM-86中可用三条定向语句来定义三种不同单位的数据。如

例 5.17 ASM-86数据的定义

```
A_SEG     SEGMENT
ALPHA     DB      ?           ;
BETA      DW      ?           ;  ALPHA,BETA,GAMMA,DELTA不被初始化。
GAMMA     DD      ?           ;
DELTA     DB      ?           ;
EPSILON   DW      5           ; EPSILON 单元中包含05H.
A_SEG     ENDS
B_SEG     SEGMENT  AT 55H; 指定的基地址
IOTA      DB      'HELLO'     ; 含有48, 45, 4C, 4C, 4FH
KAPPA     DW      'AB'        ; 含有4241H
LAMBDA    DD      B_SEG       ; 含有0000, 5500H
MU        DB      100DUP0     ; 含有(100X) 00H
B_SEG     ENDS
```

变量	属性			算子	
	段	位移地址	类型	长度	大小
ALPHA	A_SEG	0	1	1	1
BFTA	A_SEG	1	2	1	2
GAMMA	A_SEG	3	4	1	4
DELTA	A_SEG	7	1	1	1
EPSILON	A_SEG	8	2	1	2
IOTA	B_SEG	0	1	5	5
KAPPA	B_SEG	5	2	1	2
LAMBDA	B_SEG	7	4	1	4
MU	B_SEG	11	1	100	100

DB —定义字节

DW—定义字

DD —定义双字

这些定向语句告诉汇编程序要分配多少存贮单元以及这些单元中的初值应该是什么。若有初值的话，汇编程序就把带在定向语句操作数段中的初值填入这些单元。

ASM-86 汇编程序在汇编任一汇编源程序时，总要监视三个属性：段，偏移地址和

类型。段指的是包含该变量的段。偏移地址指的是从变量所在段的起始处到该变量所在处的距离，此距离以字节计算。类型代表变量的分配单位(1=字节，2=字，4=双字)。当源程序中指令引用某个变量时，ASM-86 汇编程序将利用这些属性去决定产生什么样形式的机器指令代码。如果变量的属性与其在指令中的用途相矛盾，那么 ASM-86 就会产生一个出错信息。例如：当把一个变量定义的字加到字节寄存器中去时就会产生出错信息。譬如：MOV [BX]，5 就是错的，因为汇编程序不知道 [BX] 指的是字节，字还是双字，要用下面的算符来提供属性信息：字节用 BYTE PTR，字用 WORD PTR，双字用 DWORD PTR。若把上面的错误指令改写成：MOV WORD PTR [BX]，5时，汇编程序就能把一个字 0005 传送到 [BX] 指定的内存单元中去。

除去属性信息外，ASM-86 汇编程序还提供两个算子：LENGTH（长度）和 SIZE（大小）。当用户在自已源程序的指令行中使用这两个算子之一时，汇编程序将分别作如下的工作：

- 当指令中有算子 LENGTH 时，它使 ASM-86 汇编程序能归还被一个数组所占的存贮单位（字节，字或双字）的数目。
- 当指令中有算子 SIZE 时，它使 ASM-86 汇编程序归还被一个变量或数组所占的总字节数。

这些属性和算子的作用是：当变量的属性改变时，如把一个字节数组变为字数组时，用户不必重写构成某程序的指令序列，而只要重新汇编一次就行了。ASM-86 属性及属性算子的用法见例5.18

例 5.18

```
; 求表中内容之和存入AX,
TABLE      DW      50 DUP(?)
; 注意下面的指令序列还可用于求下面DB，DW定义的表的内容之和。
; TABLE    DB      25 DUP(?)
; TABLE    DW      118 DUP(?), ETC
           SUB     AX, AX              ; 清和。
           MOV     CX, LENGTH TABLE;   循环计数。
           MOV     SI, SIZE TABLE      ; 指向表尾处的下标。
ADD_NEXT:  SUB     SI, TYPE TABLE      ; 予备的一个元素
           ADD     AX, TABLE [SI]      ; 加元素;
           LOOP    ADD_NEXT            ; 直到CX=0
           ; AX中包含有所求的和。
```

四、记　录

ASM-86 汇编程序提供了在一个字节或字中用符号定义个别位及位串的手段。用这种方式定义的数据结构称为一个记录。记录中每个指定的位串（包括可由一位组成的位串）称为一个字段。在汇编语言中提供记录提高了存储器的利用效率。同时改进了程序的可读性及减少了笔误的可能性。定义一个记录不需要分配存储器，只是通过记录来告

诉汇编程序，让汇编程序对指定的单元（字或或节）中种各位段进行分配。

ASM-86汇编程序利用记录来为 TEST（测试）、AND（逻辑与）、OR（逻辑或）等8086指令产生一个所需的“立即异蔽字”。此外，记录还可用来对移位和循环指令产生一个用于控制移位或循环次数的“立即计数值”。例5.19给出使用记录的例子。

例 5.19

```
            EMP_BYTE DB?              ；未初始化的1字节，
            ；BIT DEFINITIONS：
            ；    7-2   ：             雇用的年限
            ；      1   ：             性别（1 为女性）
            ；      0   ：             状态（1 为免税）
            EMP_BITSRECORD            ；在此定义记录
            &        YRS_EMP：6，
            &        SEX：1，
            &        STATUS：1
            •
            •
            •
            ；选择雇用10年以上的不免税的女雇员，
            MOV      AL，EMP_BYTE     ；保持初值不动，
            TEST     AL，MASK SEX     ；是女的吗？
            JZ       REJECT           ；否，退出
            TEST     AL，MASK STATUS；是不免税的吗？
            JNZ      REJECT           ；否，退出
            SHR      AL，CL           ；把年限分离出来
            CMP      AL，11           ：大于等于10年吗？
            JL       REJECT           ；不大于，退出
            ；处理所选出的雇员情况
            •
            •
            •
REJECT：    ；处理要解雇职员的情况，
            •
            •
            •                         ；这里利用记录来获得
            MOV      CL，YRS_EMP      ；移位次数的计数值。
```

五、结　　构

ASM-86 汇编程序的“结构”如同一张映象图或一个样板，用它来给一个“字段”

的集合指定名字和属性（长度，类型等等）。用DB，DW和DD定向语句来定义“结构”中的每个字段，但不为“结构”分配存储空间，只有当指令中引用一个字段名并给出一个基础时，结构就与存储器中的特定区域联系起来。汇编程序利用基值来定位一个结构。可用变量名或一个基址寄存器（BX或BP）来指定这个基值。例5.20给出了“结构”用法的例子。

例 5.20

```
EMPLOYEE        STRUC
    SSN         DB 9    DUP(?)  ┐
    RATE        DB 1    DUP(?)  │定义“结构”EMPLOYEE（雇员）
    DEPT        DW 1    DUP(?)  │
    YR_HIRED    DB 1    DUP(?)  ┘
    EMPLOYEE    ENDS
MASTER          DB 12   DUP(?)
TXN             DB 12   DUP(?)
；将MASTER中的RATE改为TXN中的值，
                MOV     AL，TXN.RATE
                MOV     MASTER.RATE，AL
；假定BX指向与EMPLOYEE“结构”具有同样格式的数据区，将SSN的第二个
；数字置为0
                MOV     SI，1    ；第二个数字的索引值，
                MOV     [BX].SSN[SI]，0
```

另外，结构中的某个字段本身又可以是一个结构，这样就会写出许多更加复杂的数据形式。

结构特别适用于：

- 对一个元件的多重缓冲器：
- 表处理；
- 栈寻址。

六、寻址方式：

参看图5.5中8086的寄存器。例5.21给出了8086寻址方式的实例。

例 5.21

```
ADD     AX，BX              ；寄存器←寄存器
ADD     AL，5               ；寄存器←立即数
ADD     CX，ALPHA           ；寄存器←存贮器（直接）
ADD     ALPHA，6            ；存贮器（直接）←立即数
ADD     ALPHA，DX           ；存贮器（直接）←寄存器
ADD     BL，[BX]            ；寄存器←存贮器（寄存器间接）
ADD     [SI]，BH            ；存贮器（寄存器间接）←立即数
```

```
ADD     [BP], ALPHA, AH           ; 存贮器（基址的）←寄存器
ADD     CX, ALPHA [SI]            ; 寄存器←存储器（变址的）
ADD     ALPHA [DI+2], 10          ; 存贮器（变址的）←立即数
ADD     [BX], ALPHA [SI], AL      ; 存贮器（基变址的）←寄存器
ADD     SI, [BP+4] [DI]           ; 寄存器←存贮器（基变址的）
IN      AL, 30                    ; 直接端口
OUT     DX, AX                    ; 间接端口
```

汇编程序把用方括号括起来的 BX、BP、SI 或 DI 解释为一个基寄存器或变址寄存器，并用它们来构造一个存储器操作数的有效地址 EA。没有用方括号起来的寄存器名被认为是寄存器本身作为操作数。

下面给出几种典型的 ASM-8 代码，用它们来说明如何存放一个数组和结构，并指出汇编程序用那种寻址方式来产生机器指令。

- 若 ALPHA 是一个数组，那么 ALPHA [SI] 是由 SI 变址指定的元素：ALPHA[SI+1]是变址指定的下一个字节，这种情况是变址寻址。
- 若 ALPHA 是一个结构的基地址，BETA 是结构中的一个字段；那么 ALPHA.BEIA 将选择 BETA 字段。这种情况为直接寻址。
- 若寄存器 BX 中包含有一个结构的基地址，且BETA 是结构中的一个字段，那么 [BX].BETA 指定 BETA 字段。这种情况为基址寻址。
- 若 BX 寄存器中包含有一个数组的地址，则 [BX] [SI]指的是由SI（基变址寻址）变址指定的元素。
- 若 BX 寄存器指向一个结构，而该结构的 ALPHA 字段本身又是一个结构，那么 [BX]. ALPHA. BETA 指的是 ALPHA 子结构的 BETA 字段。这种情况为基址寻址。
- 若 BX 寄存器指向一个结构，且该结构的 ALPHA 字段是一个数组，还有 ALPHA 数组的每一个元素是一个结构，那么 [BX]. ALPHA [SI+3]. BETA 指的是：由 [SI+3] 变址指定的 ALPHA 数组元素中的字段 BEIA。这种情况是基变址寻址。

注意：对于上述情况可用 DI 代替 SI：可用 BP 代替BX。没有段取代前缀的 SP 表达式指的是当前堆栈段，而没有段取代前缀的 BX 表达式指的是当前数据段。

关于 8086 CPU 的寄存器与标志示于图 5.5 中。

七、段控制

程序员可以把一个 ASM-86 源程序组织成为一系列指定的段，这些段都是些逻辑段；它们最终要被映象到 8086 存储器物理段中去。通常，只有当程序定位时才完成这个映象工作。ASM-86 汇编程序提供了两条定向语句：

- SEGMENT 称为“段”定向语句，程序员可用它来指定一个段的起始点。
- ENDS 称为“段结束”定向语句，程序员用它来指定一个段的末尾。

所有位于 SEGMENT 和 ENDS 之间的数据和指令是指定段的一部分。在一个程序

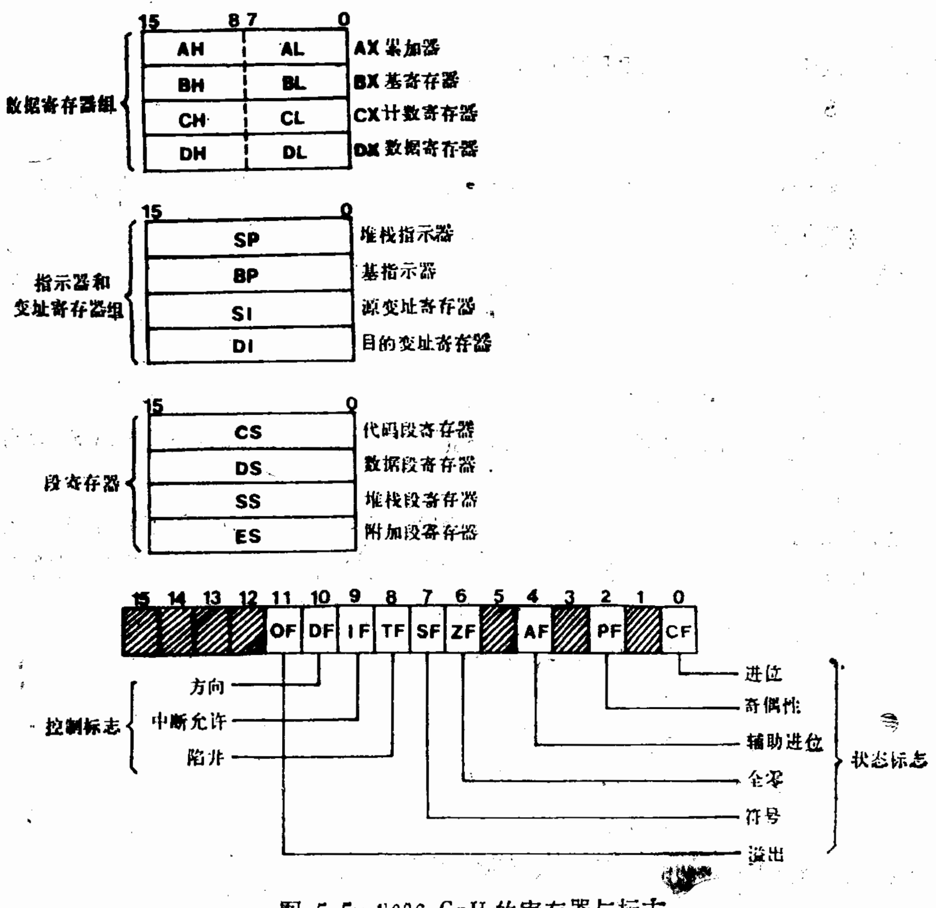


图 5.5 8086 CpU 的寄存器与标志

中，变量通常定义在在一个或两个段中：栈空间分配在另一段中：指令代码在第三或第四段中。对于小程序来说，尽量将一个完整的程序写在一个段中。若能这样，那么所有的段寄存器中将包含有同一个基地址。换句话说，就是存储器段将完全地互相复盖。大程序可以分成若干段。

通常，程序中的第一条指令用来建立段名字和段寄存器之间的对应关系，然后将其对应的基地址取入每个段寄存器。ASSUME 定向语句告诉汇编程序，在运行时寄存器中的地址是什么。汇编程序检查每条存储器指令中的操作数，确定该操作数在哪个段中及对应该段的段寄存器中包含的段地址。例5.22给出对段的设定举例。

例 5.22

```
DATA__SEG     SEGMENT
  ；数据定义在此进行：
DATA__SEG     ENDS
STACK__SEG SEGMENT
  ；为栈和标号分配 100 个字，
  ；将初始的栈顶地址取入 SP，
      DW100DUP (? )
```

```
STACK TOP LABEL WORD栈顶标号字，
STACK_SEG      ENDS
CDDE_SEG      SEGMENT
；给汇编程序最初的寄存器与段的对应关系，注意：在此程序中开始时附加段完
；全复盖数据段。
ASSUME CS：CODE_SEG，
&         DS：DATA_SEG，
&         ES：DATA_SEG，
&         SS：STACK_SEG
START：；此处为该程序的起点，LOC-86 程序将置 JMP 到地址为FFFF0H的单元
        ；中去，
  ；加载段寄存器，由于系统复位将 CS 置为 FFFFH，所以 CS 不被加载，在此的
  ；长转移IMP指令中的地址修改成为代码段的地址，由于没有立即数到段寄存器
  ；的传送 MOV 指令，故都是从 AX 中取数到段寄存器中去。
            MOV   AX，DATA_SEG
            MOV   DS，AX
            MOV   ES，AX
            MOV   AX，STACK_SEG
            MOV   SS，AX
；令栈指示器指向最初的栈顶，（对称栈底）
            MOV   SP，  OFFSET STACK_TOP
；现在段已编址好了。
；主程序代码在此编写，
CODE_SEG              ENDS
；汇编下条语句的末尾 ENDS 并告诉定位程序“LOC-86”被定位的程序的起始地址
；在何处。
            END START
```

注：LOC-86为定位程序。

现在通过例子来说明 ASSUME 定向语句的另一作用。例如：如果程序员指令中使用了寄存器 BP。那么和缺省时的情况一样。8086 CPU 希望这个存储器操作数将被定位在 SS 段寄存器指定的段中——即在当前的堆栈段中。但是，程序员可以选择使用BP去寻址一个在当前数据段中的变量，这时应由 DS 段寄存器给出段地址。ASSUME定向语句使得汇编程序能发现这种情况并自动地产生所需要的段取代前缀字节。

ASM-86 汇编程序与可重定位程序 LOC-86及连接程序 LINK-86一起能完成更多的对段进行管理的功能。如：不同的逻辑段可以组合到同一物理段中去。同一物理段可以分配给不同的逻辑段。这样，不同的程序就可使用不同的变量名和标号名去访问“公共”的存区。

八、过　　程

关于“过程”，前文PL/M-86一节已作介绍，这里只说明一点，可用ASM-86及PL/M-86来编写程序中所需要的“过程”。事实上用其中一种语言写的过程可被用另一种语言写的程序调用。ASM-86过程与PL/M-86过程的这种兼容性可简化对复杂程序的设计，并使得能在程序中的任何地方去访问公共子程序。

九：8086指令系统：

为使用户查找方便，表5.3给出按英文字母顺序排列的指令助记符及该指令在表5.4中的序号；表5.4给出了8086指令系统中各指令的汇编语言助记符格式。表中各指令是按功能分类的。

关于8086各条指令的详细介绍和举例，请参看MCS-86宏汇编语言参考手册的第五章，这里只是为阅读监控程序中的汇编语言写的子程序提供的指令表。

表 5.3　　　　指　令　索　引　表

按字母顺序的指令助记符	指令序号	按字母顺序的指令助记符	指令序号	按字母顺序的指令助记符	指令序号
AAA	48	CWD	94	JAE/JNB	183
AAD	92	DAA	49	JB/JNAE	175
AAM	83	CAS	74	JBE/JNA	176
AAS	73	DEC	62	JC	188
ADC	39	DIV	84	JCXZ	193
ADD	33	ESC	207	JE/JZ	172
AND	125	HLT	205	JG/INLE	182
CALL	157	IDIV	88	JGE/JNL	181
CBW	93	IMVL	79	JL/JNGE	173
CLC	198	IN	21	JLE/JNG	174
CLD	201	INC	45	JMP	162
CLI	203	INT	194	JNC	189
CMC	199	INTO	196	JNE/JNZ	180
CMP	67	IRET	197	JNO	186
CMPS	150	JA/JNBE	184	JNP/JPO	185

(表 5.3 续)

按字母顺序的指令助记符	指令序号	按字母顺序的指令助记符	指令序号	按字母顺序的指令助记符	指令序号
JNS	187	NEG	65	SAHF	30
JO	178	NOP	210	SAL/SHL	97
JP/JPE	177	NOT	95	SAR	105
JS	179	OR	136	SBB	56
LAHF	29	OUT	23	SCAS	151
LDS	27	POP	15	SHR	101
LEA	26	POPF	32	STC	200
LES	28	PUSH	12	STO	202
LOCK	209	PUSHF	31	STI	204
LODS	152	RCL	117	STOS	153
LOOP	190	RCB	121	SBU	50
LOOPE/LOOPZ	191	REP	148	TEST	131
LOOPNE/LOOPNZ	192	REPE/LEPZ	155	WAIT	206
MOV	1	REPNE/RETNZ	156	XCEG	18
MOVS	149	RET	168	XLAT	25
MOVSB/MOVSW	154	ROL	109	XOR	142
MVL	75	ROR	113		

表5.4　　8086 的指令表

一、数据传送指令

	序号	助记符	指令代码	字节	时钟	标志 ODITSZAPC	操作
传送指令组	1	MOVARRAY[SI],AL	1010 001W addr 低位 addr 高位	3	10		累加器内容→存贮器操作数
	2	MOV AX,TEMP-RESULT	1010 000W addr 低位 addr 高位	3	10		存贮器操作数→累加器
	3	MOV AX,CX	1000 10 dW mod reg r/m	2	2		寄存器→寄存器
	4	MOV BP,STACT-TOP	1000 10 dW mod reg r/m × ×	2—4	8+EA		存贮器→寄存器
	5	MOV COUNT[DI],CX	1000 10 dW mod reg r/m × ×	2—4	9+EA		寄存器→存贮器
	6	MOV CL,2	1011 W reg data 低位 data 高位(W=1)	2—3	4		立既数→寄存器

（续）

	序号	助记符	指令代码	字节	时钟	标志 ODITSZAPC	操作	
传送指令组	7	MOVMASK[BX][SI]，2CH	1100 011 W mod 000 r/m × × data(低位) data(高位)(w=1)	3—6	10+EA		立既数→存贮器	
	8	MOV DS, CX	1000 1110 mod 0 reg r/m	2	2		寄存器→段寄存器	reg段寄存器 00 ES 01 CS 10 SS 11 DS
	9	MOV DS, SEGMENT-BASE	1000 1110 mod 0 reg r/m × ×	2—4	8+EA		存储器→段寄存器	
	10	MOV BP, SS	1000 1100 mod 0 reg r/m	2	2		段寄存器→寄存器	reg同上
	11	MOV [BX] SEG—SAUE, CS	1000 1100 mod 0 reg r/m × ×	2—4	9+EA		段寄存器→存贮器	
进栈指令	12	PUSH SI	010 10 reg	1	11		寄存器内容进栈((SI)-1，(SP)) (SP)-2→SP	
	13	pUSH ES	000 reg 110	1	10		段寄存器内容传送到堆栈中去	

令 组	14	PUSH RETURN—CODE [SI]	1111 1111 mod 110 r/m × ×	2—4	16+EA		存贮器/寄存器操作数进栈
出	15	POP DX	0101 1 reg	1	8		寄存器操作数退栈
栈	16	POP DS	000 reg 111	1	8		段寄存器退栈
指 令 组	17	POP ARAMETER	1000 1111 mod 000 r/m × ×	2—4	17+EA		存贮器/寄存器操作数退栈 ((SP)+1:(SP))→存储器/寄存器 (PS)+2→(SP)
交	18	XCHG AX, BX	1001 0reg	1	3		寄存器和累加器交换 (BX)←→(AX)
换 指 令 组	19	XCHG SEMAPHORE, AX	1000 011 W mod reg r/m × ×	2—4	17+EA		存贮器/寄存器和寄存器交换
	20	XCHG AL, BL	1000 011 W mod reg r/m	2	4		寄存器和寄存器交换 (AL)←→(BL)
输 入	21	IN AL, 0EAH	1110 010 W 端口地址(0～255)	2	10		从输入端口传送字节/字到AL/AX (固定端口输入)
输 出 指	22	IN AX, DX	1110 110W	1	8		从DX中的端口号到间接寻址，输入字节/字到AL/AX(可变端口输入)
令 组	23	OUT 44, AX	1110 011 W 端口地址(0～255)	2	10		从AL/AX传送字节/字到输出端口 (固定端口输出)

（续）

	序号	助记符	指令代码	字节	时钟	标志 ODITSZAPC	操作
输出输入指令组	24	OUT DX,AL	1110 111W	1	8		从AL/AX传送字节/字输出到DX指定的外设端口号去
换码	25	XLAT ASCII—TAB	1101 0111	1	11		实现一个表查找的换码操作 ((BX)+(AL))→(AL)
地址目标传送指令组	26	LEA BX, [SP][DI]	1000 1101 mod reg r/m × ×	2—4	2+EA		将源操作数的有效地址EA传送到目的操作数中去 EA→BX
	27	LDS SI,DATA SEG[DI]	1100 0101 mod reg r/m × ×	2—4	16+EA		将段地址送到DS中去 偏移地址传送到寄存器中去 (REG)←EA (SI)←EA+2 寄存器为AX,BX,CX,DX,SP,BP,SI,DI之一
	28	LES DI,[BX] TEXT-BUFF	1100 0100 mod reg r/m × ×	2—4	16+EA		将段地址传送到ES中去 偏移地传址送到寄存器中去 寄存器为AX,BX,CX,DX,SP,BP,SI,DI之一
标志传送	29	LAHF	1001 1111	1	4		取标志到AH寄存器中去 (AH)←(SF)(ZF)V(AF)/(PF) V←(CF)
	30	SAHF	1001 1110	1	4		存AH到标志寄存器 (SF)(ZF)∨(AF)∨(PF) ∨(CF)←(AH,

指令组	31	PUSHF	1001 1100	1	10		标志进栈 ((SP)-1:(SP)) ←FLAGS (SP) ←(SP)-2
	32	POPF	1001 1101	1	8		标志退栈 FLAGS←((SP)+1:(SP)) (SP)←(SP)+2

二、算术运算指令

	序号	助记符	指令代码	字节	时钟	标志 ODITSZAPC	操作
加法指令组	33	ADD CX,DX	0000 00 dW mog reg r/m	2	3	× ×××××	寄存器内容加寄存器内容， (CX)+(DX)→CX
	34	ADD DI,[BX]ALPHA	0000 00 dW mod reg r/m × ×	2—4	9+EA	同 上	存贮器/寄存器内容加寄存器内容
	35	ADD TEMP,CL	0000 00 dW mod reg r/m × ×	2—4	16+EA	同 上	寄存器内容加存贮器内容
	36	ADD CL,2	1000 00 sW mod 000 r/m data 低位 data 高位(SW=01)	3—4	4	同 上	立即数加寄存器内容 ((C2)+2)→CL

(续)

	序号	助记符	指令代码	字节	钟时	标志 ODITSZAPC	操作
加法指令组	37	ADD ALPHA,2	1000 00 sW mod 000 r/m × × data低位 data高位(SW=01)	3—6	17+EA	同上	立即数加存贮器/寄存器内容 ((ALPHA)+2)←ALPHA
	38	ADD AX,200	0000 010 W data低位 data高位(W=1)	2—3	4	同上	立即数加累加器内容 (AX)←(AX)+200
带进位加法	39	ADC AX,SI	0001 00 dW mod reg r/m	2	3	同上	寄存器加寄存器加进位位 (AX)←(AX)+(SI)+(CF)
	40	ADC DX,BETA[SI]	0001 00 dW mod reg r/m × ×	2—4	9+EA	同上	存贮器内容加寄存器内容加进位位
	41	ADC ALPHA[BX][SI]DI	0001 00 dW mod reg r/m × ×	2—4	16+EA	同上	寄存器内容加存贮器内容加进位位
	42	ADC BX,256	1000 00 sW mod 010 r/m data 低位 data 高位SW=01	3—4	4	同上	立即数加寄存器内容加进位位 (BX)←(BX)+256+(CF)

指令组	43	ADC GAMMA,30H	1000 00 sW mod 010 r/m × × data 低位 data 高位	3—6	17+EA	同　上	立即数加存贮器内容加进位位 (CF)+(GAMMA)+30H→GAMMA
	44	ADC AL,5	0001 010 W data 低位 data 高位	2—3	4	同　上	立即数加累加器内容加进位位
加1指令组	45	INC CX	0100 0 reg	1	2	×　××××	寄存器内容加1 (CX)+1→CX
	46	INC BL	1111 111 W mod 000 r/m	2	3	同　上	寄存器内容加1 (BL)+1→BL
	47	INC ALPHA[DI][BX]	1111 111 W mod 000 r/m × ×	2—4	15+EA	同　上	存贮器内容加1
加法的ASCII修正	48	AAA	0011 0 111	1	4	U　UU×U×	若((AL)&0FH)>9或(AF)=1 则(AL)←(AL)+6 (AH)←(AH)+1 (A"F")←1 (CF)←(AF) (AL)←(AL)&0FH 加法的ASCII修正

（续）

	序号	助记符	指令代码	字节	时钟	标志 ODITSZAPC	操作
十进制加法修正	49	DAA	0010 0 111	1	4	× ×××××	若((AL)&0FH)>9或(AF)=0 则(AL)←(AL)+6 (AF)←1 若(AL)>9FH或(CF)=1 则(AL)←(AL)+60H (CF)←1 十进制加法修正
减法指	50	SUB CX,BX	0010 10 dW mod reg r/m	2	3	× ×××××	目的寄存器减源寄存器内容
	51	SUB DX,MATH-TOTAL[SI]	0010 10 dW mod reg r/m × ×	2—4	9+EA	× ×××××	寄存器内容减存贮器内容
	52	SUB[BP+2],CL	0010 10 dW mod reg r/m × ×	2—4	16+EA	同上	存贮器内容减寄存器内容
	53	SUB AL,10	0010 110 W data 低位 data低位(W=1)	2—3	4	同上	从累加器减立既数

令 组	54	SUB SI, 5280	1000 00 SW mod 101 r/m data 低位 data高位(SW=01)	3—4	4	同　上	从寄存器减立既数
	55	SUB[BP].BALANCE, 1000	1000 00 sW mod 101 r/m data 低位 data高位(SW=01) × ×	3—6	17+EA	同　上	从存贮器减立既数
带借位的减法指令组	56	SBB BX,CX	0001 10 dW mod reg r/m	2	3	同　上	从目的寄存器减源寄存器减进位位（借位）
	57	SBB DI,[BX] PAYMENT	000 110 dW mod reg r/m × ×	2—4	9+EA	同　上	从寄存器减存贮器减进位位（借位）
	58	SBB BALAWCE,AX	000 110 dW mod reg r/m × ×	2—4	16+EA	同　上	从存贮器减寄存器减进位位（借位）
	59	SBB AX,2	000 111 0 W data 低位 data 高位(W=1)	2—3	4	同　上	从累加器立既数减进位位（借位）
	60	SBB CL, 1	1000 00 SW mod 011 r/m data 低位 data高位(SW=01)	3—4	4	同　上	从寄存器减立既数减进位位（借位）

（续）

	序号	助记符	指令代码	字节	时钟	标志 ODITSZAPC	操作
带借位的减法指令组	61	SBB COUNT[SI],10	1000 00 SW mod 011 r/m × × data 低位 data高位(SW=01)	3—6	17+EA	× ×××××	从存贮器减立即数减进位位（借位）
减1指令组	62	DEC AX	0100 1 reg	1	2	× ×××××	寄存器（16位）操作数减1
	63	DEC AL	1111 111 W mod 001 r/m	2	3	同上	寄存器（8位）操作数减1
	64	DEC ARRAY[SI]	1111 111 W mod 001 r/m × ×	2—4	15+EA	同上	存贮器操作数减1
求补	65	NEG AL	1111 011 W mod 011 r/m	2	8	× ×××××	寄存器内容求补码
	66	NEG MULTIPLIER	1111 011 W mod 011 r/m × ×	2—4	16+EA	同上	存贮器内容求补码
比	67	CMP BX,CX	0011 10 dW mod reg r/m	2	8	同上	寄存器与寄存器比较（减法）结果不回送

数指令组	68	CMP DH,ALPHA	0011 10 dW mod reg r/m × ×	2—4	9+EA	同 上	存贮器与寄存器比较，结果不回送
	69	CMP [BP+2],SI	0011 10 dW mod reg r/m × ×	2—4	9+EA	同 上	寄存器与存贮器比较，结果不回送
	70	CMP BL,02H	1000 00sW mod 111 r/m data 低位 data高位(SW=01)	3—4	4	同 上	立既数与寄存器比较，结果不回送
	71	CMP[BX]RADAR[DI], 3420H	1000 00 sW mod 111 r/m data 低位 data高位(SW=01) × ×	3—6	10+EA	同 上	立既数与寄贮器比较，结果不回送
	72	CMP AL,00010000B	0011 110W dats 低位 data 高位(W=1)	2—3	4	同 上	立既数与累加器比较，法结不回送
减法ASCII码修正	73	AAS	0011 1111	1	4	U UU×U×	减法的ASCIJ修正 若((AL)&0FH)>9或(AF)=1 则(AL)=(AL)-6, (AH)←(AH)-1 (AF)←1,(CF)←(AF) (AL)←(AL)&0FH

（续）

	序号	助记符	指令代码	字节	时钟	标志 ODITSZAPC	操作
减法十进制调整	74	DAS	0010 1111	1	4	U ×××××	减法的十进制调整 若（（AL）&0FH）>9 或（AF）=1 则（AL）←（AL）－6，（AF）←1 若（AL）>9FH或（CF）=1 则（AL）←（AL）－60H （CF）←1
乘法指令组	75	MUL CL	1111 011 W mod 100 r/m	2	70—77	× UUUU×	（8位）寄存器数的乘法
	76	MUL CX	1111 011 W mod 100 r/m	2	118－133	同 上	（16位）寄存器数的乘法
	77	MUL MONTE[SI]	1111 011 W mod 100 r/m × ×	2—4	（76＋83）＋EA	同 上	（8位寄存器数的乘法
	78	MUL BAUD-RATE	1111 011 W mod 100 r/m × ×	2—4	（124－139） ＋EA		（16位）存贮器数的乘法
有附号	79	IMUL CL	1111 011 W mod 101 r/m	2	80—98	同 上	（8位）寄存器有附号数的整数乘法
	80	IMUL BX	1111 011 W mod 101 r/m	2	128—154	同 上	（16位）寄存器有附号数的整数乘法

数的调整乘法	81	IMUL RATE-BYTE	1111 011 W mod 101 r/m × ×	2—4	(86—104) +EA	同　上	(8位) 存贮器有附号数的整数乘法
	82	IMUL RATE-BYTE[BP] [DI]	1111 011 W mod 101 r/m × ×	2—4	(134—160) +EA	同　上	(16位) 存贮器有附号数的整数乘法
ASCII乘法调整	83	AAM	1101 0100 0000 1010	2	83	U ××U×U	ASCII数乘法调整 (AH)←(AL)/0AH (AL)←(AL)%0AH(%为模数)
除	84	DIV CL	1111 011W mod 110 r/m	2	80—90	U UUUUU	(8位)寄存器的除法
法	85	DIV BX	1111 011W mod 110 r/m	2	144—162	同　上	(16位)寄存器数的除法
指令	86	DIV ALPHA	1111 011 W mod 100 r/m × ×	2—4	(86—96) +EA	同　上	(8位)存贮器数的除法
组	87	DIV TABLE[SI]	1111 110W mod 100 r/m × ×	2—4	(150—168) +EA	同　上	(16位)存贮器数的除法

(续)

	序号	助记符	指令代码	字节	时钟	标志 ODITSZAPC	操作
有附号数整数除法指令组	88	IDIV BL	1111 011W mod 111r/m	2	101—112	同上	(8位)寄存器有附号数的整数除法
	89	IDIV CX	1111 011 W mod 111 rm	2	165—184	同上	(16位)寄存器有附号数的整数除法
	90	IDIV DIVISOR-BYTE [SI]	1111 011 W mod 111 r/m × ×	2—4	(107—118 +EA	同上	(8位)存贮器有附号数的整数除法
	91	IDIV[BX] DIVIVISOR-WORD	1111 011 W mod 111 r/m × ×	2—4	(171—190) +EA	同上	(16位)存贮器有附号数的整数除法
ASCII除法调整	92	AAD	1101 0101 0000 1010	2	60	U ××U×U	除法的ASCII调整 (AL)←(AH)*0AH+(AL) (AH)←0
转换指令	93	CBW	1001 1000	1	2		将字节转换成整字 若(AL)<80H则(AH)←0 否则(AH)←FFH
	94	CWD	1001 1001	1	5		将单倍长字转换成双倍字 否(AX)<8000H则(DX)←0 否则(DX)←FFFFH

三、逻辑运算指令

	序号	助　记　符	指令代码	字节	时　钟	标志 ODITSZAPC	操　作
逻辑非	95	NOT AX	1111 011 W mod 010 r/m	2	8		寄存器内容求反 $(AX) \leftarrow (\overline{AX})$
	96	NOT CHARACTRE	1111 011 W mod 010 r/m × ×	2—4	16 + EA		存贮器/寄存器内容求反
算术/逻辑左移指令组	97	SAL/SHL SAL AL,1	1101 000 W mod 100 r/m	2	2	×　　×	寄存器算术/逻辑左移 1 位 CF ← [　　] ←∘ 0
	98	SHL DI,CL	1101 001 W mod 100 r/m	2	8 + 4/bit	同　上	寄存器算术/逻辑左移(CL)位
	99	SHL[BX]OVERDRAW,1	1101 000 W mod 100 r/m × ×	2—4	15 + EA	同　上	存储器逻辑/算术左移 1 位
	100	SHL STORE-COUNT,CL	1101 001 W mod 100 r/m × ×	2—4	20 + EA + 4/bit	同　上	存贮器逻辑/算术左移(CL)位

(续)

	序号	助记符	指令代码	字节	时钟	标志 ODITSZAPC	操作
逻辑右移指令组	101	SHR SI,1	1101 000 W mod 101 r/m	2	2	同上	寄存器逻辑右移 1 位 0 ∘→[ ]→CF
	102	SHR SI,CL	11011 001 W mod 101 r/m	2	8 + 4/bit	同上	寄存器逻辑右移(CL)位
	103	SHR ID-BYTE[SI][BX],1	1101 000 W mod 101 r/m × ×	2—4	15 + EA	同上	存贮器逻辑右移 1 位
	104	SHRINPUT-WDRD,CL	1101 001 W mod 101 r/m × ×	2—4	20 + EA + 4/bit	同上	存储器逻辑右移(CL)位
算术右移指	105	SAR DX,1	1101 000 W mod 111 r/m	2	2	× ××U××	寄存器算术右移 1 位 [→]→CF
	106	SAR DI,1	1101 001 W mod 111 r/m	2	8 + 4/bit	同上	寄存器算术右移(CL)位
	107	SAR N-BLOCKS,1	1101 000 W mod 111 r/m × ×	2—4	15 + EA	同上	存贮器算术右移 1 位

指令组	序号	指令	编码	字节数	时钟数	标志	说明
令组	108	SARN-BLOCKS,CL	1101 001 W mod 111 r/m × ×	2—4	20 + EA + 4/bit	×　××U××	存贮器算术右移(CL)位
左循环移位指令组	109	ROL BX,1	1101 000 W mod 000 r/m	2	2	×　　×	寄存器左循环1位 CF ←
	110	ROL DI,CL	110 1001 W mod 000 r/m	2	8 + 4/bit	同　上	寄存器左循环(CL)位
	111	ROL FLAG-BYTE [DI],1	1101 000 W mod 000 r/m × ×	2—4	15 + EA	同　上	存贮器左循环1位
	112	ROL ALPHA,CL	1101 001 W mod 000 r/m × ×	2—4	20 + EA + 4/bit	同　上	存贮器左循环(CL)位
右循环移位指令组	113	ROR AL,1	1101 000 W mod 001 r/m	2	2	同　上	寄存器右循环1位 → CF
	114	ROR BX,CL	1101 001 W	2	8 + 4/bit	同　上	寄存器右循环(CL)位
	115	ROR RORT-STATU S,1	1101 000 W mod 001 r/m × ×	2—4	15 + EA	同　上	存贮器右循环1位

	序号	助 记 符	指 令 代 码	字节	时 钟	标 志 ODITSZAPC	操 作
右位循指环令移组	116	ROR CMD-WDRD,CL	1101 001 W mod 001 r/m × ×	2—4	20 + EA + 4/bit	同 上	存贮器右循环(CL)位
带进位左循环移位指令组	117	RCL CX,1	110 1000 W mod 010 r/m × ×	2	2	同 上	带进位的寄存器左循环 1 位 ←CF←[ ]←
	118	ROL AL,CL	1101 001 W mod 010 r/ m	2	8 + 4/bit	同 上	带进位的寄存器左循环(CL)位
	119	RCL ALPHA,1	1101 000 W mod 010 r/m × ×	2—4	15 + EA	同 上	带进位的存贮器左循环 1 位
	120	RCL[BP]PARM,CL	1101 001 W mod 010 r/m × ×	2—4	20 + EA + 4/bit	× ×	带进位的存贮器左环循(CL)位
带进位	121	RCR BX,1	1101 000 W mod 011 r/m	2	2	同 上	带进位寄存器右循环 1 位 —CF→[ ]→

右循环移位指令组	122	RCR BL,CL	1101 000 W mod 011 r/m	2	8+4/bit	× ×	带进位的寄存器右循环(CL)位
	123	RCR[BX]STATU,1	1101 000 W mod 011 r/m × ×	2—4	15+EA	同 上	带进位的寄存贮器右循环1位
	124	RCR ARRAY[DI],CL	1101 001 W mod 011 r/m × ×	2—4	20+EA+ 4/bit	同 上	带进位的存贮器右循环(CL)位
逻辑与指令组	125	AND AL,BL	0010 00 dW mod reg r/m	2	3	0 ××U×0	寄存器和寄存器内容逻辑与
	126	AND CX,FLAG-WORD	0010 00 dW mod reg r/m × ×	2—4	9+EA	同 上	存器贮和寄存器内容逻辑与
	127	AND ASCII[DI],AL	0010 00 dW mod reg r/m × ×	2—4	16+EA	同 上	寄存器和存贮器内容逻辑与
	128	AND CX,0F0H	1000 000 W mod 100 r/m data低位 data高位(W=1)	3—4	4	同 上	立既数和寄存器内容逻辑与

	序号	助记符	指令代码	字节	时钟	标志 ODITSZAPC	操作
逻辑与指令组	129	AND BETA,01H	1000 000 W mod 100 r/m × × data(高位)(W=1) data(低位)	3—6	17+EA	同上	立既数和存贮器内容逻辑与
	130	AND AX,01010000B	001 0010 W aata 低位 data高位(W=1)	2—3	4	同上	立既数和累加器内容逻辑与
检测指令	131	TEST SI,DI	1000 010 W mod reg r/m	2	3	0 ××U×0	寄存器同寄存器递辑与，结果不回送
	132	TEST SI,END COUNT	1000 010 W mod reg r/m × ×	2—4	9+EA	同上	存贮器同寄存器逻辑与，结果不回送
	133	TEST AL,00100000B	1010 100 W ata低位 data高位(W=1)	2—3	4	同上	立既数同累加器逻辑与，结果不回送
	134	TEST BX,0CC4H	1111 011 W mod 000 r/m data 低位 data高位(W=1)	3—4	5	同上	立既数同寄存器逻辑与，结果不回送

组	135	TEST RETURN-CODE, 01H	1111 011 W mod 000 r/m × × data(低位) data(高位)(W=1)	3—6	11+EA	同　上	立既数同存贮器逻辑与，结果不回送
逻辑或指令组	136	OR AL,BL	0000 10 dW mod reg r/m	2	3	同　上	寄存器同寄存器逻辑或
	137	OR DX,PORT-ID[DI]	0000 10 dW mod reg r/m × ×	2—4	9+EA	同　上	存贮器同寄存器逻辑或
	133	OR FLAG-BYTE,CL	0000 10 dW mod reg r/m × ×	2—4	16+EA	同　上	寄存器同存贮器逻辑或
	139	OR AL,01101100B	0000 110 W data (低位) data(高位)(W=1)	2—3	4	同　上	立既数同累加器逻辑或
	140	OR CX,01H	1000 000 W mod 001 r/m data(低位) data(高位)(W=1)	3—4	4	同　上	立既数同寄存器逻辑或

	序号	助记符	指令代码	字节	时钟	标志 ODITSZAPC	操作
	141	OR[BX].CMD-WDRD 0CFH	1000 000 W mod 001 r/m × × data(低位) data(高位)(W=1)	3—6	17+EA	同 上	立即数同存贮器逻辑或
	142	XOR CX,BX	0011 00 dW mod reg r/m	2	3	同 上	寄存器与寄存器逻辑异或
逻辑异或指	143	XOR CL MASK-BYTE	0011 00 dW mod reg r/m × ×	2—4	9+EA	0 ××U×0	存贮器与寄存器逻辑异或
	144	XOR MEM-MORD,AX	0011 00 dW mod reg r/m × ×	2—4	16+EA	同 上	寄存器与存贮器逻辑异或
	145	XOR AL,01000010B	0011 010 W data(低位) data(高位)(W=1)	2—3	4	同 上	立既数与累加器逻辑异或
	146	XOR SI,00C2H	1000 000 W mod 110 r/m data(低位) data(高位)(W=1)	3—4	4	同 上	立既数与寄存器逻辑异或

	序号	助记符	指令代码	字节	时钟	标志 ODITSZAPC	操作
令组	147	XOR RETURN-CODE, OD2H	1000 000 W mod 110 r/m × × data（低位） data（高位）(W = 1)	3—6	17 + EA	同上	立即数与存贮器逻辑异或

四、字符串操作指令

	序号	助记符	指令代码	字节	时钟	标志 ODITSZAPC	操作
字符串指令组	148	REP MOVS DEST,SRCE	1111 001Z	1	2		重复(迭代)前缀
	149	MOVS LINE EDIT-DATA	1010 010W	1	18		传送字节/字串
	150	CMPS BUFF1,BUFF2	1010 011W	1	22	× ×××××	比较字节/字串
	151	SCAS INPUT-LINE	1010 111W	1	15	同上	搜索字节/字串
	152	LODS CUSTOMER-NAME	1010 110W	1	12		传送字节/字到累加器
	153	STOS PRINT-LINE	1010 101W	1	11		由累加器存字节/字到存贮器
	154	MOVSB/MOVSW	1010 010W	1	18		传送字节/字串
	155	REPE/REPZ CMPS DATA,REY	1111 001Z	1	2		当等于/为0时重复(迭代)
	156	REPNE/REPNZ SCAS LNPUT-LINE	1111 001Z	1	2		当不等于/不为0时重复(迭代)

五、转移指令程序流程控制指令

	序号	助记符	指令代码	字节	时钟	标志 ODITSZAPC	操作
调用子程序指令组	157	CALL AEAR-PROC	1110 1000 disp—low cisp—hish	3	19		段内直接转子程序
	158	CALL FAR-PROC	1001 1010 offset—low offset—hish seg—low seg—hish	5	28		段间直接转子程序
	159	CALL PROC-TABLE (BI)	1111 1111 mod 011 r/m × ×	2—4	21＋EA		段间间接调用子程序
	160	CALL AX	1111 1111 mod 010 r/m	2	16		段内间接调用子程序
	161	CALL[BZ]TASK[SI]	1111 1111 mod 010 r/m × ×	2—4	37＋EA		段内间接调用子程序
转	162	JMP SHORT	1110 1011 disp	2	15		段内直接转移
	163	JMP WITHIN-SEGMENT	1110 1001 disp—low disp—hish	3	15		段内直接转移

移指令组	164	JMP FAR-LABEL	1110 1010 cffset-low offeet-high seg-low seg-high	5	15		段间直接转移
	165	JMP [BX].TARGET	1111 1111 mod 101 r/m × ×	2—4	18+EA		段间间接转移
	166	JMP CX	1111 1111 mod 101 r/m	2	11		段内间接转移
	167	JMP OTHER.SEG(SI)	1111 1111 mod 101 r/m × ×	2—4	24+EA		段间间接转移
返回指令组	168	RET	1100 0011	1	8		段内返回
	169	RET 4	1100 0010 data-low data-hish	3	12		段内返回并加立即数到栈指针
	170	RET	1100 1011	1	18		段间返回
	171	RET 2	1100 1010 data-low data-hish	3	17		段间返回并加立即数到栈指针
	172	JE/JZ JZ ZERO	0111 0100 disp	2	16 or 4		当前条指令比较的两数相等或相减为0则转移 若ZF=1则(IP)←(IP)+disp符号扩展到16位

	符号	助记符	指令代码	字节	时钟	标志 ODITSZAPC	操作
条件转移	173	JL/JNGE JL LESS	0111 1100	2	16 or 4		比较的结果"小于""或"不大于或等于"时则转移 若(SF) ¦¦ (OF) = 1则(IP)←(IP) + diep 符号扩展到16位
	174	JLE/JNG JNG NOT-GREATER	0111 1110 disp	2	16 or 4		当结果"小于或等于"或"不大于"时转移 若((SF) ¦¦ ((OF)i(ZF)) = 1 则(IP)←(IP) + disp符号扩展到16位
	175	JB/JNAE JB BELOW	0111 0010 disp	2	16 or 4		当结果"低于"或"不高于或等于"时转移 若(CF) = 1则(IP)←(IP) + disp 同上
	176	JBE/JNA JNA NOT-ABOVE	0111 0110 disp	2	16 or 4		当结果"低于或等于"或"不高于"时转移 若(CF) ¦ (ZF) = 1则(IP)←(IP) + disp 同上
	177	JP/JPE JPE EVEN-PARITY	0111 1010 disP	2	16 or 4		当"奇偶性"或"偶奇偶性"时则转移 若(PF) = 1则(IP)←(IP) + disp 同上
	178	JO SIGNED-OVRFLW	0111 0000	2	16 or 4		当溢出时则转移 若 (OF) = 1则(IP)←(IP) + disp
	179	JS NEGATIVE	0111 1000	2	16 or 4		当符号位置位时则转移 若 (SF) = 1 则 (IP)←(IP) + disp 符号扩展到16位
	180	JNZ/JNE JNZ NOT-EQUAL	0111 0101 disp	2	16 or 4		当结果"不等于"或"不是零"时则转移 若(ZF) = 0则(IP)←(IP) + disp 符号扩展到16位
	181	JNL/JGE JGE GREATER-EQUAL	0111 1101 disp	2	16 or 4		当结果"不少于"或"大于或等于"时则转移 若(SF) ¦¦ (OF) = 0则(IP)←(IP) + disp符号扩展到16位

指令组	182	JNLE/JG JG GRFATER	0111 1111 disp	2	16 or 4		当结果"不小于或等于"或"大于"时则转移 若((SF)¦(OF)¦(ZF))=0 则(IP)←(IP)+disp符号扩展到16位
	183	JNB/JAE JAE ABOVE-EQUAL	0111 0011 disp	2	16 or 4		当结果"不低于"或"高于或等于"时则转移 若(CF)=0则(IP)←(IP)+disp同上
	184	JNBE/JA JA ABOVE	0111 0111 disp	2	16 or 4		当"不低于或等于"或"高于"时则转移 若(CF)¦(ZF)=0 则(IP)←(IP)+disp符号扩展到16位
	185	JNP-JPO JPO ODD-PARITY	0111 1011 disp	2	16 or 4		当"非奇偶性"或"奇性"时则转移 若(PF)=0则(IP)←)IP)+disp同上
	186	JNO NO-OVERFLOW	0111 0001 disp	2	16 or 4		当无溢出时则转移 若(OF)=0则(IP)←(IP)+disp同上
	187	JNS POSITIVE	0111 1001 disp	2	16 or 4		当无符号时则转移 若(SF)=0则(IP)←(IP)+disp同上
	188	JC CARRY-SET	0111 0010 disp	2	16 or 4		若(CF)=1则(IP)←(IP)+disP 符号扩展到16位
	189	JNC NOT-EQUAL	0111 0011 disp	2	16 or 4		若(CF)=0则(IP)←(IP)+disp 符号扩展到16位
迭代控制指令	190	LOOP AGAIN	1110 0010 disp	2	17 or 5		循环控制(CX)←(CX)-1 若(CX)≠0则(IP)←(IP)+disp同上

	序号	助记符	指令代码	字节	时钟	标志 ODITSZAPC	操作
迭代控制指令	191	LOOPZ/LOOPE LOOPE AGAIN	1110 0001 disp	2	18 or 6		当"为零"和"相等"循环 (CX)←(CX)－1 若(ZF)＝1及(CX)≠0 则(IP)←(IP)＋disp同上
	192	LOOPNZ/LOOPNE LOOPNE AGAIN	1110 0000 disp	2	19 or 5		当"不为0"及"不相等"循环 (CX)←(CX)－1 若(ZF)＝0及(CX)≠0 则(IP)←(IP)＋disp同上
	193	JCXZ COUNT-DONE	1110 0011 disp	2	18 or 6		当(CX)＝0则转移(IP)←(IP)＋disp 符号展扩16位
中断指令组	194	INT 3	1100 1100	1	52	00	将标志进栈(TF)←0，(IF)←0
	195	INT 67	1100 1101 type	2	51	00	将标志进栈(TF)←0，(IF)←0并间接调用256个中断矢量之一
	196	INTO	1100 1110	1	53 or 4	00	若(OF)＝1将标志进栈，TF←0，IF←0并间接转中断矢量4 若(OF)＝0则不操作
	197	IRET	1100 1111	1	24		中断返回

六、处理器控制指令

	序号	助记符	指令代码	字节	时钟	标志 ODITSZAPC	操作
标	198	CLC	1111 1000	1	2	0	(CF)←0
	199	CMC	1111 0101	1	2	×	(CF)求反(CF)←$(\overline{CF})$

志指令组	200	STC	1111 1001	1	2			1	(CF)←1
	201	CLD	1111 1100	1	2	0			(DF)←0
	202	STD	1111 1110	1	2	1			(DF)←1
	203	CLI	1111 1010	1	2		0		(IF)←0
	204	STI	1111 1011	1	2		1		(IF)←1
处理器控制指令组	205	HLT	1111 0100	1	2				暂停
	206	WAIT	1001 1011	1	3+510				等待不操作既空转
	207	ESC 6, ARRAY[SI]	1101 1××× mod × r/m × ×	2－4	8+EA				处理器交权（存贮器到立既数）
	208	ESC 20, AL	1101 1 ××× mod × r/m	2	2				处理器交权（寄存器到立既数）
	209	LOCK XCHG FLAG, AL	1111 0000	1	2				总线封锁前缀，不操作
	210	NOP	1001 0000	1	3				空操作

注：1. 此表中“助记符”一栏中给出的是助记符指令的一个例子，不是唯一的指令格式。

2. 此表中“标志”一栏中若“空白”则说明指令操作不影响标志。

3. 此表是根据原文改编，可参看原文《The 8086 Family User,s Manual》和《MCS 86™ASSEMBLY LANGUAGE REFERE CE GUIDE》。

4. 此表“×、×”表示位移量地址，低位在前，高位在后。

5.3 TP-86A监控程序簡介

我们选了英特尔公司SDK-86单板机的监控程序，并对它作了修改和功能扩充，然后移植到TP-86A上，形成了TP-86A的监制程序，TP-86A监控程序由两大部分组成：

·键盘监控程序：

用户通过TP-86A单板机上的小键盘使用键盘监控程序提供的十三条命令与主机进行通讯，这些命令的功能，格式及使用步骤请参看本手册的第三章。

·串行监控程序：

用户可通过外接的（或大键盘与屏幕显示器）使用串行监控程序提供的十条命令与主机进行通讯，并可使用打印机进行输出。这些命令的功能，格式及使用步骤请参看本手册的第四章。

TP-86A监控程序是用PL/M-86高级语言与ASM-86汇编语言写成的，用ASM-86汇编语言写的过程可被PL/M-86写的程序调用，键盘监控程序和串行监控程序均为模块化的结构程序。

一、监控程序的內存分配

8086CPU的寻址范围为1兆字节，因此其地址范围为00000H到FFFFFH。装有监控程序的EPROM安放在高地址端，用户可使用的RAM存储器安放在低地址端，见图5.9。

由于这两个监控程序的地址是相对的浮动地址，所以它们可以互换位置。譬如，只用串行监控程序时，可把键盘监控程序的EPROM拔下，而把串行监控程序的EPROM插在键盘监控程序EPROM所处的位置上去。

二、键盘监控程序的组成

TP-86A键盘监控程序由三个模块组成。它们是：总的数据说明模块；公共过程模块；命令模块。

如图5.7所示键盘监控程序中主程序及各条程序的调用路经只能沿箭头方向行进。

1. 总的数据说明模块

·该模块中包括所有总的数据说明及文字说明（等式）

·该模块的组成：

①实用部分：总的标志，变量和等式。

②I/O部分：5/0端口、屏蔽字、和特殊的符号。

③存储器自变量部分：用于中间指示器的结构。

④寄存器部分：用户寄存器保存区。

⑤引导和8089部分—引导和8089描述器。

2. 公共过程模块

·在这个模块中包含了那些较低一级的过程。这些过程只能被较高一级的例行程序调用

·模块组织：这个模块可分为三部分：

①基本I/O部分：

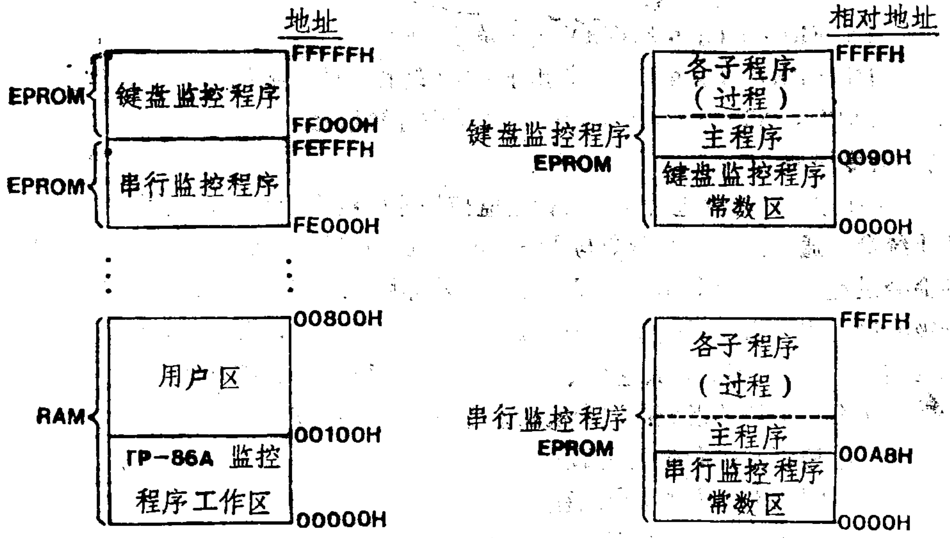


图5.6 监控程序内存分配示意图

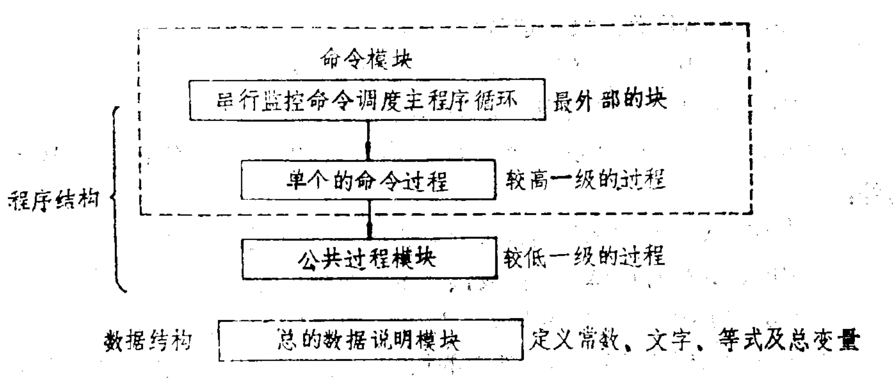


图 5-7 键盘监控程序结构示意图

KB$DISPLAY 到数据段/地址显示器上去显示。

KB$BLANK$DATA$FIELD 使数据段显示器显示带提示符的空白。

KB$BLANK$ADDR$FIELD 使地址段显示器显示带提示符的空白。

KB$OUTB$BYTE 输出一个字节去显示。

KBOUTWORD 输出一个字去显示。

KBGETCHAR 从键盘输入一个符号。

②自变量表达式求值部分

KBGETEXPR 取字表达式

KBGETADDR 取地址表达式

KB$UPDATE$IP 取任选的 CS:IP

③中断和恢复/执行例行程序,

SAVE$REGISTERS 保存用户寄存器。

RESTORE$EXECUTE 恢复机器状态并执行,

INTERRUPT 1 $ENTRY 用于单步中断

INTERRUPT 8 $ENTRY 用于 GO（转移）的中断例行程序

INITINTVECTOR 对中断矢量进行初始化

3. 命令模块

• 在这个模块中包含了所有键盘监控程序中给出的命令，用户在 TP-86A 单板机的小统盘上通过键盘监控命令与 TP-86A 主机进行对话。每个命令为一个独立的过程，这些命令过程只能被命令调度主程序循环的外部块所调用。同时命令过程又可作为较高一级的例行程序去调用较低一级的过程（即公共过程），命令模块中有三个过程是用 ASM-86 汇编语言编写的，其它命令过程是用 PL/M-86 语言编写的。

• 命令模块的组成：这个模块由下列两部分组成：

①键盘监控命令过程部分：

KB$GO 转移命令过程；

KB$SINGLE$SKEP 单步命令过程；

KB$EXAM$MEM 存储器检测与置换命令过程；

KB$EXAM$REC 检测器寄存器命令过程；

KB$MOVE 数据块传送命令过程；

KB$INPUT 从外部端口输入的命令过程；

KB$OUTPUT 往外部端口输出的命令过程。

以上的过程是用 PL/M-86 高级语言编写的，下面的三个过程是用 ASM-86 汇编语言编写的：

KB$CASS$DUMP 转录或写磁带的命令过程；

KB$CASS$LOAD 转储或读磁带的命令过程；

KB$PROG$TYPE$EPROM EPROM 写入的命令过程。

②命令调度主程序循环（外部的块，主程序循环），它是用 PL/M-86 语言写成的。

NEXT $ COMMAND 命令调度处理程序；

ERROR 出错处理程序；

AFTER$INTERRUPT 中断后的处理程序。

三、TP—86A串行监控程序的组成：

TP-86A 串行监控程序由三个模块组成，如图 5.8 所示，它们是：总的数据说明模块；公共过程模块；命令模块。

1. 总的数据说明模块

• 该模块中包含了所有的总的数据说明及文字的说明（等式）。

• 模块组织：

①实用部分：总的标志，变量和等式。

②I/O部分：I/O端口，屏蔽字及特殊符号。

③存储器自变量部分：用于中间指示器的结构。

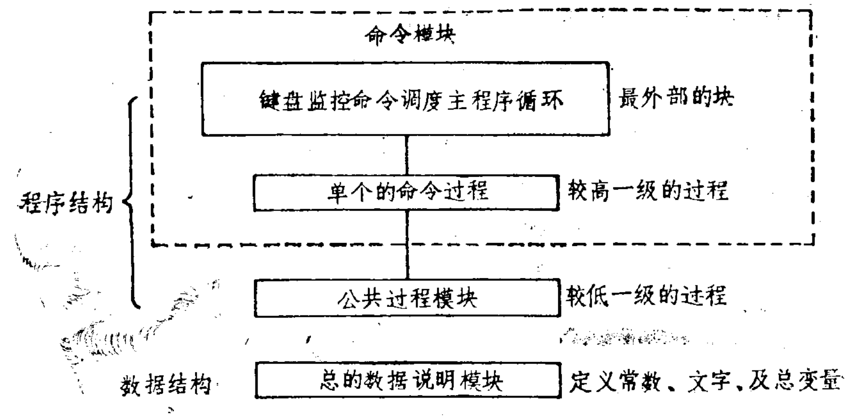


图 5.8 TP-86A 串行监控程序结构示意图

④寄存器部分：用户寄存器的保存区。

⑤引导和8089部分：引导和8089描述器。

2. 公共过程模块

· 在这个过程中包含了那些较低一级的过程。这些过程只能被较高一级的例行程序调用

· 模块组织：

①基本的部分

SIO$CHAR$RDY 输入字符就绪

SIO$CHECK$CONTROL$CHAR 检查控制字符

SIOOUTCHAR 输出字符

SIOGETCHAR 输入一个字符

SIOOUTBYTE 以16进制形式输出一个字节

SIOOUTWORD 以16进制形式输出一个字

SIOOUTBLANK 输出单个的空白

SIOOUTSTRING 输出一个“串”

SIOOUTHEADER 输出一个纸带标题

SIO8251SETTLING$DELAY产生延迟时间供写操作后异步安排使用

②实用例行程序部分：

SIO$VALID$HEX 对有效的16进制字符进行测试。

SIO$HEX 从 ASCII 变换到 HEX（16进制）。

SIO$VALID$REG$FIRST 对有效的寄存器第一个字符进行测试。

SIO$VALID$REG 对有效的寄存器名字进行测试。

SIO$CRLF 输出一个回车 CR 和换行 LF。

SIO$TEST$WORD$MODE 对命令中的‘W’进行测试。

SIO$SCAN$BLANK 对任意插入的空白进行搜索。

③对自变量表达式求值部分：

SIOGETWORD 取字表达式。

SIOGETADDR 取地址表达式。

SIO$UPDATE$IP 任选的 CS:IP 修正。

④读纸带部分

SIO$READ$CHAR 从 TTY（电传打字机）读入器读取一个字符。

SIO$READ$BYTE 读一个字节。

SIO$READ$WORD 读一个字。

⑤中断及恢复/执行例行程序

SAVE$REGISTERS 保存用户寄存器。

RESTORE$EXECUTE 恢复机器状态并执行。

INTERRUPT1$ENTRY 用于单步的中断例行程序。

INTERRUPT3$ENTRY 用于 GO 的中断例行程序。

INITINTVECTOR 用于对中断矢量进行初始化的例行程序。

3. 命令模块

·这个模块中包含了所有串行监控程序中给出的命令，用户在 CRT 键盘上通过串行监控命令与 TP-86A 主机进行对话，每条串行监控命令为一个独立的过程，这些命令过程只能被命令调度主程序循环的外部块所调用。同时命令过程又可作为较高一级的例行程序去调用较低一级的过程（公共过程）。

·命令模块的组成：这个模块由下列两部分组成：

①串行监控命令过程部分：

SIO$GO 转移命令过程；

SIO$SINGLE$STEP 单步命令过程；

SIO$EXAM$MEM 存储器测试及置换命令过程；

SIO$EAXM$REG 测试寄存器命令过程；

SIO$MOVE 传送命令过程；

SIO$DISPLAY 显示字节过程；

SIO$INPUT 读外部端口（从外部端口输入）命令过程；

SIO$OUTPUT 写外部端口（往外部端口输出）命令过程；

SIO$WRITE 写数据记录命令过程；

SIO$READ 读数据记录命令过程。

②串行监控命令调度主程序循环（外部的块）：

NEXT$COMMAND 命令调度处理程序；

ERROR 出错处理程序；

AFTER$INTERRUPT 中断后的处理程序。

四、键盘监控程序及串行监制程序的入口地址表：

表5.5给出了键盘监控程序的入口地址，表5.6给出了串行监控程序的入口地址。要注意入口地址是相对地址。入口参数，出口参数的变量在 RAM的0页（0～255地址）中。

键盘监控程序源程序见附录 I。

表 5.5　键盘监控程序透明化表

入口地址	名字	功能说明	入口参数	出口参数
009CH	MONITOR (MAIN PROGR)	主程序、用来扫描键盘读人键盘命令，并转向各命令的处理程序上去，包括：·显示“—”及版本号·初始化用户寄存器·读人下一命令并转向对应程序·出错处理：显示出错信息后去取下一命令·当通过中断矢量重人监控程序时，检查最后的命令。		
0251H	KB DISPLY	KB$DISPLY：这个程序用来在由第二个参数FIELD（数据段或地址段DATA/ADDR）指定的字段上显示由第一个参数（PTR）指定的数组的内容，小数点或提示符的数目由第三个参数PROMPTS决定。PROCEDURE（PTR，FIELD，PROMPTS）；	PTR＝0008H　4 FIELD＝0006H Addr/Data＝1/0 PROMPTS＝0004H	
02A7H	KB BL ANK DATA FIELD	KB　BLANK　DATA　FIELD： 这个程序用来将数据字段中填人空白，并显示参数“PROMPTS”中指定数目的“小数点”作为提示符，PROCEDURE（PROKPTS）；	PROMPTS＝0004H	
02BEH	KB BLANKADDR FIELD	KB$BLANK$ADDR　FIELD： 这个程序用来将地址字段中填人空白，并显示参数“PROMPTS”中指定目的“小数点”作为提示符，PROCEDURE（PROMPTS）；	PROMPTS＝0004H	
02D5H	KB OUTBYTE	KB$OUT BYTE：这个程序用来把从第一个参数中来的输人字节输出到数据字段上去，并具有由第二个参数指定题目的提示符PROCEDURE（B，PROMPTS）；	B＝0006H PROMPTS＝0004H	
0315H	KB OUT WORD	KB$OUT　WORD：这个程序把第一个参数W送到第二个参数FIELD指定的数据/地址字段上去，并具有第三个PROMPTS指定数目的提示符，若第四个参数指定的话，就要实现引导零空白，PROCEDURE(W、FIELD，PROMPTS、BLANING）；	W＝000AH FIELD＝0008H Addr/Data＝1/0 PROMPTS＝0006H BLANKING＝0004H	

（续）

入口地址	名　　字	功　能　说　明	入 口 参 数	出 口 参 数
039BH	KB GET CHAR	KBGETCHAR：从(键双控制器)的 FIFO（先进先出）队列中读入一个字符，等待直到字符可用，然后将此字符送回总变量中定义的CHAR中去。		CHAR＝005AH
03S5H	KB GET EXPR	KBGETEXPR：此程序收集输入的字符流，并建立一个表达式，这个表达式作为该过程的值送回，由第一个参数 FIELD 指定输入字符送到地址字段"(FIELD＝0）还是数据字段（FIELD＝1），这时表达式总是作为一个字输出的，如果FIELD＝－1,则总是作为一个字节在数据字段中，第二个参数PROMPTS指定提示符的数目，第三个参数 BLANKING 指定引导零空格，表达式由，·或:终结，出口时'CHAR'中包含有它们中的一个，PROCEDURE（FIELD、PROMPTS、BLANKING）WORD;	FIELD＝0008H PROMPTS＝0006H BLANKING＝0004H	CHAR＝005H中应为"："＝14H "·"＝10H "；"＝11H之一
0501H	KB GET AD DR	KBGETADDR:这个程序收集输入字符，并建立段和位移量地址的表达式，要求第一个字符已经读入总变量CHAR中，如果用户没有输入段地址则使用的是DEFAULT$BASE（缺省的基）即CS段寄存器，在地址字段中显示出现地址表达式并具有第三个参数PROMPTS指定的提示符。	PTR＝0008H　4 DEFAULTBASE ＝0006H PROMPTS＝0004	
0550H	KB UPDATE IP	KB$UPDATE$IP：该程序被单步（ST）程序调用，并转去输出当前的 CS:IP 和当前的指令字节，对任意的输入CS:IP是打开的。		
05B5H	SAVE REGIS-TERS	SAVE$REGISTERS$：这个程序把进栈的用户的寄存器保存到监控程序保护区中去。		
0605H	RESTOREEXE-CUTE	RESTORE$EXECUTE：这个程序用来恢复用户机器的状态并将控制转给用户程序，为了实现用户的寄存器的退栈操作，这个过程中包含有一段机器语言的子程序，在此机器语言子程序末尾执行一条"IRET"中断返回指令把控制转给用户程序。		
067DH	INTERRUPT1 ENTRY	INTERRUPTI$ENTRY：当CPU被一条具有陷井位置位单（单步）的指令中断时，要调用此过程。（注括号内为机器码指令开始的入口地址）		

06ECH (06D3H)	INTERRUPT3 ENTRY	INTERRUPT 3 $ENTRY：当CPU执行一条INT3指令时，调用这个过程，监控这个程序插入(0CCH)作为一个断点，还有一个外部的中断或一用户的软件中断可能引起调用这个过程。(注：括号内为机器码指令开始的入口地址。)		
0709H	INIT INT VECTOR	INIT $INT $VECTOR：这个程序按下列步骤去初始化一个中断矢量，从INT $ROUTINE地址来的偏移值(OFFSET)要被合适的字节数校正，以便中断PLM序言。从当前CS寄存器来的段地址(SEGMENT)要由一个机器语言代码的子程序来确定。		
072AH	KBGO	KB $GO：处理GO命令，显示IP和当前指令字节的内容，并对输入及一个任选的断点打开CS:IP，按下句号键"•"，控制转去执行用户的程序，并在地址字段显示"E"(执行的字头)。如果遇到了一个"断点"则在数据字段上显示BR(断点的字头)。		
07B9H	EBSINGLE STEP	KB $SINGLE $STEP：处理单步命令ST，显示与当前的指令字，对输入打开CS:IP，按下"，"豆号键，单步执行命令；按下"•"句号键，停止用行单步命令ST。		
07E0H	KB EXAM MEN	KB $EXAM $MEN：处理检测存储器命令EB（检测字节），EW(检测字)，在地址字段上显示这个地址，在数据段上显示该地址单元中的内容（字节或字)，然后对输入随意地打开。"，"号使地址增量到下一个单元，"•"句号终止该命令的执行，"•"号使地址减量到下一个单元。		
0903H	KB EXAM REG	KB $EXAM $REG：处理检测寄存器命令ER，在地址字段提示要检查的寄存器名，在数据字段显示该寄存器的内容。然后对输入随意地打开。"，"键按下，用来增量到下一个寄存器，一直到"FL"寄存器，"L"与"•"作用一样，停止命令的执行。用户可键入任选的寄存器名字，名写不对，则显示ERR		
09E5H	KB MOVE	KB $MOVE：处理传送命令MV，在地段提示显示三个地址自变量：起始地址，结束的偏移地址及目的地址，并将由地址自变量ARG1-ARG2，指定的存储器块中的内容送到由第三个地址自变量ARG 3 指定的存储器块中去，如果读回时有差别，即出错。		

（续）

入口地址	名　　字	功　能　说　明	入　口　参　数	出　口　参　数
0A92H	KB INPUT	KB INPUT：处理输入字节或字命令 IB 和 IW，在地址字段显示响应端口地址，（按下命令后），此时按下“，”键在数据字段上再出现该端口地址中的内容，可用来读入该端口地址单元中的多个数据，按下“-”键停止此命令的执行。		
0AF0H	KB OUTPUT	KB$OUTPUT：处理输出字节或字命令 OB 或 OW，按下 OB/OW命令后，再按端口地址在地址字段上的响应显示此端口地址然后按下“，”号，再按 REG 名或要输出的数据，在数据段显示输出的数据，“·”停止命令的执行，对指定的端口该命令只输出一次数据。		
0C08H	KB CASS DUMP	KB$CASS$DUMP：此过程来完成 CD（转录或写磁带）命令的功能，即将用户在命令中指定的首，末地址的 RAM 存区中的信息转录到盒式录音机磁带上，在磁带上建立用户在命令中给出文件名的文件。磁带作为 TP-86A的外存，以保存暂时不用的程序和数据。		
0D33H	KB CASS LOAD	KB$CASS$LOAD：该过程用来完成CL（转储或读磁带）命令的功能，它把录音机磁带中的信息（由用户键入的文件名指定的文件）输入到由用户键入的地址指定的内存（RAM）区域中去。		
0E13H	KB PROGRAM-MING TYPF EPROM	KB$PROGRAMMING$TYPE$EPROM：该过程的功能是将用户键入的命令中“原文件首，末地址”指定的 RAM 中的信息写到由命令中“编程类型”指定的2716/2732 型 EPROM 中去，EPROM 的地址由命令中输入的EPROM“相对地址”给出。		

表5.6　　串行监控程序透明化表

类别		入口地址(16进制)	名称(略去插入的符号)	功能说明	入口参数	出口参数
					此栏中跟在变量地址后的数为变中所占的单元数	
公共的过程	基本I/O部分	02A0	SIO CHAR RDY	此过程（例行程序）用来检查正在读的状态端口中有无挂起的输入字符，并对SIO$RXRDY进行屏蔽(屏蔽出“读数据就绪”状态)。如果不空(有挂起的字符).则返回值为“真”；如果没有挂起的字符(空)，则返回值为“假”。		
		02B3	SIO CHECK CONTROL CHAR	此过程(例行程序)检查是否有一个控制字符已输入到串行端口了。如果有一个控制字符 CONTROL-S，则在返回调用程序之前该过程要等待另一个控制字符CONTROL-Q。如果输入的是控制字符 CONTROL-C，则转移到出错处理例行程序上去执行。		
		02EF	SIO OUT CHAR	此过程(例行程序)的功能是：当异步输出就绪时(即发送缓冲器 XMIT空时)，该过程要把输出的“输入参数”送到异步输出端口中去。	C = 0004H 单元	
		030F	SIO GET CHAR	此过程(例行程序）用来“从输入口端”输入一个字符，并且在返回时将该输入字符置入总变量单元“CHAR”中		CHAR = 0066H
		0330	SIO OUT BYTE	此过程(例行程序)的功能是：将单个的“输入参数”以 ASCII 十六进制格式输出到异步串行端口去。	B = 0004H 单元	
		0360	SIO OUT WORD	此过程例行程序把“输入参数”作为4个ASCII 十六进制的字符输出到异步输出端口去。	W = 0004H　2	
		0377	SIO OUT BLANK	此过程(例行程序)的功能是：输出一个空白。		
		0382	SIO OUTSTRING	此过程(例行程序)的功能是：输出由第一个参数PTR所指向的“串”。	PTR = 0004H　4	

类别		入口地址（16进制）	名称（略去插入的$符号）	功能明说	入口参数（此栏中跟在变量地址后的数为变量所占的单元数）	出口参数
公共的过程	基本I/O部分	03B5	SIO OUT HEADER	此过程(例行程序)的功能是：输出由“:”(冒号)后跟以“记录长度，“寄存地址”和“记录类型”的纸带标题。该例行程序还将“检查和”的初值置为“0”(零)。	LENGTH =0008H 1 LOAD ADDR =0006H 2 RECTYPE =0004H 1	
		03D9	SIO 8251 SETTLING DELAY	此过程（例行程序）产生足够的(合适的)延迟时间供写操作后异步端口进行自己的安排		
	实用例行程序部分	03E7	SIO VALID HEX	此过程(例行程序)用来测试“输入参数”是否是一个有效的ASCII十六进制数字。如果是，该过程返回时的值为“真”；如果不是，则返回值为“假”。	H=0004H 1	
		0416	SIO HEX	此过程(例行程序)用来将ASCII形式的“输入参数”变换成它的二进制等效值，还把该二进制等效值作为返回值送回。此例行程序对输入的有效性(确实性)不做检查。	C=0004H 1	
		0431	SIO VALID REG FIRST	此过程（例行程序）用来检查变量“CHAR”中是否包含有一个寄存器名的第一个有效字母，如果是，则返回值为“真”，如不是有效字母，则返回值为“假”。		
		0460	SIO VALID REG	此过程(例行程序)用来检查两个“合起来看的输入参数”是不是一个有效的寄存器名字。该例行程序在“寄存器名表”中进行查找。如果在表中找到了与输入参数一样的寄存器名字，则把这个有效寄存器在表中所处的索引值置入总的变量“REG$INDEX”(寄存器索引)中去。并且返回值为“真”。如果在表中没有找到与输入参数一样的寄存器名字，则该过程(例行程序)的返回值为“假”，且总变量REG $NIDEX为不确定的。	C_1=0006H 1 C_2=0004H 1	

公共的过程	实用程序部分	04A7	SIO CRLF	此过程(例行程序)用来输出一个回车 CR 和换行 LF 到输出端口中去。		
		04B8	SIO TEST WORO MODE	此过程(例行程序)用来假查跟在命令后的是不是字母 W(以 ASC II 式给出的"W"=57H)。若是，将标志 WORD $MODE"置为"真"值为 0FFH表示"真");若不是字母 W，则将标志 WORD $ MODE"置为"假"(值为 00H 表示"假")。该过程还要搜索完跟在命令后任意插入的空白。		WORDMODE= 0069H 1
		04DE	SIO SCAN BLANK	在输入一个命令字母后要调用这个过程（例行程序）来搜索完任意插入的空白。		
	自变量表达求值部分	04F0	SIO GET WORD	此过程（例行程序）用来从输入端口读入字符，并对十六进制数和寄存器名作为操作数和"+"(加)，"—"(减)算符组成的表达式求值。		
		05B4	SIO GET ADDR	此过程（例行程序）来接收一个由任选的（SEG）：和位移量组成的地址表达式。	PTR=0006H 4 DEFAULTBASE 0004H 2	
		0625	SIO UPDATE IP	单步程序调用这个过程，以便转向输出当前的 IP 和指令字节，并对输入打开 IP。		
	纸带读部分	0682	SIO READ CHAR	这个过程（例行程序）用来从 TTY 的纸带阅读器中读入一个字符,读入字符是通过触发 DTR，采样 CSR 用于监视启动位，且当启动位出现时，解除对 DTR 的触发，并读入异步格式数据的方法来实现的,为了只对"寄存命令(LAD COMMAND)"提供HOLD-OFF能力（不是对TTY纸带）,在读入 CONTROL-S 之后。该过程（例行程序）要等待读入一个对应的 CONTROL-Q，然后再继续往下进行。		
		06CD	SIO READ BYTE	此过程（例行程序）用来从纸带阅读器读入一个字节。		
		06F8	SIO READ WORD	此过程（例行程序）从纸带阅读器读入一个字，并将此字作为自己返回值		

（续）

类别		入口地址（十六进制）	名称（略去插入的 $ 符号）	功能说明	入口参数	出口参数
					（此栏中跟在变量地址后的数为变量所占的单元数）	
公共的过程	中断和恢复/执行部分	0715	SAVE REGISTERS	此过程（例行程序）用来将栈中的用户寄存器内容保存入监控程序的保存区中去。		
		0766	RESTORE EXE-CUTE	此过程（例行程序）恢复用户的机器状态，并将控制转回给用户程序。在这个过程中有一段机器语言的子程序。这个子程序用来完成用户寄存器的退栈和执行一条‘IRET 中断返回指令将控制传给用户程序。		
		07c6	INTERRUPT 1 ENTRY	当执行一条带“TRAP（陷井）位”置位的指令（单步）使CPU中断时，要调用这个过程（例行程序）		
		0838	INTERRUPT 3 ENTRY	当CPU执行一条‘INT 3” 中断指令时要调用这个过程（例行程序）。监控程序插入‘INT 3” 指令的代码（OCCH)作为一个断点。还有，外部中断和用户的软件中断也可能调用此过程		
		0871	INIT INT VECTOR	此过程按照下列步骤来初始化一个中断矢量： ·要用合适的字节数去校正‘INT$ROUTINE’ 地址的偏移值(OFFSET)，以便中断 PLM 语言。 ·用一个机存语言编码的子程序来确定从当前CS寄存器来的段地址值（SEGMENT）		
		0892	SIO GO	该过程用来完成‘GO” 转移命令的功能，用户可以指定一个新的 IP:PC 和一个任选的断点		
	单	08FF′	SIO SINGLE STEP	此过程用来完成单步命令的功能，显示IP 和当前的指令字节。对输入打开CS:IP，按下豆号键‘，”使监控程序单步执行指令，按下句号键‘.”终止该命令的执行。		

串行监控命令	个的命令部分	0926	SIO EXAM MEM	此过程用来完成存储器检测命令的功能		
		0A06	SIO EXAM REG	此过程用来完成寄存器检测命令的功能，搜索有效的寄存器名并显示该寄存器的内容，然后对输入随意地打开，按下逗号键‘，’可增量到下一个寄存器，若下一个寄存器为‘LF’，则停止该命令的执行，其作用与按下回车键‘CR’使命令停止执行一样		
		0B55	SIO MOVE	该过程完成 MOVE（数据块传送）命令的功能。它接收三个自变量 ARG1，ARG_2和ARG_3，把由 ARG_1-ARG_2 指定的存储器数据块传送到 ARG_3 指定的存储区中去。若ARG_2<ARG_1或当读回的一个字节有差异时，则转FRROR 出错处理程序		
		0BE6	SIO DISPLAY	该过程完成显示字节命令的功能。如果用一个参数来调用它，则输出单个的字节。如果用两个参数来调用它，则输出这两个参数地址之间所包含的范围中的内容。若 0FFSET<BEGIN（偏移值<起始值），则只输出一个单个的字节。		
		0CA6	SIO INPUT	此过程完成“INPUT”输入命令的功能。用户指定一个端口，该端口中的数值被显示出来		
		0CE6	SIO OUTPUT	此过程完成“OUTPUT”输入命令的功能。将用户指定的数值输出到指定的端口中去		
		0D47	SIO WRITE	此过程完成纸带写命令的功能，输出引导零，扩展的地址记录（只对8086），起始地址记录(只对8086)，数据记录，EOF)文件末端记录，和尾部的零		
		OED8	SIO READ	此过程用来完成读纸带命令的功能		

（续）

类别		入口地址（十六进制）	名称（略去插入的$符号）	功能说明	入口参数	出口参数
					此栏中跟在变量地址后的数为变量所占的单元数	
串行监控命令	命令调度主程序循环	00A8	MONITOP (MAIN PROGRAM)	主程序，用来扫描键盘命令并转到各命令处理程序上去，其功能包括： • 在键盘主显示灯和 CRT 上显示提示符及版本号 • 保证在硬件复位(期待方式)或软件重启动(期待命令)时，对异步操作进行正确的初始化 • 对用户寄存器进行初始化 • 读入用户键入的下一个命令，并转到对应该命令的程序上去 • 当出错时，显示出错信息后去取下一条命令 • 当中断显示CS:IP和恢复被断点断开的指令后，调用其"AFTER $INTERRUPT'程序段作中断后的处理，并转向下一条命令		

附 录 I

TP-86A 键盘监控程序PL／M-86 源程序

附 录 I

TP-86A键盘监控程序PL/M-86源程序

```
/* ****************************************************************
   ****************************************************************
   TP-86A 键盘监控程序.VA.1
     1982年 6 月
   TP-86A 键盘监控程序.版本VA.1是由英特尔SDK-86
   键盘监控程序1版本VI.1经修改并移植到TP-86A上而成的。
   ****************************************************************
   ****************************************************************
```

简 介:

这个程序是装在 EPROM 中的 TP-86A 键盘监控程序，它给用户提供了检测/修改存储器和寄存器、读/写、I/O端口、控制程序执行、读/写盒式磁带及写 EPROM 的能力。

环 境:

TP-86A 键盘监控程序通过板上的24键键盘和8个7段数码显示器（4个显示地址，4个显示数据）与用户进行通讯。

程序的构造:

该程序分为一个数据模块和两个代码（程序）模块。

1. 数据定义模块，总的数据定义。
2. 公共程序、较低一级的过程。
3. 命令模块，单个的命令和外部的块。

调 用 路 径:

≫命令处理，调度模块（外部的块）。

  单个的命令过程。

    公共程序。

总的数据结构:

监控程序在一个字数组中保持用户的机器状态（寄存器），寄存器内容通过 PL/M-86 中断过程被推入用户栈而贮存起来，TP-86A的 2**20 地址空间的指示器由双字指示器结构来提供。

```
1       MONITOR: DO;          /*BEGINNING OF MODULE */
        /* ***********************************************
           ***********************************************
                 GLOBAL DATA DECLARATIONS MODULE
        ABSTRACT
        THIS MODULE CONTAINS ALL THE GLOBAL DATA DECLARATIONS
        AND LITERALS (EQUATES) .
        MODULE ORGAINIZATION
        THE MODULE IS DIVIDED INTO 5 SECTIONS:
        1. UTILITY SECTION        GLOBAL FLAGS, VARIBLES,
                                   EQUATES
        2. I/O SECTION            I/O PORTS, MASKS, AND SPECIAL
                                   CHARS
        3. MEMORY ARGUMENTS/      STRUCTURES FOR MEDIUM
           SECTIONS               POINTERS
        4. REGISTER SECTION       USER REGISTER SAVE AREA
        5. BOOT AND 8089 SECTION  BOOTSTRAP AND 8089 DESCRIPTOR
      */
          /**********************
          *  UTILITY SECTION  *
          **********************/
2   1   DECLARE
          INT$VECTOR(5)          POINTER;  /*INTERRUPT VECTORS*/
3   1   DECLARE
          MONITOR$STACKPTR     WORD,
          MONITOR$STACKBASE    WORD;      /*MONITOR SS SAVE*/
4   1   DECLARE
          COPYRIGHT(*) BYTE/DATA ('(C)1982 BPU DEP 3') ;
5   1   DECLARE
          BRK1$FLAG        BYTE,           /* TRUE IF BREAK SET */
          BRK1$SAVE        BYTE,           /* INST BREAK SAVE */
          CHAR             BYTE,           /*ONE CHAR LOOK AHEAD */
          DIST(4)          BYTE,           /*DISPLAY ARRAY */
          I                BYTE,
          END$OFF          WORD,           /*END OFFSET ADDRESS*/
          WORD$MODE        BYTE,
          LAST$COMMAND     BYTE,
6   1   DECLARE
```

```
TRUE            LITERALLY '0FFH',
FALSE           LITERALLY '000H',
ADDR$FIELD      LITERALLY '1',       /* ADDR FIELD WORD
                                        OUTPUT */
DATA$FIELD      LITERALLY '0',       /* DATA FIELD WORD
                                        OUTPUT */
DATA$BYTE       LITERALLY '-1',      /* DATA FIELD BYTE
                                        OUTPUT */
BLANK           LITERALLY '1',
NOBLANK         LITERALLY '0',
BREAK$INST      LITERALLY '0CCH',    /* BREAKPOINT
                                        TRAP */
STEP$TRAP       LITERALLY '0100H',   /* SS TRAP FLAG
                                        MASK */
USER$INIT$SP LITERALLY '100H',       /* USER STACK
                                        INITIAL */
GO$COMMAND      LITERALLY '2',       /* GO COMMAND
                                        CODE /*
SS$COMMAND      LITERALLY '3';

/***********************************
*  I/O DECLARATIONS SECTION        *
***********************************/

7  1  DECLARE
   KB$STAT$PORT    LITERALLY '0FFEAH',  /*STATUS/
                                           COMMAND
                                           PORT */
   KB$DATA$PORT    LITERALLY '0FFE8H',  /*DATA PORT*/
   KB$INIT$MODE    LITERALLY '00H',     /*8 8-BIT, LEFT
                                          ENTRY,
                                          ENCODE, 2
                                          KEY
                                          LOCKOUT*/
   KB$INIT$SCAN    LITERALLY '39H',     /*10MS SCAN
                                           RATE */
   KB$INRDY        LITERALLY '07H',     /*IN CHAR RDY
                                           MASK */
   KB$CMND$PMT(*)BYTE DATA              /*'-' */
                   (40H,00H,00H,00H),
```

```
KB$SIGNON(*)        BYTE DATA              /* '-86' */
                     (40H,00H,7FH,7DH),
KB$VERSION(*)       BYTE DATA              /* '1.1' */
                     (00H,00H,86H,06H),
KB$REC$PMT(*)       BYTE DATA              /* 'R' */
                     (50H,00H,00H,00H),
KB$ERR$MSG(*)       BYTE DATA              /* '-ERR' */
                     (40H,79H,50H,50H),
KB$EXEC(*)          BYTE DATA              /* 'E' */
                     (79H,00H,00H,00H),
KB$BRK1(*)          BYTE DATA              /* 'BR' */
                     (7CH,50H,00H,00H),
KB$BLANKS(*)        BYTE DATA              /* '    ' */
                     (00H,00H,00H,00H),
LED(*)              BYTE DATA
    (3FH,06H,5BH,4FH,66H,6DH,7DH,07H,  /* '0'-'7' */
     7FH,6FH,77H,7CH,39H,5EH,79H,71H);  /* '8'-'F' */
8  1  DECLARE
    KBPER        LITERALLY '10H',   /* PERIOD           */
    KBCOM        LITERALLY '11H',   /* COMMA            */
    KBREGKEY     LITERALLY '15H',   /* REGISTER KEY     */
    KBCOL        LITERALLY '14H',   /* COLON(SEGMENT)   */
    KBPLUS       LITERALLY '13H',   /* PLUS KEY         */
    KBMINUS      LITERALLY '12H',   /* MINUS KEY        */

      /***********************
      * POINTER SECTION  *
      ***********************/

9  1  DECLARE
    MEMORY$ARG1$PTR POINTER,    /* ARGUMENT 1 */
    ARG1 STRUCTURE (OFF WORD,SEG WORD)
      AT (@MEMORY$ARC1$PTR),
    MEMORY$ARG1BASED MEMORY$ARG1 PTR BYTE,
    MEMORY$WORD$ARG1 BASED MEMORY$ARG1 $PTR WORD,

    MEMORY$ARG3 $PTR POINTER,    /* ARGUMENT 3 /
    ARG3 STRUCTURE (OFF WORD, SEG WORD)
      AT (@MEMORY$ARG3$PTR),
    MEMORY$ARG3 BASED MEMORY$ARG3$PTR BYTE,
```

```
MEMORY$BRK1$PTR POINTER,     /*BREAKPOINT */
BRK1 STRUCTURE (OFF WORD, SEG WORD)
  AT(@MEMORY$BRK1$PTR),
MEMORY$BRK1 BASED MEMORY$BRK1$PTR BYTE,

MEMORY$CSIP$PTR POINTER,      /* CS:IP INSTRUCTION */
CSIP STRUCTURE (OFF WORD, SEG WORD)
  AT (@MEMORY$CSIP$PTR),
MEMORY$CSIP BASED MEMORY$CSIP$PTR BYTE,

MEMORY$USERSTACK$PTR POINTER, /* USER'S STACK */
USERSTACK STRUCTURE (OFF WORD, SEG WORD)
  AT (@MEMORY$USERSTACK$PTR),
MEMORY$USERSTACK BASED MEMORY$USERSTACK$
  PTR WORD;
  /************************
  * REGISTER SECTION *
  ************************/
10 1DECLARE
KB$REG(*)    BYTE DATA
  (00H,77H,00H,7CH,00H,39H,00H,5EH,     /* AXBXCXDX */
   6DH,73H,7CH,73H,6DH,30H,5EH,30H,     /* SPBPSIDI   */
   39H,6DH,5EH,6DH,6DH,6DH,79H,6DH,     /* CSDSSSES   */
   30H,73H,71H,38H),                    /*IPFL        */
REG$SAV(14)     WORD,          /* USER'S SAVED REGS */
REG$ORD(*)      BYTE DATA
  (7,6,1,3,2,0,9,11,12,8,13),     /* STACKED REG ORDER */
SP         LITERALLY 'REG$SAV(4)',
BP         LITERALLY 'REG$SAV(5)',
CS         LITERALLY 'REG$SAV(8)',
DS         LITERALLY 'REG$SAV(9)',
SS         LITERALLY 'REG$SAV(10)',
ES         LITERALLY 'REG$SAV(11)',
IP         LITERALLY 'REG$SAV(12)',
FL         LITERALLY 'REG$SAV(13)',

  /****************************************
  * BOOTSTRAP JUMP AND 8089 VECTOR *
  ****************************************/
/* THIS BOOT CONSISTS OF A LONG JUMP TO THE BEGINNING OF
```

```
          THE MONITOR AT FF00+XXXX WHERE  XXXX IS THE STARTING
          ADDRESS OFFSET (IP) AND MUST BE DETERMINED AFTER EACH
          COMPILE. */
11   1  DECLARE
          START$ADDR LITERALLY '009CH',
                                                  /* STARTING ADDRESS */
          BOOT1(*)  BYTE AT (0FFFF0H) DATA (0EAH),
                                                  /* LONG JUMP OPCODE */
          BOOT2(*)  WORD AT (0FFFF1H) DATA (START$ADDR),
          BOOT3(*)  WORD AT (0FFFF3H) DATA (0FF00H);
                                                  /*SEGMENT ADDRESS   */

        /* THIS TWO-WORD DATA IS A JUMP TO THE STARTING ADDRESS
          AND IS LOCATED AT THE FIRST LOCATION OF ROM (NO OTHER
          DATA OR CONSTANT DECLARATIONS MAY PRECEDE IT).  THE
          JUMP IS ACTUALLY TO  (START-ADDR) -4 SINCE THE INSTRU-
          CTION IS A RELATIVE JUMP OF LENGTH 3.*/
12   1  DECLARE
            BOOT4(*)  WORD DATA (0E990H, START$ADDR-4);
                                                /* NOP, JMP START-ADDR */

        /* THIS BLOCK  OF  ROM  ATFFFF6- FFFFA  IS INITIALIZED  FOR
          THE  8089  DEVICE  AND  POINTS  TO  A  BLOCK  OF  RAM  AT
          LOCATION 100H. */
13   1  DECLARE
            BLOCK$8089         WORD      AT (0FFFF6H)  DATA(00001H),
            BLOCK$8089$PTR  POINTER   AT (0FFFF8H) DATA (00100H),

            /* ***********************************************
               ***********************************************

                          COMMON PROCEDURES

            ABSTRACT
            THIS MODULE CONTAINS THOSE LOWER  LEVEL  PROCEDURES
            CALLED BY HIGHER LEVEL ROUTINES.
            MODULE ORGANIZATION

            THIS MODULE CONTAINS THE FOLLOWING SECTIONS:
              1. BASIC I/O SECTION
```

```
        KB$DISPLAY                    DISPLAY TO LED FIELDS
        KB$BLANK$DATA$FIELD           BLANK DATA FIELD WITH
                                        PROMPTS
        KB$BLANK$ADDR$FIELD           BLANK ADDRESS FIELD
                                        WITH PROMPTS
        KB$OUT$BYTE                   OUTPUT A BYTE TO
                                        DISPLAY
        KB$OUT$WORD                   OUTPUT A WORD TO
                                        DISPLAY
        KB$GET$CHAR                   INPUT A CHAR FROM
                                        KEYPAD
      2. ARGUMENT EXPRESSION EVALUATOR
        KB$GET$EXPR                   GET WORD EXPRESSION
        KB$GET$ADDR                   GET ADDRESS EXPRESSION
        KB$UPDATTE$IP                 GET OPTIONAL CS:IP
      3. INTERRRUPT AND RESTORE/EXECUTE ROUTINES
        SAVE$REGISTERS                SAVES USERS REGISTERS
        RESTORE$EXECUTE               RESTORE MACHINE STATE
                                        AND EXEC
        INTERRUPR 1$ENTRY             INTERRUPT FOR SINGLE
                                        SPEP
        INITERRUPT 3$ENTRY            INTERRUPT ROUTINE FOR
                                        GO
        INIT$INT$VECTOR               INITIALLZES INTERRRUPT
                                        VECTORS

   */

      /**************
       * BASIC I/O SECTION *
       **************/

14  1   KB$DISPLAY:
        /* THIS ROUTINE DISPLAYS THE CONTENTS OF THE ARRAY
          POINTED TO BY THE FIRST PARM TO THE FIELD SPECIFIED
          BY THE SECOND (ADDR OR DATA).THE NUMBER OF DECIMAL
          POINTS OR PROMPTS IS DETERMINED BY THE THIRD PARA-
          METER */
         PROCEDURE (PTR,FIELD,PROMPTS);
15  2     DECLARE PTR POINTER,(FIELD,PROMPTS,T) BYTE,
            DISPLAY BASED PTR (1) BYTE;
```

```
16  2       IF FIELD=ADDR$FIELD THEN
17  2         OUTPUT (KB$STAT$PORT) =94H;  /*ADDRESS FIELD */
          ELSE
18  2         OUTPUT (KB$STAT$PORT) =90H;     /* DATA FIELD */
19  2       DO I=0 TO 3;
20  3         T=DISPLAY (3—I);           /* DISPLAY BACKWARDS! */
21  3         IF PROMPTS>I THEN T=T OR 80H;
23  3         OUTPUT (KB$DATA$PORT) =T;
24  3       END;
25  2   END;

26  1   KB$BLANK$DATA$FIELD:
         /* THIS ROUTINE BLANKS THE DATA FIELD OF THE  DISPLAY
            WITH THE NUMBER OF PROMPTS(DECIMAL POINTS) AS SPE-
            CIFIED BY THE PARAMETER. */
        PROCEDURE (PROMPTS);
27  2      DECLARE PROMPTS BYTE);
28  2      CALL KP$DISPLAY(OKB$BLANKS,DATA$FIELD,PROMPTS) ;
29  2   END;

30  1   KB$BLANK$ADDR$FIELD:
        /* THIS PROCEDURE BLANKS THE ADDRESS FIELD OF  THE DISPALY
           WITH THE NUMBER OF PROMPTS SPECIFIED BY THE PARAME-
           TER 'PROMPTS'. */
        PROCEDURE (PROMPTS);
31  2      DECLARE PROMPTS BYTE;
32  2      CALL KB$DISPLAY (OKB$BLANKS,ADDR$FIELD,PROMPTS);

33  2   END;
34  1   KB$OUT$BYTE:
        /* THIS ROUTINE OUTPUTS THE BYTE INPUT FROM THE FIRST
           PARAMETER TO THE DATA  FIELD  WITH  THE  NUMBER OF
           PROMPTS SPECIFIED BY  THE SECOND PARAMETER. */
        PROCEDURE (B,PROMPTS);
35  2      DECLARE (B,PROMPTS) BYTE;
36  2      DISP(0),DISP(1)=0;                  /* FIPST TWO BLANK */
37  2      DISP(2)=LED(SHR(B,4)AND OFH);
38  2      DISP(3)=LED(B AND OFH);
```

```
39  2      CALL KB$DISPLAY(ODISP,DATA$FIELD,PROMPTS);
40  2   END;

41  1   KB$OUT$WORD:
        /* THIS  ROUTINE  OUTPUTS THE FIRST PARM  TO  THE  FIELD
           SPECIFIED BY THE SECOND WITH THE NUMBER OF PROMPTS
           SPECIFIED BY  THE  THIRD.  LEADING  ZERO  BLANKING IS
           PERFORMED IF SPECIFIED BY THE  FOURTH  PARAMETER. */
        PROCEDURE (W,FIELD,PROMPTS,BLANKING);
42  2      DECLARE W WORD,(FIELD,PROMPTS,BLANKING) BYTE;
43  2      DO I=0 TO 3;
44  3       DISP(I)=LED(SHR(W,(3—I)*4) AND 000FH);
45  3      END;
46  2      IF BLANKING=BLANK THEN          /* BLANK LEADING0'S*/
47  2       DO;
48  3       I=0;
49  3      DO WHILE DISP(I)=3FH AND I<3;
50  4        DISP(I)=0;
51  4        I=I+1;
52  4      END;
53  3      END;
54  2    CALL KB$DISPLAY (ODISP,FIELD,PROMPTS);
55  2   END;

56  1  KB$GET$CHAR:
       /* READS ONE CHARACTER FROM THE FIFO OF THE 8279. WAITS
          UNTIL CHARACTER IS AVAILABLE AND THEN RETURNS THE
          CHARACTER IN GLOBAL VARIABLE 'CHAR'. */
       PROCEDURE;
57  2      DO WHILE(INPUT(KB$STAT$PORT)AND KB$INRDY)=0;END;
59  2      OUTPUT (KB$STAT$PORT)=040H; /* ENABLE INPUT DATA */
60  2      CHAR=INPUT(KB$DATA$PORT);  /* READ CHARACTER    */
61  2   END;

        /*************************************
        * ARGUMENT EXPRESSION EVALUATOR SECTION *
        *************************************/

62  1   KB$GET$EXPR:
```

```
        /* THIS  ROUTINE  GATHERS CHARACTERS FROM  THE INPUT
           STREAM AND FORMS A WORD EXPRESSION WHICH IS RETURNED
           AS THE VALUE OF THE PROCEDURE. THE CHARACTERS INPUT
           ARE ECHOED TO THE ADDRESS OR DATA FIELD AS SPECIFIED
           BY THE FIRST PARAMETER. IF THE FIRST PARM IS 0(ADDR)
           OR 1(DATA) THEN THE EXPRESSION IS OUTPUT  AS  A  WORD
           AND IF-1(DATA$BYTE) THEN AS A BYTE  ALWAYS  IN  DATA
           FIELD. THE NUMBER OF PROMPTS BY THE SECOND PARM,AND
           LEADING ZERO  BLANKINGN  BY  THE  THIRD.  EXPRESS  ION
           ARETERMINATED WITH A COMMA,PERIOD  OR  COLON.'CH  AR
           'WILL CONTAIN ONE OF THESE ON EXIT.*/
        PROCEDURE (FIELD,PROMPTS,BLANKING) WORD;
63  2      DECLARE (FIELD,PROMPTS,BLANKING) BYTE,
             (SAVE,W) WORD, OPER BYTE;
64  2      OPER=KBPLUS;
65  2      W=0;
66  2      DO WHILE TRUE;
67  3        IF CHAR=KBREGKEY THEN
68  3          DO;                                  /* REGISTER NAME */
69  4      CALL KB$DISPLAY(OKB$REG$PMT,FIELD,PROMPTS);
70  4      CALL KB$GET$CHAR;
71  4      IF CHAR>0DH THEN GOTO ERROR;
                                                  */INVALID REG KEY */
73  4      SAVE=REG$SAV (CHAR);
74  4      IF FIELD=DATA$BYTE THEN
             CALL KB$OUT$BYTE(LOW(SAVE),PROMPTS);
           ELSE
76  4        CALL KB$OUT$WORD(SAVE,FIELD,PROMPTS,
               BLANKING);
77  4      CALL KB$GET$CHAR;
           END;
         ELSE
79  3      DO;                                    /* NUMBER */
80  4      IF CHAR>0FH THEN GOTO ERROR;
                                                  /* INVALID DIGIT */
82  4      SAVE=0;
83  4      DO WHILE CHAR<=0FH;
84  5       SAVE=SHL(SAVE,4)V⊕DOUBLE(CHAR);
```

```
 85  5          IF FIELD=DATA$BYTE THEN
 86  5            CALL KB$OUT$BYTE(LOW)SAVE),PROMPTS);
                ELSE
 87  5            CALL KB$OUT$WORD(SAVE,FIELD,PROMPTS,
                    BLANKING);
 88  5          CALL KB$GET$CHAR;
 89  5        END;
 90  4        END;
 91  3     IF OPER=KBPLUS THEN                   /* EVAL PREV OPER */
 92  3       W=W+SAVE;
           ELSE
 93  3       W=W-SAVE;
 94  3     IF FIELD=DATA$BYTE THEN
 95  3       CALL KB$OUT$BYTE(LOW(W),PROMPTS);
           ELSE
 96  3       CALL KB$OUT$WORD(W,FIELD,PROMPTS,BLANKING);
 97  3     IF CHAR=KBCOM OR CHAR=KBPER  OR CHAR=KBCOL THEN
 98  3       RETURN W;
 99  3     IF CHAR=KBPLUS OR CHAR=KBMINUS THEN
100  3       OPER=CHAR;
           ELSE
101  3       GOTO ERROR;
102  3     CALL KB$GET$CHAR;                     /* GET NEXT CHAR /*
103  3     END;
104  2     END;

105  1     KB$GET$ADDR:
           /* THIS ROUTINE  GATHERS CHARACTERS FROM  THE INPUT
             STREAM AND FORMS AN ADDRESS EXPRESSION OF SEGMENT
             PART AND  OFFSET PART. THE FIRST CHARACTER HAS
             ALREADY BEEN  READ INTO  GLOBAL 'CHAR'.IFNOSEGMENT
             IS ENTERED, THE SECOMD PARM DEFAULT IS USED, THE
             ADDRESS EXPRESSION IS DISPLAYED IN THE ADDRESS FIELD
             WITH THE NUMBER OF  PROMPTS SPECIFIED BY THE THIRD
             PARM.*/
          PROCEDURE(PTR,DEFAULT$BASE,PROMPTS);
106  2       DECLARE PTR POINTER, DFFAULT$BASE WORD,  PROMPTS
               BYTE,  ARG BASED PTR STRUCTURE (OFF  WORD,  SEG
```

```
          WORD)
107   2    ARG.SEG=DEFAULT$BASE;
108   2   ARG.OFF=KB$GET$EXPR (ADDR$FIELD, PROMPTS, BLANK);
109   2    IF CHAR=KBCOL  THEN               /* SEGMENT SPEC'D */
110   2      DO;
111   3      CALL KB$GET$CHAR;
112   3      ARG.SEG=ARG.OFF;
113   3    ARG.OFF=KB$GET$EXPR(ADDR$FIELD,PROMPTS,
             BLANK);
114   3    IF CHAR=KBCOL THEN GOTO ERROR;
116   3    END;
117   2  END;

118   1  KB$UPDATE$IP:
         /* THIS ROUTINE IS CALLED BY  SINGLE  STEP  AND  GO TO
           OUTPUT  THE CURRENT CS: IP AND THE CURRENT  INSTRU
           CTION BYTE.CS: IP IS OPENED FOR OPTIONAL INPUT.*/
         PROCEDURE;
119   2    CALL KB$OUT$WORD(IP,ADDR$FIELD,1,BLANK);
                                             /* DISPLAY IP */
120   2    CSIP.OFF=IP;
121   2    CSIP.SEG=CS;
122   2    CALL KB$OUT$BYTE(MEMORY$CSIP,0);
123   2    CALLKB$GET$CHAR;
124   2    IF CHAR< >KBCOM AND CHAR< >KBPER THEN
125   2      DO;                          /* CHANGE CS:IP */
126   3    CALL KB$BLANK$ADDR$FIELD(1);
127   3    CALL KB$BLANK$DATA$FIELD(0);
128   3    CALL KB$GET$ADDR(@CSIP,CS,1);
129   3    END;
130   2    END;

          /* * * * * * * * * * * * * * * * * * * * * * * *
          * INTERRUPT AND RESTORE/EXECUTE SECTION *
          * * * * * * * * * * * * * * * * * * * * * * * * */

131   1    SAVE$REGISTERS:
           /* THIS ROUTINE IS USED TO SAVE THE STACKED USER'S
             REGISTERS IN THE MONITOR'S SAVE AREA. */
           PROCEDORE;
```

```
132  2      BP = MEMORY$USERSTACK;
133  2      USERSTACK.0FF = USERTACK.0FF + 4;
134  2      DO I = 0 TO 10;          /* POP REGISTERS OFF OF STACK */
135  3       REG$SAV(REG$ORD(I) = MEMORY$USERSTACK;
136  3       USERSTACK.OFF = USERSTACK.OFF + 2;
137  3      END;
138  2      SS = USERSTACK.SEG;
139  2      SP = USERSTACK.OFF;
140  2      END;

141  1      RESTORE$EXECUTE:
            /* THIS PROCEDURE RESTORES THE STATE OF THE USER
               MACHINE AND PASSES CONTROL BACK TO THE USER PRO-
            GRAM. IT CONTAINS A MACHINE LANGUAGE SUBROUTINE
            TO PERFORM THE POPPING OF THE USER REGISTERS AND
            TO EXECUTE AN 'IRET' TO TRANSFER CONTROL TO THE
            USER'S PROGRAM. */
          PROCEDURE;
142  2      DECLARE RESTORE$EXECUTE$CODE(*)BYTE DATA
            (08BH,0ECH,          /* MOV BP,SP          */
             08BH,046H,002H,     /* MOV AX,/BP/.PARM2  */
             08BH,05EH,004H,     /* MOV BX,/BP/.PARM1  */
             08EH,0D0H           /* MOV SS,AX          */
             08BH,0E3H           /* MOV SP,BX          */
             05DH,               /* POP BP             */
             05FH,               /* POP DI             */
             05EH,               /* POP SI             */
             05BH,               /* POP BX             */
             05AH,               /* POP DX             */
             059H,               /* POP CX             */
             058H,               /* POP AX             */
             01FH,               /* POP DS             */
             007H,               /* POP ES             */
             0CFH),              /* IRET               */
             RESTORE$EXECUTE$CODE$PTR WORD DATA
               (.RESTORE$EXECUTE$CODE);

143  2      USERSTACK.SEG = SS;
```

```
144  2      USERSTACK.OFF=SP;
145  2      DO I=0 TO 10
                      /* PUSH USEH'S REGISTERS ONTO HIS STACK */
146  3      USERSTACK.OFF=USERSTACK.OFF-2;
147  3      MEMORY$USERSTACK=REC$SAV(REG$ORD(10-I));
148  3      END;
149  2      USERSTACK.OFF=USERSTACK.OFF-2;
150  2      MEMORY$USERSTACK=BP;
151  2      CALL RESTORE$ EXECUTE$ CODE$PTR (USERSTACK.OFF,
             USERSTACK.SEG);
152  2    END;

153  1    INTERRUPT1$ENTRY:
          /* THIS PROCEDURE IS CALLED WHEN THE CPU IS INTER-
             RUPTED BY EXECUTING AN INSTRUCTION WITH THE  TRAP
             BIT SET (SINGLE STEP). */
             PROCEDURE INTERRUPT1;
154  2      USERSTACK.OFF=STACKPTR; /*SAVE USER  STACK  INFO */
            USERSTACK.SEG=STACKBASE;
156  2      STACKPTR=MONITOR$STACKPTR;
157  2      STACKBASE=MONITOR$STACKBASE;
158  2      CALL SAVE$REGISTERS;
159  2      FL=FL AND (NOT STEP$TRAP);     /*CLEAR STEP  FLAG */
160  2      IF LAST$COMMAND< >SS$COMMAND THEN
161  2        CALL RESTORE$EXECUTE;       /* COTINUE IF NOT SS /*
162  2      CALL KB$UPDATE$IP;
163  2      IF CHAR=KBCOM THEN
164  2       DO;
165  3        IP  =CSIP.OFF;
166  3        CS  =CSIP.SEG;
167  3        FL  =FL OR STEP$TRAP;        /* SET STEP FLAG */
168  3        CALL RESTORE$EXECUTE;
169  3        END;
170  2       IF CHAR< >KBPER THEN GOTO ERROR;
172  2       GOTO AFTER$COMMAND;
173  2    END;

174  1    INTERRUPT3$ENTRY:
```

```
        /* THIS PROCEDURE IS CALLED WHEN THE CFU EXECUTES A
           'INT 3' INSTRUCTION.THE MONITOR INSERTS THIS(OCCH)
           FOR A BREAKPOINT. ALSO AN EXTERNAL INTERRUPT OR
           A USER SOFTWARE INTERRUPT MAY CAUSE THIS
           PROCEDURE TO BE CALLED. */
        PROCEDURE INTERRUPT 3;
175  2      USERSTACK.OFF=CTACKPTR;/* SAVE USER STACK INFO */
176  2      USERSTACK.SEG=STACKBASE;
177  2      STACKPTR=MONITOR$STACKPTR;
178 2       STACKBASE=MONITOR$STACKBASE;
179  2      CALL SAVE$REGISTERS;
170  2      GOTO AFTER$INTERRUPT;
171  2      END;

182  1  INIT$INT$VECTOR:
        /* THIS ROUTINE INITIALIZES AN INTERRUPT VECTOR AS
           FOLLOWS:THE OFFSET FROM THE ADDRESSOF`INT$ROUTINE'
           CORRECTED BY THE APPROPRIATE NUMBER OF BYTES FOR
           THE INTERRUPT PLM PROLOGUE.THE SEGMENT FROM THE
           CURRENT CS REGISTER IS DETERMINED BY A MACHINE
           LANGUAGE CODED SUBROUTINE. */
           PROCEDURE(INT$VECTOR$PTR,INT$ROUTINE$OFFSET);
183  2     DECLARE INT$VECTOR$PTR POINTER,
             INT$ROUTINE$OFFSET WORD,VECTOR BASED
             INT$VECTOR$PTR STRUCTURE (OFF WORD,SEG WORD),
             CORRECTION LITERALLY'19H',/*OFFSET FOR FROLOGUE*/
             INIT$INT$VECTOR$CODE(*)BYTE DATA
                (055H,                  /* PUSH BP          */
                 08BH,0ECH,             /* MOV BP,SP        */
                 08CH,0C8H,             /* MOV AX,CS        */
                 0C4H,05EH,004H,        /* LES BX,/BP/.PAEM1 */
                 026H,089H,007H,        /* MOV ES:W/BX/,AX  */
                 05DH,                  /* POP BP           */
                 0C2H,004H,000H),       /* RET 4            */
                INIT$INT$VECTOR$CODE$PTR WORD DATA
                (.IEIT$INT$VECTOR$CODE);

184  2          CALL INIT$INT$VECTOR$CODE$PTR (@VECTOR.SEG);
```

```
        /* SEGMENT PORTION /
185  2      VECTOR.OFF = INT$ROUTINE$OFFSET - CORRECTION;
        /* OFFSET PORTION */
186  2      END;

      / ******************************************************
        ******************************************************

                        COMMAND MODULE

      ABSTRACT

      THIS MODULE CONTAINS ALL THE COMMANDS IMPLEMENTED
      AS INDIVIDUAL PROCEDURES AND CALLED FROM THE OUTER
      BLOCK OF THE COMMAND DISPATCH LOOP.

      MODULE ORGANIZATION

      THIS MODULE CONTAINS THE FOLLOWING SECTIONS:
        1. COMMANDS SECTION
          KB$GO                   GO
          KB$SINGLE$STEP          SINGLE STEP
          KB$EXAM$MEM             SUBSTITUTE MEMORY
          KB$EXAM$REG             EXAMINE REGISTER
          KB$MOVE                 NOVE
          KB$INPUT                INPUT PORT
          KB$OUTPUT               OUTPUT PORT
          KB$CASSETE$DUMP         STORE MEMORY TO CASSETE TAPE
          KB$CASSTE $LOAD         LOAD CASSETE TAPE TO MEMORY
          KB$PROGR $EPROM         WRITE IMFORMATION IN RAM TO
                                  EPROM
        2 COMMAND DISPATCH(OUTER BLOCK,MAIN PROGRAM LOOP)
          NEXT$COMMAND                 DISPATCH
          ERROR                        ERROR ROUTINE
      */

          /****************************/
          *    COMMANDS SECTION        *
          ******************************/

187  1  KB$GO:
          /* IMPLEMENTS THE 'GO' COMMAND. DISPLAYS IP AND
             CURRENT INSTRUCTION BYTE AND OPENS CS: IP FOR
             INPUT AND ONE OPTIONAL BREAKPOINT.BEGINS EXECUTION
```

```
             WHEN A PERIOD IS DEPRESSED AND DISPLAYS 'E' IN THE
              ADDRESS FIELD. UPON ENCOUNTERING A BREAKPOINT.'BR'
              IS DISPLAYED IN THE DATA FIELD.*/
           PROCEDURE;
188  2        CALL KB$UPDATE$IP;     /* OPTIONAL CHANGE CS:IP */
189  2        IF CHAR=KBCOM THEN
190  2          DO;                         /* BREAKPOINT */
191  3          CALL KB$BLANK$ADDR$FIELD(1);
192  3          CALL KB$BLANK$DATA$FJELD(0);
193  3          CALL KB$GET$CHAR;
194  3          CALL KB$GET$ADDR(@BRK1,CSIP.SEG,1);
195  3          IF CHAR<>KBPER THEN GOTO ERROR;
197  3          BRK1$SAVE=MEMORY$BRK1;
198  3          MERORY$BRK1=BREAK$INST;
199  3          IF MEMORY$BRK1<>BREAK$INST THEN GOTO ERROR;
201  3          BRK1$FLAG=TRUE;
202  3          END;
            ELSE
203  2          IF CHAR<>KBPER THEN GOTO ERROR;
             CALL KB$DISPLAY(@KB$EXEC,ADDR$FIELD,0);
206  2       CALL KB$BLANK$DATA$FIELD(0);
207  2       IP=CSIP.OFF;
208  2       CS=CSIP.SEG;
209  2       FL=FL AND(NOT STEP$TRAP);
210  2       CALL RESTORE$EXECUTE;
211  2      END;

212  1  KB$SINGLE$STEP;
          /* IMPLEMENTS THE SINGLE STEP COMMAND.DISPLAYS IP AND
             THE CURRENT INSTRUCTION WORD. OPENS CS:IP FOR INPUT.
             DEPRESSINC COMMA CAUSES THE MONITOR TO SINGLE
             STEP THE INSTRUCTION,  AND  PERIOD  TERMINATES THE
             COMMAND */
          PROCEDURE;
213  2       CALL KB$UPDATE$IP;  /* OPTIONAL CRANGE OF CS:IP */
214  2       IF CHAR<>KBCOM THEN GOTO ERROR;
216  2       IP=CSIP.OFF;
217  2       CS=CSIP.SEG;
```

```
218  2      FL=FL OR STEP$TRAP;  /* SET TRAP FLAG BIT IN PSW*/
219  2      CALL RESTORE$EXECUTE;
220  2    END;

221  1  KB$EXAM$MEM:
        /* IMPLEMENTS THE EXAMINE MEMORY COMMAMD. PROMPTS
           FOR AN ADDRESS AND THEN DISPLAYS THE BYTE OR WORD
           AT THAT LOCATION. IT THEN IS OPTIONALLY OPENED FOR
           INPUT. COMMA INCREMENTS TO THE NEXT LOCATION.
           COLON DECREMENTS TO THE PREVIOUS LOCATION. PERIOD
           TERMINATES.*/
        PROCEDURE;
222  2      DECLARE W WORD;
223  2      CALL KB$BLANK$ADDR$FIELD(1);        /* PROMPT FOR
            ADDRESS */
224  2      CALL KB$GET$CRAR;
225  2      CALL KB$GET$ADDR(@ARG1.CS.1);    /*GET ADDRESS */
226  2      IF CHAR< >KBCOM THEN GOTO ERROR;
228  2      DO WHILE TRUE;
229  3          CALL KB$OUT$WORD (ARG1. OFF, ADDR$FIELD, 0,
            BLANK);      /* CLEAR PROMPT */
230  3          IF WORD$MODE THEN
231  3              CALL KB$OUT$WORD (MEMORY$WORD$ARG1,
                DATA$FIELD, 1, NOBLANK);
            ELSE
232  3              CALL KB$OUT$BYTE(MEMORY$ARG1,1);
233  3          CALL KB$GET$CHAR;
234  3          IF CHAR=KBPER THEN RETURN;
236  3          IF CHAR< >KBCOM AND CRAR< >KBCOL THEN
237  3          IF WORD$MODE THEN
238  3                  DO;
239  4                  W=KB$GET$EXPR(DATA$FIELD,1,NOBLANK);
240  4                  IF (CHAR< >KBCOM) AND (CRAR< >KBPER) AND
                    (CHAR< >KBCOL)THEN GOTO ERROR;
242  4                  MEMORY$WORD$ARG1=W;
243  4                  IF MEMORY$WORD$ARG1< >W THEN GOTO ERROR;
245  4                  END;
                ELSE
```

```
246  3                    DO;
247  4                    W=KB$GET$EXPR(DATA$BYTE,1,NOBLANK);
248  4                    IF (CHAR<>KBCOM) AND (CHAR<>KBPER) AND
                           (CHAR<>KBCOL) THEN GOTO ERROR;
250  4                    MEMORY$ARG1=LOW(W);
251  4                    IF MEMORY$ARC1<>LOW(W)THEN GOTO ERROR;
253  4                    END;
254  3               IF CHAR=KBPER THEN RETURN;
256  3               IF WORD$MODE THEN
257  3                 IF CHAR=KBCOL THEN
                          ARG1·0FF=ARG1·0FF∓2;
                       ELSE
                          ARG1·0FF=ARG1=ARG1·0FF+2;
                     ELSE
258  3                 IF CHAR=KBCOL THEN
                          ARG1·0FF=ARG1·0FF-1;
                       ELSE
                          ARG1·0FF=ARG1·0FF+1;
259  3           END;
260  2         END;

261  1     KB$EXAM$REG;
             /* IMPLEMENTS THE EXAMINE REGISTER COMMAND.PROMPTS
                FOR A VALID REGISTER KEY AND DISPLAYS THE VALUE OF
                THAT REGISTER WHICH IS OPTIONALLY OPENED FOR INPUT
                COMMA INCRENTS TO NEXT REGI STER UNLE SSIT IS 'FL'
                WHICH TERMINATESS AS DOES PERIOD. */
             PROCEDURE;
262  2         DEOLARE I BYTE, SAVE WORD;
263  2         CALL KB$BLANK$ADDR$EIELD(1);
264  2         CALL KB$GET$CHAR;
265  2         IF CHAR)ODH THEN GOTO ERROR; /* INVALID REG KEY*/
267  2         I=CHAR;
268  2         DO WHILE TRUE;
269  3         DISP(0), DISP(1)=0;                /*DISPLAY REG NAME*/
270  3         DISP(2)=KB$REG(I*2);
271  3         DISP(3)=KB$REG(I*2+1);
272  3         CALL KB$DISPLAY(@DISP, ADDR$FIELD,0);
```

```
273  3        CALL KB$OUT$WORD (REG$SAV(I), DATA$FIELD, 1,
              NOBLANK); /* VALUE */
274  3        CALL KB$GET$CHAR;
275  3        IF CHAR< >KBCOM AND CHAR< >KBPER THEN
276  3        DO;
277  4        SAVE=KB$GET$EXPR (DATA$FIELD,1,NOBLANK);
              /* UPDATE */
278  4        IF CHAR< >KBCOM AND CHAR< >KBPER THEN GOTO ERROR;
280  4        REG$SAV(I)=SAVE;
281  4        END;
282  3        IF CHAR=KBPER OR I=13 THEN RETURN;    /*DONE? */
284  3        I=I+1;
285  3        END;
286  2      END;

287  1  KB$MOVE;
          /* IMPLEMENTS THE MOVE COMMAND. PROMPTS FOR 3
              ARGUMENTS AND MOVES THE BLOCK OF MEMORY SPECIFIED
              BY ARG1-ARG2 TO ARG3. IF THERE IS A DIFFERENCE WHEN
              READ BACK, THEN ERROR. */
           PROCEDURE;
288  2        CALL KB$BLANK$ADDR$FIELD(3); /*FIRST ARGUWENT */
289  2        CALL KB$GET$CHAR;
290  2        CALL KB$GET$ADDR(@ARG1, CS, 3);
291  2        IF CHAR< >KBCOM THEN GOTO ERROR;
293  2        CALL KB$BLANK$ADDR$FIELD (2) ;/*SECOND  ARGUMENT*/
294  2        CALL KB$GET$CHAR;
295  2        END$OFF=KB$GET$EXPR(ADDR$FIELD,2,BLANK);
296  2        IF END$OFF<ARG1.OFF THEN GOTO ERROR;
298  2        IF CHAR< >KBCOM THEN GOTO ERROR;
300  2        CALL KB$BLANK$ADDR$FIELD(1); /* THIRD ARGUMENT*/
301  2        CALL KB$GET$CHAR;
302  2        CALL KB$GET$ADDR(@ARG3,ARG1,SEG,1);
303  2        IF CHAR< >KBPER THEN GOTO ERROR;
305  2  LOOP:
              MEMORY$ARG3=MEMORY$ARG1;
306  2        IF MEMORY$ARG3< >MEMORY$ARG1 THEN GOTO ERROR;
308  2        IF ARG1.OFF=END$OFF THEN RETURN;
```

```
310  2        ARG1.0FF=ARG1.0FF+1;
311  2        ARG3.0FF=ARG3.0FF+1;
312  2        GOTO LOOP;
313  2      END;

314  1   KB$INPUT:
        /* PROMPTS FOR A PORT WORD IN THE ADDRESS FIELD.WHEN
          A COMMA IS ENTERED THE BYTE OR WORD IS DISPLAYED
          IN THE DATA FIELD. THIS MAY BE REPEATED FOR MULTIPLE
          READING OF THE PORT.  PERIOD TERMINATES THE
          COMMAMD.*/
        PROCEDURE;
315  2        DECLARE PORT WORD;
316  2        CALL KB$BLANK$ADDR$FIELD(1); /*PROMPT FOR PORT*/
317  2        CALL KB$GET$CHAR;
318  2        PORT=KB$GET$EXPR (ADDR$FIELD,1,BLANK) ;
              /* GET PORT NUMBER */
319  2        CALL KB$OUT$WORD (PORT, ADDR$FIELD,0,BLANK) ;
              /* REMOVE PROMPT */
320  2  LOOP;
              IF CHAR< >KBCOM THEN GOTO ERROR;
322  2        IF WORD$MODE THEN
323  2            CALL KB$OUT$WORD(INWORD (PORT), DATA$FIELD,
                  0,NOBLANK);
              ELSE
324  2            CALL KB$OUT$BYTE(INPUT(PORT),0);
325          CALL KB$GET$CHAR;
326  2       IF CHAR=KBPER THER THEN RETURN;
328  2       GOTO LOOP;
326  2     END;

330  1  KB$OUTPUT:
        /* PROMPTS FOR A PORT WORD IN THE ADDRESS FIELD AND
           A BYTE OR WORD DATUM IN THE DATA FIELD. THIS DATUM
           IS OUTPUT A SINGLE TIME TO THE SPECIFIED PORT. */
         PROCEDURE:
331  2       DECLARE(DATUM, PORT)WORD;
332  2       CALL KB$BLANK$ADDR$FIELD(1); /*PROMPT FOR PORT*/
```

```
333  2        CALL KB$GET$CHAR;
334  2        PORT=KB$GET$EXPR(ADDR$FIELD,1,BLANK);
335  2        IF CHAR< >KBCOM THEN GOTO ERROR;
337  2        CALL KB$OUT$WORD(PORT,ADDR$FIELD,0,BLANK);
              /* REMOVE PROMPT */
338  2        CALL KB$BLANK$DATA$FIELD(1);        /* PROMPT FOR
              DATUM*/
339  2        CALL KB$GET$CHAR;
340  2   LOOP:
             IF WORD$MODE THEN
341  2          DATUM=KB$GET$EXPR (DATA$FIELD,1,NOBLANK);
                /* GET DATUM WORD */
             ELSE
342  2          DATUM=KB$GET$EXPR (DATA$BYTE,1,NOBLANK);
                /* GET DATUN BYTE */
343  2       IF CHAR=KBCOL THEN GOTO ERROR;
345  2       IF WORD$MODE THEN
346  2          OUTWORD(RORT)=DATUM;
             ELSE
347  2          OUTPUT(PORT)=LOW(DATUN);
348  2       IF CHAR=KBCOM THEN      /* MULTIPLE OUTPUTS */
349  2          DO;
350  3          CALL KB$BLANK$DATA$FIELD(1);
351  3          CALL KB$GET$CHAR;
352  3          IF CHAR< >KBPER THEN GOTO LOOP;
354  3          END;
355  2     END;

           /*********************************************************
           *      COMMAND DISPATCH MAIN PROGRAM LOOP                 *
           *********************************************************/

356  1            DISABLE;
357  1            OUTPUT(KB$STAT$PORT)=KB$INIT$MODE;
                  /* INIT 8279 */
358  1            OUTPUT(KB$STAT$PORT)=KB$INIT$SCAN;
359  1            CALL KB$DISPLAY (@KB$SIGNON, ADDR$FIELD,0);
                  /* SIGN ON MESSAGE */
360  1            CALL KB$DISPLAY (@KB$VERSION, DATA$FIELD,0);
                  /* VERSION NUMBER */
```

```
                    /* INITIALIZE USER'S REGISTERS */
361  1              CS,SS,DS,ES,FL,IP=0;
362  1              SP=USER$INIT$SP;

363  1              CALL INIT$INT$VECTOR(@INT$VECTOR(1),.
                    INTERRUPT1$ENTRY);
364  1              CALL INIT$INT$VECTOR(@INT$VECTOR(2),.
                    INTERRUPT3$ENTRY);
365  1              CALL INIT$INT$VECTOR(@INT  VECTOR(3),.
                    INTERRUPT3$ENTRY);
366  1              BRK1$FLAG=FALSE
367  1              MONITOR$STACKBASE=STACKBASE;  /* SAVE MONITOR
                    STACK VALUES */
368  1              MONITOR$STACKPTR=STACKPTR;
369  1              GOTO AFTER$ERROR;

370  1     NEXT$COMMAND:
             /* THIS IS THE PERPETUAL COMMAND LOOP WHICH DISPATCHES
                TO EACH COMMAND WHICH IS A SEPARATE PROCEDURE.*/

                CALL KB$DISPLAY(@KB$CMND$PMT,ADDR$FIELD,0);
371  1     AFTER$ERROR:
                CALL KB$GET$CHAR;
372  1          CALL KB$BLANK$ADDR$FIELD(0);
373  1          CALL KB$BLANK$DATA$FIELD(0);
374  1          IF (LAST$COMMAND:=CHAR) >OCH THEN GOTO ERROR;

376  1          WORD$MODE=FALSE;
377  1          DO CASE CHAR;
378  2              CALL KB$EXAM$MEM;
379  2              CALL KB$EXAM$REG;
380  2              CALL KB$GO;
381  2              CALL KB$SINGLE$STEP;
382  2              CALL KB$INPUT;
383  2              CALL KB$OUTPUT;
384  2              CALL KB$MOVE;
385  2              CALL KB$CASS$DUMP;
386  2              CALL KB$CASS$LOAD.
```

```
387  2          CALL KB$PROGR$EPROM;
388  2          DO; WOR$DMODE=TRUE;GALL KB$EXAM$MEM;   END;
389  2          DO; WORD$MODE=TRUE;GALL KB$INPUT;      END;
390  2          DO; WORD$MODE=TRUE;GALL KB$OUTPUT      END;
391  2       END;

392  1   AFTER$COMMAND:
             CALL KB$BLANK$DATA$FIELD(0)
393  1       GOTO NEXT$COMMAND;

394  1   ERROR:
           /* THIS ROUTINE HANDLES ALL ERRORS DETECTED BY THE
              MONITOR AND WILL OUTPUT THE ERROR MESSAGE TO THE
              DISPLAY AND THEN JUMP TO 'AFTER$ERROR' TO GET
              ANOTHER COMMAND.*/

              CALL KB$DISPLAY(@KB$ERR$MSG, ADDR$FIELD,0);
395  1        CALL KB$BLANK$DATA$FIELD(0);
396  1        GOTO AFTER$ERROR;
397  1    AFTER$INTERRUPT;

           /* THIS ROUTINE CHECKS FOR THE LAST COMMAND WHEN
              THE MONITOR IS REENTERED VIA THE INTERRUPT VECTOR.*/
          IF BRK1$FLAG THEN
404  1        DO;
405  2        MEMORY$BRK1=BRK1$SAVE;
406  2        BRK1$FLAG=FALSE;
407  2        IF ((IF-1) AND 000FH)=(BBK1.0FF AND 000FH) AND
              (SHR(IP-1,4) +CS)=(SHR(BHK1.OFF,4)+BRK1.SEG) THEN
408  2        DO:
409  3        IP=IP-1;
410  3        CALL KB$DISPLAY(@KB$BRK1,DATA$FIELD,0);
411  3        GOTO NEXT$COMMAND;
412  3        END;
413  2        END;
404  1     GOTO AFTER$COMMAND;

415  1     END MONITOR;      /* END OF MODULE */
```

EOF

MODULE INFORMATION:

```
CODE AREA SIZE          = 0B82H   2946D
CONSTANT AREA  SIZE     = 0000H      0D
VARIABLE AREA  SIZE     = 0065H    101D
MAXIMUM STACK SIZE      = 0050H     80D
865 LINES READ
0 PROGRAM ERROR(S)
```

END OF PL/M-86 COMPILATION

••
••

TP-86 A 键盘监控程序子程序

CASS DUMP PROCEDURE AND CASS LOAD

PROCEDURE OF 8086

••
••

```
KB$CASS$DUMP
C08:        PUSH   BP
            MOV    BP, SP
            MOV    AL, 02H              ；在地址段显示两个提示符“.”
            PUSH   AX
            CALL   KBBLANKADDRFIELD
            MOV    AL, 01H              ；在数据段显示一个提示符“.”
            PUSH   AX
            CALL   KBBLANKDATAFIELD
            CALL   KBGETCHAR
            CMP    CHAR, 0 FH           ；只能用16个数据键，键入文件名
            JBE    @1
            JMP    ERROR
    @1:     MOV    AL, 00H              ；由数据段接收文件名
            PUSH   AX
            MOV    AL, 01H
```

```
        PUSH    AX
        PUSH    AX
        CALL    KBGETEXPR
        MOV     AX, SAVE
        PUSH    AX                  ; 文件名进栈
        DEC     SP
        PUSH    AX
        INC     SP
        CMP     CHAR, 11H           ; 以“,”结束文件名
        JZ      @2
        POP     AX
        POP     AX
        JMP     ERROR
@2:     PUSH    AX                  ; 在数据段显示文件名并去掉提示符
        MOV     AL, 00H
        PUSH    AX
        PUSH    AX
        IN      AX
        PUSH    AX
        CALL    KBOUTUORD
        CALL    KGETCHAR            ; 接收首地址
        LEA     AX, ARG1
        PUSH    DS
        PUSH    AX
        PUSH    AX
        PUSH    REGSAV+10H
        MOV     AL, 02H
        PUSH    AX
        CALL    KBGETADDR
        CMP     CHAR, 11H           ; 以“,”结束输入首址
        Jz      $+5H
        JMP     ERROR
        MOV     AL, 01H             ; 在地址段上显示一个提示符
        PUSH    AX
        GALL    KBBLANKADDR
        CALL    KBGETCHAR           ; 接收末地址
        MOV     AL, 01H
        PUSH    AX
```

```
PUSH    AX
PUSH    AX
CALL    KBGETEXPR
MOV     ENDOFF, AX
CMP     AX, AG1
JNB     $+5H
JMP     ERROR
CMP     CHAR, 10H          ; 以“·”结束键入命令，并执行转
                           录功能
JZ      $+5H
JMP     ERROR
PUSH    CS                 ; 显示转录提示符“d”
MOV     AX, SIGN1
PUSH    AX
MOV     AL, 01H
PUSH    AX
MOV     AL, 00H
PUSH    AX
CALL    KBDISPLAY
MOV     DX, 0FFF2H         ; 8251初始化模式
MOV     AL, FFH
OUT     (DX), AL
MOV     AL, 31H            ; 命令
OUT     (DX), AL
MOV     AX, 0045H          ; 30 秒延时
CALL    KBDELAY
MOV     AL, 3AH            ; 发“: ”
PUSH    AX
CALL    KBTRAN
CALL    KBTRAN             ; 发文件名高八位
CALL    KBTRAN             ; 发文件名低八位
MOV     AX, 0002H          ; 1 秒延时
CALL    KBDELAY
MOV     AX, ENDOFF         ; 发文件长度-1
SUB     AX, ARG1
PUSH    AX
DEC     SP
PUSH    AX
```

```
          INC     SP
          CALL    KBTRAN
          CALL    KBTRAN
          PUSH    CS                         ; 消除数据段上的文件名
          MOV     AX, SIGN1+1
          PUSH    AX
          MOV     AL, 00H
          PUSH    AX
          PUSH    AX
          CALL    KRDISPLAY
          LES BX, MEMORYARG 1 PTP            ; 循环发送
          MOV AL, ES: MEMORYARG1[BX]
          PUSH    AX
          CALL    KBTRAN
          MOV     AX, ARG1
          CMP     AX, ENDOFF
          JNZ     NOTEND
          MOV     AX, 0010H                  ; 发送结束，送五秒全“1”
          CALL    KBDELAY
          MOV     AL, 50H                    ; 复位 8251
          OUT     (DX), AL
          POP     BP
          RET
NOTEND:   ADD     ARG1, 01H
          JMP     LOOP
SING1:    DW      5E, 00, 00, 00, 00
KBDELAY:  MOV     CX, 0FFFFH
LOOP1:    DEC     CX
          JNZ     LOOP1
          DEC     AX
          JNZ     KBDELAY
          RET
KBTRAN:   PUSH    BP
          MOV     BP, SP
          MOV     DX, 0FFF2H
WAIT1:    IN      AL, (DX)
          AND     AL, 01H
          JR      Z, WAIT1
```

```
         MOV    AL, 11H
         OUT    (DX), AL
         MOV    DX, 0FFF0H
         MOV    AL, [BP]·TRANBYTE
         OUT    (DX), AL
         MOV    DX, 0FFF2H
         MOV    AL, 31H
         OUT    (DX), AL
WAIT2:   IN     AL, (DX)
         AND    AL, 04H
         JZ     WAIT 2
         POP    BP
D 30     RET    02H
      ——END——
```

```
KB$CASS$LOAD
D 33     PUSH   BP
         MOV    BP, SP
         MOV    AL, 01H                显示提示符
         PUSH   AX
         PUSH   AX
         CALL   KBBLANKADDRFIELD
         CALL   KBBLANKDATAFIELD
         CALL   GETCHAR                ; 文件名只能使用16个数据键
         CMP    CHAR, 0FH
         JBE    @3
         JMP    ERROR
     @3: MOV    AL, 00H                ; 接收键入文件名带有提示符，并将
                                         文件名存入栈。
         PUSH   AX
         MOV    AL, 01H
         PUSH   AX
         PUSH   AX
         CALL   KBGETEXPR
         PUSH   SAVE
```

```
        CMP     CHAK, 11 H              ; 文件名以“，”结束
        JZ      $ + 5H
        POP     AX
        JMP     ERROR
        POP     AX                      ; 文件名出栈
        PUSH    AX                      ; 在线上保留一个文件名
        PUSH    AX                      ; 显示文件名不带提示符
        MOV     AL, 00 H
        PUSH    AX
        PUSH    AX
        MOV     AL, 01 H
        PUSH    AX
        CALL    KBOUTWORD
        CALL    KBGETCHAR               ; 接收转储起始地址
        LEA     AX, ARG1
        PUSH    DS
        PUSH    XA
        PUSH    REGSAV + 10 H
        MOV     AL, 01 H
        PUSH    AX
        CALL    KBGETADDR
        CMP     CHAR, 10 H              ; 地址结束用“·”，并执行
        JZ      @4
        POP     AX
        JMP     ERROR
SING2:  DW      38, 00, 00, 00
   @4:  PUSH    CS                      ; 显示转储运行提示符“L”
        MOV     AX, SIGN2
        PUSH    AX
        MOV     AL, 01
        PUSH    AX
        MOV     AL, 00H
        PUSH    AX
        CALL    KBDISPLAY
        MOV     DX, 0 FFF 2 H           ; 8251初始化
        MOV     AL, 0 BFH
        OUT     (DX), AL
        MOV     AL, 50 H
```

```
        OUT     (DX), AL
        MOV     AL, 0FFH
        OUT     (DX), AL
        MOV     AL, 16H
        OUT     (DX), AL
REPEAT: CALL    RBRECIV          ；直到接收到“：”为止
        CMP     AL, 3AH
        JNZ     REPEAT
        POP     CX               ；将键入名记在CX中
        CALL    KBRECIV          ；接收文件名
        MOV     AH, AL
        CALL    KBRECIV
        PUSH    CX
        INC     CX               ；从栈中取，取出用户键入文件名
        JZ      START            ；如果为全“F”，不与磁带上的文件
                                   名比较
        POP     CX
        CMP     AX, CX
        JZ      START            ；否则比较磁带上的名与用户键入名
        JMR     REPEAT           ；不一致重新接收“：”
SLART:  POP     CX
        PUSH    CS               ；文件名相符，消除文件名
        MOV     AX, SING1+1
        PUSH    AX
        MOV     AL, 00H
        PUSH    AX
        PUSH    AX
        CALL    KBDISPLAY
        CALL    KBRECIV          ；接收文件长度并放入CX
        MOV     CH, AL
        CALL    KBRECIV
        MOV     CL, AL
        INC     CX
LOOP:   CALL    KBRECIV          ；循环接收
        LES     BX, MEMORYARG1PTP
        MOV     ES: MEMORYARG1 [BX], AL
        ADD     ARG1+1
        DEC     CX
```

```
            JNZ     LOOP
            MOV     AL, 50H
            OUT     (DX), AL
            POP     BP
            RET
KBRECIV:    MOV     DX, 0FFF2
            IN      AL, (DX)
            AND     AL, 02H
            JZ      RECIN
            IN      AL, (DX)
            AND     AL, 70H          ; 如有溢出、校验、帧误错之一，
                                     转 ERROR 处理
            JZ      $+5H
            JMP     ERROR
            MOV     DX, 0FFF0H
            IN      AL, (DX)         ; 出口参数，AL=接收字符 DX=
                                     FFF2, 8251 壮志口
E12         RET

——          END


KB$PROC$EPROM
E13         PUSH    BP
            MOV     BP, SP
            MOV     AL, 01H          ; 显示提示符，数据段 1 个地址段
                                     3 个
            PUSH    AX
            CALL    KBBLANKDATAFIELD
            MOV     AL, 03H
            PUSH    AX
            CALL    KBBLANKADDRFIELD
            CALL    KBGETCHAR        ; 编程类型2716或2732，不能用功
                                     能键
            CMP     CHAR, 0FH
            JBE     @2
```

```
$$1:    JMP     ERROR
 @2:    MOV     AL, 00 H              ; 接收编程类型
        PUSH    AX                    ; 带提示符在数据段上
        MOV     AL, 01 H
        PUSH    AX
        PUSH    AX
        CALL    KBGETEXPR
        CMP     CHAR, 11 H            ; 需用“,”结束
        JNZ     $$1
        MOV     AX, SAVE              ; 编程类型辨别是 2716 还是 2732
        SUB     AX, 2716 H
        JZ      @5
        SUB     AX, 1 CH
        JNZ     $$1                   ; 都不是，则出错
        INC     AX
 @5:    INC     AX                    ; 2732“2”标志,2716“1”标志压入栈
        PUSH    AX
        NOP
        MOV     AL, 00 H              ; 清除编程类型
        PUSH    AX
        CALL    UBBLANKDATAFIELD
        CALL    KBGETCHAR             ; 接收源首地址
        LEA     AX, ARG1
        PUSH    DS
        PUSH    AX
        PUSH    REGSAV + 10 H
        MOV     AL, 3 H
        PUSH    AX
        PUSH    AX
        CALL    KBGETADDR
        CMP     CHAR, 11 H
        JZ      $ + 5 H
$$2:    POP     AX
        JMP     $$1
        MOV     AL, 02 H              ; 显示两个提示符
        PUSH    AX
        CALL    KBBLANKADDRFIELD
        CALL    KBGETCHAR             ; 接收源末地址
```

```
MOV     AL, 1 H
PUSH    AX
MOV     CL, 2 H
PUSH    CX
PUSH    AX
CALL    KBGETEXPR
MOV     ENDOFF, AX              ; 末地址小于首地址为错误
SUB     AX, ARG1
JNB     $ + 5H
POP     AX
JMP     $ $ 1
POP     BX                      ; BX = 编程类型
NOP                             ; 将 2716/2732 标记 1/2 变换成
                                  1000/2000
MOV     CL, 4 H
ROR     BX, CL
DEC     BX                      ; 2716/2732 对应于FFF/1FFF 文
                                  件最大长度
CMP     BX, AX                  ; 2716 文件长度
JB      $ $ 1
PUSH    BX                      ; 标记 FFF/1FFF 进栈
PUSH    AX                      ; 文件长度进栈
CMP     CHAR, 11 H
IZ      $ + 5 H
POP
POP
JMP     ERROR
MOV     AL, 1 H                 ; 显示一个提示符
PUSH    AX
CALL    KBBLANKADDRFIELD
CALL    KBGETCHAR               ; 接收编程相对地址
MOV     AL, 01 H
PUSH    AX
PUSH    AX
PUSH    AX
CALL    KBGETEXPR
NOV     ARG3, AX                ; 传送相对首地址
```

```
        POP     BX                          ; 编程相对首地址+文件长不能
                                              大于 FFF/1FFF
        ADD     BX, AX
        POP     AX
        CMP     AX, BX
        JB      ERROR
        MOV     CL, 04 H                    ; 2716 段地址送 F900 H
        ROL     AX, CL                      ; 2732 段地址送 F 800 H
        XOR     AX, 09 FFH
        MOV     ARG3+2, AX
        CMP     CHAR, 10 H                  ; 以"·"结束，并执行
        JNZ     ERROR
        PUSH    CS                          ; 显示编程提示符"P"
        MOV     AX, SIGN4                   ; SIGN4: 73, 00, 00, 00
        PUSH    AX
        MOV     AL, 01 H
        PUSH    AX
        DEC     AX
        PUSH    AX
        CALL    KBDISPLAY
        MOV     DX, 0 FFE 6 H               ; 8253 初始化
        MOV     AL, 30 H
        OUT     (DX), AL
LOOP3:  MOV     CX, 0002 H                  ; 误错重写次数
LOOP1:  MOV     DX, 0 FFE 0 H               ; 8253 置时间常数
        MOV     AL, 00 H
        OUT     (DX), AL
        MOV     AL, 30 H
        OUT     (DX), AL
        LES     BX,MEMORYARGIPTR
        MOV     AL,ES:MEMORYARG1 [BX]; 取数据
        LES     BX,MEMERYARG 3 PTR
        MOV     ES:MEMORYARG 3[BX],AL: 写入 EPROM
        MOV     DX, 0005H                   ; 写后延时
LOOP2:  DEC     DX
        JR      NZ LOOP 2
        CMP     AL,ES,MEMORYARG 3 [BX]
        JZ      GOOD
```

```
         DEC     CX                        ; 如有错再写一次
         JNZ     LOOP 1
         PUSH    BX                        ; 两次写有错
         MOV     AL, 01 H                  ; 将相对地址显示在地址段
         PUSH    AX
         PUSH    AX
         PUSH    AX
         CALL    KBOUTWORD
         PUSH    CS
         MOV     AX  SIGN3                 ; SING 3: 40, 7C, 77, 5E
         PUSH    AX                        ; 将错误标志"-bAd"显示在数
                                             据段
         MOV     AL, 00 H
         PUSH    AX
         PUSH    AX
         CALL    DISPLAY
         CALL    KBGETCHAR                 ; 等待直到收到字符
         CMP     CHAR, 10 H
         JZ      $ +5 H                    ; "·"为正确结束
         JMP     ERROR
         POP     BP
         RET
SIGN 3:  DW:     40, 7C, 77, 5E
SIGN 4:  DW:     73, 00, 00, 00
GOOD :   MOV     AX, ARG 1
         CMP     AX, END 0FF
         JNZ     @5
         POP     BP
         RET
@5:      ADD     ARG1, 1 H                 ; 地址加 1 写下一个单元
         ADD     ARG 3, 1 H
F55      JMP     LOOP 3
——       END     ——
```

附　录　Ⅱ

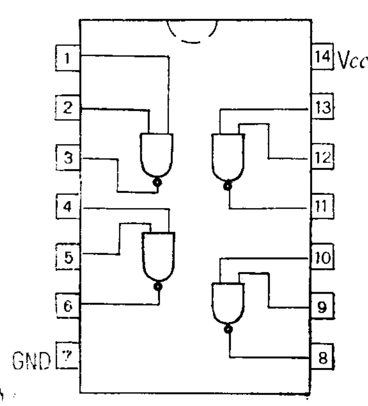
TP−86A 所用主要集成电路引脚图

图　序　表

图号	名	称
I.1	74LS00	四单元二输入与非门
I.2	74LS02	四单元二输入或非门
I.3	74LS04	六单元反相器
I.4	7406	六单元反相缓冲/驱动器
I.5	74LS08	四单元二输入与门
I.6	74LS10	三单元三输入非门
I.7	74LS14	六单元施密斯反相器
I.8	74LS20	二单元四输入与非门
I.9	74LS30	八输入与非门
I.10	74LS32	四单元二输入或门
I.11	7474	二单元正沿触发 D 触发器
I.12	74LS164	串行送入并行输出移位寄存器
I.13	74LS393	二单元模数 16 位计数器
I.14	7445	十中选一译码/驱动器
I.15	74LS133	十三输入与非门
I.16	74LS367A	六单元三态缓冲器
I.17	74LS155	二单元四中选一译码器
I.18	74LS138	八中选一译码器
I.19	MC14538B	二单元精密可触发/可复位单稳多谐振荡器
I.20	6116	2K ×8 Bit 静态 RAM
I.21	74LS373	8位锁存器
I.22	74LS244	八位缓冲/总线驱动器
I.23	8251A	可编程通讯接口
I.24	8286	八位总线收发器
I.25	2716	2K ×8 Bit EPROM
I.26	2732	4K ×8 Bit EPROM
I.27	8284	8086时钟发生器
I.28	8253	可编程计数/计时器
I.29	8086CPU	中央处理器CPU
I.30	8255A	可编程并行接口
I.31	8279	可编程键盘/显示接口

74LS00：四单元二输入与非门

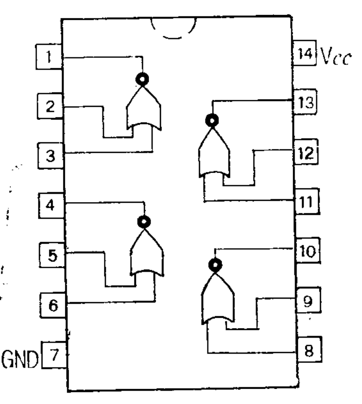
引脚图



图Ⅱ.1

74LS02：四单元二输入或非门

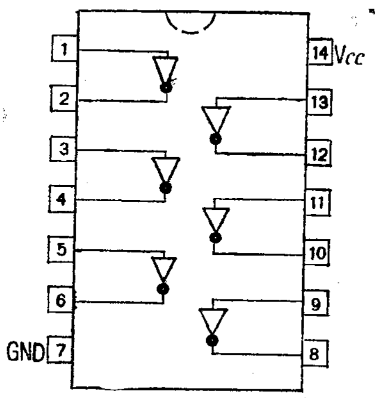
引脚图



图Ⅱ.2

74LS04：六单元反相器

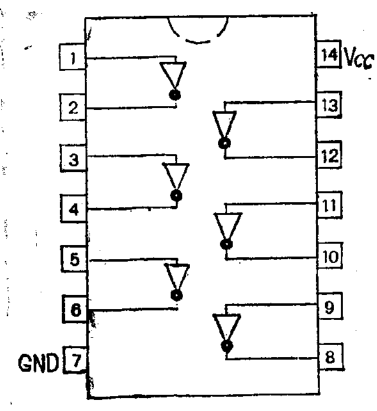
引脚图



图Ⅱ.3

7406：六单元反相缓冲/驱动器

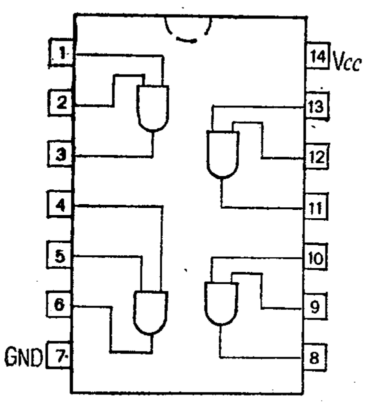
引脚图



图Ⅱ.4

74LS08：四单元二输入与门

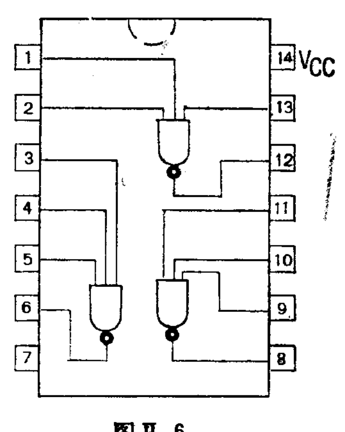
引脚图



图Ⅱ.5

74LS10：三单元三输入与非门

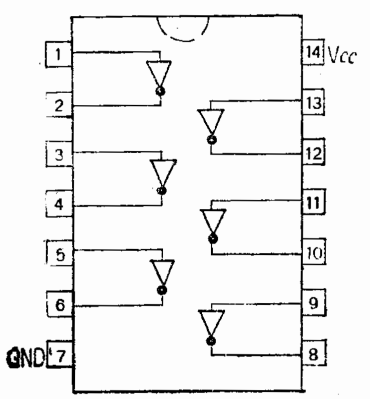
引脚图



图Ⅱ.6

74LS14：六单元施密斯反相器

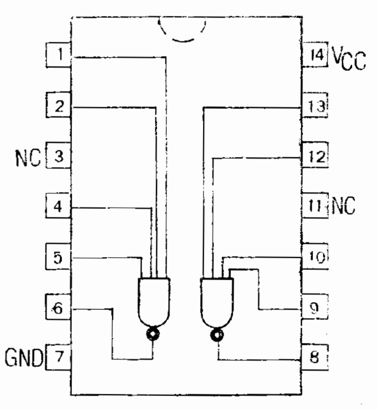
引脚图



图Ⅱ.7

74LS20：二单元四输入与非门

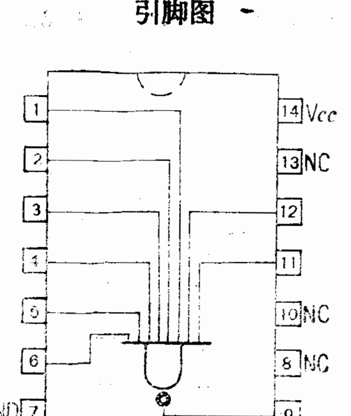
引脚图



图Ⅱ.8

74LS30：八输入与非门

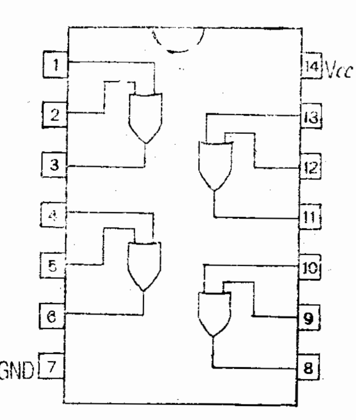
引脚图



图Ⅱ.9

74LS32：四单元二输入或门

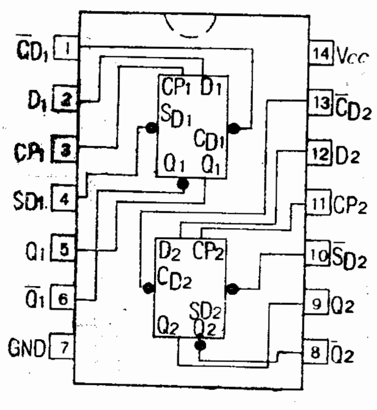
引脚图



图Ⅱ.10

74 74 二单元正沿触发D 触发器

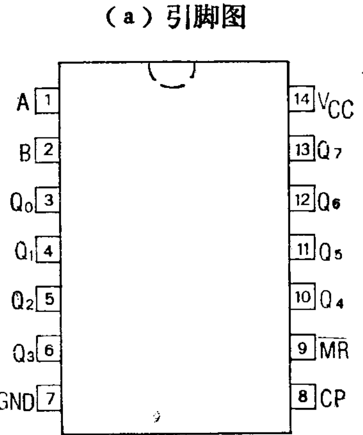
引脚图



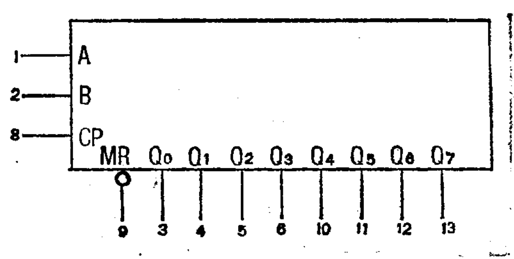
图Ⅱ.11

74LS164：串行输入并行输出移位寄存器

（a）引脚图



（b）功能图

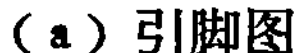


（c）真值表

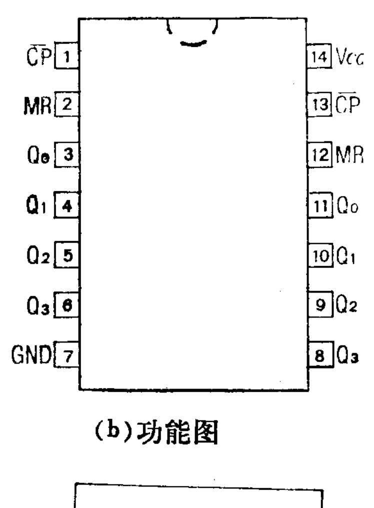
OPERATING MODE	INPUTS			OUTPUTS	
	$\overline{MR}$	A	B	Q_0	$Q_1 - Q_7$
Reset (Clear)	L	X	X	L	L — L
Shift	H	l	l	L	$q_0 - q_6$
	H	l	h	L	$q_0 - q_6$
	H	h	l	L	$q_0 - q_6$
	H	h	h	H	$q_0 - q_6$

图Ⅱ.12

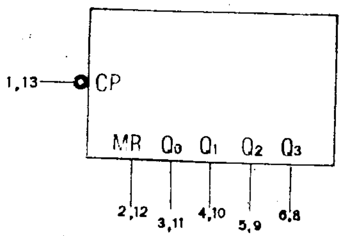
74LS393：二单元模数16位计数器



（a）引脚图



（b）功能图



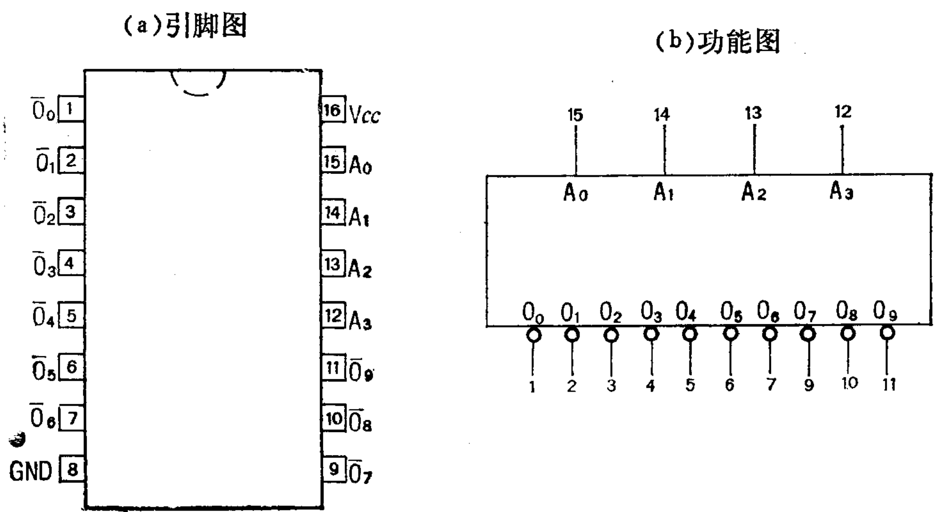
（c）真值表

COUNT	OUTPUTS			
	Q_3	Q_2	Q_1	Q_0
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	L	H	L	H
6	L	H	H	L
7	L	H	H	H
8	H	L	L	L
9	H	L	L	H
10	H	L	H	L
11	H	L	H	H
12	H	H	L	L
13	H	H	L	H
14	H	H	H	L
15	H	H	H	H

H＝高电位

L＝低电位

图Ⅱ.13



(c)真值表

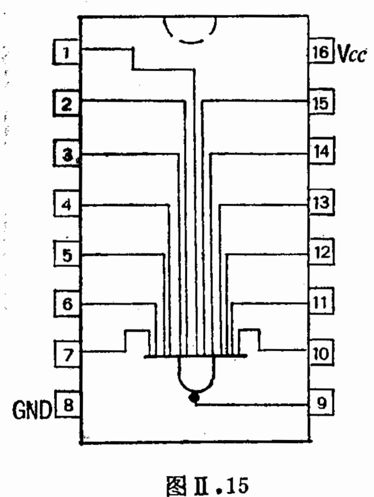
INPUTS				OUTPUTS									
A_0	A_1	A_2	A_3	$\overline{O}_0$	$\overline{O}_1$	$\overline{O}_2$	$\overline{O}_3$	$\overline{O}_4$	$\overline{O}_5$	$\overline{O}_6$	$\overline{O}_7$	$\overline{O}_8$	$\overline{O}_9$
L	L	L	L	L	H	H	H	H	H	H	H	H	H
H	L	L	L	H	L	H	H	H	H	H	H	H	H
L	H	L	L	H	H	L	H	H	H	H	H	H	H
H	H	L	L	H	H	H	L	H	H	H	H	H	H
L	L	H	L	H	H	H	H	L	H	H	H	H	H
H	L	H	L	H	H	H	H	H	L	H	H	H	H
L	H	H	L	H	H	H	H	H	H	L	H	H	H
H	H	H	L	H	H	H	H	H	H	H	L	H	H
L	L	L	H	H	H	H	H	H	H	H	H	L	H
H	L	L	H	H	H	H	H	H	H	H	H	H	L
L	H	L	H	H	H	H	H	H	H	H	H	H	H
H	H	L	H	H	H	H	H	H	H	H	H	H	H
L	L	H	H	H	H	H	H	H	H	H	H	H	H
H	L	H	H	H	H	H	H	H	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	H
H	H	H	H	H	H	H	H	H	H	H	H	H	H

H＝高电位
L＝低电位

图Ⅱ.14

74LS133 十三输入与非门

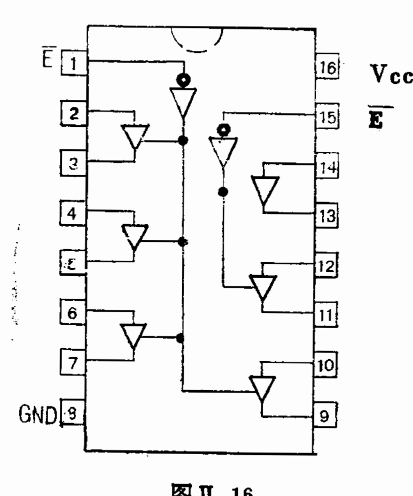
引脚图



图Ⅱ.15

74LS367A：六单元三态缓冲器

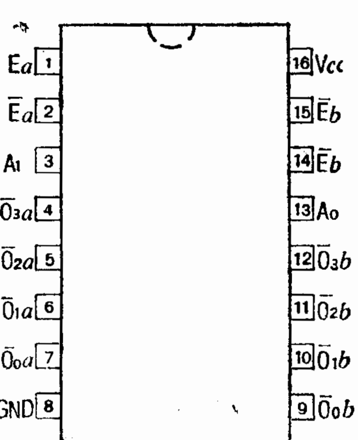
引脚图



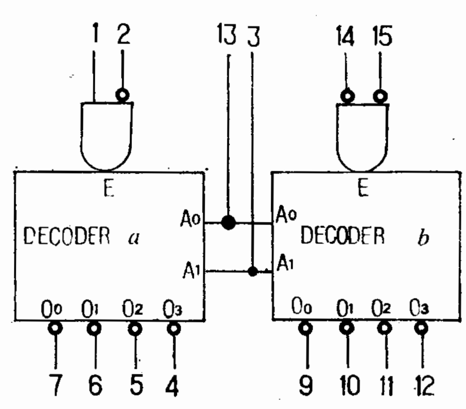
图Ⅱ.16

74LS155：二单元四中选一译码器

（a）引脚图



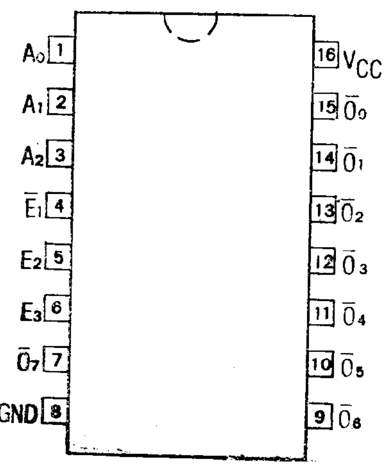
（b）功能图



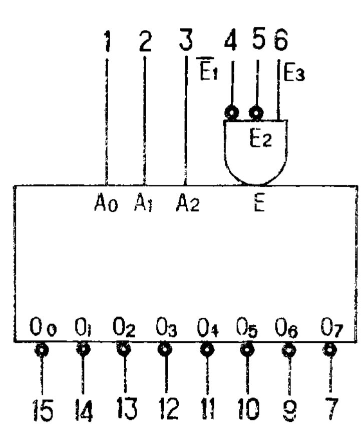
图Ⅱ.17

74LS138 八中选一译码器

（a）引脚图



（b）功能图



（c）真值表

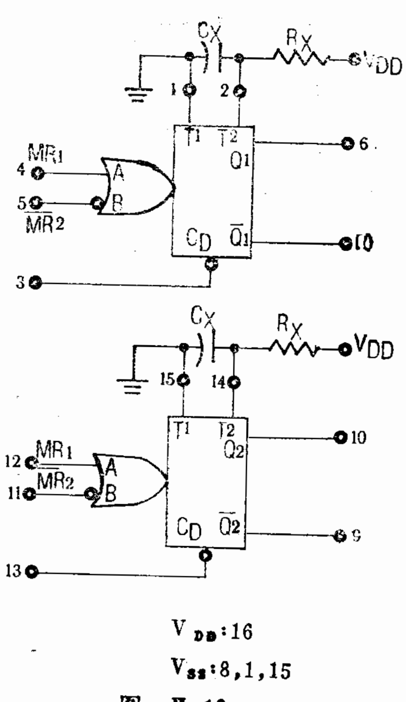
INPUTS						OUTPUTS							
$\overline{E}_1$	$\overline{E}_2$	E_3	A_0	A_1	A_2	$\overline{O}_0$	$\overline{O}_1$	$\overline{O}_2$	$\overline{O}_3$	$\overline{O}_4$	$\overline{O}_5$	$\overline{O}_6$	$\overline{O}_7$
H	X	X	X	X	X	H	H	H	H	H	H	H	H
X	H	X	X	X	X	H	H	H	H	H	H	H	H
X	X	L	X	X	X	H	H	H	H	H	H	H	H
L	L	H	L	L	L	L	H	H	H	H	H	H	H
L	L	H	H	L	L	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	L	H	H	H	H	H
L	L	H	H	H	L	H	H	H	L	H	H	H	H
L	L	H	L	L	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	L	H	H
L	L	H	L	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	L

H＝高电位
L＝低电位
X＝任意

图Ⅱ.18

MC14538B：二单元精密可触发/可复位单稳多谐振荡器

功能图



V_{DD}:16

V_{SS}:8,1,15

图 Ⅱ.19

6116：2K×8 Bit静态RAM

引脚图

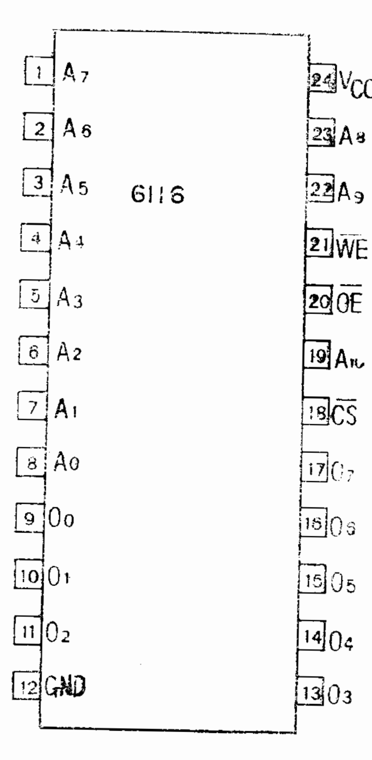


图 Ⅱ.20

74LS373：八位锁存器

(a)引脚图

OE	1	20	Vcc
O0	2	19	O7
D0	3	18	D7
D1	4	17	D6
O1	5	16	O6
O2	6	15	O5
D2	7	14	D5
D3	8	13	D4
O3	9	12	O4
GND	10	11	LE

(b)功能图

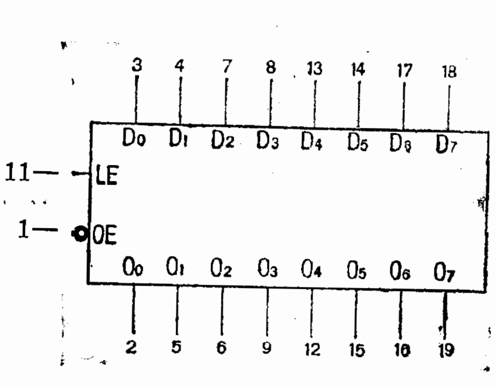


图 Ⅱ.21

74LS244　八位缓冲/总线驱动器（三态输出）

引脚图

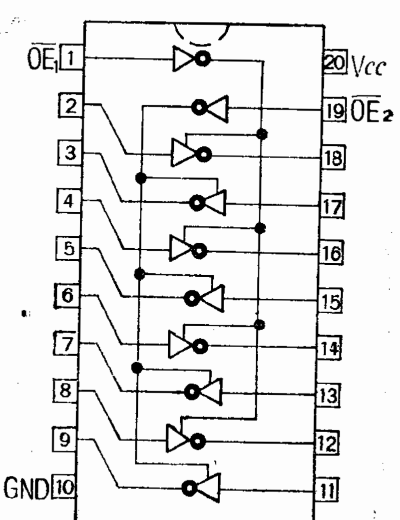


图　Ⅱ.22

8251A　可编程通讯接口

引脚图

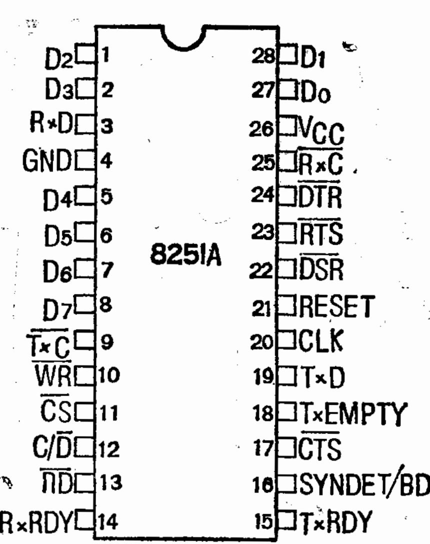


图　Ⅱ.23

8286：八位总线收发器

(a)引脚图

(b)功能图

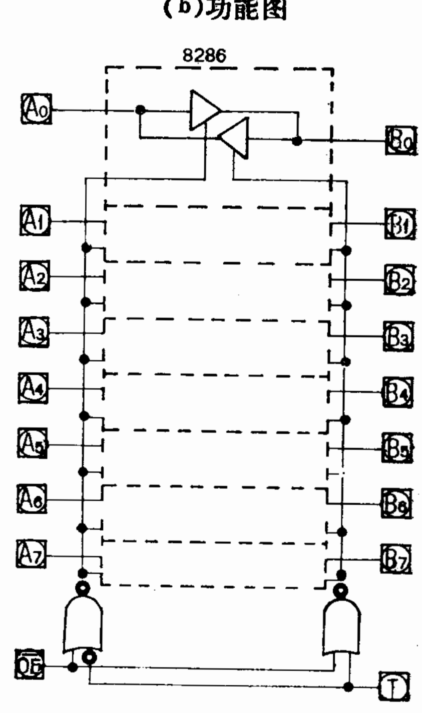
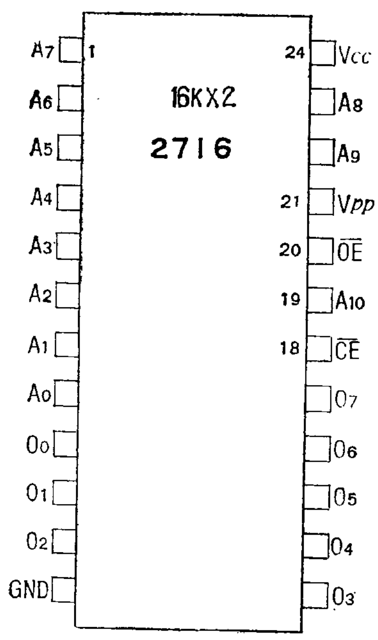


图　Ⅱ.24

2716：2K×8 Bit EPROM

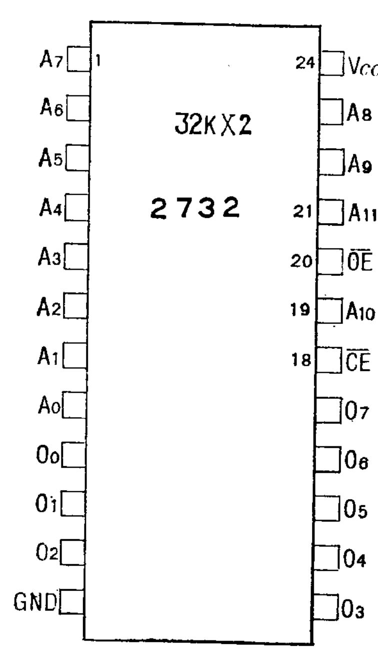
引脚图



图Ⅱ.25

2732：4K×8 Bit EPROM

引脚图



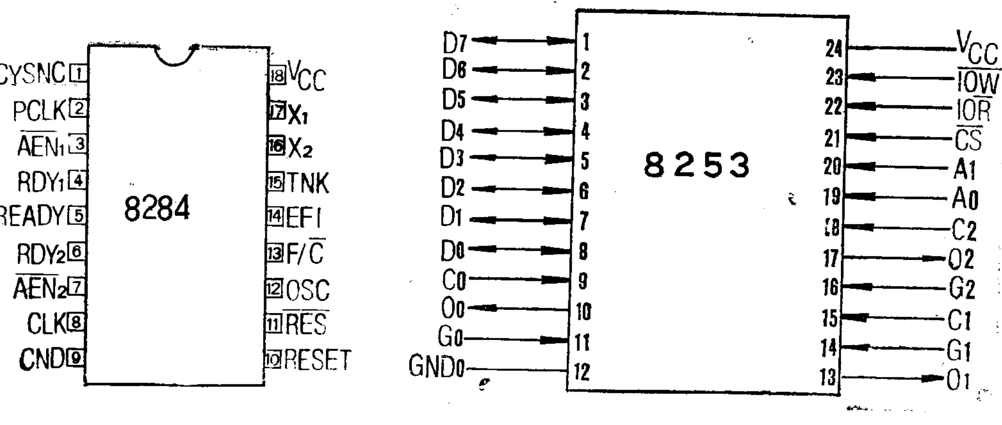
图Ⅱ.26

8284：8086，8088，8089

处理器时钟发生驱动器

引脚图

8253　可编程计数/计时器



图Ⅱ.27

图Ⅱ.28

8086CPU：8086中央处理器引脚图

8086 CPU

Pin	Name	Pin	Name
1	GND	40	VCC
2	AD_{14}	39	AD_{15}
3	AD_{13}	38	A_{16}/S_3
4	AD_{12}	37	A_{17}/S_4
5	AD_{11}	36	A_{18}/S_5
6	AD_{10}	35	A_{19}/S_6
7	AD_9	34	$\overline{BHE}/S_7$
8	AD_8	33	$MN/\overline{MX}$
9	AD_7	32	$\overline{RD}$
10	AD_6	31	$\overline{RQ}/\overline{GT_0}$ (HOLD)
11	AD_5	30	$\overline{RQ}/\overline{GT_1}$ (HLDA)
12	AD_4	29	$\overline{LOCK}$ ($\overline{WR}$)
13	AD_3	28	$\overline{S_2}$ ($M/\overline{IO}$)
14	AD_2	27	$\overline{S_1}$ ($DT/\overline{R}$)
15	AD_1	26	$\overline{S_0}$ ($\overline{DEN}$)
16	AD_0	25	QS_0 (ALE)
17	NMI	24	QS_1 ($\overline{INTA}$)
18	INTR	23	$\overline{TEST}$
19	CLK	22	READY
20	GND	21	RESET

图Ⅱ.29

8255A：可编程并行接口引脚图

8255A

Pin	Name	Pin	Name
1	PA_3	40	PA_4
2	PA_2	39	PA_5
3	PA_1	38	PA_6
4	PA_0	37	PA_7
5	$\overline{RD}$	36	$\overline{WR}$
6	$\overline{CS}$	35	RESET
7	GND	34	D_0
8	A_1	33	D_1
9	A_0	32	D_2
10	PC_7	31	D_3
11	PC_6	30	D_4
12	PC_5	29	D_5
13	PC_4	28	D_6
14	PC_0	27	D_7
15	PC_1	26	V_{CC}
16	PC_2	25	PB_7
17	PC_3	24	PB_6
18	PB_0	23	PB_5
19	PB_1	22	PB_4
20	PB_2	21	PB_3

图Ⅱ.30

8279：可编程键盘/接口显示引脚图

8279

Pin	Name	Pin	Name
1	RL_2	40	VCC
2	RL_3	39	RL_1
3	CLK	38	RL_0
4	IRO	37	CNTL/STB
5	RL_4	36	SHIFT
6	RL_5	35	SL_3
7	RL_6	34	SL_2
8	RL_7	33	SL_1
9	RESET	32	SL_0
10	$\overline{RD}$	31	$OUTB_0$
11	$\overline{WR}$	30	$OUTB_1$
12	DB_0	29	$OUTB_2$
13	DB_1	28	$OUTB_3$
14	DB_2	27	$OUTA_0$
15	DB_3	26	$OUTA_1$
16	DB_4	25	$OUTA_2$
17	DB_5	24	$OUTA_3$
18	DB_6	23	$\overline{BD}$
19	DB_7	22	$\overline{CS}$
20	V_{SS}	21	A_0

图Ⅱ.31

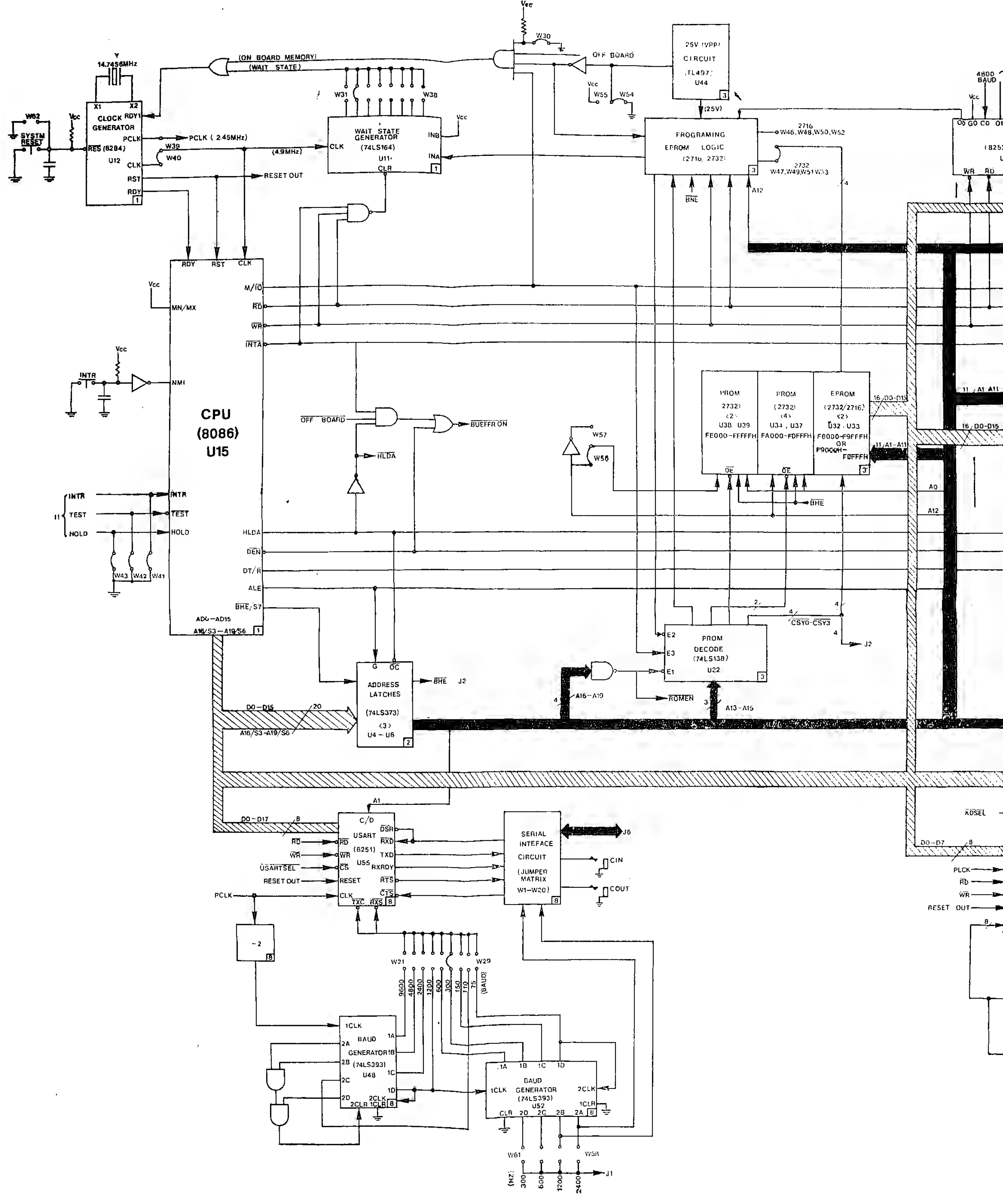
附　录 Ⅲ

TP-86A电原理綫路图

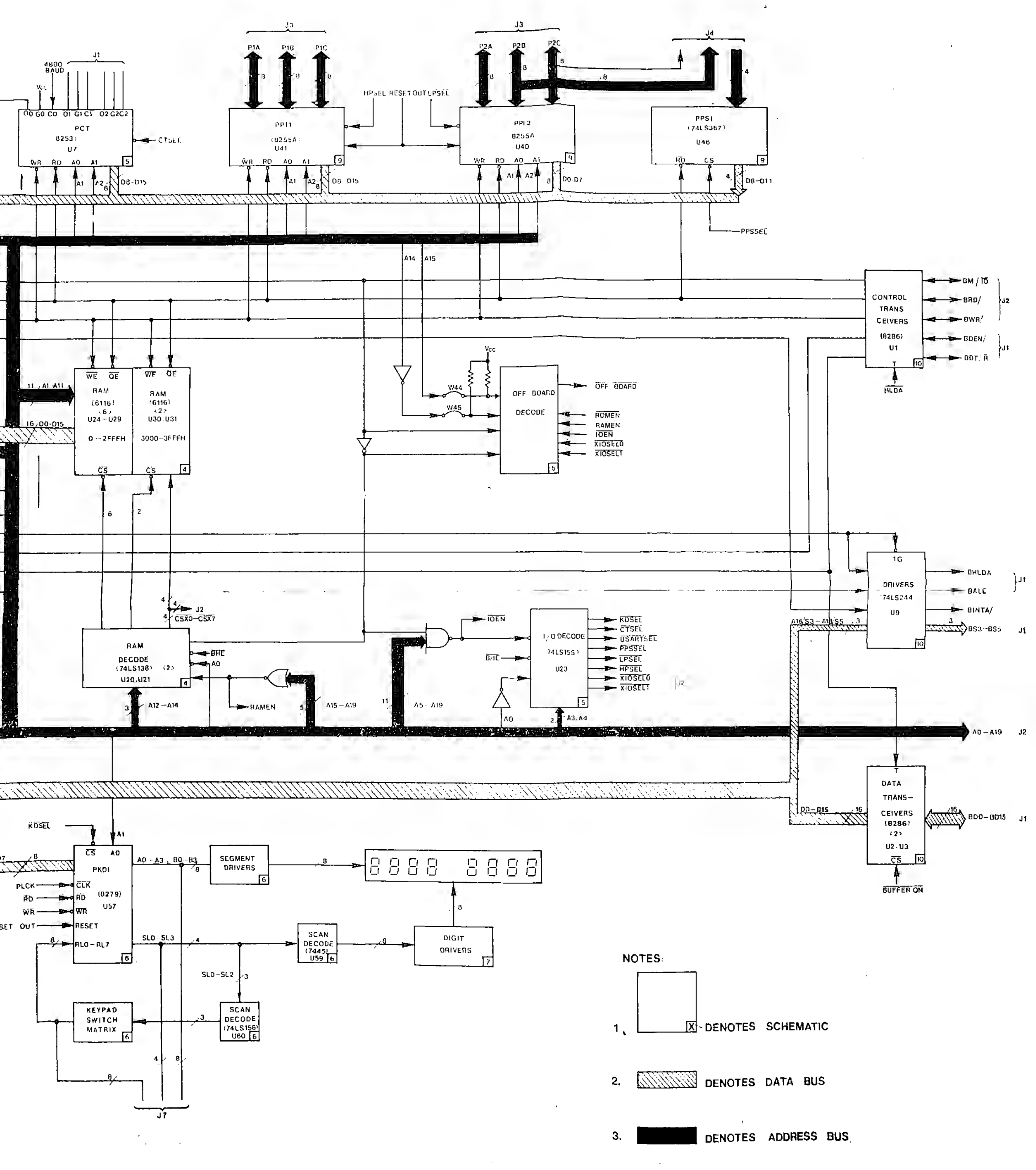
（附图一—十）

图 序 表

图号	名称
（一）	时钟、等待电路及CPU
（二）	地址锁存电路
（三）	PROM
（四）	RAM
（五）	I/O、译码及计数计时电路
（六）	键盘显示控制电路
（七）	显示驱动电路
（八）	串行接口电路
（九）	并行接口电路
（十）	总线扩展电路
（插页）	TP-86A单板计算机安装位置图



TP-86A



-86A Detailed Block Diagram