INSTRUCTION MANUAL

TMA PC-27

POWER MODULE CONTROL INTERFACE BOARD

KEPCO INC.

MODEL
TMA PC-27
POWER MODULE CONTROL INTERFACE BOARD

ORDER NO.	REV. NO.

IMPORTANT NOTES:

1) This manual is valid for the following Model and associated serial numbers:

MODEL	SERIAL NO.	REV. NO.

2) A Change Page may be included at the end of the manual. All applicable changes and revision number changes are documented with reference to the equipment serial numbers. Before using this Instruction Manual, check your equipment serial number to identify your model. If in doubt, contact your nearest Kepco Representative, or the Kepco Documentation Office in New York, (718) 461-7000, requesting the correct revision for your particular model and serial number.

3) The contents of this manual are protected by copyright. Reproduction of any part can be made only with the specific written permission of Kepco, Inc.

Data subject to change without notice.

KEPCO®
THE POWER SUPPLIER™

KEPCO, INC. ● 131-38 SANFORD AVENUE ● FLUSHING, NY. 11352 U.S.A. ● TEL (718) 461-7000 ● FAX (718) 767-1102
email: hq@kepcopower.com ● World Wide Web: http://www.kepcopower.com

TABLE OF CONTENTS

TABLE OF CONTENTS Continued

LIST OF ILLUSTRATIONS

LIST OF TABLES

APPENDICES

SECTION 1
INTRODUCTION

1.1 PURPOSE OF MANUAL

This manual describes the installation and operation of KEPCO's TMA PC-27 Power Module Control Interface Board and associated software. The associated software is contained on a Kepco-supplied disk and allows operation of the TMA PC-27 with various languages.

This manual contains five sections and an appendix. Section 1 contains a General Description. Section 2 describes Equipment Configuration. Section 3 describes Installation. Section 4 describes Programming With The PC-27. Section 5 describes the Kepco Program Manager. An appendix contains program examples in various languages.

1.2 GENERAL DESCRIPTION

The Kepco TMA PC-27 power module control interface board (see Figure 1-1) will allow you to communicate with up to 27 of Kepco's MAT line of programmable linear DC supplies from an IBM (or compatible) computer. The board plugs into any expansion slot in the computer.

The associated software can either be installed as a TSR (Terminate Stay Resident) or stand alone program. As a TSR it can be accessed by many popular programming languages, including the following:

TURBO C
QUICK C
POWER C
TURBO PASCAL
QUICK PASCAL
TopSpeed PASCAL
QUICK BASIC
INTERPRETED BASIC
TopSpeed MODULA-2

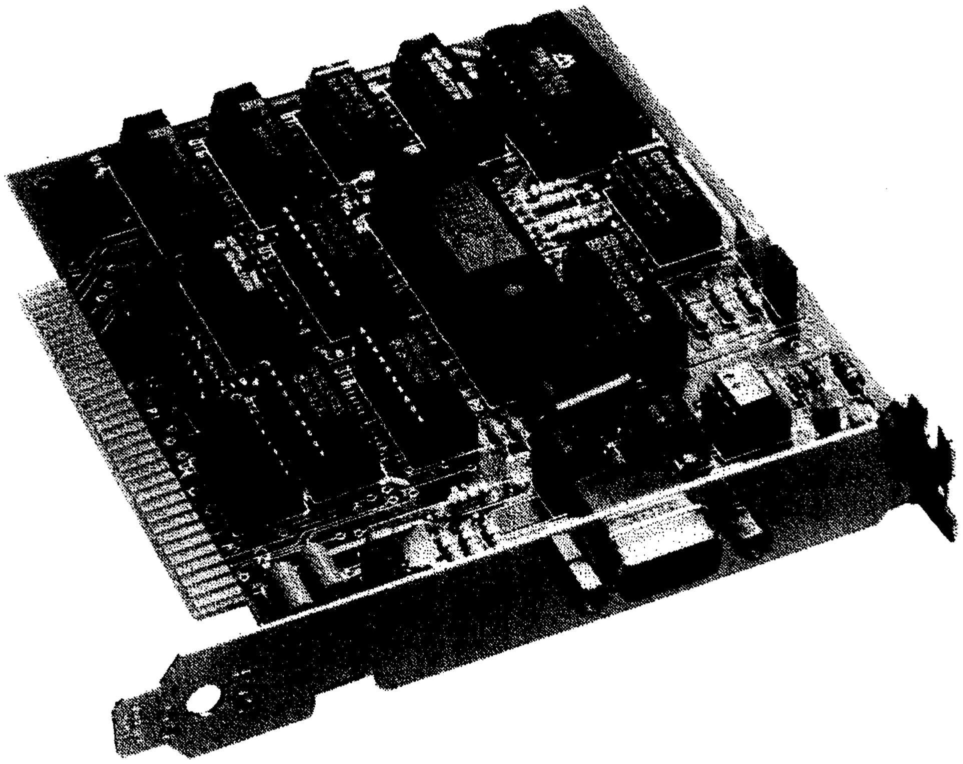

Figure 1-1. Model TMA PC-27

SECTION 2
EQUIPMENT CONFIGURATION

2.1 BASE ADDRESS AND HARDWARE INTERRUPT LEVEL

Configuring the TMA PC-27 board includes making selections to the following features:

Base Address
Hardware Interrupt Level

The TMA PC-27 is supplied with factory default settings of base address 208 (hex) and hardware interrupt level 2 (IRQ2). If desired (e.g. defaults in conflict with other programs or hardware), optional settings are possible (see Table 2-1). Figure 2-1 illustrates the location of the base address and hardware interrupt jumper blocks on the TMA PC-27 for alternate settings.

ITEM	DEFAULT	OPTIONAL
Base Address (Hex)	208H	Within the range of 200H to 3F8H, refer to Table 2-2.
Interrupt Level (IRQ)	2	2 to 7

Table 2-1. Base Address and Hardware Interrupt level

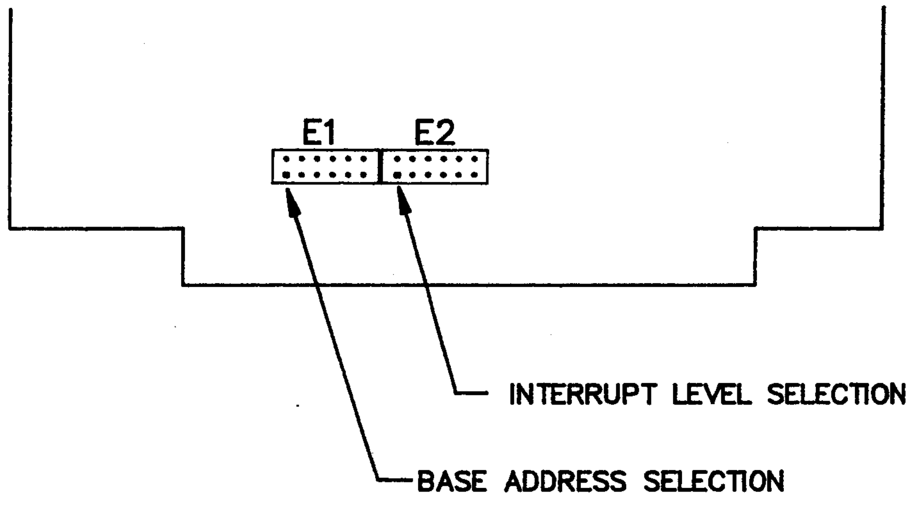

Figure 2-1. Base Address and Hardware Interrupt Level Jumper Blocks, Location

2.2 CONFIGURING THE BASE ADDRESS

The default base address value, 208H, is appropriate for most applications. Examples of jumper configurations for some of the optional base addresses are shown in Figure 2-2. Refer to Table 2-2 for a complete listing of optional base addresses. In configuring the base address interrupt, a jumper OUT sets the bit to 1.

Each base address consists of 12 bits (A0 through A11) with A0 being the least significant bit (LSB). The 12 base address bits are arranged in three 4-bit groups, corresponding to the three characters in each base address. Three of the four bits associated with the first and last characters of all base addresses are preset (e.g. 2 and 8 of base address 208).

Using the base address 208H as an example:

2	(binary 0010)	A11 is always preset to 0.
		A10 is always preset to 0.
		A9 is always preset to 1.
		A8 is selectable via the jumper block (in this example it is 0).
0	(binary 0000)	A7 is selectable via the jumper block.
		A6 is selectable via the jumper block.
		A5 is selectable via the jumper block.
		A4 is selectable via the jumper block.
8	(binary 1000)	A3 is selectable via the jumper block (in this example it is 1).
		A2 is always preset to 0.
		A1 is always preset to 0.
		A0 is always preset to 0.

The six jumper positions on the base address jumper block reflect, in ascending order, the six accessible bits (A3 through A8). The jumper block position for bit A3 is by the "E1" jumper block designation on the printed circuit board.

2.3 CONFIGURING THE HARDWARE INTERRUPT

The default hardware interrupt, "5", is appropriate for most applications. Jumper configurations for optional hardware interrupts are shown in Figure 2-3.

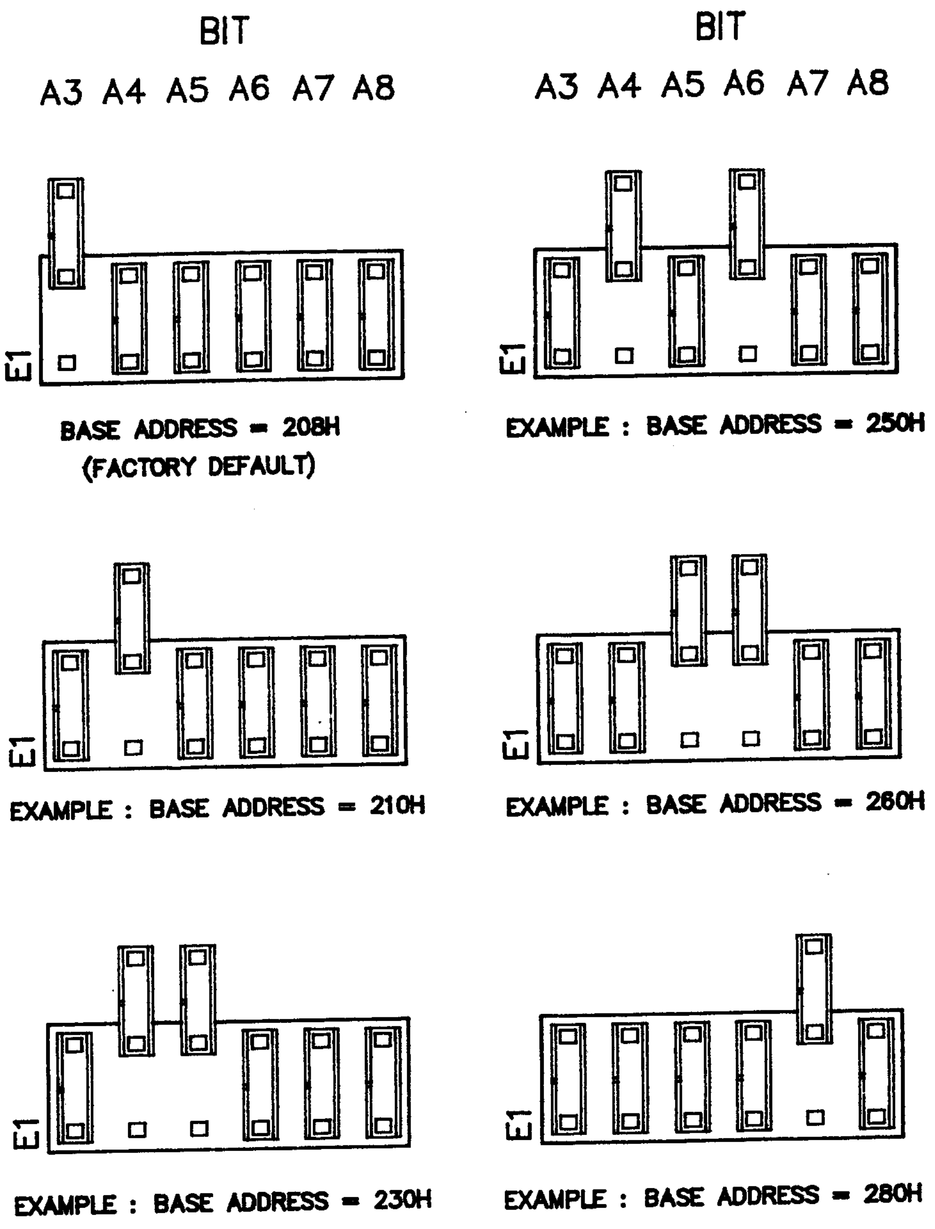
BIT
A3 A4 A5 A6 A7 A8
E1
BASE ADDRESS = 208H
(FACTORY DEFAULT)
BIT
A3 A4 A5 A6 A7 A8
E1
EXAMPLE : BASE ADDRESS = 250H
E1
EXAMPLE : BASE ADDRESS = 210H
E1
EXAMPLE : BASE ADDRESS = 260H
E1
EXAMPLE : BASE ADDRESS = 230H
E1
EXAMPLE : BASE ADDRESS = 280H

Figure 2-2. Base Address Jumper Examples

	BIT					
ADDRESS	A8	A7	A6	A5	A4	A3
200						
208						*
210					*	
218					*	*
220				*		
228				*		*
230				*	*	
238				*	*	*
240			*			
248			*			*
250			*		*	
258			*		*	*
260			*	*		
268			*	*		*
270			*	*	*	
278			*	*	*	*
280		*				
288		*				*
290		*			*	
298		*			*	*
2A0		*		*		
2A8		*		*		*
2B0		*		*	*	
2B8		*		*	*	*
2C0		*	*			
2C8		*	*			*
2D0		*	*		*	
2D8		*	*		*	*
2E0		*	*	*		
2E8		*	*	*		*
2F0		*	*	*	*	
2F8		*	*	*	*	*

* = JUMPER OUT (BIT = 1)

Table 2-2. Optional Base Addresses (Sheet 1 of 2)

	BIT					
ADDRESS	A8	A7	A6	A5	A4	A3
300	*					
308	*					*
310	*				*	
318	*				*	*
320	*			*		
328	*			*		*
330	*			*	*	
338	*			*	*	*
340	*		*			
348	*		*			*
350	*		*		*	
358	*		*		*	*
360	*		*	*		
368	*		*	*		*
370	*		*	*	*	
378	*		*	*	*	*
380	*	*				
388	*	*				*
390	*	*			*	
398	*	*			*	*
3A0	*	*		*		
3A8	*	*		*		*
3B0	*	*		*	*	
3B8	*	*		*	*	*
3C0	*	*	*			
3C8	*	*	*			*
3D0	*	*	*		*	
3D8	*	*	*		*	*
3E0	*	*	*	*		
3E8	*	*	*	*		*
3F0	*	*	*	*	*	
3F8	*	*	*	*	*	*

* = JUMPER OUT (BIT = 1)

Table 2-2. Optional Base Addresses (Sheet 2 of 2)

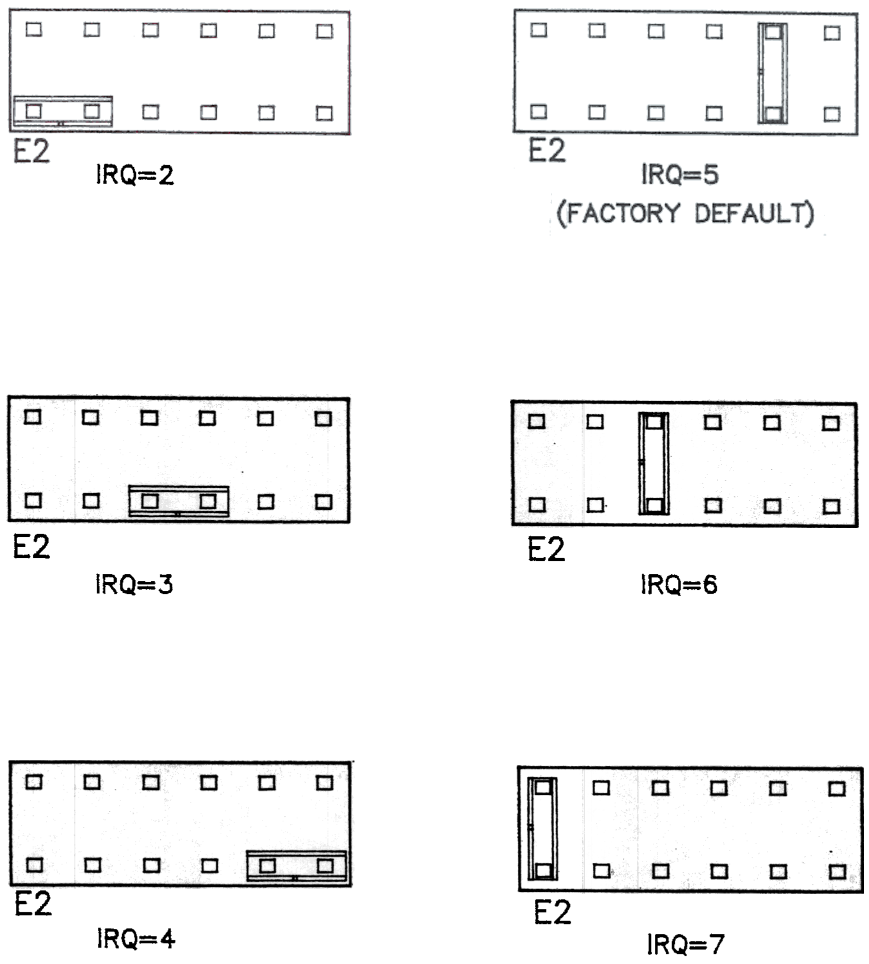

Figure 2-3. Hardware Interrupt Level, Jumper Locations

SECTION 3
INSTALLATION

3.1 TMA PC-27 BOARD

The following steps should be followed when installing the TMA PC-27 board:

A) Turn off the computer and unplug the power cord.

B) Remove the computer cover.

C) Select an empty expansion slot and remove the mounting screw and bracket.

D) Insert the TMA PC-27 board into the slot and replace the cover.

E) Replace the computer cover and connect the power cord.

F) Connect the 9-pin side (J1) of the interface cable (Kepco Part Number 118-0749) to the TMA PC-27. The other side of the cable can now be connected to the MAT control bus. Pin assignments for connector J1 are as follows:

Pin 2	Ground
Pin 3	DATA
Pin 7	Ground
Pin 8	DATA
Pins 1,4,5,6,9	No Connection

3.2 TMA PC-27 INTERFACE TO MAT POWER MODULE

One shielded, twisted-pair cable (2 meters long) with a mating connector at each end (Kepco part number 118-0699) is supplied with each rack adapter (RA 50 or RA 51) and with each full rack MAT Power Module. The TMA PC-27 Power Module Interface and up to 27 MAT Power Modules can be connected in a daisy chain configuration (see Figure 3-1). The last (in the daisy chain) Power Module Control Bus Outlet must be terminated with a terminating connector assembly (Kepco part number 195-0075) that is supplied with the TMA PC-27.

3.3 DRIVER SOFTWARE

CAUTION

If your computer does not have a hard drive, make a "working" copy of the disk provided. If your computer has a hard drive, copy the disk onto it. Store the original in a safe place.

One of two command sets may be used to program the TMA PC-27. The two command sets are CIIL (Control Interface Intermediate language) and KPSL (Kepco Power Supply Language). The TMA.BAT batch file (included on supplied disk) is used to select which driver is to be used. The drivers are TMACIIL.EXE (for CIIL command set) and TMAKPSL.EXE (for KPSL command set).

To install the TMA PC-27 driver from the DOS command line, type: **TMA** and strike the **ENTER** key.

To install the PC-27 driver from your AUTOEXEC.BAT file, the statement TMA must be included in the file. The program is then loaded into memory and remains installed until the computer is rebooted or reset. The TMA program uses software interrupt 60 (hex) and assumes hardware interrupt IRQ2 and base address 208. These defaults may be altered by using an option code when installing the TSR as follows:

-i allows selection of the software interrupt (e.g. **TMA -i67** selects software interrupt 67 (hex)).

-h allows selection of the hardware interrupt (e.g. **TMA -h3** selects hardware interrupt IRQ3). Ensure that the hardware level interrupt jumper position coincides with the hardware level interrupt selection made (see Figures 2-1 and 2-3).

-b allows selection of the base address (e.g. **TMA -b200** selects base address 200 (hex)). Ensure that the base address jumper position coincides with the base address selection made (see Figures 2-1, 2-2 and Table 2-2).

-s runs a stand-alone, non-TSR program (e.g. **TMA -s**), which allows direct entry of commands from the keyboard.

NOTE: Multiple options (e.g. **TMA -b200 -h5**) are permitted.

The TMA .BAT file may be modified to accommodate alternate settings. For example, from DOS type:

COPY CON TMA.BAT and strike the **ENTER** key
TMACIIL -b200 -h5 -i67 and strike the **F6** key
Strike the **ENTER** key to update the file and return to DOS

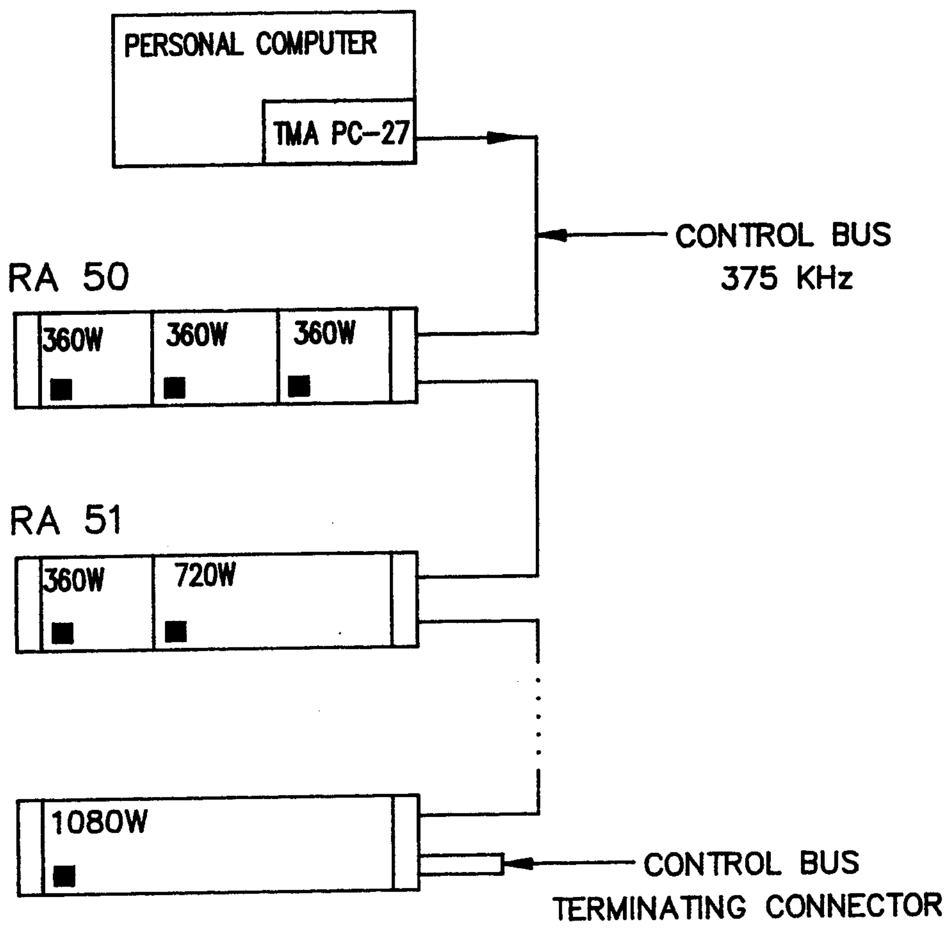

Figure 3-1. TMA PC-27 To MAT Power Module, Interface

SECTION 4
PROGRAMMING WITH THE TMA PC-27

4.1 GENERAL

With the TMA software loaded, the KEPCO MAT Power Supply Modules may be programmed over the control bus using CIIL (Control Interface Intermediate Language) or KPSL (Kepco Power Supply Language) commands. CIIL, defined by military standard 2806763 provides a common language for instruments used in an automatic test system. KPSL is a simplified command set based on CIIL. In addition, some functions that are not supported by CIIL have been included (refer to paragraphs 4.4 and 4.6).

4.2 USING THE PC-27 SOFTWARE

Data is passed to the TMA PC-27 through the use of registers and a software interrupt. Register AX must be loaded with the value 0 for C, 3 for Pascal, and 4 for Basic (see sample programs in appendix). Registers BX and ES must be loaded with the address of the data string to be sent. Following this, software interrupt 60 (hex) must be called. Upon returning from the interrupt, the AX register will contain a status byte, this value will be zero if execution was normal or a value other then 0 if the data sent was invalid. If the command sent requires a response, the response message will be returned in the same data string used for the command, otherwise that data string will be nulled.

NOTE: Examples for each programming language are contained on the included program disk. You may use them as a model for your application programs.

4.3 INTRODUCTION TO THE KPL PROGRAM MANAGER

Kepco has provided an alternate way to create application programs quickly and easily. This is through the use of our own language interpreter, KPL (Kepco Programming Language). KPL is used at Kepco to create test programs for our products. It provides for instant I/O communications for the TMA PC-27 and is compatible with National Instruments NI-488 DOS Handler for the IEEE-488 GPIB. This means if you are controlling other instruments with a National Instruments IEEE-488 interface card, you can easily communicate with them as well. Selecting between the TMA PC-27 and the IEEE-488 is a simple matter (refer to paragraph 5.8.6).

KPL can be used to try out different programming ideas, or it can be used for your actual test programs. It was designed for ease of use and quick results. KPL offers many advantages due to its modularity and editing features. Because KPL is an interpreted language no time is lost waiting for the program to recompile every time you make a small change or correct a "bug". Most test applications do not require the extra speed of a compiled program. From the Manager environment you can trace through, edit, or run programs. You can also call subroutines directly or execute whole programs from disk while maintaining variables. Programs may call other programs without losing data and DOS is only a keystroke away. You can control instruments directly from the KPL Program Manager menu or read back data by pressing a single key. You can easily build up a library of routines and execute them from the manager menu or put them together to create larger programs. Routines are called by names, not meaningless numbers. Once a program is created and "debugged", it may be executed directly from DOS. Section 5 gives a detailed description of the KPL Program Manager.

4.4 COMMAND SET SUMMARY

CIIL COMMANDS:

Operation codes (op codes):

FNC	FUNCTION
SET	SET
SRX	SET MAXIMUM
SRN	SET MINIMUM
INX	INITIATE
FTH	FETCH
CLS	CLOSE
OPN	OPEN
IST	INTERNAL SELF TEST
CNF	CONFIDENCE TEST
RST	RESET
STA	STATUS

Nouns :

DCS	DIRECT CURRENT SOURCE

Noun Modifiers:

VOLT	VOLTAGE MODE OPERATION
VLTL	VOLTAGE LIMIT
CURR	CURRENT MODE OPERATION
CURL	CURRENT LIMIT

KPSL COMMANDS:

SET	SET
FTH	FETCH
CLS	CLOSE
OPN	OPEN
IST	INTERNAL SELF TEST
CNF	CONFIDENCE TEST
RST	RESET
STA	STATUS

UTILITY COMMANDS FOR CIIL AND KPSL:

REN	REMOTE ENABLE
GTL	REMOTE DISABLE (GO TO LOCAL)
DCL	DEVICE CLEAR
S0	NO SIGNAL ON ERROR
S1	MESSAGE SIGNAL ON ERROR
S2	BEEP SIGNAL ON ERROR
S3	MESSAGE AND BEEP SIGNALS ON ERROR
T0	DON'T SAVE NON CATASTROPHIC ERRORS
T1	SAVE NON CATASTROPHIC ERRORS
REST	SYSTEM RESET
F0	NO ERROR REPORT ON FETCH
F1	ERROR REPORT ON FETCH
P0	REPORT POWER LOSS ONCE
P1	REPORT POWER LOSS CONTINUOUSLY

Power loss mode 1. Continuously reports a power loss message until power is restored to the power module.

DELIMITERS

All op codes and operands must be separated by the ASCII space (ASCII value 32).

4.5 CIIL COMMAND DESCRIPTIONS

4.5.1 FUNCTION Command

FNC - Function Op Code - This operator sets up a power module's output (stimulus mode), or sets up the power module to read its output settings (sensor mode).

```
SYNTAX:     FNC DCS :CHnn SET VOLT (stimulus mode)
            FNC DCS VOLT :Chnn (sensor mode)
```

The first operand contains the three (3) letter mnemonic pertaining to the device on the control bus, in this case **DCS** (Direct Current Source). If a reading is being set up, the modifier **VOLT** or **CURR** follows.

The next operand is used to select the specific channel of the device being programmed or read from. The TMA PC-27 can control up to 27 MAT power modules with control bus addresses in the range of 1 to 31.

4.5.2 SET,SRX,and SRN Commands

SET - Set Op Code -This operator is used in conjunction with FNC (in stimulus mode) to specify the output mode of the power module being programmed. The first operand is the noun modifier and the second operand specifies the value.

```
SYNTAX:     SET VOLT value SET CURL value
            SET CURR value SET VLTL value
```

NOTE: A complete command must include **FNC DCS :CHnn**
i.e. **FNC DCS :CH1 SET VOLT 5 SET CURL 2**

The first operand field of the command contains the four(4) letter mnemonic for the output mode of the power module. The choices are:

VOLT	VOLTAGE MODE OPERATION
VLTL	VOLTAGE LIMIT
CURR	CURRENT MODE OPERATION
CURL	CURRENT LIMIT

The second operand field of the command contains the value assigned to the chosen output mode. This value may be specified as accurately as the resolution of the MAT power module allows (12 bits, or .024% of maximum rated voltage or current). It can be directly specified in ASCII integer, decimal, or in scientific notation.

There may be two (2) set commands, separated by a space (ASCII 32), for each power module being programmed. The following are the only allowable combinations:

VOLT with CURL
CURR with VLTL

The limit parameter (CURL or VLTL) may not be set without the main parameter. A polarity sign may precede the VOLT or CURR value so that the power module's polarity may be selected.

In the case of Kepco's MAT power modules, the two(2) related Op Codes, SRX and SRN are functionally identical to the SET command, since there is only one range, 0 - maximum. The commands are included only for compatibility.

SRX	Set Range Maximum
SRN	Set Range Minimum

4.5.3 OPEN/CLOSE RELAY Commands

These commands are used to enable or disable the power modules from their loads.

OPN	opens the power module relay specified by the operand.
CLS	Closes the power module relay specified by the operand.
SYNTAX	OPN :CHnn CLS :CHnn

4.5.4 RESET Command

RST - Reset Op Code. This operator is used to return a power module to its power-on state. The output voltage and current are programmed to zero and the power relay is opened.

SYNTAX: RST DCS :CHnn

4.5.5 CONFIDENCE TEST Command

CNF - Confidence Test Command - This operator commands the TMA PC-27 to execute the confidence test procedure defined for the MAT power modules. The procedure consists of opening all power relays, programming voltage and current to their maximum values, switching polarity, checking for error flags, then programming voltage and current to zero.

SYNTAX: CNF

The results of **CNF** may be obtained through the **STA** command.

4.5.6 INTERNAL SELF TEST Command

IST -Internal Self Test Command. For Kepco's MAT power modules, this command is functionally equivalent to the **CNF** command.

4.5.7 STATUS Command

STA - Status Request Op Code - This operator commands the TMA PC-27 to report its present operating status. Any catastrophic error conditions which exist will be reported, until the error condition is corrected (refer to paragraph 4-8).

SYNTAX: STA

4.5.8 INITIATE Command

INX-Initiate Op Code - commences a data acquisition process in accordance with the preceding FNC command. The response to the **INX** command is a dynamic time-out value, unless a catastrophic error condition exists, in which case an error message will be returned. If the time-out value returned is not zero,this indicates the power module's output voltage or current has not yet settled. A time delay should be observed before proceeding with the **FTH** command or the command may be repeated until a zero value is returned, but the preceding **FNC** command must also be repeated. The advantage of this method is that if the module settles quickly following the first time-out value returned, the reading may be obtained without need for a delay.

SYNTAX: INX VOLT (initiate voltage reading)
INX CURR (initiate current reading)

4.5.9 FETCH Command

FTH - Fetch Op Code - commands the previously designated power module to return the requested data reading. It must immediately follow an **INX** command. The value returned is the value of the output voltage or current, whichever was requested, unless a catastrophic error condition exists, in which case an error message will be returned (value observed will be in scientific notation).

SYNTAX: FTH VOLT (fetch voltage reading)
FTH CURR (fetch current reading)

4.6 KPSL COMMAND DESCRIPTIONS

4.6.1 SET Command

SET - Allows setting of voltage or current

SYNTAX: SET n Vx Ly

Sets channel n to x volts and the current limit to y amps.

SYNTAX: SET n Cx Ly

Sets channel n to x amps and the voltage limit to y volts.

4.6.2 FETCH Command

FTH - Allows reading (fetching) of a voltage or current

SYNTAX: FTH Vn

Fetches the voltage on channel n.

SYNTAX: FTH Cn

Fetches the current on channel n.

4.6.3 CLOSE Command

CLS - Closes a power module's power relay, which connects the load.

SYNTAX: CLS n

Closes the relay on channel n.

4.6.4 OPEN Command

OPN - Opens a power modules power relay, which disconnects the load.

SYNTAX: OPN n

Opens the relay on channel n.

4.6.5 RESET Command

RST - Resets a power module's voltage and current to zero and opens its power relay.

SYNTAX: RST n Resets channel n.
RST ALL - Resets all active channels.
Functionally equivalent to the REST utility command.

4.6.6 CONFIDENCE TEST Command

CNF - Perform a confidence test on all active modules. All power relays are opened and each module's voltage and current are programmed to their maximum values and then reset to zero.

4.6.7 INTERNAL SELF TEST Command

IST - Performs an internal self test on all modules. The command is Functionally equivalent to the CNF command.

4.6.8 STATUS Command

STA - Returns the current status of the power modules.

4.7 UTILITY COMMAND DESCRIPTIONS

4.7.1 REMOTE ENABLE Command

REN - Remote enable - Puts all power modules in the remote enabled mode, which allows response to subsequent commands.

4.7.2 GO TO LOCAL Command

GTL - Go To Local - This command disables remote operation and causes the TMA PC-27 to ignore all subsequent commands except for the REN command.

4.7.3 DEVICE CLEAR Command

DCL - Device clear - This command causes all power modules to be reset to their power on state, and will initialize any modules which have been added to the control bus since the last initialization. The command takes about 3 seconds to execute.

4.7.4 SYSTEM RESET Command

REST - System Reset - This command is similar to "DCL", but does not poll all 27 channels to detect if new modules have been added. Thus, execution time is much faster than the "DCL" command.

4.7.5 VERSION Command

VER - Returns the version number of the software.

4.7.6 ERROR HANDLING Commands

S0 Causes no signal to be generated if a catastrophic error condition is present.

S1 Causes the message "ERROR CONDITION PRESENT !" to be printed on the screen, following any command to the TMA PC-27, if a catastrophic error condition is present.

S2 Causes an audible beep to be generated, following any command to the TMA PC-27, if a catastrophic error condition is present.

S3 Causes both a message and a beep if a catastrophic error condition is present.

T0	Instructs non-catastrophic error messages to be erased from memory if any command is sent prior to an STA command.
T1	Instructs non-catastrophic error messages to be stacked in memory until an STA command is sent.
F0	Fetch mode 0. Ignores error conditions when performing a FTH command.
F1	Fetch mode 1. Reports any error conditions which are present during a FTH command.
P0	Power loss mode 0. Reports a power loss message only once until power is restored to the power module.
P1	Power loss mode 1. Continuously reports a power loss message until power is restored to the power module.

NOTE: The defaults are S0, T0, F1 and P1.

4.8 ERROR MESSAGES

Certain conditions will cause an error message to be generated. These messages are of two(2) types, catastrophic and non-catastrophic. Catastrophic errors are those in which the condition may cause damage if not corrected, non-catastrophic errors are less serious. All error messages are obtained by sending an "STA" command. The message is returned in the same variable that was used to send "STA". Catastrophic error messages will remain in memory until an "STA" command is sent, and the error condition is corrected, while non-catastrophic error messages will be erased as soon as any command other than "STA" is sent to the TMA PC-27.

As required by CIIL, all error messages begin with an ASCII "F" (FAULT) followed by a 2 digit code, "07" (HALT). The code that follows (DCSnn) indicates the type of device and the channel number. The next 3 digit code describes the nature of the fault, "DEV" for device related errors or "MOD" for non-device errors, such as syntax.

The following messages have the prefix "F07 DCSnn DEV:":

CATASTROPHIC ERRORS

POWER LOSS	The module has lost its power.
CROWBARRED	A shut-down occurred due to overvoltage or overcurrent.
OVER TEMPERATURE	A shut-down occurred due to thermal causes.
OVERLOAD	The voltage or current limit point was exceeded.
VOLTAGE COMPARISON ERROR	The output voltage is not within limits (voltage mode).
CURRENT COMPARISON ERROR	The output current is not within limits (current mode).
RELAY NOT OPENED	The power relay failed to open.
RELAY NOT CLOSED	The power relay failed to close.
POLARITY ERROR	The output polarity is not correct.

NON-CATASTROPHIC ERRORS

NOT READY The output voltage or current has not settled.

DEVICE NOT PRESENT The specified module was not present during power-up or during the last DCL.

DEVICE NOT RESPONDING The module has failed to communicate to the TMA PC-27.

INVALID VOLTAGE RANGE The programmed voltage is outside the module's range.

INVALID CURRENT RANGE The programmed current is outside the module's range.

SET MODIFIER ERROR An improper SET command was sent.

INVALID DEVICE ID The selected channel was not between 1-31.

The following non-catastrophic message has the prefix "F07 DCSnn MOD:"

INVALID COMMAND Improper syntax was used.

SECTION 5
KPL PROGRAM MANAGER

5.1 GENERAL

The KPL Program Manager provides a working environment for running, editing, or debugging KPL programs. It may be called up from DOS by typing KP or KP filename. If filename is specified, the KPL file with that name will be loaded into the KPL Program Manager, otherwise no file will be loaded. If you wish only to run a KPL program then type KP ^filename, and the specified KPL program will be run, after which you will be returned to DOS.

One of the main objectives of KPL is to facilitate input/ output operations to a variety of different devices, whether they be IEEE-488, Kepco's TMA PC-27, data files, printer or screen. In a test environment all of these may, and probably will, be used. KPL makes it easy to switch between any of these devices without having to write lots of programming code. Set up routines and data acquisition routines may be created for particular instruments, and big projects may be broken up into smaller modular components without spending lots of time creating complex code.

Routines are easily put together or rearranged to suit the need. Creating a new program doesn't mean starting from scratch each time or spending hours trying to follow someone else's code in order to modify it, then more hours debugging it. The KPL program manager allows you to trace through a program's execution one step at a time and allows program data to be examined at each step.

Once the KPL Program Manager has been loaded, a menu will appear. You may now create, run, edit, or debug a program, or perform various other tasks involving input/output operations. Any KPL command may be issued directly from the KPL Program Manager environment.

5.2 USING THE MENU

The following tasks can be performed by selecting the appropriate command from the menu.

A)	Selects the active input/output device.
B)	Begins execution of the currently loaded program.
C)	Restores a program file that had been cleared.
D)	Issues a DEVICE CLEAR to all devices.
H)	Defines keys
I)	Puts you into program entry mode.
L)	Lists the currently loaded program.
M)	Toggles the menu on and off.
P)	Sends a command to the active device.
Q)	Quits the KPL Program Manager and returns to DOS.
R)	Inputs data from the active device.
S)	Allows execution of DOS commands.
T)	Prints the current time and date.
V)	Prints information about the most recent file loaded or saved.
b)	Continues a program that was interrupted.
c)	Clears all variables, or clears a program file or both.
e)	Allows editing of the current program file.
i)	Allows insertion of additional commands into the current program file.
l)	Lists the program starting from a particular line.
p)	Allows execution of a KPL command.
s)	Redefines the beginning and end of a program.
t)	Allows single stepping of a program.

#) Loads a KPL program, or merges it to the current one.
$) Saves the current program file.
*) Toggles the printer on and off for printer commands.
>) Executes an existing subroutine.
^) Loads and executes a program from disk, then reloads the current program.
?) Display variables

5.3 USING 'HOT KEYS'

KPL allows you to program up to 20 keys, which will allow you to perform just about any task by pressing a single key. Once a key is programmed, any time it is pressed, even while a program is being executed, it will execute the command that was programmed into it. The command may be any KPL command, which includes loading and running a file, executing a subroutine, etc. The best way to implement hot keys is to create a KPL program or subroutine to define them; when the application for which they were called is done, then reset or redefine them. You must be careful that a program you are running will not need, for some other purpose, the keys you have assigned as 'hot keys'. See the command section for information on implementation of 'hot keys'.

5.4 INIT.KPL FILE

Whenever KP is called up, the file INIT.KPL is loaded and run. This is any KPL program which you have created and is made up of KPL commands. The INIT file may contain setup information which you may always want executed, such as defining 'hot keys' , or it may even call up another KPL program, in which case all you need to do to execute your application from DOS is to type KP. The program called may be, for example, a control program that allows you to select from among other KPL programs. The instruction for calling up another KPL program from within a KPL program is ^filename. Thus, putting this instruction in your INIT.KPL file will automatically load and run any KPL program. Another method would be to type KP filename, which would load and run the INIT.KPL file, then load and run the KPL program specified by filename.

5.5 INITIATING THE IEEE-488 GPIB AND TMA PC-27

In order to use the GPIB it must first be initialized.

The TMA PC-27 may also be instructed to respond when certain IEEE-488 commands are issued. See the section on GPIB commands for details.

5.6 CREATING PROGRAMS AND SUBROUTINES WITH KPL

KPL was created to allow easy access to program routines that have been previously written and tested. Once a routine is created the user need not be concerned about its contents, only about what the routine does. The more programs you write with KPL the more time you may save, if the programs will perform similar tasks. Creating a routine is quite simple, the following guidelines may be followed:

A) Choose a name for the routine. The more descriptive the name is, the easier it will be for someone else to follow the program. The first line of the routine must contain a colon (:) immediately followed by the routine's name. The routine may have parameters passed to it, these are defined in parentheses following the name, e.g. **:sub(%a,%b)**. No other instructions are allowed on this line. The name is used by KPL to locate the routine during program execution and is only read when the file is loaded or edited. NOTE: Variables in KPL are notated by the % symbol.

B) Enter the programming commands needed. Any line of the program that needs to be jumped to from another part of the program may receive a label. Labels are globally known to the whole program and should, therefore, be chosen carefully. Routine names and labels are functionally the same. A given label or name should be used in only one place for any given program. General terms such as **:NEXT** should be avoided and might better be replaced with ones such as **:NEXT_READ** or **:NEXT_OUT**.

C) Terminate the routine with a return instruction. The symbol for this is **<**. Certain routines may have to return to a different place from where they were called, so a label may follow such as **<TOP**.

D) The main body of a KPL program may or may not begin with a label but if subroutines follow it, then it must terminate with the symbol < . This will terminate the program unless it was called from another program, in which case the calling program will be reloaded and will continue from where it left off. It is good practice to terminate all programs with a **<.** The symbol << terminates a program, even if it is in the middle of a subroutine. Control will return to the calling program if there was one otherwise the program will terminate.

E) A KPL program may have up to 500 lines. However, any KPL program may call another one while retaining all data in memory, so that virtual longer programs are possible. In fact, the called program may also call a program and so on up to 5 programs deep. Each program will return to the program that called it.

5.7 KPL EDITOR

The editor allows you to make changes in your program. It is invoked by striking **e**. You may start at the top of the program by striking the **ENTER** key or you may go directly to any line by entering the line number. You may also go to any label by entering a : followed by the label name. The editor commands are defined at the top of the screen. You may add lines, delete lines or alter lines. To add new lines to the end of the program use the **i** command from the KPL Program Manager menu and press enter for the line number. Use **I** from the KPL Program Manager menu to start entering a new program.

5.8 KPL COMMANDS AND FEATURES

The following section describes the KPL commands and how to use them to create KPL programs. More than one command may be put on a line by separating the commands by the | symbol. Comments may be inserted by using the // symbol.

5.8.1 Calling Subroutines and Programs

COMMAND	DESCRIPTION
^ filename	Load and run a program, then return to the current program.
>>	Enter and execute a KPL command.
>subname (a,b)	Execute the named subroutine, passing parameters **a** and **b**. Any number of parameters may be passed. Variables or direct values may be used. The number of parameters should agree with the number defined in the subroutine. Each value will be assigned to the corresponding variable defined in the subroutine.
<	Terminate a routine and return to where it was called from.
<label	Terminate a routine and return to the location of the given label.
<Mn	Terminate a routine and return to location of the last **MJn** command.
!command	Shell to DOS and execute a DOS command or run a file from DOS.
<<	Terminate program.

5.8.2 Defining Variables

Variables in KPL are specified by use of the **%** symbol. Data typing is done automatically, depending on the operation being done. Variables may be operated on mathematically or may be treated as character strings, depending on the KPL instruction used. The total number of digits used before and after the decimal is fixed by the **N** command. The default value is **N0**, which specifies that all math operations will return only integer values. If a non-integer value is desired, use the **N** command to specify how many digits you need. Arrays may be created by attaching a variable suffix to a variable name. For instance, if **%ARY** is defined as **MEASURED** and **%INDEX** is defined as **3** then the array element **MEASURED.3** may be represented by the symbol **%ARY.%INDEX**. The following % commands may be used.

COMMAND	DESCRIPTION
%A=n	Create a variable.
%A+n	Add **n** to the value of **%A**.
%A-n	Subtract **n** from the value of **%A**.
%A*n	Multiply **%A** by **n**.
%A/n	Divide **%A** by **n**.
%A&%B	Append **%B** to **%A**.
%A@n	Redefine **%A** starting with the nth character.
%A@-n	Truncate **n** characters from **%A**.
%A.%B=n	Define an array element whose name is defined by **%A** and element # by **%B**.
%A.=n	Assign value **n** to the variable whose name is defined by **%A**

Note: **n** may also be a variable

5.8.3 Built In Variables

%rc_str	Contains the last data string received from the active device.
%xpos	The current **x** coordinate of the cursor.
%ypos	The current **y** coordinate of the cursor.

5.8.4 Comparing Data Strings

One of the more frequent tasks a program must perform is to compare a response or a data value to a known or expected one. This is done in KPL by the use of the * symbol. If the items agree with each other the next program line is skipped, otherwise it is executed. The opposite will be true if ***!** is used. By making the command on the next line a jump instruction, the program can then decide between two paths.

COMMAND	DESCRIPTION
*n	Jump if received data matches **n**.
*!n	Jump if received data doesn't match **n**.
*%A=n	Jump if **%A** matches **n**.
*%A>n	Jump if **%A** is greater than **n**.
*%A<n	Jump if **%A** is less than **n**.
*Dn, *DMn	Jump if the value of received data equals **n** or the value in memory **n** within the limits set by the **S** command.
*In	Jump if counter value is greater than or equal to **n**.
*Kn1n2n3...	Jump if the key pressed matches any of the listed keys **n1**, **n2**, **n3** etc.
*K/n	Jump if the ASCII code of the key pressed is n (e.g. \32 is a space)
*Ln	Jump if the length of received data equals **n**.
*Rn	Jump if bus status equals **n**

Note: **n** may also be a variable. The ! symbol may be used for any of the cases to reverse the result.

5.8.5 GPIB Commands

COMMAND	DESCRIPTION
/I	Initiate the GPIB.
/K	Instruct TMA PC-27 to respond to the commands **/C**, **/D**, **/R**, and **/L**.
/B	Check bus status.
/C	Selected Device Clear (TMA command DCL).
/D	All Devices clear (TMA command DCL).
/E0	Set end of string to none.
/E1	Set end of string to cr (ASCII 13).
/E2	Set end of string to lf (ASCII 10).
/E3	Set end of string to cr lf (ASCII 1310).
/F	Send Interface Clear.
/L	Set Remote line false (local mode) (TMA command GTL).
/P	Conduct a Serial Poll.
/R	Set Remote line true (remote mode).(TMA command REN).
/Tn	Set the time-out value to **n**.

5.8.6 Selecting An Active Device

The active device will determine where to direct all input/output operations. It may be the TMA PC-27, a GPIB instrument, a data file, the printer or the screen. Thus any routine's output may be directed to a device such as printer or screen in order to test the routine, then redirected elsewhere by changing the active device.

COMMAND	DESCRIPTION
A	Selects the screen (output only).
ATMA or @TMA	Selects the TMA PC-27, this is the default device if no other device is selected.
ADEVn or @DEVn	Selects an IEEE-488 device defined by IBCONF (National Instruments GPIB software).
A%dev or @%dev	Selects the device defined by a variable, in this case, **%dev**.
AP or @P	Selects the printer (output only).
AF filename	Opens the file named by filename and closes any previously opened file.

NOTE: files are written to or read from sequentially. The first operation done deterrnines the read/write mode and may only be changed by another AF command.

5.8.7 I/O Commands

COMMAND	DESCRIPTION
'data string	Sends a data string to the active device. The string may be made up partially or entirely of variables.
R	Receives a data string from the active device.
R%var	Receives data and assigns it to a variable, **%var** in this case.
RP	Receives data and prints it to the screen.
RM,RMn	Receives data and stores it in memory n or memory 0 if n is not specified.
R*	Receives data and does a compare. See Comparing Data Strings (refer to paragraph 5.8.4).

5.8.8 Creating An Audible Beep

COMMAND	DESCRIPTION
Bn	Causes a BEEP of **n** counts.

5.8.9 Creating A Time Delay

Time delays may be created anywhere in a program using the **D** command as follows:

COMMAND	DESCRIPTION
Dn	Causes a time delay of **n** seconds.
Dmn	Causes a time delay of **n** milliseconds.

Note: **n** may be a variable , e.g. **D%T**

5.8.10 Getting Input From The Keyboard

Variables may be input from the keyboard as follows:

COMMAND	DESCRIPTION
G %A message	Print message prompt and wait for input.
G message %A	Same as above.
G %A %B	Print variable **%B** and wait for input of **%A**.
G %A.%B %A%.%B	Print variables **%A** and **%B** separated by a and wait for input of array element **%A.%B**.
W message	Display a message (optional) and wait for a key to be pressed.

5.8.11 Defining 'Hot Keys'

COMMAND	DESCRIPTION
HnCOMMAND	Program key **n** to execute the KPL command indicated.
Hn	Unprogram key **n**. see also Clearing Hot Keys (refer to paragraph 5.8.14)

5.8.12 Using The Built In Counter

There is a built-in counter, that may be used for controlling a program process as follows:

COMMAND	DESCRIPTION
I	Increment the counter by one.
IC	Clear the counter.
IMn	Retrieve a counter value from memory **n**.see also MEMORY COMMANDS (refer to paragraph 5.8.15)

5.8.13 Jumping To Another Part Of A Program

COMMAND	DESCRIPTION
J:LABEL	Jump to the specified label.
J%L	Jump to a label named by a variable, %L.
JN	Jump to the next line.
Jn	Jump to line n.
JMn	Jump to location of MJn command. See also MEMORY COMMANDS

5.8.14 Clearing The Keyboard Buffer, 'Hot Keys', And The Menu

COMMAND	DESCRIPTION
K	Clears keyboard buffer.
KH	Clears all 'hot keys'.
KM	Turns off the KPL Program Manager menu.

5.8.15 Memory Commands

There are 10 memory locations that can be used for storing certain information temporarily, for later retrieval. When a program needs to run as fast as possible, using one of these memory locations is better than using a variable. Variables must be searched by location, which takes more time.

COMMAND	DESCRIPTION
MIn	Store counter value.
MJn	Store current program line for jumping back to by the command.
MKn	Store currently pressed key.
MRn	Store currently received data.
MTn	Store current time.
M%var	Transfer contents of memory 0 to a variable, in this case **%var**.
M=%var	Store a variable in memory 0. Note: Valid values for **n** are 0-9. If **n** is left out location 0 is used.

5.8.16 Setting Precision

This command sets the number of digits to use when performing math operations on variables. The variable is padded with zeros if less digits are generated. This is also useful in formatting data for printout. After executing an **N** instruction, any variable may be reformatted by multiplying it by 1 (i.e. **%a*1**).

COMMAND	DESCRIPTION
Nx	Set the minimum # of digits to be used when math is performed on a variable (x=0 for integers).

5.8.17 Printing To The Screen

COMMAND	DESCRIPTION
P text	Print text to the screen. The text may contain variables.
P! text	Clear the screen and print text (the text is optional).
P; text	Print text and remain at the last screen position.
P\n	Print an ASCII character, **n** is the ASCII code for the character.
PB	Print the GPIB status after a **/B** or **/P** command.
PC x,y text	Go to screen location x,y, clear the line and print text at that location. x is the column (1-25) and y is the row (1-80).
PF n%A n%B	Print formatted text, using n characters for each variable indicated and padding with spaces.
PI	Print the current value of the counter.
PL x,y text	Print text at screen location x,y.
PMn	Print contents of memory **n**(0-9).
PR	Print last data received by **R** command.
PT	Print the current time.
Lx,y	Place cursor at location x,y.

The **%** symbol may be used at the end of a variable if no space is desired e.g. **P %A%%B** will print the variables **%A** and **%B** with no space between. If spaces follow the text, the **|** may be used e.g. **P; Enter value |**. Another command does not necessarily have to follow a **|**.

5.8.18 Printing To The Printer

The printer commands are the same as the screen commands. The **P** is replaced by the **?** symbol.

COMMAND	DESCRIPTION
? text	Print text.
?!	Print a form feed, go to next page.
?; text	Print text but no line feed.
?B	Print GPIB status.
?F n%A n%B	Print formatted text.
?I	Print the current value of the counter.
?Mn	Print contents of memory **n**.
?R	Print last data received by **R** command.
?T	Print current time.

5.8.19 Setting Tolerance

COMMAND	DESCRIPTION
Sn	Set the tolerance range for the ***D command. n** may be a variable.

5.8.20 Video Display Commands

The following commands may be used to alter the video display screen characteristics.

COMMAND	DESCRIPTION
VBn	Set background color.
VTn	Set text color.
VWx1,y1,x2,y2	Set display window limits. Full screen is 1,1,80,25. x1,y1 is the upper left corner, x2,y2 is the lower right corner.
VCn	Select cursor type n (0=no cursor, 1=normal cursor, 2=solid cursor)

APPENDIX A
QUICK CHECKOUT

A.1 Getting Started

A control menu program has been provided on the program disk, in order to speed up the time required to check out your set-up. The program is written in KPL and uses the KPSL driver for the TMa PC-27.

Follow these steps to get started:

1) Install your TMa PC-27 following the instructions in your manual.
2) Boot your PC with DOS and press the caps lock key.
3) Insert your program disk into drive A or B and make that the default drive. i.e. type A: (for drive A) or type B: (for drive B).
4) Connect the MATs to the control bus and turn their power on.
5) At the DOS prompt type "demo" and press the enter key. The KPSL driver will be loaded into memory and a menu will appear on the screen.
6) Selections may be made by using the up/down arrow keys or by pressing the appropriate letter for each item. The item letter will be highlighted and you may then press the enter key to select that item.
7) The left/right arrow keys will change the channel # from any position on the menu and the data will then reflect the new channel. This will allow you to quickly program all channels.

The control keys function as follows:

[N] Allows manual selection of any channel

[V] Allows entry of a new voltage value

[C] Allows entry of a new current value

[M] Toggles the selected channel between voltage and current mode operation.

[R] Toggles the state of the output relay

[P] Allows manual entry of any KPSL command. Refer to the TMa PC-27 manual for a complete listing of all commands

[X] Toggles between auto and manual execute mode. In auto mode, voltage and current settings are programmed as soon as they are entered. In manual mode, voltage and current values for each channel may be set up first then executed by pressing the F1 key.

[T] Toggles between continuous reading or manual reading of the output voltage and current. In manual mode the F2 key is used to read the outputs.

[S] Toggles between continuous reading and manual reading of the system status. In manual mode the F3 key is used to check status.

[E] Allows automatic resetting of all channels, when hardware error conditions such as an overload are reported during a status check. The F4 key allows manual resetting of all channels at any time.

APPENDIX B
PROGRAMMING

B.1 KPL Programming Examples

B.1.1 Init.KPL File To Initialize IEEE-488 And TMA PC-27

```
/I                                  // Initialize IEEE
/F                                  // Interface Clear
/K                                  // Initialize TMA PC-27
/R                                  // Remote Enable
<<                                  // End Program
```

B.1.2 Read An Array From A Data File

```
:READ_ARRAY
AF MYFILE                           // OPEN A FILE NAMED MYFILE
G enter name of array %NAME         // GET THE ARRAY NAME
%X=1¦NO                             // SET INDEX , USE INTEGERS
R %MAX_DATA                         // READ ARRAY LENGTH
:NEXT_READ                          // MARK THIS SPOT
R %NAME.%X                          // READ DATA ELEMENT
%X+1                                // INCREMENT INDEX
*%X>%MAX_DATA                       // JUMP OUT IF X > MAX_DATA
J:NEXT_READ                         // OTHERWISE JUMP BACK
AF                                  // CLOSE FILE
<                                   // END OF ROUTINE
```

B.1.3 Write To A File

```
:WRITE_FILE
AF TESTDATA                          //  OPEN FILE NAMED TESTDATA
%X=1¦NO                              //  SET INDEX, USE INTEGERS
'%MAX_DATA                           //  WRITE FILE LENGTH
:NEXT_WRITE                          //  MARK THIS SPOT
'%NAME.%X                            //  WRITE DATA
%X+1                                 //  INCREMENT INDEX
*%X>%MAX_DATA                        //  JUMP OUT IF X > MAX_DATA
J:NEXT_WRITE                         //  OTHERWISE JUMP BACK
AF                                   //  CLOSE FILE
<                                    //  END
```

B.1.4 Set Up a Menu

```
:TEST_MENU
P:      TEST MENU                    //  PRINT THE MENU ON SCREEN
P
P   1) SEND  2) READ  3) PRINT
P
W   ENTER YOUR CHOICE                //  WAIT FOR KEY TO BE PRESSED
*!K1¦>SEND                           //  EXECUTE SUBROUTINE SEND
*!K2¦>READ                           //  EXECUTE SUBROUTINE READ
*!K3¦^PRINT                          //  EXECUTE PROGRAM PRINT.KPL
J:TEST_MENU
:SEND
P       SEND ROUTINE
<
:READ
P       READ ROUTINE
<
```

B1.5 Passing Parameters

```
:MAIN
>READ_DATA                          //  EAD IN THE PROGRAM DATA
%CH=1                               //  START WITH FIRST CHANNEL
:LOOP
>SETV(%CH,%VMAX.%CH,%IMAX.%CH)      //  SEND DATA
%CH+1                               //  INCREMENT CHANNEL NUMBER
*%CH>%LAST_CH                       //  SEE IF IT'S THE LAST CHANNEL
J:LOOP                              //  JUMP BACK
<
:SETV(%N,%V,%I)                     //  DATA IS PLACED INTO %N,%V,%I
ATMA
'FNC DCS :CH%N SET VOLT %V SET CURL %I
<                                   //  END SETV ROUTINE
:READ_DATA                          //  READ DATA ROUTINE
AF DATA
R %CH_MAX
%CH=1
NEXT_DATA
R %VMAX.%CH
R %IMAX.%CH
%CH+1
*%CH>%CH_MAX
J:NEXT_DATA
<                                   //  END READ DATA ROUTINE
```

B.2 "C" Language Program Example

```
#include <stdio.h>
#include <string.h>
#include <dos.h>

char far *tma(char far *io_str)
{
     union REGS inregs, outregs;
      struct SREGS segregs;
      inregs.x.ax = 0;                          /* USE 0 FOR C LANGUAGE    */
      inregs.x.bx = FP_OFF(io_str);             /* STRING OFFSET           */
      segregs.es  = FP_SEG(io_str);             /* STRING SEGMENT          */
      int86x(0x60, &inregs, &outregs, &segregs); /* CALL INTERRUPT 60 HEX  */
      if(outregs.x.ax)                          /* CHECK STATUS BIT FOR 0 */
        printf("STATUS ERROR \n\a");
      return(io_str);                           /* RETURN DATA             */
}

void main(void)
{
      int i;
      char in_str[128];
      do{
            printf("enter command - ");
            gets(in_str);                       /* ENTER DATA              */
            i=strlen(in_str);                   /* CHECK LENGTH            */
            if(i)                               /* SEND OUT DATA AND       */
                  printf("%s", tma(in_str) );   /* PRINT RETURNED DATA     */
      }while(i);                                /* EXIT IF NO MORE DATA    */
}
```

B.3 "Pascal" Language Program Example

```
uses crt, dos;

var
   io_str : string;
   i : word;

function tma (var in_str : string): string;

var  r : registers;

     begin
             r.AX:= 3;                                { USE 3 FOR PASCAL     }
             r.BX:= ofs (in_str);                     { DATA STRING OFFSET   }
             r.ES:= seg (in_str);                     { DATA STRING SEGMENT  }
             intr($60, r);                            { CALL INTERRUPT 60    }
             if (r.AX>0) then                         { CHECK STATUS FLAG    }
                 writeln('status error', chr(7));
             tma := in_str;                           { RETURN DATA          }
     end; { tma }

begin
     i := 1;
     while (i>0) do
          begin
             write('enter command > ');
             readln(io_str);                          { ENTER DATA           }
             i := length(io_str);                     { CHECK LENGTH         }
             if (i>0) then                            { SEND DATA AND        }
                 write(tma(io_str));                  { PRINT RETURNED DATA  }
          end; { while }                              { EXIT IF NO DATA      }
end.
```

B.4 "Quick Basic" Language Program Example

```
NOTE: Start QB with the /l option.

DECLARE FUNCTION TMA$ (IN$)

DO

    INPUT "Enter Command > ", cmd$                          'ENTER DATA
    i = LEN(cmd$)                                           'CHECK LENGTH
    IF (i) THEN PRINT RTRIM$(TMA$(cmd$))                    'SEND DATA AND
                                                            'PRINT RETURNED DATA
LOOP WHILE (i)                                              'EXIT IF NO DATA

FUNCTION TMA$ (IN$)

    DIM INARY%(9), OUTARY%(9)                               ' REGISTER ARRAYS
    DIM IOSTR AS STRING * 128                               ' I/O BUFFER
    CONST ax = 0, bx = 1, es = 9                            ' DEFINE REGISTERS
    IOSTR = IN$                                             ' LOAD DATA
    INARY%(ax) = 4                                          ' USE 4 FOR BASIC
    INARY%(bx) = VARPTR(IOSTR)                              ' DATA STRING OFFSET
    INARY%(es) = VARSEG(IOSTR)                              ' DATA STRING SEGMENT
    CALL INT86XOLD(&H60, INARY%(), OUTARY%())               ' CALL INTERRUPT 60
    IF OUTARY%(ax) THEN PRINT CHR$(7); "STATUS ERROR"       ' CHECK STATUS FLAG
    TMA$ = IOSTR                                            ' RETURN DATA

END FUNCTION
```

B.5 "Interpreted Basic" Language Program Example

NOTE: File TMABAS.INT must be loaded in the same directory as the INTERPRETED BASIC sample program (IBSAMP.BAS), listed below, for the program to function. Both files are contained in the Kepco supplied disk.

```
10 CLEAR, 64511!                          'MAKE ROOM AT TOP OF MEMORY
20 TMA=64512!                             'ASSIGN ADDRESS
30 BLOAD"TMABAS.INT",TMA                  'LOAD INTERFACE MODULE
40 POKE(64560!),VAL("&H60")               'SET INTERRUPT VECTOR TO 60 (HEX)
50 INPUT"Enter Command >  ",D$            'ENTER DATA
60 L%=LEN(D$):IF L%=0 THEN 130            'CHECK DATA LENGTH
70 T=127-L%:T$=SPACE$(T)                  'ADD SPACES
80 D$=D$+T$+""                            'MAKE LENGTH = 128
90 CALL TMA(S%,D$,L%)                     'MAKE CALL
100 IF S% THEN PRINT "COMMAND ERROR"      'CHECK STATUS FLAG
110 PRINT D$                              'PRINT RETURN DATA
120 GOTO 50                               'GET MORE DATA
130 END                                   'DONE
```

B.6 "TopSpeed Modula - 2" Language Program Example

```
(* THIS PROGRAM IS COMPILER-SPECIFIC TO "TopSpeed Modula-2",
     AND HAS BEEN TESTED AS FUNCTIONAL WITH V2 r1.04. *)


MODULE modsamp;
  FROM IO IMPORT RdStr, WrLn, WrChar, WrStr;
  FROM Str IMPORT Length;
  FROM SYSTEM IMPORT Ofs, Seg, Registers;
  FROM Lib IMPORT Intr;

  VAR
  result: CARDINAL;
  io_string: ARRAY [0..128] OF CHAR;

  PROCEDURE tma(VAR Inp: ARRAY OF CHAR);
    VAR r: Registers;
  BEGIN
    r.AX := 0;                          (* USE 0 FOR MODULA-2  *)
    r.BX := Ofs(Inp);                   (* STRING OFFSET       *)
    r.ES := Seg(Inp);                   (* STRING SEGMENT      *)
    Intr(r, 60H);                       (* CALL INTERRUPT 60   *)
    IF (r.AX > 0) THEN                  (* CHECK STATUS FLAG   *)
      WrStr('STATUS ERROR');
      WrChar(7C);
      WrLn;
    END; (* IF *)
  END tma;

BEGIN
  result := 1;
  WHILE (result > 0) DO
    WrStr('Enter Command > ');
    RdStr(io_string);                   (* ENTER DATA          *)
    result := Length(io_string);        (* CHECK LENGTH        *)
      IF (result > 0) THEN
        tma(io_string);                 (* SEND OUT DATA AND   *)
        WrStr(io_string);               (* PRINT RETURNED DATA *)
      END; (* IF *)
  END; (* WHILE *)                      (* EXIT IF NO DATA     *)
END modsamp.
```

B.7 "TopSpeed Pascal" Language Program Example

```
                    (* Sample Program Using "TopSpeed Pascal" *)

PROGRAM TSPSAMP (input, output);

Import PasDos *;

Var
   io_str : MAXSTRING;
   status : Word;
   i : Word;

Function TMA(io_str : MAXSTRING): MAXSTRING;

Var  Regs : Registers;

     Begin
     With Regs Do
           Begin
                AX:= 3;                                    { USE 3 FOR PASCAL     }
                BX:= Ofs (io_str);                         { DATA STRING OFFSET   }
                ES:= Seg (io_str);                         { DATA STRING SEGMENT  }
                Intr(60H, Regs);                           { CALL INTERRUPT 60    }
                If(AX > 0) Then                            { CHECK STATUS FLAG    }
                   WriteLn('STATUS ERROR', CHR(7));
                TMA := io_str                              { RETURN DATA          }
           End { With Regs Do }
     End; { TMA }

Begin
     i := 1;
     While (i>0) DO
            Begin
                 Write( 'Enter Command > ' );
                 ReadLn(io_str);                            { ENTER DATA           }
                 i := Length(io_str);                       { CHECK LENGTH         }
                 If (i > 0) Then                            { SEND DATA AND        }
                         Write( TMA(io_str) );              { PRINT RETURNED DATA  }

               End { While }                                { EXIT IF NO DATA      }
End.
```