IMPORTANT NOTES:

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

<table>
<thead>
<tr>
<th>MODEL</th>
<th>SERIAL NO.</th>
<th>REV. NO.</th>
</tr>
</thead>
</table>

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.
SECTION 1 - INTRODUCTION
1.1 Scope of Manual .................................................................................................................. 1-1
1.2 General Description ............................................................................................................ 1-1
1.3 Specifications, BIT 232/BIT 232-F .................................................................................... 1-1
1.4 Accessories ....................................................................................................................... 1-1

SECTION 2 - INSTALLATION
2.1 Unpacking and Inspection ................................................................................................ 2-1
2.2 Set Start-up Defaults ............................................................................................................ 2-1
2.2.1 Start-up Language Default ............................................................................................ 2-1
2.2.2 Set Power Supply Identification Switch ......................................................................... 2-1
2.3 Installation of BIT Card into the BOP ................................................................................ 2-1
2.4 Input/Output Signals ........................................................................................................... 2-5
2.5 RS 232 Connections .......................................................................................................... 2-5

SECTION 3 - CALIBRATION
3.1 Equipment Required .......................................................................................................... 3-1
3.2 Adjustment of the Bop ±10 Volt Calibration Controls (R31, R32) .................................... 3-1
3.3 Adjustment of the Ammeter Zero (R50) ............................................................................. 3-1
3.4 Adjustment Of The Output Voltage Zero (R81) ............................................................... 3-2
3.5 Adjustment of the Full Scale Output Voltage (R21) ......................................................... 3-2
3.6 Voltage Reading Zero Calibration (R35) ........................................................................... 3-3
3.7 Voltage Reading Calibration (R19) .................................................................................... 3-3
3.8 Adjustment of the Output Current Zero (R83) ................................................................. 3-4
3.9 Adjustment of the Full Scale Output Current (R22) .......................................................... 3-4
3.10 Current Reading Zero Calibration (R36) ............................................................................ 3-4
3.11 Current Reading Calibration (R20) .................................................................................. 3-5

SECTION 4 - OPERATION
4.1 General ............................................................................................................................... 4-1
4.2 RS232-C Bus Protocol ...................................................................................................... 4-1
4.3 RS232 Implementation ...................................................................................................... 4-1
4.4 SCPI Programming .......................................................................................................... 4-2
4.4.1 SCPI Messages ............................................................................................................. 4-2
4.4.2 Common Commands/Queries ..................................................................................... 4-2
4.4.3 SCPI Subsystem Command/Query Structure ............................................................. 4-2
4.4.4 Program Message Structure ....................................................................................... 4-3
4.4.4.1 Keyword .................................................................................................................. 4-3
4.4.4.2 Keyword Separator ............................................................................................... 4-5
4.4.4.3 Query Indicator ....................................................................................................... 4-5
4.4.4.4 Data ....................................................................................................................... 4-6
4.4.4.5 Data Separator ....................................................................................................... 4-6
4.4.4.6 Message Unit Separator ......................................................................................... 4-6
4.4.4.7 Root Specifier ...................................................................................................... 4-6
4.4.4.8 Message Terminator ............................................................................................. 4-6
4.4.5 Understanding The Command Structure ................................................................... 4-6
4.4.6 Program Message Syntax Summary .......................................................................... 4-7
4.5 CIIL Programming ............................................................................................................ 4-8
4.6 Programming Examples ................................................................................................... 4-8
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>SECTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPENDIX A - SCPI COMMON COMMAND/QUERY DEFINITIONS</td>
<td></td>
</tr>
<tr>
<td>A.1 Introduction</td>
<td>A-1</td>
</tr>
<tr>
<td>APPENDIX B - SCPI COMMAND/QUERY DEFINITIONS</td>
<td></td>
</tr>
<tr>
<td>B.1 Introduction</td>
<td>B-1</td>
</tr>
<tr>
<td>APPENDIX C - CIIL COMMAND DEFINITIONS</td>
<td></td>
</tr>
<tr>
<td>C.1 Introduction</td>
<td>C-1</td>
</tr>
<tr>
<td>APPENDIX D - TERMINAL EMULATION PROGRAM</td>
<td></td>
</tr>
<tr>
<td>D.1 Introduction</td>
<td>D-1</td>
</tr>
<tr>
<td>APPENDIX E - RS232 COMMAND LOOP PROGRAM</td>
<td></td>
</tr>
<tr>
<td>E.1 Introduction</td>
<td>E-1</td>
</tr>
<tr>
<td>APPENDIX F - C LANGUAGE FUNCTIONS</td>
<td></td>
</tr>
<tr>
<td>F.1 Introduction</td>
<td>F-1</td>
</tr>
<tr>
<td>F.2 Open Serial Port Using ROM BIOS Routine</td>
<td>F-1</td>
</tr>
<tr>
<td>F.3 Put Character to Serial Port</td>
<td>F-1</td>
</tr>
<tr>
<td>F.4 Send, Wait for Echo</td>
<td>F-2</td>
</tr>
<tr>
<td>F.5 Get Character from Serial Port</td>
<td>F-2</td>
</tr>
<tr>
<td>F.6 Check to See If a Character Was Received</td>
<td>F-3</td>
</tr>
<tr>
<td>F.7 Send Command</td>
<td>F-3</td>
</tr>
<tr>
<td>F.8 Send Command and Wait Until Ready</td>
<td>F-4</td>
</tr>
<tr>
<td>APPENDIX G - BASIC LANGUAGE TERMINAL EMULATION</td>
<td></td>
</tr>
<tr>
<td>G.1 Introduction</td>
<td>G-1</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Remotely Controlled Power Supply Configurations Using Kepco Products</td>
<td>vi</td>
</tr>
<tr>
<td>2-1</td>
<td>Installation of BIT Card into BOP</td>
<td>2-2</td>
</tr>
<tr>
<td>2-2</td>
<td>BIT 232/BIT 232-F Switch and Adjustment Locations</td>
<td>2-4</td>
</tr>
<tr>
<td>2-3</td>
<td>RS 232C Connector</td>
<td>2-5</td>
</tr>
<tr>
<td>3-1</td>
<td>BOP Power Supply, Internal Calibration Control Locations</td>
<td>3-2</td>
</tr>
<tr>
<td>3-2</td>
<td>Current Shunt Connections</td>
<td>3-5</td>
</tr>
<tr>
<td>4-1</td>
<td>Tree Diagram of SCPI Commands Used with BIT 232/BIT 232-F Interface Card</td>
<td>4-3</td>
</tr>
<tr>
<td>4-2</td>
<td>Message Structure</td>
<td>4-5</td>
</tr>
<tr>
<td>A-1</td>
<td>*CLS — Clear Status Command</td>
<td>A-1</td>
</tr>
<tr>
<td>A-2</td>
<td>*ESE — Standard Event Status Enable Command</td>
<td>A-2</td>
</tr>
<tr>
<td>A-3</td>
<td>*ESE? — Standard Event Status Enable Query</td>
<td>A-2</td>
</tr>
<tr>
<td>A-4</td>
<td>*ESR? — Event Status Register Query</td>
<td>A-3</td>
</tr>
<tr>
<td>A-5</td>
<td>*IDN? — Identification Query</td>
<td>A-3</td>
</tr>
<tr>
<td>A-6</td>
<td>*OPC — Operation Complete Command</td>
<td>A-4</td>
</tr>
<tr>
<td>A-7</td>
<td>*OPC? — Operation Complete Query</td>
<td>A-4</td>
</tr>
<tr>
<td>A-8</td>
<td>*RST — Reset Command</td>
<td>A-5</td>
</tr>
<tr>
<td>A-9</td>
<td>*SRE — Service Request Enable Command</td>
<td>A-6</td>
</tr>
<tr>
<td>A-10</td>
<td>*SRE? — Service Request Enable Query</td>
<td>A-6</td>
</tr>
<tr>
<td>A-11</td>
<td>*STB? — Status Byte Register Query</td>
<td>A-7</td>
</tr>
<tr>
<td>A-12</td>
<td>*TRG — Trigger Command</td>
<td>A-7</td>
</tr>
<tr>
<td>A-13</td>
<td>*TST? — Self Test Query</td>
<td>A-8</td>
</tr>
<tr>
<td>A-14</td>
<td>*WAI — Wait-to-continue Command</td>
<td>A-8</td>
</tr>
<tr>
<td>B-1</td>
<td>INITiate[:IMMediate] Command</td>
<td>B-1</td>
</tr>
<tr>
<td>B-2</td>
<td>INITiate:CONTInuous Command</td>
<td>B-2</td>
</tr>
<tr>
<td>B-3</td>
<td>INITiate:CONTInuous Query</td>
<td>B-2</td>
</tr>
<tr>
<td>B-4</td>
<td>MEASure:CURRent? and MEASure:VOLTage? Queries</td>
<td>B-3</td>
</tr>
<tr>
<td>B-5</td>
<td>CURRent Command</td>
<td>B-4</td>
</tr>
<tr>
<td>B-6</td>
<td>CURRent Query</td>
<td>B-4</td>
</tr>
<tr>
<td>B-7</td>
<td>CURRent:TRIGgered Command</td>
<td>B-5</td>
</tr>
<tr>
<td>B-8</td>
<td>CURRent:TRIGgered Query</td>
<td>B-5</td>
</tr>
<tr>
<td>B-9</td>
<td>VOLTage Command</td>
<td>B-6</td>
</tr>
<tr>
<td>B-10</td>
<td>VOLTage Query</td>
<td>B-6</td>
</tr>
<tr>
<td>B-11</td>
<td>VOLTage:TRIGgered Command</td>
<td>B-7</td>
</tr>
<tr>
<td>B-12</td>
<td>VOLTage:TRIGgered Query</td>
<td>B-7</td>
</tr>
<tr>
<td>B-13</td>
<td>FUNCTION:MODE Command</td>
<td>B-8</td>
</tr>
<tr>
<td>B-14</td>
<td>STATus:OPERation:CONDition Query</td>
<td>B-8</td>
</tr>
<tr>
<td>B-15</td>
<td>STATus:OPERation:ENABLE Command</td>
<td>B-9</td>
</tr>
<tr>
<td>B-16</td>
<td>STATus:OPERation:ENABLE Query</td>
<td>B-9</td>
</tr>
<tr>
<td>B-17</td>
<td>STATus:OPERation QUERY</td>
<td>B-10</td>
</tr>
<tr>
<td>B-18</td>
<td>STATus:PRESet Command</td>
<td>B-10</td>
</tr>
<tr>
<td>B-19</td>
<td>STATus:QUESTionable? Query</td>
<td>B-11</td>
</tr>
<tr>
<td>B-20</td>
<td>STATus:QUESTionable:CONDition? Query</td>
<td>B-11</td>
</tr>
<tr>
<td>B-21</td>
<td>STATus:QUESTionable:ENABLE Command</td>
<td>B-12</td>
</tr>
<tr>
<td>B-22</td>
<td>STATus:QUESTionable:ENABLE? Query</td>
<td>B-12</td>
</tr>
<tr>
<td>B-23</td>
<td>SYStem:ERROR? Query</td>
<td>B-13</td>
</tr>
<tr>
<td>B-24</td>
<td>SYStem:LANGuage Command</td>
<td>B-13</td>
</tr>
<tr>
<td>C-1</td>
<td>FNC — Function Command</td>
<td>C-1</td>
</tr>
<tr>
<td>C-2</td>
<td>INX — Initiate Op Code Command</td>
<td>C-2</td>
</tr>
<tr>
<td>C-3</td>
<td>FTH — Fetch Command</td>
<td>C-2</td>
</tr>
<tr>
<td>C-4</td>
<td>SET Command</td>
<td>C-3</td>
</tr>
<tr>
<td>C-5</td>
<td>RST — Reset Command</td>
<td>C-4</td>
</tr>
<tr>
<td>C-6</td>
<td>CNF, IST — Confidence Test, Internal Self Test Commands</td>
<td>C-4</td>
</tr>
<tr>
<td>C-7</td>
<td>STA — Status Command</td>
<td>C-5</td>
</tr>
<tr>
<td>C-8</td>
<td>GAL — Go to Alternate Language Command</td>
<td>C-6</td>
</tr>
<tr>
<td>TABLE</td>
<td>TITLE</td>
<td>PAGE</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>1-1</td>
<td>Kepco BIT 232, 488 and 4882 Digital Programming Cards</td>
<td>1-1</td>
</tr>
<tr>
<td>1-2</td>
<td>Applicability of BIT Interface Cards to Specific BOP Models</td>
<td>1-2</td>
</tr>
<tr>
<td>1-3</td>
<td>Specifications, BIT 232 and BIT 232-F</td>
<td>1-2</td>
</tr>
<tr>
<td>2-1</td>
<td>Power Supply Identification Switch S2 Setting</td>
<td>2-4</td>
</tr>
<tr>
<td>2-2</td>
<td>Input/Output Pin Assignments</td>
<td>2-5</td>
</tr>
<tr>
<td>3-1</td>
<td>BOP Power Supply, Internal Calibration Controls</td>
<td>3-3</td>
</tr>
<tr>
<td>4-1</td>
<td>SCPI Command Index</td>
<td>4-4</td>
</tr>
<tr>
<td>4-2</td>
<td>Rules Governing Shortform Keywords</td>
<td>4-4</td>
</tr>
<tr>
<td>A-1</td>
<td>SCPI Common Command/query Index</td>
<td>A-1</td>
</tr>
<tr>
<td>B-1</td>
<td>SCPI Subsystem Command/query Index</td>
<td>B-1</td>
</tr>
<tr>
<td>C-1</td>
<td>CIIL Subsystem Command/query Index</td>
<td>C-1</td>
</tr>
<tr>
<td>C-2</td>
<td>CIIL Error Messages</td>
<td>C-5</td>
</tr>
<tr>
<td>C-3</td>
<td>CIIL Error Handling Utility Commands</td>
<td>C-6</td>
</tr>
<tr>
<td>D-1</td>
<td>ROM BIOS Routines to be replaced with equivalent for non-IBM-PC-compatible PC’s</td>
<td>D-1</td>
</tr>
<tr>
<td>E-1</td>
<td>ROM BIOS Routines to be replaced with equivalent for non-IBM-PC-compatible PC’s</td>
<td>E-1</td>
</tr>
</tbody>
</table>
FIGURE 1.1. REMOTELY CONTROLLED POWER SUPPLY CONFIGURATIONS USING KEPCO PRODUCTS
SECTION 1 - INTRODUCTION

1.1 SCOPE OF MANUAL

This manual contains instructions for the installation, operation and maintenance of the BIT 232 and BIT 232-F Interface Cards manufactured by Kepco, Inc., Flushing, NY, U.S.A. References to “BIT Card” refer to both models.

1.2 GENERAL DESCRIPTION

The Kepco BIT Card Series were designed as an accessory for the Kepco BOP series bipolar power supplies. The BIT cards make it possible to control the BOP output by means of digital input signals (see Figure 1-1). The BIT card acts as an interface between the digital data bus and the BOP, accepting the digital input data and converting it to an analog signal, which in turn, controls the BOP output. The BIT 232 provides RS232 communication capability. It is fully compliant with SCPI and CIIL high level programming languages.

The BIT design group consists of five models. Field installable interface cards carry the prefix “BIT”. BOP bipolar power supplies with an installed BIT card carry the suffixes shown in Table 1-1.

1.3 SPECIFICATIONS, BIT 232/BIT 232-F (SEE TABLE 1-3)

Refer to Table 1-3 for BIT 232/BIT 232-F specifications.

1.4 ACCESSORIES

The sample programs illustrated in Appendices E through G are available on 3.5 in. diskette, Kepco P/N 254-0019.

---

### TABLE 1-1. KEPCO BIT 232, 488 AND 4882 DIGITAL PROGRAMMING CARDS

<table>
<thead>
<tr>
<th>FIELD INSTALLABLE PROGRAMMING CARD MODEL</th>
<th>FACTORY INSTALLED PROGRAMMING CARD BOP SUFFIX</th>
<th>INPUT CODING</th>
<th>RESOLUTION</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT 232 (See Table 1-2) BIT 232-F (See Table 1-2)</td>
<td>-232</td>
<td>SERIAL: 8 DATA BITS, NO PARITY BIT 1 STOP BIT</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>BIT 488-B</td>
<td>-488-B</td>
<td>BYTE-SERIAL</td>
<td>12 BITS (BINARY)</td>
<td>8 BITS (BINARY)</td>
</tr>
<tr>
<td>BIT 488-D</td>
<td>-488-D</td>
<td>BYTE-SERIAL</td>
<td>3-DIGIT (BCD)</td>
<td>2-DIGIT (BCD)</td>
</tr>
<tr>
<td>BIT 4882 (See Table 1-2) BIT 4882-F (See Table 1-2)</td>
<td>-4882</td>
<td>BYTE-SERIAL</td>
<td>12 BITS (BINARY)</td>
<td>12 BITS (BINARY)</td>
</tr>
<tr>
<td>BIT TMA-27</td>
<td>-TMA</td>
<td>2-WIRE-SERIAL</td>
<td>12 BITS (BINARY)</td>
<td>12 BITS (BINARY)</td>
</tr>
</tbody>
</table>

Except for the installation procedures, the BIT 232 and 232-F cards are identical. The BIT 232F includes a replacement transformer needed to modify earlier BOP Models indicated in Table 1-2.
### TABLE 1-2. APPLICABILITY OF BIT INTERFACE CARDS TO SPECIFIC BOP MODELS

<table>
<thead>
<tr>
<th>BOP TO BE MODIFIED</th>
<th>APPLICABLE CARD</th>
<th>BOP TO BE MODIFIED</th>
<th>APPLICABLE CARD</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODEL</td>
<td>REVISION NO.</td>
<td>CARD</td>
<td>MODEL</td>
</tr>
<tr>
<td>20-5M</td>
<td>1</td>
<td>BIT 232-F, BIT 4882-F</td>
<td>50-8M</td>
</tr>
<tr>
<td></td>
<td>2 AND LATER</td>
<td>BIT 232, BIT 4882</td>
<td></td>
</tr>
<tr>
<td>20-10M</td>
<td>12 TO 16</td>
<td>BIT 232-F, BIT 4882-F</td>
<td>72-3M</td>
</tr>
<tr>
<td></td>
<td>17 AND LATER</td>
<td>BIT 232, BIT 4882</td>
<td></td>
</tr>
<tr>
<td>20-20M</td>
<td>8 TO 14</td>
<td>BIT 232-F, BIT 4882-F</td>
<td>72-6M</td>
</tr>
<tr>
<td></td>
<td>15 AND LATER</td>
<td>BIT 232, BIT 4882</td>
<td></td>
</tr>
<tr>
<td>36-6M</td>
<td>12 TO 18</td>
<td>BIT 232-F, BIT 4882-F</td>
<td>100-1M</td>
</tr>
<tr>
<td></td>
<td>19 AND LATER</td>
<td>BIT 232, BIT 4882</td>
<td></td>
</tr>
<tr>
<td>36-12M</td>
<td>7 TO 12</td>
<td>BIT 232-F, BIT 4882-F</td>
<td>100-2M</td>
</tr>
<tr>
<td></td>
<td>13 AND LATER</td>
<td>BIT 232, BIT 4882</td>
<td></td>
</tr>
<tr>
<td>50-2M</td>
<td>16 TO 20</td>
<td>BIT 232-F, BIT 4882-F</td>
<td>100-4M</td>
</tr>
<tr>
<td></td>
<td>21 AND LATER</td>
<td>BIT 232, BIT 4882</td>
<td></td>
</tr>
<tr>
<td>50-4M</td>
<td>6 TO 12</td>
<td>BIT 232-F, BIT 4882-F</td>
<td>200-1M</td>
</tr>
<tr>
<td></td>
<td>13 AND LATER</td>
<td>BIT 232, BIT 4882</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** For modification of BOP Models with revision numbers that do not appear in this table, contact Kepco for assistance.

### TABLE 1-3. SPECIFICATIONS, BIT 232 AND BIT 232-F

<table>
<thead>
<tr>
<th>SPECIFICATION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPUT VOLTAGE (MAIN CHANNEL)</td>
<td>0 ± 10V</td>
</tr>
<tr>
<td>OUTPUT VOLTAGE (LIMIT CHANNEL)</td>
<td>0 to +10V</td>
</tr>
<tr>
<td>OUTPUT CURRENT (EACH CHANNEL)</td>
<td>0 to ± 2 mA max.</td>
</tr>
<tr>
<td>OUTPUT IMPEDANCE</td>
<td>&lt;0.05 ohms</td>
</tr>
</tbody>
</table>
| TEMPERATURE COEFFICIENT | Full scale: ± 35 ppm/°C max  
Zero: ± 20μV/°C max |
| OPTICAL ISOLATION | Digital and Analog grounds can be separated by a maximum of 500 Volts. |
| DIGITAL INPUT FORMAT | Serial, 8 data bits, no parity, 1 stop |
| POWER REQUIREMENT | Supplied by BOP |
| PROGRAMMING RESOLUTION | VOLTAGE CURRENT |
| RATING/DESCRIPTION | CONDITION |
| 0.024% | 12 Bits |
| DATA READBACK ACCURACY | VOLTAGE CURRENT |
| 0.2% | of Max. Voltage |
| | of Max. Current |
SECTION 2 - INSTALLATION

2.1 UNPACKING AND INSPECTION

The BIT Card has been thoroughly inspected and tested prior to packing and is ready for operation following installation. Unpack, saving original packing material. If any indication of damage is found, file a claim immediately with the responsible transport service.

2.2 SET START-UP DEFAULTS (SEE FIGURE 2-2)

Start-up defaults, consisting of Start-up Language and Power Supply Identification are initially set by means of DIP switches as described in the following paragraphs.

2.2.1 START-UP LANGUAGE DEFAULT (SEE FIGURE 2-2)

DIP switch S1 position 6 sets the Start-up Language Default:

- OFF (0) = SCPI (factory default)
- ON (1) = CIIL

2.2.2 SET POWER SUPPLY IDENTIFICATION SWITCH (SEE FIGURE 2-2)

Power Supply Identification switch S2 (Figure 2-2) identifies the BOP model to be controlled by the BIT Card. Set Switch S2 positions 1 through 6 in accordance with Table 2-1.

2.3 INSTALLATION OF BIT CARD INTO THE BOP

Refer to Figure 2-1 to install the BIT 232-F BIT Card.
NOTE: Step numbers coincide with encircled numbers on Figure 2-1, sheet 2.

Step 1. Disconnect a-c power from BOP by removing line cord.

Step 2. Remove BOP cover (see Section 5, Figure 5-1 of your BOP Instruction Manual).

Step 3. Remove and discard Rear Cover Plate (PN 128-1434) and associated hardware.

Step 4. Remove J204 Connector Assembly (PN 241-0680) from Location #1, save for Step 11.

Step 5. Locate Transformer T202 and note part number (stamped on top):
   if PN 100-2167, remove from unit and discard (applicable to BIT 232-F only).
   if PN 100-2354, unplug connector from Location #4 only.

Step 6. Locate Rear Bracket and note part number (stamped on outside left edge)
   if PN 128-1566, remove and discard, replace with Bracket noted in Step 9.

Step 7. Unpack the BIT Card Installation Components
   BIT 232-F: (Transformer, PCB Assembly, Cables #1 and #2, Connector Assembly, three
              (3) Knurled Nuts, five (5) washers, Spacer and Bracket).
   BIT 232: PCB Assembly, Cables #1 and #2, Connector Assembly, three (3) Knurled
            Nuts, five (5) washers, Spacer and Bracket).

Step 8. BIT 232-F only: Mount Transformer T202 (PN 100-2354) if required (ref. Step 5, above).


Step 10. Install PCB Assembly (PN 235-1166) into the guides, slide into position so that mounting
         holes in PCB Assembly line up with the three mounting posts on the BOP mounting bracket.

Step 11. Secure the BIT Card to the Mounting Posts using the Knurled Nuts and Lockwashers.
         Mount Connector J204 (Ref. Step 4, above) into Location #3 using the Lockwashers, Hex
         Spacer and Knurled Nut.

Step 12. Install Cable #1 (18-position connectors) to the BIT Card; mate the other end of the cable
         with Location #1 on BOP A1 Assembly.

Step 13. Step 7A Install Cable #2 (5-position connectors) to the BIT Card; mate the other end of the
         cable with Location #2 on BOP A1 Assembly.

Step 14. Plug in 3 and 9 pin Connectors and Primary Leads from Transformer 100-2354 as shown.

Step 15. Mark ",-232" after Model No. on Nameplate (see Detail A).

Step 16. Remove "Control Identification" label (PN 188-1107) and "Address Label" (PN 188-1012).
         Affix revised "Control Identification" label (PN 188-1826) and "Address Label" (PN
         188-1726) in vacated positions (with part numbers facing front panel).

Step 17. Affix "Connector Identification" label (PN 188-1912) to rear bracket (PN 188-1810) between
         rear programming connector and Interface Card (see Detail C).

Step 18. Reinstall BOP cover.

Step 19. Perform calibration procedure detailed in Section 3 of this manual.

FIGURE 2-1. INSTALLATION OF BIT CARD INTO BOP (SHEET 1 OF 2)
FIGURE 2-1. INSTALLATION OF BIT CARD INTO BOP (SHEET 2 OF 2).
FIGURE 2-2. BIT 232/BIT 232-F SWITCH AND ADJUSTMENT LOCATIONS

TABLE 2-1. POWER SUPPLY IDENTIFICATION SWITCH S2 SETTING

<table>
<thead>
<tr>
<th>MODEL</th>
<th>SELECTOR SWITCH S2 SECTION</th>
<th>HEX VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOP 50-2M</td>
<td>0 0 0 0 0 0</td>
<td>00</td>
</tr>
<tr>
<td>100-1M</td>
<td>1 0 0 0 0 0</td>
<td>01</td>
</tr>
<tr>
<td>20-10M</td>
<td>0 1 0 0 0 0</td>
<td>02</td>
</tr>
<tr>
<td>36-6M</td>
<td>1 1 0 0 0 0</td>
<td>03</td>
</tr>
<tr>
<td>50-4M</td>
<td>0 0 1 0 0 0</td>
<td>04</td>
</tr>
<tr>
<td>72-3M</td>
<td>1 0 1 0 0 0</td>
<td>05</td>
</tr>
<tr>
<td>100-2M</td>
<td>0 1 1 0 0 0</td>
<td>05</td>
</tr>
<tr>
<td>20-20M</td>
<td>1 1 1 0 0 0</td>
<td>07</td>
</tr>
<tr>
<td>36-12M</td>
<td>0 0 0 1 0 0</td>
<td>08</td>
</tr>
<tr>
<td>50-8M</td>
<td>1 0 0 1 0 0</td>
<td>09</td>
</tr>
<tr>
<td>72-6M</td>
<td>0 1 0 1 0 0</td>
<td>0A</td>
</tr>
<tr>
<td>100-4M</td>
<td>1 1 0 1 0 0</td>
<td>0B</td>
</tr>
<tr>
<td>200-1M</td>
<td>1 0 1 1 0 0</td>
<td>0C</td>
</tr>
<tr>
<td>20-5M</td>
<td>0 1 1 1 0 0</td>
<td>0D</td>
</tr>
<tr>
<td>BOP 50-2M</td>
<td>0 0 0 0 0 0</td>
<td>00</td>
</tr>
</tbody>
</table>
2.4 INPUT/OUTPUT SIGNALS

The RS232 port is a standard 9 pin connector (Figure 2-4) conforming to the IBM AT 9-Pin RS 232 Serial Interface. Refer to Table 2-3 for pin assignments.

**TABLE 2-2. INPUT/OUTPUT PIN ASSIGNMENTS**

<table>
<thead>
<tr>
<th>PIN</th>
<th>SIGNAL NAME</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SGND</td>
<td>Signal Ground</td>
</tr>
<tr>
<td>2</td>
<td>RXD</td>
<td>Receive Data</td>
</tr>
<tr>
<td>3</td>
<td>TXD</td>
<td>Transmit Data</td>
</tr>
<tr>
<td>4</td>
<td>DTR</td>
<td>Data Terminal Ready</td>
</tr>
<tr>
<td>5</td>
<td>SGND</td>
<td>Signal Ground</td>
</tr>
<tr>
<td>6</td>
<td>DSR</td>
<td>Data Set Ready</td>
</tr>
<tr>
<td>7</td>
<td>RTS</td>
<td>Request To Send</td>
</tr>
<tr>
<td>8</td>
<td>CTS</td>
<td>Clear To Send</td>
</tr>
<tr>
<td>9</td>
<td>SGND</td>
<td>Signal Ground</td>
</tr>
</tbody>
</table>

**FIGURE 2-3. RS 232C CONNECTOR**

2.5 RS 232 CONNECTIONS

Since the BIT Card uses a 9-pin male connector, it is classified as a Data Terminal Equipment (DTE) in accordance with the RS 232 Standard (equipment using a female connector is classified as Data Communication Equipment, DCE).

Either a DTE to DTE or a null modem cable is required to connect the BIT Card to an IBM-PC compatible computer. This cable connects RXD at one end to TXD at the other end, DTR at one end to DSR at the other end, and CTS at one end with RTS at the other end.
SECTION 3 - CALIBRATION

NOTE: The calibration procedures below are for the purpose of recalibration and for the case where the BIT card is installed by the user. Unless otherwise noted, syntax is in SCPI.

3.1 EQUIPMENT REQUIRED
The following is a listing of equipment required for calibration of the BIT Card installed in a Kepco “BOP” Series Power Supply:

A. Precision digital voltmeter (DVM), 5 digit minimum resolution (suggested).

B. An RS232 compatible Controller, (with appropriate software) connected to BOP Power Supply with an RS232 cable.

C. Precision four-terminal current shunt (with suitable power rating and tolerance for the currents to be measured).

3.2 ADJUSTMENT OF THE BOP ±10 VOLT CALIBRATION CONTROLS (R31, R32)
NOTE: BOP cover removal required for this procedure.

1. Connect a DVM to the REAR PROGRAMMING CONNECTOR (PC-12); between common and pin 28 (+10Vdc REFERENCE).

2. Turn the BOP Power Supply “ON” and locate the calibration controls (see Figure 3-1, refer to Table 3-1). Adjust R31 for +10.000Vdc.

3. Turn the BOP Power Supply “OFF” and connect DVM between common and pin 22 (–10Vdc REFERENCE).


5. Turn the BOP Power Supply “OFF”.

3.3 ADJUSTMENT OF THE AMMETER ZERO (R50)

1. Without a load connected to the BOP output, connect the DVM to the REAR PROGRAMMING CONNECTOR (PC-12); between COMMON and pin 10.

2. Turn the BOP Power Supply “ON” and locate AMMETER ZERO control R50 (see Figure 3-1, refer to Table 3-1).

3. Adjust the control for zero, ±100 microvolts.

4. Turn the BOP Power Supply “OFF”.
3.4 Adjustment of the Output Voltage Zero (R81)

1. Without a load connected to the BOP output, connect a DVM between the FRONT PANEL SENSING TERMINALS of the BOP Power Supply.

2. Turn the BOP Power Supply “ON”, program the BOP Power Supply to ZERO VOLTAGE AND MAXIMUM CURRENT LIMIT.

3. Locate Eo COMP AMP ZERO control R81 (see Figure 3-1, refer to Table 3-1).

4. Adjust control R81 for zero, ±100 microvolts.

3.5 Adjustment of the Full Scale Output Voltage (R21)

1. Program the BOP Power Supply for PLUS FULL SCALE VOLTAGE.

2. Locate VOLTAGE FULL SCALE control R21 (see Figures 2-1 and 3-1, refer to Table 3-1).

3. Adjust control R21 for FULL SCALE VOLTAGE, ±1 millivolt.

4. Program the BOP Power Supply for MINUS FULL SCALE. The output should be NEGATIVE FULL SCALE ±0.024% (±1 LSB).
3.6 VOLTAGE READING ZERO CALIBRATION (R35)
1. Program the BOP power supply for ZERO VOLTAGE and MAXIMUM CURRENT LIMIT.

2. ’MEASURE’ (‘FETCH’ in CIIL syntax) the OUTPUT VOLTAGE of the BOP.

3. Locate VOLTAGE READBACK ZERO control R35 (see Figures 2-1 and 3-1, refer to Table 3-1).

4. Continue to ’MEASURE’ (‘FETCH’ in CIIL syntax) the VOLTAGE READING while adjusting control R35 until the ‘MEASUREd’ (‘FETCHed’ in CIIL syntax) value is not 0.0.

5. Continue to ’MEASURE’ (‘FETCH’ in CIIL syntax) the VOLTAGE READING while adjusting control R35 until the ‘MEASUREd’ (‘FETCHed’ in CIIL syntax) value is 0.0. Once a stable value of 0.0 is reached, continue rotating R35 four full (360°) turns in the same direction. Verify VOLTAGE READING is 0.0.

3.7 VOLTAGE READING CALIBRATION (R19)
1. Program the BOP Power Supply for PLUS FULL SCALE VOLTAGE, less one percent (verify by reading external DVM).

NOTE: If the unit is calibrated using CIIL syntax, send the ‘GAL’ command followed by the ‘FØ’ switch command.

2. ’MEASURE’ (‘FETCH’ in CIIL syntax) the OUTPUT VOLTAGE of the BOP Power Supply.

3. Locate VOLTAGE READ. CAL. control R19 (see Figures 2-1 and 3-1, refer to Table 3-1).

<table>
<thead>
<tr>
<th>REFERENCE DESIGNATION</th>
<th>CONTROL NAME</th>
<th>PURPOSE</th>
<th>ADJUSTMENT PROCEDURE (PAR.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R19</td>
<td>VOLTAGE READING</td>
<td>'MEASURE'd voltage reading adjustment</td>
<td>3.7</td>
</tr>
<tr>
<td>R20</td>
<td>CURRENT READING</td>
<td>'MEASURE'd current reading adjustment</td>
<td>3.11</td>
</tr>
<tr>
<td>R21</td>
<td>VOLTAGE FULL SCALE</td>
<td>Full scale output voltage adjustment</td>
<td>3.5</td>
</tr>
<tr>
<td>R22</td>
<td>CURRENT FULL SCALE</td>
<td>Full scale output current adjustment</td>
<td>3.9</td>
</tr>
<tr>
<td>R31, R32</td>
<td>(±) 10V CAL.</td>
<td>Reference voltage calibration</td>
<td>3.2</td>
</tr>
<tr>
<td>R35</td>
<td>VOLTAGE READBACK ZERO</td>
<td>Aero output voltage adjustment</td>
<td>3.6</td>
</tr>
<tr>
<td>R36</td>
<td>CURRENT READBACK ZERO</td>
<td>Zero output current adjustment</td>
<td>3.10</td>
</tr>
<tr>
<td>R50</td>
<td>AMMETER ZERO</td>
<td>Sensing amplifier offset adjustment</td>
<td>3.3</td>
</tr>
<tr>
<td>R81</td>
<td>E COMP AMP ZERO</td>
<td>Voltage channel zero adjustment</td>
<td>3.4</td>
</tr>
<tr>
<td>R83</td>
<td>I COMP AMP ZERO</td>
<td>Current channel zero adjustment</td>
<td>3.8</td>
</tr>
</tbody>
</table>
4. Continue to 'MEASURE' ('FETCH' in CIIL syntax) the VOLTAGE READING while adjusting control R19, until the 'MEASUREd' ('FETCHed' in CIIL syntax) value matches the programmed value.

5. Turn the BOP Power Supply "OFF".

3.8 ADJUSTMENT OF THE OUTPUT CURRENT ZERO (R83)

1. With the BOP Power Supply "OFF", connect a precision current shunt between the FRONT PANEL OUTPUT TERMINALS.

2. Connect the DVM to the REAR PROGRAMMING CONNECTOR (PC 12); between COMMON and pin 10.

3. Turn the BOP "ON" and program the BOP Power Supply to ZERO CURRENT and MAXIMUM VOLTAGE LIMIT.

4. Locate Io COMP AMP ZERO control R83 (see Figure 3-1, refer to Table 3-1).

5. Adjust control R83 for zero, ±0.5 millivolts.

6. Turn the BOP Power Supply "OFF".

3.9 ADJUSTMENT OF THE FULL SCALE OUTPUT CURRENT (R22)

1. With the BOP Power Supply "OFF", connect the DVM to the PRECISION 4-TERMINAL SHUNT (see Figure 3-2)

2. Turn the BOP Power Supply "ON" and program the BOP Power Supply for PLUS FULL SCALE CURRENT and MAXIMUM VOLTAGE LIMIT.

3. Locate CURRENT FULL SCALE control R22 (see Figures 2-1 and 3-1, refer to Table 3-1).

4. Adjust control R22 for exactly FULL SCALE CURRENT.

5. Program the BOP Power Supply for MINUS FULL SCALE CURRENT. The output should be NEGATIVE FULL SCALE within ±0.024% (±1 LSB or ±1 mA [which ever is greater] ) of the POSITIVE VALUE.

3.10 CURRENT READING ZERO CALIBRATION (R36)

1. Program the BOP Power Supply for ZERO CURRENT and MAXIMUM VOLTAGE LIMIT.

2. 'MEASURE' ('FETCH' in CIIL syntax) the OUTPUT CURRENT of the BOP Power Supply.

3. Locate CURRENT READBACK ZERO control R36 (see Figures 2-1 and 3-1, refer to Table 3-1).

4. Continue to 'MEASURE' ('FETCH' in CIIL syntax) the CURRENT, while adjusting control R36, until the 'MEASUREd' ('FETCHed' in CIIL syntax) value is not 0.0.

5. Continue to 'MEASURE' ('FETCH' in CIIL syntax) the OUTPUT CURRENT while adjusting control R36 until the 'MEASUREd' ('FETCHed' in CIIL syntax) value is 0.0. Once a stable value of 0.0 is reached, continue rotating R36 four full (360°) turns in the same direction. Verify OUTPUT CURRENT is 0.0.
**FIGURE 3-2. CURRENT SHUNT CONNECTIONS**

3.11 CURRENT READING CALIBRATION (R20)

1. Program the BOP Power Supply for PLUS FULL SCALE CURRENT, less one percent (verify by reading external DVM).

   NOTE: If the unit is calibrated using CIIL syntax, send the ‘GAL’ command followed by the ‘FØ’ switch command.

2. ‘MEASURE’ (‘FETCH’ in CIIL syntax) the OUTPUT CURRENT of the BOP Power Supply.

3. Locate CURRENT READ. CAL. control R20 (see Figures 2-1 and 3-1, refer to Table 3-1).

4. Continue to ‘MEASURE’ (‘FETCH’ in CIIL syntax) the CURRENT while adjusting control R20, until the ‘MEASUREd’ (‘FETCHed’ in CIIL syntax) value matches the programmed value.
SECTION 4 - OPERATION

4.1 GENERAL

The Kepco BOP Power Supply, with an installed BIT 232/BIT 232-F Interface Card, may be pro-
grammed over the RS232C bus using either SCPI (Standard Commands for Programmable
Instruments) or CIIL (Control Interface Intermediate Language) commands. SCPI and CIIL pro-
vide a common language used in an automatic test system. (Refer to Table 2-3 for input/output
signal allocations.)

4.2 RS232-C BUS PROTOCOL

The BIT Card may be operated via an RS232-C terminal, or from a PC using a terminal emula-
tion program. The following settings must be observed:

• Baud rate: 9600
• Parity: None
• Data Bits 8
• Stop Bits 1

The above settings are established by the firmware and cannot be changed by the user
although firmware with alternate settings can be ordered. It is recommended that the user pro-
gram the computer’s serial interface to match the above setting. Refer to PAR. 2.5 for RS232
connections.

4.3 RS232 IMPLEMENTATION

The following paragraphs are provided to help the user understand how the RS232 interface is
implemented in the BIT 232 and BIT 232-F Interface Cards. Since the RS232 protocol does not
use a parity bit, the echo method is used to ensure reliable communication between the com-
mand originator (computer) and the BIT Card, thus avoiding a more complex “handshake” pro-
tocol.

When a character is received through the RS232 Interface, the software checks for the back-
space character, hex value 08 (08H). If the backspace character is not detected, the received
character is put in a buffer, the buffer pointer is incremented, and it is sent (echoed) back to the
originator. If the backspace character is detected, the buffer pointer is decremented, and three
characters are sent to the originator: Backspace (08H), Blank (20H), and Backspace (08H).
These three characters permit a direct interface with a terminal emulation program, effectively
erasing the last character.

When either a Carriage Return, CR, (0D decimal value 13, 1310) or Line Feed, LF, (0A10, 1010)
is received by the BIT Card, the buffer contents are transferred to the SCPI (PAR.4.4) or CIIL
(PAR. 4.5) parser for analysis and execution of command(s). If the command requires the power
supply to respond, the response message is immediately returned to the command originator
via the RS232 interface.

To inform the command originator that the parsing and execution phases are complete, the BIT
Card returns the following three characters: CR, LF, and > (3EH, 6210). These characters create
the “prompt” effect if a terminal emulation program is the command(s) originator.
Sample programs provided in Appendices D through G guide the user in setting up a program to communicate with the BOP via the BIT 232/BIT 232-F Interface Card.

4.4 **SCPI PROGRAMMING**

SCPI (Standard Commands for Programmable Instruments) is a programming language conforming to the protocols and standards established by IEEE 488.2 (reference document ANSI/IEEE Std 488.2, IEEE Standard Codes, Formats, Protocols, and Common Commands). SCPI commands are sent to the BIT Card as output strings within the selected programming language (PASCAL, BASIC, etc.) in accordance with the manufacturer’s requirements for the particular interface card used.

Different programming languages (e.g., BASIC, C, PASCAL, etc.) have different ways of representing data that is to be put on the RS232C bus. It is up to the programmer to determine how to output the character sequence required for the programming language used. Address information must be included before the command sequence.

4.4.1 **SCPI MESSAGES**

There are two kinds of SCPI messages: program messages from controller to power supply, and response messages from the power supply to the controller. Program messages consist of one or more properly formatted commands/queries and instruct the power supply to perform an action; the controller may send a program message at any time. Response messages consist of formatted data; the data can contain information regarding operating parameters, power supply state, status, or error conditions.

4.4.2 **COMMON COMMANDS/QUERIES**

Common commands and queries are defined by the IEEE 488.2 standard to perform overall power supply functions (such as identification, status, or synchronization) unrelated to specific power supply operation (such as setting voltage/current). Common commands and queries are preceded by an asterisk (*) and are defined and explained in Appendix A (see Table 4-4). Refer also to syntax considerations (PARs 4.4.3 through 4.4.6).

4.4.3 **SCPI SUBSYSTEM COMMAND/QUERY STRUCTURE**

Subsystem commands/queries are related to specific power supply functions (such as setting output voltage, current limit, etc.) Figure 4-1 is a tree diagram illustrating the structure of SCPI subsystem commands used in the BIT Card with the “root” at the left side, and specific commands forming the branches. The subsystem commands are defined and explained in Appendix B (see Table 4-4).
4.4.4 PROGRAM MESSAGE STRUCTURE

SCPI program messages (commands from controller to power supply) consist of one or more message units ending in a message terminator (required by Kepco power modules). The message terminator is not part of the syntax; it is defined by the way your programming language indicates the end of a line (such as a “newline” or “end-of-line” character). The message unit is a keyword consisting of a single command or query word followed by a message terminator (e.g., CURR?<newline> or TRIG<end-of-line>). The message unit may include a data parameter after the keyword separated by a space; the parameter is usually numeric (e.g., CURR 5<newline>), but may also be a string (e.g., OUTP ON<newline>). Figure 4-2 illustrates the message structure, showing how message units are combined. The following subparagraphs explain each component of the message structure.

NOTE: An alternative to using the message structure for multiple messages defined in the following paragraphs is to send each command as a separate line. In this case each command must use the full syntax shown in Appendix B.

4.4.4.1 KEYWORD

Keywords are instructions recognized by a decoder within the BIT Card, referred to as a “parser.” Each keyword describes a command function; all keywords used by the BIT Card are listed in Figure 4-1.

Each keyword has a long form and a short form. For the long form the word is spelled out completely (e.g. STATUS, OUTPUT, VOLTAGE, and TRIGGER are long form keywords). For the short form only the first three or four letters of the long form are used (e.g., STAT, VOLT, OUTP, and TRIG). The rules governing short form keywords are presented in Table 4-5.
### TABLE 4-1. SCPI COMMAND INDEX

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>COMMAND</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>*CLS</td>
<td>A-1</td>
<td>[SOUR]:CURR?</td>
<td>B-5</td>
</tr>
<tr>
<td>*ESE</td>
<td>A-2</td>
<td>[SOUR]:CURR:TRIG</td>
<td>B-6</td>
</tr>
<tr>
<td>*ESE?</td>
<td>A-2</td>
<td>[SOUR]:CURR:TRIG?</td>
<td>B-6</td>
</tr>
<tr>
<td>*ESR?</td>
<td>A-3</td>
<td>[SOUR]:VOLT</td>
<td>B-7</td>
</tr>
<tr>
<td>*IDN?</td>
<td>A-3</td>
<td>[SOUR]:VOLT?</td>
<td>B-7</td>
</tr>
<tr>
<td>*OPC</td>
<td>A-4</td>
<td>[SOUR]:VOLT:TRIG</td>
<td>B-8</td>
</tr>
<tr>
<td>OPC?</td>
<td>A-4</td>
<td>[SOUR]:VOLT:TRIG?</td>
<td>B-8</td>
</tr>
<tr>
<td>*RST</td>
<td>A-5</td>
<td>[SOUR]:FUNC:MODE</td>
<td>B-9</td>
</tr>
<tr>
<td>*SRE</td>
<td>A-6</td>
<td>STAT:OPER:COND?</td>
<td>B-9</td>
</tr>
<tr>
<td>*SRE?</td>
<td>A-6</td>
<td>STAT:OPER:ENAB</td>
<td>B-10</td>
</tr>
<tr>
<td>*TRG</td>
<td>A-7</td>
<td>STAT:OPER?</td>
<td>B-11</td>
</tr>
<tr>
<td>*TST</td>
<td>A-8</td>
<td>STAT:PRES</td>
<td>B-11</td>
</tr>
<tr>
<td>*WAI</td>
<td>A-8</td>
<td>STAT:QUES?</td>
<td>B-12</td>
</tr>
<tr>
<td>INIT[:IMM]</td>
<td>B-1</td>
<td>STAT:QUES:COND?</td>
<td>B-12</td>
</tr>
<tr>
<td>INIT:CONT</td>
<td>B-1</td>
<td>STAT:QUES:ENAB</td>
<td>B-13</td>
</tr>
<tr>
<td>MEAS:CURR?</td>
<td>B-3</td>
<td>SYST:ERR?</td>
<td>B-14</td>
</tr>
<tr>
<td>MEAS:VOLT?</td>
<td>B-3</td>
<td>SYST:LANG</td>
<td>B-14</td>
</tr>
<tr>
<td>[SOUR]:CURR</td>
<td>B-4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### TABLE 4-2. RULES GOVERNING SHORTFORM KEYWORDS

<table>
<thead>
<tr>
<th>IF NUMBER OF LETTERS IN LONGFORM KEYWORD IS:</th>
<th>AND FOURTH LETTER IS A VOWEL?</th>
<th>THEN SHORT FORM CONSISTS OF:</th>
<th>EXAMPLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 OR FEWER</td>
<td>(DOES NOT MATTER)</td>
<td>ALL LONG FORM LETTERS</td>
<td>MODE</td>
</tr>
<tr>
<td>5 OR MORE</td>
<td>NO</td>
<td>THE FIRST FOUR LONG FORM LETTERS</td>
<td>MEAS, OUTp, EVENt</td>
</tr>
<tr>
<td></td>
<td>YES</td>
<td>THE FIRST THREE LONG FORM LETTERS</td>
<td>LEVel, IMMEDIATE, ERRor</td>
</tr>
</tbody>
</table>
You must use the rules above when using keywords. Using an arbitrary short form such as ENABL for ENAB (ENABLE) or IMME for IMM (IMMEDIATE) will result in an error. Regardless of which form chosen, you must include all the letters required by that form.

To identify the short form and long form in this manual, keywords are written in upper case letters to represent the short form, followed by lower case letters indicating the long form (e.g., IMMEDIATE, EVENT, and OUTPUT). The parser, however, is not sensitive to case (e.g., outp, OutP, OUTPut, ouTPut, or OUTp are all valid).

4.4.4.2 KEYWORD SEPARATOR

If a command has two or more keywords, adjacent keywords must be separated by a colon (:) which acts as the keyword separator (e.g., CURR:LEV:TRIG). The colon can also act as a root specifier (paragraph 4.4.4.7).

4.4.4.3 QUERY INDICATOR

The question mark (?) following a keyword is a query indicator. This changes the command into a query. If there is more than one keyword in the command, the query indicator follows the last keyword. (e.g., VOLT? and MEAS:CURR?).
4.4.4 DATA

Some commands require data to accompany the keyword either in the form of a numeric value or character string. Data always follows the last keyword of a command or query (e.g., VOLT:LEV:TRIG 14 or SOUR:VOLT? MAX)

4.4.5 DATA SEPARATOR

Data must be separated from the last keyword by a space (e.g., VOLT:LEV:TRIG 14 or SOUR:VOLT? MAX)

4.4.6 MESSAGE UNIT SEPARATOR

When two or more message units are combined in a program message, they must be separated by a semicolon (;) (e.g., VOLT 15;MEAS:VOLT? and CURR 12; CURR:TRIG 12.5).

4.4.7 ROOT SPECIFIER

The root specifier is a colon (:) that precedes the first keyword of a program message. This places the parser at the root (top left, Figure 4-3) of the command tree. Note the difference between using the colon as a keyword separator and a root specifier in the following examples:

VOLT:LEV:IMM 16    Both colons are keyword separators.

:CURR:LEV:IMM 4    The first colon is the root specifier, the other two are keyword separators.

VOLT:LEV 6;:CURR:LEV 15  The second colon is the root specifier, the first and third are keyword separators

:INIT ON;:TRIG;:MEAS:CURR?;VOLT?  The first three colons are root specifiers.

4.4.8 MESSAGE TERMINATOR

The message terminator defines the end of a message. Three message terminators are permitted:

• new line (<NL>), ASCII 10 (decimal) or 0A (hex)
• (<CR>), ASCII 13 (decimal) or 0D (hex)
• both of the above (<CR> <NL>)

NOTE: Kepco power modules require a message terminator at the end of each program message. The examples shown in this manual assume a message terminator will be added at the end of each message. Where a message terminator is shown it is represented as <NL> regardless of the actual terminator character.

4.4.5 UNDERSTANDING THE COMMAND STRUCTURE

Understanding the command structure requires an understanding of the subsystem command tree illustrated in Figure 4-3. The “root” is located at the top left corner of the diagram. The parser goes to the root if:

• a message terminator is recognized by the parser
• a root specifier is recognized by the parser
Optional keywords are enclosed in brackets [ ] for identification; optional keywords can be omitted and the power supply will respond as if they were included in the message. The root level keyword [SOURce] is an optional keyword. Starting at the root, there are various branches or paths corresponding to the subsystems. The root keywords for the BIT Card are :INITiate, :MEASure, :OUTPut, [:SOURce], :STATus, and :SYSTem. Because the [SOURce] keyword is optional, the parser moves the path to the next level, so that VOLTage, CURRent, and FUNCTION commands are at the root level.

Each time the parser encounters a keyword separator, the parser moves to the next indented level of the tree diagram. As an example, the STATus branch is a root level branch that has three sub-branches: OPERation, PRESet, and QUESTionable. The following illustrates how SCPI code is interpreted by the parser:

STAT:PRES
The parser returns to the root due to the message terminator.

STAT:OPER?:PRES
The parser moves one level in from STAT. The next command is expected at the level defined by the colon in front of OPER?. Thus you can combine the following message units STAT:OPER? and STAT:PRES;

STAT:OPER:COND?:ENAB 16
After the OPER:COND? message unit, the parser moves in one level from OPER, allowing the abbreviated notation for STAT:OPER:ENAB.

4.4.6 PROGRAM MESSAGE SYNTAX SUMMARY

- Common commands begin with an asterisk (*).
- Queries end with a question mark (?).
- Program messages consist of a root keyword and, in some cases, one or more message units separated by a colon (:) followed by a message terminator. Several message units of a program message may be separated by a semicolon (;) without repeating the root keyword.
- If a program message has more than one message unit, then a colon (:) must precede the next keyword in order to set the parser back to the root (otherwise the next keyword will be taken as a subunit of the previous message unit).

  e.g., the command meas:volt?:curr? will read output voltage and output current since both volt? and curr? are interpreted as subunits of the meas command.

- Several commands may be sent as one message; a line feed terminates the message. Commands sent together are separated by a semicolon (;). The first command in a message starts at the root, therefor a colon (:) at the beginning is not mandatory.

  e.g., the command meas:volt?:;curr? will read output voltage and programmed current since the colon preceding curr? indicates that curr? is not part of the meas command and starts at the root.

- UPPER case letters in mnemonics are mandatory (short form). Lower case letters may either be omitted, or must be specified completely (long form)

  e.g., INStrument (long form) has the same effect as INST (short form).
• Commands/queries may be given in upper/lower case (long form) e.g., \texttt{SoUrCe} is allowed.

• Text shown between brackets [] is optional. e.g., \texttt{[:SOUR]}\texttt{VOLT[:LEV]} \texttt{TRIG} has the same effect as \texttt{:VOLT TRIG}

4.5 CIIL PROGRAMMING

The CIIL command language is used on early models of Kepco power supplies and controllers. The command functions are included here for compatibility with other equipment programmed with CIIL commands. The CIIL command set for the BIT Card is defined and explained in Appendix C.

4.6 PROGRAMMING EXAMPLES

Appendices D through G provide sample programs which can be altered and adapted by the user for specific applications, or used exactly as shown. All programs shown use SCPI commands; these can be adapted to use CIIL commands by changing the command strings to CIIL.

Appendix D is a C language program used with an IBM compatible computer running a terminal emulation program.

Appendix E is a C language program which automatically programs a series of commands

Appendix F is comprised of C language function prototypes used as building blocks for the programs in Appendices D and E. These function prototypes can also aid the user in creating new programs for unique applications.

Appendix G is a QuickBasic language program which executes some commands and also implements terminal emulation.
A.1 INTRODUCTION

This appendix defines the SCPI common commands and queries used with the BIT 232/BIT 232-F Interface Card. Common commands and queries are preceded by an asterisk (*) and are defined and explained in Figures A-1 through A-14, arranged in alphabetical order. Table A-1 provides a quick reference of all SCPI common commands and queries used in the BIT Card.

### TABLE A-1. SCPI COMMON COMMAND/QUERY INDEX

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>COMMAND</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>*CLS</td>
<td>A-1</td>
<td>*RST</td>
<td>A-5</td>
</tr>
<tr>
<td>*ESE</td>
<td>A-2</td>
<td>*SRE</td>
<td>A-6</td>
</tr>
<tr>
<td>*ESE?</td>
<td>A-2</td>
<td>*SRE?</td>
<td>A-6</td>
</tr>
<tr>
<td>*ESR?</td>
<td>A-3</td>
<td>*STB?</td>
<td>A-7</td>
</tr>
<tr>
<td>*IDN?</td>
<td>A-3</td>
<td>*TRG</td>
<td>A-7</td>
</tr>
<tr>
<td>*OPC</td>
<td>A-4</td>
<td>*TST</td>
<td>A-8</td>
</tr>
<tr>
<td>*OPC?</td>
<td>A-4</td>
<td>*WAI</td>
<td>A-8</td>
</tr>
</tbody>
</table>

### Syntax:
*CLS

### Function:
Clear status data

### Response:
Not Applicable

### Description:
Forces power supply to "operation complete idle" and "operation complete query" state. Clears all Event Registers summarized in Status Byte Register.

Clears Standard Event Status, Operation Status Event, Questionable Status Event, and Status Byte Registers. Clears the error queue.

### Example:
*CLS

Power supply clears status data.

FIGURE A-1. *CLS — CLEAR STATUS COMMAND
**ESE**

**Syntax:**

```
*ESE <integer>
```

*integer* = positive whole number: 0 to 255 per **STANDARD EVENT STATUS ENABLE REGISTER BITS** table below.

**Function:**

Sets ESE (standard Event Status Enable) register bits to enable the Standard events to be summarized in the Status Byte register (1 = set = enable function, 0 = reset = disable function).

**Response:**

Not applicable

**Description:**

Contents of Standard Event Status Enable register (*ESE) determine which bits of Standard Event Status register (*ESR) are enabled, allowing them to be summarized in the Status Byte register (*STB). All of the enabled events of the Standard Event Status Enable Register are logically ORed to cause ESB (bit 5) of the Status Byte Register to be set.

<table>
<thead>
<tr>
<th>CONDITION</th>
<th>NU</th>
<th>NU</th>
<th>CME</th>
<th>EXE</th>
<th>DDE</th>
<th>QUE</th>
<th>NU</th>
<th>OPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>VALUE</td>
<td>128</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

**Example:**

*ESE 49  Power supply enables bits 0, 4 and 5 allowing command error, execution error and operation complete conditions to be recorded in the Event Status Register.

**FIGURE A-2.  *ESE — STANDARD EVENT STATUS ENABLE COMMAND**

---

**ESE?**

**Syntax:**

```
*ESE?
```

**Response:**

*integer* value per **STANDARD EVENT STATUS ENABLE REGISTER BITS** table below.

**Function:**

Returns the status of the Standard Event Status Enable Register

**Description:**

Contents of Standard Event Status Enable register (*ESE) determine which bits of Standard Event Status register (*ESR) are enabled, allowing them to be summarized in the Status Byte register (*STB). All of the enabled events of the Standard Event Status Enable Register are logically ORed to cause ESB (bit 5) of the Status Byte Register to be set (1 = set = enable function, 0 = reset = disable function).

<table>
<thead>
<tr>
<th>CONDITION</th>
<th>NU</th>
<th>NU</th>
<th>CME</th>
<th>EXE</th>
<th>DDE</th>
<th>QUE</th>
<th>NU</th>
<th>OPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>VALUE</td>
<td>128</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

**Example:**

*ESE 49  Power Supply enables bits 0, 4 and 5
*ESE?  Controller reads <value> 49, verifying that bits 0, 4 and 5 have been enabled.

**FIGURE A-3.  *ESE? — STANDARD EVENT STATUS ENABLE QUERY**
**ESR?**

Syntax: *ESR?*

Response: <integer> value per STANDARD EVENT STATUS REGISTER BITS table below.

Function: This query reads the Standard Event Status Event register, clearing the register at the same time.

Description: The Standard Event Status Event register bit configuration is shown below (1 = set = enable function, 0 = reset = disable function):

<table>
<thead>
<tr>
<th>CONDITION</th>
<th>NU</th>
<th>NU</th>
<th>CME</th>
<th>EXE</th>
<th>DDE</th>
<th>QUE</th>
<th>NU</th>
<th>OPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>VALUE</td>
<td>128</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Example: *ESE 49*  
*ESR?*  
Power supply enables bits 0, 4 and 5  
Controller reads <value> 48 (bits 4 and 5 set), indicating Command Error and Execution error have occurred since the last time the register was read.

FIGURE A-4. *ESR? — EVENT STATUS REGISTER QUERY*

**IDN?**

Syntax: *IDN?*

Function: Identifies the instrument.

Response: Character string.

Description: Power Supply responds with model and version.

Example: *IDN?*  
Controller reads character string: “KEPCO BOP BIT 232 REV n”  
where n = applicable revision number.

FIGURE A-5. *IDN? — IDENTIFICATION QUERY*
**FIGURE A-6.  *OPC — OPERATION COMPLETE COMMAND**

**Syntax:**  
*OPC

**Function:**  
Causes power supply to set status bit 0 (Operation Complete) when pending operations are complete

**Description:**  
This command sets Standard Event Status Register bit 0 to “1” when all previous commands have been executed and changes in output level have been completed. This command does not prevent processing of subsequent commands, but bit 0 will not be set until all pending operations are completed. (1 = set = enable function, 0 = reset = disable function)

<table>
<thead>
<tr>
<th>CONDITION</th>
<th>NU</th>
<th>NU</th>
<th>CME</th>
<th>EXE</th>
<th>DDE</th>
<th>QUE</th>
<th>NU</th>
<th>OPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>VALUE</td>
<td>128</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

**Example:**  
Controller sends command(s), then sends *OPC
If controller then sends *ESR?, the power supply responds with either a “0” (if the power supply is busy executing the programmed commands), or a “1” (if the previously programmed commands are complete).

**FIGURE A-7.  *OPC? — OPERATION COMPLETE QUERY**

**Syntax:**  
*OPC?

**Function:**  
Indicates when pending operations have been completed.

**Response:**  
<1>

**Description:**  
When all pending operations are complete (all previous commands have been executed and changes in output level have been completed) a “1” is placed in the Output Queue. Subsequent commands are inhibited until the pending operations are completed. *OPC? is intended to be used at the end of a command line so that the application program can monitor the bus for data until it receives the “1” from the power supply Output Queue.

**Example:**  
Controller sends command(s), then sends *OPC?
Controller waits until power supply responds with “1” on bus, indicating previous commands are complete, then proceeds to execute subsequent commands.
**Syntax:** *RST

**Function:** Resets power supply as defined below:

**Response:** None

**Description:** Establishes the following power supply parameters:

- CURR[:LEV][:IMM] 0
- VOLT[:LEV][:IMM] 0
- FUNC:MODE VOLT

**Example:**

*RST

Power supply responds by establishing default states defined above.

---

**FIGURE A-8. ** *RST — RESET COMMAND**
**SRE**

Syntax:  
*SRE<integer>*  
<integer> = value from 0 - 255 per SERVICE REQUEST ENABLE REGISTER table below, except bit 6 cannot be programmed.

Function:  
Programs the Service Enable Register to determine which events of the Status Byte Register will cause the power supply to generate a service request.

Description:  
When a Service Request Enable Register bit is set ("1"), the corresponding Status Byte Register bit (set to "1") causes the RQS and MSS bits to be set (1 = set = enable function, 0 = reset = disable function). All enabled Service Request Enable Register bits are logically ORed to set bit 6 (MSS/RQS) of the Status byte register. Bit 6 is also set by the power supply to request service (RQS). (See also Figure 3-12, *STB?*

<table>
<thead>
<tr>
<th>SERVICE REQUEST ENABLE REGISTER BITS</th>
<th>MSS</th>
<th>RQS</th>
<th>ESB</th>
<th>MAV</th>
<th>NU</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>VALUE</td>
<td>128</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
</tr>
</tbody>
</table>

Example:  
*SRE112*  
Power supply will set bit 6 (MSS/RQS) if any event status register bit is set (ESB = 1), or any message is available (MAV = 1).

**FIGURE A-9.  *SRE — SERVICE REQUEST ENABLE COMMAND***

**SRE?**

Syntax:  
*SRE?*

Response:  
<integer> = value from 0 - 255 per SERVICE REQUEST ENABLE REGISTER table below.

Function:  
Reads the Service Enable Register to determine which events of the Status Byte Register are programmed to cause the power supply to generate a service request (1 = set = function enabled, 0 = reset = function disabled).

Description:  
See *SRE command* (Figure 3-11) and *STB? command* (Figure 3-13).

<table>
<thead>
<tr>
<th>SERVICE REQUEST ENABLE REGISTER BITS</th>
<th>MSS</th>
<th>RQS</th>
<th>ESB</th>
<th>MAV</th>
<th>NU</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>VALUE</td>
<td>128</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
</tr>
</tbody>
</table>

Example:  
*SRE?*  
Power supply responds with 32 to indicate that the power supply will request service if the Event Status Byte summary bit is set.

**FIGURE A-10.  *SRE? — SERVICE REQUEST ENABLE QUERY***
Syntax: *STB?
Response: <integer> value from 0 to 255 per table below
Function: Read Status Byte Register without clearing it
Description: This Query reads the Status Byte Register (bit 6 = MSS) without clearing it (1 = set = function enabled, 0 = reset = function disabled). The register is cleared only when subsequent action clears all set bits. MSS is set when the power supply has one or more reasons for requesting service. (A serial poll also reads the Status Byte Register, except that bit 6 = RQS, not MSS; ands RQS will be reset.)

<table>
<thead>
<tr>
<th>CONDITION</th>
<th>N#HI</th>
<th>MSS</th>
<th>RQS</th>
<th>ESB</th>
<th>MAV</th>
<th>NODE NUMBERS (ADDRESS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>VALUE</td>
<td>128</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

Example: *STB? Power supply responds with 96 (64 + 32) to indicate MSS and the Event Status Byte summary bit have been set.

**FIGURE A-11. *STB? — STATUS BYTE REGISTER QUERY**

Syntax: *TRG
Function: Triggers the power supply to be commanded to preprogrammed value of output current and voltage.
Description: This command causes the power supply to be commanded to the output voltage and current levels specified by VOLT:TRIG and CURR:TRIG commands, respectively (see Figures B-13 and B-9).

Example 1:
- VOLT 25 Power supply voltage commanded to 25V.
- VOLT:TRIG 15 Programs power supply voltage to 15V when *TRG received.
- INIT Trigger event is initialized.
- *TRG Power supply reverts to commanded output voltage of 15V.

**FIGURE A-12. *TRG — TRIGGER COMMAND**

### Syntax:

*TST?

### Response:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>pass test</td>
</tr>
<tr>
<td>1</td>
<td>fail test</td>
</tr>
</tbody>
</table>

### Function:

Power Supply test

### Description:

This query causes the power supply to do a self-test and provide the controller with pass/fail results.

**CAUTION: TO AVOID DAMAGE TO THE LOAD, DISCONNECT THE LOAD BEFORE ISSUING THIS COMMAND. (DURING THE SELF-TEST, THE BOP IS PROGRAMMED TO FULL SCALE POSITIVE AND FULL SCALE NEGATIVE OUTPUT.)**

### Example:

*TST?  
Power supply executes self test and responds with 0 if test completed successfully, with 1 if test failed.

**FIGURE A-14.  *WAI — WAIT-TO-CONTINUE COMMAND**

### Syntax:

*WAI

### Function:

Causes the power supply to wait until all previously issued commands and queries are complete before executing subsequent commands or queries.

### Description:

This command can be used to guarantee sequential execution of commands and queries. When all pending operations are complete (all previous commands have been executed, changes in output level have been completed), the WAI command is completed and execution of subsequent commands can continue.

### Example:

- Controller sends Command 1
- Controller sends *WAI
- Controller sends Command 2

Power supply begins execution of command 1.  
Power supply waits for command 1 to be completed before executing command 2.  
Command 2 executed after command 1 is completed.
This appendix defines the SCPI subsystem commands and queries used with the BIT 232/BIT 232-F Interface Card. Subsystem commands are defined in Figures B-1 through B-26, arranged in groups as they appear in the tree diagram, Figure 3-3. Table B-1 provides a quick reference of all SCPI subsystem commands and queries used in the BIT Card.

### TABLE B-1. SCPI SUBSYSTEM COMMAND/QUERY INDEX

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>COMMAND</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT[:IMM]</td>
<td>B-1</td>
<td>[SOUR]:FUNC:MODE</td>
<td>B-9</td>
</tr>
<tr>
<td>INIT:CONT</td>
<td>B-2</td>
<td>STAT:OPER:COND?</td>
<td>B-9</td>
</tr>
<tr>
<td>INIT:CONT?</td>
<td>B-2</td>
<td>STAT:OPER:ENAB</td>
<td>B-10</td>
</tr>
<tr>
<td>MEAS:VOLT?</td>
<td>B-3</td>
<td>STAT:OPER[EVENT]?</td>
<td>B-11</td>
</tr>
<tr>
<td>[SOUR]:CURR</td>
<td>B-5</td>
<td>STAT:PRES</td>
<td>B-11</td>
</tr>
<tr>
<td>[SOUR]:CURR?</td>
<td>B-5</td>
<td>STAT:QUES[EVENT]?</td>
<td>B-12</td>
</tr>
<tr>
<td>[SOUR]:CURR:TRIG</td>
<td>B-6</td>
<td>STAT:QUES:COND?</td>
<td>B-12</td>
</tr>
<tr>
<td>[SOUR]:VOLT</td>
<td>B-7</td>
<td>STAT:QUES:ENAB?</td>
<td>B-13</td>
</tr>
<tr>
<td>[SOUR]:VOLT?</td>
<td>B-7</td>
<td>SYST:ERR?</td>
<td>B-14</td>
</tr>
<tr>
<td>[SOUR]:VOLT:TRIG</td>
<td>B-8</td>
<td>SYST:LANG</td>
<td>B-14</td>
</tr>
<tr>
<td>[SOUR]:VOLT:TRIG?</td>
<td>B-8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Syntax:**

Short Form: INIT

Long Form: :INITiate[:IMMediate]

**Function:** Enables a single trigger.

**Description:** This command enables a single trigger. A *TRG command completes the sequence. Upon receipt of the *TRG command, the power supply will return to the commanded values of voltage and current established by the VOLT:TRIG and CURR:TRIG commands. After a *TRG command has been received, subsequent *TRG commands have no effect unless preceded by INIT or INIT:CONT ON.

**Example:**

- **VOLT 21; CURR 5**
  Power supply output commanded to go to 21V, 5A
- **VOLT:TRIG 15; CURR:TRIG 3**
  Power supply output programmed to return to 15V, 3A upon receipt of *TRG trigger.
- **INIT:*TRG**
  Upon receipt of *TRG command power supply is commanded to 15V, 3A.

**NOTE:** Commanded voltage and current parameters will either be output or limit parameters, depending on whether output is enabled, load and mode.

**FIGURE B-1. INITiate[:IMMediate] COMMAND**
**INIT:CONT**

**Syntax:**
- Short Form: `INIT:CONT <value>`
- Long Form: `:INITiate:CONTinuous <value>`

**Function:**
- `INIT:CONT ON`: Enables continuous triggers.
- `INIT:CONT OFF`: Disables continuous triggers.

**Description:**
This command enables/disables triggers. *TRG commands complete the sequence. Once `INIT:CONT ON` enables continuous triggers, subsequent *TRG commands return the power supply output to the commanded values of voltage and current established by the VOLT:TRIG and CURR:TRIG commands until `INIT:CONT OFF` is received.

**Example:**
- **VOLT 21; CURR 5**
  - Power supply output commanded to go to 21V, 5A
- **INIT:CONT ON**
  - Continuous triggers enabled.
- **VOLT:TRIG 15; CURR:TRIG 3**
  - Power supply output programmed to return to 15V, 3A upon receipt of trigger.
  - *TRG
    - Upon receipt of *TRG command power supply output returns to 15V, 3A.
- **VOLT 17; CURR 2**
  - Power supply output commanded to go to 17V, 2A
  - *TRG
    - Upon receipt of *TRG command power supply output returns to 15V, 3A.
- **INIT:CONT OFF**
  - Triggers disabled.

**FIGURE B-2. INITiate:CONTinuous COMMAND**

**INIT:CONT?**

**Syntax:**
- Short Form: `INIT:CONT?`
- Long Form: `:INITiate:CONTinuous?`

**Function:**
Determines whether continuous triggers are enabled or disabled.

**Response:**
- "1" = continuous triggers are enabled (INIT:CONT ON)
- "0" = continuous triggers disabled (INIT:CONT OFF)

**Description:**
Power supply returns value of INIT:CONT flag to controller

**Example:**
- **VOLT 21; CURR 5**
  - Power supply output commanded to go to 21V, 5A
  - Continuous triggers enabled.
- **INIT:CONT ON**
  - Power supply output programmed to return to 15V, 3A upon receipt of *TRG trigger.
  - *TRG
    - Power supply returns to 15V, 3A.
  - **INIT:CONT?**
    - Controller reads "1" to indicate that continuous triggers are enabled.

**FIGURE B-3. INITiate:CONTinuous QUERY**
**MEAS:CURR? MEAS:VOLT?**

**Syntax:**
- Short Form: MEAS:CURR?
- MEAS:VOLT?
- Long Form: MEASure:CURRent[:DC]?
  MEASure:VOLTage[:DC]?

**Function:**
Measures actual current or voltage

**Description:**
This query returns the actual value of output current or voltage (measured at the sense terminals) as determined by the commanded value of voltage and current and load conditions.

**Example:**
- VOLT 21; CURR 5
  
  Power supply output commanded to go to 21V, 5A
  
  MEAS:CURR?  
  Controller reads actual value of output current, e.g., 4.83A.
  
  MEAS:VOLT?  
  Controller reads actual value of output voltage, e.g., 20.9.

**FIGURE B-4. MEASURE:CURR? and MEASURE:VOLT? QUERIES**
**Syntax:**

Short Form:  

```
CURR <value>
```

Long Form:  

```
[SOURce]:CURRent[:LEVel][:IMMediate] <value>
```

**Function:**
Sets commanded current or current limit to specified level.

**Description:**
This command programs output current (Current mode) or current limit (Voltage mode) to a specific value. Actual output current will depend on load conditions.

**Example:**

```
VOLT 21; CURR 5
```

Power supply output commanded to go to 21V, 5A

---

**Syntax:**

Short Form:  

```
CURR?
```

Long Form:  

```
[SOURce]:CURRent?
```

**Function:**

- **CURR?**
  Returns programmed current value.
- **CURR? MAX**
  Returns maximum current allowed for power supply.
- **CURR? MIN**
  Returns minimum current allowed for power supply (always 0).

**Description:**
The CURR? query returns the programmed current (Current mode) or current limit (Voltage mode) of the power supply. Actual output current will depend on load conditions. The CURR?MAX query returns the maximum current allowed for a particular model.

**Example:**

```
VOLT 21; CURR 1.1
CURR?
CURR? MAX
```

Power supply commanded to go to 21V, 1.1A

Controller reads 1.1, indicating programmed current value = 1.1A

Controller reads 3.6 (assuming maximum allowable current for power supply being addressed is 3.6 A).
**CURR:TRIG**

**Syntax:**
- Short Form: CURR:TRIG <value>
- Long Form: [SOURce]:CURRent:TRIGgered <value>

**Function:** Programs current value to be implemented by *TRG command. Actual output current will depend on load conditions.

**Description:** This command can be used to reset many power supplies to preselected parameters by issuing a single *TRG command.

**Example:**
- VOLT 21; CURR 1.1: Power supply commanded to go to 21V, 1.1A
- CURR:TRIG 2.3: Power supply current programmed to 2.3A upon receipt of *TRG
- CURR? Controller reads 1.1, indicating programmed current value = 1.1A
- *TRG: Power supply commanded to current value established by CURR:TRIG command (2.3) and voltage value established by VOLT:TRIG
- CURR? Controller reads 2.3, indicating programmed current value = 2.3A

**FIGURE B-7. CURRent:TRIGgered COMMAND**

**CURR:TRIG?**

**Syntax:**
- Short Form: CURR:TRIG?
- Long Form: [SOURce]:CURRent:TRIGgered?

**Function:** Returns value to controller which represents current value to be programmed by *TRG command.

**Description:** This command returns the current value established by CURR:TRIG command.

**Example:**
- VOLT 21; CURR 1.1: Power supply commanded to go to 21V, 1.1A
- CURR:TRIG 2.3: Power supply current programmed to 2.3 upon receipt of *TRG
- CURR? Controller reads 1.1, indicating programmed current value = 1.1A
- CURR:TRIG? Controller reads 2.3 (current value established by CURR:TRIG command)

**FIGURE B-8. CURRent:TRIGgered QUERY**
**VOLT**

**Syntax:**
- Short Form: `VOLT <value>`
- Long Form: `[SOURce]:VOLTage[:LEVel][:IMMediate] <value>`

**Function:** Sets commanded voltage or voltage limit to specified level.

**Description:** This command programs commanded output voltage (Voltage mode) or voltage limit (Current mode) to a specific value. Actual output voltage will depend on load conditions.

**Example:**
- `VOLT 21; CURR 5` Power supply commanded to go to 21V, 5A

**FIGURE B-9. VOLTage COMMAND**

**VOLT?**

**Syntax:**
- Short Form: `VOLT?`
- Long Form: `[SOURce]:VOLTage?`

**Function:**
- `VOLT?` Returns programmed voltage value.
- `VOLT? MAX` Returns maximum voltage allowed for power supply.
- `VOLT? MIN` Returns minimum voltage allowed for power supply (always 0).

**Description:** The `VOLT?` query returns the programmed voltage (Voltage mode) or voltage limit (Current mode) of the power supply to the controller. Actual output voltage will depend on load conditions. The `VOLT?MAX` query returns the maximum voltage allowed for a particular model.

**Example:**
- `VOLT 21; CURR 1.1` Power supply commanded to go to 21V, 1.1A
- `VOLT?` Controller reads 21, indicating programmed voltage value = 21V
- `VOLT? MAX` Controller reads 100 (assuming maximum allowable voltage for power supply being addressed is 100V).

**FIGURE B-10. VOLTage QUERY**
VOLT:TRIG

Syntax:
- Short Form: VOLT:TRIG <value>
- Long Form: [SOURce]:VOLTage:TRIGgered <value>

Function: Programs voltage value to be implemented by *TRG command. Actual output voltage will depend on load conditions.

Description: This command can be used to reset many power supplies to preselected parameters by issuing a single *TRG command.

Example:
- VOLT 21; CURR 1.1  
  Power supply commanded to go to 21V, 1.1A
- VOLT:TRIG 29.3  
  Power supply current programmed to 29.3 upon receipt of *TRG
- VOLT?  
  Controller reads 21, indicating programmed current value = 21V
- *TRG  
  Power supply commanded to voltage value established by VOLT:TRIG command (29.3) and current value established by CURR:TRIG
- VOLT?  
  Controller reads 29.3, indicating programmed current value = 2.3A

FIGURE B-11. VOLTage:TRIGgered COMMAND

VOLT:TRIG?

Syntax:
- Short Form: VOLT:TRIG?
- Long Form: [SOURce]:VOLTage:TRIGgered?

Function: Returns value to controller representing voltage value to be programmed by *TRG command.

Description: This command returns the voltage value established by VOLT:TRIG command.

Example:
- VOLT 21; CURR 1.1  
  Power supply commanded to go to 21V, 1.1A
- VOLT:TRIG 29.3  
  Power supply voltage programmed to 29.3 upon receipt of *TRG
- VOLT?  
  Controller reads 21, indicating programmed voltage value = 21V
- *TRG  
  Controller reads 29.3 (voltage value established by VOLT:TRIG command)
- VOLT:TRIG?  
  Controller reads 29.3 (voltage value established by VOLT:TRIG command)

FIGURE B-12. VOLTage:TRIGgered QUERY
**FUNC:MODE**

**Syntax:**
- Short Form: `FUNC:MODE`
- Long Form: `[SOURce]:FUNCtion:MODE`

**Function:**
- `FUNC:MODE VOLT`: Commands power supply to Voltage mode
- `FUNC:MODE CURR`: Commands power supply to Current mode

**Description:**
Commanded mode establishes parameters (voltage or current) monitored for error conditions. Actual mode depends upon load conditions. When commanded to Voltage mode, if load conditions cause the power supply to try to exceed the current limit, the unit will automatically switch to Current mode and flag an error condition. When commanded to Current mode, if load conditions cause the power supply to try to exceed the voltage limit, the unit will automatically switch to Voltage mode and flag an error condition.

**Example:**
- `FUNC:MODE VOLT`: Power supply commanded to Voltage mode.
- `VOLT 21; CURR 1.1`: Power supply commanded to output 21V, current limit set to 1.1A.
- `FUNC:MODE CURR`: Power supply commanded to Current mode.
- `CURR 1.1; VOLT 21`: Power supply commanded to output 1.1A, voltage limit set to 21V.

**FIGURE B-13.  FUNCTION:MODE COMMAND**

---

**STAT:OPER:COND?**

**Syntax:**
- Short Form: `STAT:OPER:COND?`
- Long Form: `STATus:OPERation:CONDition?`

**Function:**
Returns the value of the Operation Condition Register to controller (1 = set = function enabled, 0 = reset = function disabled).

**Description:**
The Operation Condition Register contains unlatched real-time information about the operating conditions of the power supply.

**OPERATION CONDITION REGISTER BITS**

```
<table>
<thead>
<tr>
<th>CONDITION</th>
<th>CURRENT MODE</th>
<th>RELAY CLOSED</th>
<th>VOLTAGE MODE</th>
<th>NOT USED</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>1 0</td>
</tr>
<tr>
<td>VALUE</td>
<td>1024</td>
<td>512</td>
<td>256</td>
<td>1 0 1</td>
</tr>
</tbody>
</table>
```

**Example:**
- `STAT:OPER:COND?`: Power supply returns `<value> 1536 (1024 + 512)` to indicate that power supply is operating in Current mode with relay closed.

**FIGURE B-14.  STATus:OPERation:CONDition QUERY**
**STAT:OPER:ENAB**

**Syntax:**
- Short Form: STAT:OPER:ENAB <value>
- Long Form: STATus:OPERation:ENABle <value>

**Function:** Programs Operational Condition Enable Register

**Description:**
The Operation Condition Enable Register determines which conditions are allowed to set the Operation Condition Register. The value sent by the controller sets the corresponding bits of the Operation Condition Enable Register in the power supply (1 = set = function enabled, 0 = reset = function disabled).

**Example:**
STAT:OPER:ENAB 1280

Bits 8 and 10 of Operation Condition Enable Register are set, allowing the Operation Condition Register to monitor Voltage and Current mode conditions, but not report the status of the relay.

**FIGURE B-15. STATus:OPERation:ENABle COMMAND**

---

**STAT:OPER:ENAB?**

**Syntax:**
- Short Form: STAT:OPER:ENAB?
- Long Form: STATus:OPERation:ENABle?

**Function:** Reads Operational Condition Enable Register

**Description:** Power supply returns value of Operation Condition Enable Register to controller, indicating which conditions are being monitored. (1 = set = function enabled, 0 = reset = function disabled)

**Example:**
STAT:OPER:ENAB?

Controller reads <value> 1792 (1024 + 512 + 256) to indicate that bits 8, 9, and 10 of the Operation Condition Register are set, allowing Voltage mode, relay closed, and Current mode conditions to be monitored.

**FIGURE B-16. STATus:OPERation:ENABle QUERY**
**STAT:OPER?**

**Syntax:**
- Short Form: STAT:OPER[EVEN]?  
- Long Form: STATus:OPERation[EVENT]?  

**Function:** Indicates changes in conditions monitored by Operational Event Register.

**Description:** Power supply returns value to controller indicating conditions of Operation Event Register which have changed since the last STAT:OPER? query. This value is cleared once reported...

**OPERATION EVENT REGISTER BITS**

<table>
<thead>
<tr>
<th>CONDITION</th>
<th>CURRENT MODE</th>
<th>RELAY CLOSED</th>
<th>VOLTAGE MODE</th>
<th>NOT USED</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>VALUE</td>
<td>1024</td>
<td>512</td>
<td>256</td>
<td>128</td>
</tr>
</tbody>
</table>

**Example:** STAT:OPER?

Controller reads <value> 768 (512 + 256) to indicate that bits 8 and 9 are set, indicating that the power supply has changed to Voltage mode and the relay has closed since the last STAT:OPER? query.

**FIGURE B-17. STATus:OPERation QUERY**

---

**STAT:PRES**

**Syntax:**
- Short Form: STAT:PRES  
- Long Form: STATus:PRESet  

**Function:** Disables reporting of all status events.

**Description:** This command sets all bits of the Operation Condition and Questionable Condition Registers to 0, preventing all status events from being reported...

**Example:** STAT:PRES  

Operation Condition and Questionable Condition registers can no longer be set.

**FIGURE B-18. STATus:PRESet COMMAND**
Syntax: Short Form: STAT:QUES[EVEN]?  
Long Form: STATus:QUESTionable[EVEN]?  

Function: Indicates changes in conditions monitored by Questionable Event Register.

Description: Power supply returns value to controller indicating conditions of Questionable Event Register which have changed since the last STAT:OPER? query. This value is cleared once reported.

Example: STAT:OPER? Controller reads <value> 1026 (1024 + 2) to indicate that bits 1 and 10 are set, indicating that the power supply has detected overload and current error conditions since the last STAT:QUES? query.

**FIGURE B-19.** STATus:QUESTionable? QUERY

---

Syntax: Short Form: STAT:QUES:COND?  
Long Form: STATus:QUESTionable:CONDition?

Function: Returns the value of the Questionable Condition Register to controller. (1 = set = function enabled, 0 = reset = function disabled)

Description: The Questionable Condition Register contains unlatched real-time information about questionable conditions of the power supply.

Example: STAT:QUES:COND? Power supply returns <value> 1545 (1024 + 512 + 8 + 1) to indicate that overload, relay, overtemperature and voltage errors were detected.

**FIGURE B-20.** STATus:QUESTionable:CONDition? QUERY
STAT:QUES:ENAB

Syntax:
Short Form: STAT:QUES:ENAB <value>
Long Form: STATus:QUESionable:ENABLE <value>

Function: Programs Questionable Condition Enable Register

Description: The Questionable Condition Enable Register determines which conditions are allowed to set the Questionable Condition Register. The value sent by the controller sets the corresponding bits of the Questionable Condition Enable Register in the power supply (1 = set = function enabled, 0 = reset = function disabled).

Example: STAT:QUES:ENAB 3595
Bits 0, 1, 3, 9, 10, and 11 of the Questionable Condition Enable Register are set, causing the power supply to monitor voltage error, current error, overtemperature, relay error, overload and power loss conditions.

FIGURE B-21. STATus:QUEStionable:ENABle COMMAND

STAT:QUES:ENAB?

Syntax:
Short Form: STAT:QUES:ENAB?
Long Form: STATus:QUESionable:ENABLE?

Function: Reads Questionable Condition Enable Register

Description: Power supply returns value of Questionable Condition Enable Register to controller, indicating which conditions are being monitored.

Example: STAT:QUES:ENAB?
Controller reads <value> 3595 (2048 + 1024 + 512 + 8 + 2 + 1) to indicate that voltage error, current error, overtemperature, relay error, overload and power loss conditions are being monitored.

FIGURE B-22. STATus:QUEStionable:ENABle? QUERY
**SYST:ERR?**

**Syntax:**
- Short Form: SYST:ERR?
- Long Form: SYSTem:ERRor?

**Function:** Provides error messages to controller.

**Description:** Returns error number and character string containing error message. Error messages are defined as follows:

<table>
<thead>
<tr>
<th>ERROR MESSAGE</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 &quot;No error&quot;</td>
<td>No error</td>
</tr>
<tr>
<td>-100 &quot;Command error&quot;</td>
<td>Wrong syntax; command not understood.</td>
</tr>
<tr>
<td>-222 &quot;Voltage out of range&quot;</td>
<td>Value Exceeds Power Supply Rating</td>
</tr>
<tr>
<td>-222 &quot;Current out of range&quot;</td>
<td>Value exceeds power supply rating</td>
</tr>
<tr>
<td>-240 &quot;Hardware error&quot;</td>
<td>Power supply did not respond to command</td>
</tr>
</tbody>
</table>

**Example:** SYST:ERR? Controller reads: 0, "No error"

**FIGURE B-23.** SYSTem:ERRor? QUERY

---

**SYST:LANG**

**Syntax:**
- Short Form: SYST:LANG CIIL
- Long Form: SYSTem:LANGuage

**Function:** Selects the CIIL language command set (See Appendix C.)

**Description:** This command allows the CIIL command language to be used to program the power supply. (CIIL is included to provide compatibility with earlier Kepco equipment.) Once CIIL is selected, the CIIL command ‘GAL’ followed by the command ‘SCPI’ must be sent for the power supply to respond to SCPI commands.

**Example:** SYST:LANG CIIL Power supply responds to CIIL command set.

**FIGURE B-24.** SYSTem:LANGuage COMMAND
APPENDIX C - CIIL COMMAND DEFINITIONS

C.1 INTRODUCTION

This appendix defines the CIIL commands used with the BIT 232/BIT 232-F Interface Card. Table C-1 provides a quick reference of all CIIL commands used in the BIT Card.

TABLE C-1. CIIL SUBSYSTEM COMMAND/QUERY INDEX

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PAGE</th>
<th>COMMAND</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNF</td>
<td>C-4</td>
<td>RST</td>
<td>C-4</td>
</tr>
<tr>
<td>FNC</td>
<td>C-1</td>
<td>SET</td>
<td>C-3</td>
</tr>
<tr>
<td>FTH</td>
<td>C-2</td>
<td>SRN</td>
<td>C-3</td>
</tr>
<tr>
<td>GAL</td>
<td>C-6</td>
<td>SRX</td>
<td>C-3</td>
</tr>
<tr>
<td>INX</td>
<td>C-2</td>
<td>STA</td>
<td>C-5</td>
</tr>
<tr>
<td>IST</td>
<td>C-4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Syntax:
Stimulus mode: FNC DCS :CH1 <SET Command>
Sensor mode: FNC DCS <VOLT or CURR command> :CH1

Function: This operator is used with either the SET command to program a power supply's output (stimulus mode), or with the VOLT and CURR commands to read its output settings (sensor mode).

Description: 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.

Example:
FNC DCS :CH1 SET VOLT 15 Power supply commanded to 15V
FNC DCS :CH1 SET CURR 3 Power supply commanded to 3A
FNC DCS VOLT :CH1 Power supply returns value which represents actual output voltage
FNC DCS CURR :CH1 Power supply returns value which represents actual output current

NOTE: Actual output voltage and current depends on load conditions

FIGURE C-1. FNC — FUNCTION COMMAND
INX

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

Function:  
Commences a data acquisition process in accordance with the preceding FNC command.  

Description:  
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 supply'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 FTH command must also be repeated.  

Example:  
INX VOLT  
Power supply initiates voltage reading)  
FTH VOLT  
Power supply sends voltage reading to controller)  

FIGURE C-2.  INX — INITIATE OP CODE COMMAND

FTH

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

Function:  
Commands the previously designated power supply to return the requested data reading.  

Description:  
This command 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. The value observed will be in scientific notation.  

Example:  
INX VOLT  
Power supply initiates voltage reading)  
FTH VOLT  
Power supply sends voltage reading to controller)  

FIGURE C-3.  FTH — FETCH COMMAND
Syntax: FNC DCS :CH1 SET VOLT <value> CURL <value>
FNC DCS :CH1 SET CURR <value> VLTL <value>

Function: This operator is used in conjunction with FNC (in stimulus mode) to specify the output mode of the power supply being programmed.

Description: The first operand is the noun modifier and the second operand specifies the value. The first operand field of the command contains the four(4) letter mnemonic for the output mode of the power supply. 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 power supply allows. 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 supply 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 supply's polarity may be selected.

In the case of Kepco's BOP power supplies, the two 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.

Example: FNC DCS :CH1 SET VOLT 5 CURL 3  Power supply commanded to 5V (Voltage mode) with current limit of 3A.
FNC DCS :CH1 SET CURR 2 VLTL 17  Power supply commanded to 2A Current mode) with voltage limit of 17V

FIGURE C-4. SET COMMAND
**RST**

Syntax: \[ \text{RST DCS :CH1} \]

Function: This operator is used to return a power supply to its power-on state. The output voltage and current are programmed to zero.

Example: \[ \text{RST DCS :CH1} \] The power supply is reset.

**FIGURE C-5.  RST — RESET COMMAND**

**CNF, IST**

Syntax: \[ \text{CNF or IST} \]

Function: Causes power supply to execute confidence test.

Description: The CNF operator commands the BOP to execute the confidence test procedure defined for the BOP power supplies (IST is functionally identical to CNF for BOP power supplies). The procedure consists of programming voltage and current to their maximum values, checking for error flags, then programming voltage and current to zero. The results of CNF are obtained through the STA command.

Example: \[ \text{CNF} \] Power supply executes confidence test.
\[ \text{IST} \] Power supply executes self test.

**FIGURE C-6.  CNF, IST — CONFIDENCE TEST, INTERNAL SELF TEST COMMANDS**
STA

Syntax: STA

Function: Causes power supply to return operating status to controller.

Description: This operator commands the power supply to report its present operating status. Status is reported in the form of a message (character string) as defined below. Any catastrophic error conditions (indicated by * in the table below) which exist will be reported, until the error condition is corrected. 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.

<table>
<thead>
<tr>
<th>ERROR MESSAGE</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>F07 DCSnn DEV Power Loss</td>
<td>The power supply has lost its input power. *</td>
</tr>
<tr>
<td>F07 DCSnn DEV Device Turned Off (BOP)</td>
<td>A shutdown occurred due to overvoltage or overcurrent. *</td>
</tr>
<tr>
<td>F07 DCSnn DEV Over Temperature</td>
<td>A shutdown occurred due to thermal causes. *</td>
</tr>
<tr>
<td>F07 DCSnn DEV Overload</td>
<td>The voltage or current limit point was exceeded. *</td>
</tr>
<tr>
<td>F07 DCSnn DEV Voltage Fault</td>
<td>The output voltage is not within limits (voltage mode). *</td>
</tr>
<tr>
<td>F07 DCSnn DEV Current Fault</td>
<td>The output current is not within limits (current mode). *</td>
</tr>
<tr>
<td>F07 DCSnn DEV Load Path Fault</td>
<td>Open or miswired load or error sense leads detected. *</td>
</tr>
<tr>
<td>F07 DCSnn MOD Invalid Command</td>
<td>Improper syntax was used. **</td>
</tr>
<tr>
<td>F07 DCSnn DEV Not Ready</td>
<td>The output voltage or current has not settled. **</td>
</tr>
<tr>
<td>F07 DCSnn DEV Device Not Present</td>
<td>The specified power supply was not present during power up or during the last DCL. **</td>
</tr>
<tr>
<td>F07 DCSnn DEV Device Not Responding</td>
<td>The power supply has failed to communicate to the controller. **</td>
</tr>
<tr>
<td>F07 DCSnn DEV Invalid Voltage Range</td>
<td>The programmed voltage is outside the power supply's range. **</td>
</tr>
<tr>
<td>F07 DCSnn DEV Invalid Current Range</td>
<td>The programmed current is outside the power supply's range. **</td>
</tr>
<tr>
<td>F07 DCSnn DEV Set Modifier Error</td>
<td>An improper SET command was sent. **</td>
</tr>
<tr>
<td>F07 DCSnn DEV Invalid Device ID</td>
<td>The selected channel was not between 1-31. **</td>
</tr>
</tbody>
</table>

* Catastrophic error
** Non-Catastrophic error

FIGURE C-7. STA — STATUS COMMAND
Syntax:  
GAL

Function: Enables utility commands which change error handling defaults.

Description: This command enables the utility commands listed below. If no GAL command is issued, the default conditions are T0, F1, and P1. Once the GAL command is issued, the appropriate utility command may be sent to change the default condition.

TABLE C-3. CIIL ERROR HANDLING UTILITY COMMANDS

<table>
<thead>
<tr>
<th>UTILITY COMMAND</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>Instructs non-catastrophic error messages to be erased from memory if any command is sent prior to STA command.</td>
</tr>
<tr>
<td>T1</td>
<td>Instructs non-catastrophic error messages to be stacked in memory until STA command is sent.</td>
</tr>
<tr>
<td>F0</td>
<td>Fetch Mode 0. Ignores error conditions when performing FTH command.</td>
</tr>
<tr>
<td>F1</td>
<td>Fetch Mode 1. Reports any error conditions which are present during FTH command.</td>
</tr>
<tr>
<td>P0</td>
<td>Power Loss Mode 0. Reports a power loss message only once until power is restored to the power module.</td>
</tr>
<tr>
<td>P1</td>
<td>Power Loss Mode 1. Continuously reports a power loss message until power is restored to the power module.</td>
</tr>
</tbody>
</table>

Note: The defaults are T0, F1 and P1

Example:  
GAL Enables utility commands.
F0 Causes controller to ignore error conditions during FTH command.

FIGURE C-8. GAL — GO TO ALTERNATE LANGUAGE COMMAND
APPENDIX D - TERMINAL EMULATION PROGRAM

D.1 INTRODUCTION

Appendix D is a C language program used with an IBM-PC-compatible computer as a terminal emulator, allowing the BOP to be controlled directly from a keyboard. Refer to Appendix F for all functions. (See PAR. 1.4 to order Sample Programs diskette.)

To modify this program for computers other than IBM-PC-compatible (e.g., Macintosh), four ROM BIOS routines must be replaced with their equivalent (see Table D-1).

TABLE D-1. ROM BIOS ROUTINES TO BE REPLACED WITH EQUIVALENT FOR NON-IBM-PC-COMPATIBLE PC’S

<table>
<thead>
<tr>
<th>ROM BIOS ROUTINE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>open_port(n) n=0 for COM 1 or n=1 for COM 2</td>
<td>Initialize the serial port.</td>
</tr>
<tr>
<td>put_serc(n,c)</td>
<td>Write the character c to COM 1 or COM 2 serial port.</td>
</tr>
<tr>
<td>get_serc(n)</td>
<td>Read a character from the serial port.</td>
</tr>
<tr>
<td>in_ready(n)</td>
<td>Fetch the current status of the serial port.</td>
</tr>
</tbody>
</table>

The Terminal Emulation program is a loop in which typed characters from the computer keyboard are sent to the BIT Card using the SendWecho function (see Appendix F). This allows the BOP to be exercised by typing the appropriate commands (SCPI or CIIIL) directly from the keyboard. Responses to the typed commands are received from the BIT Card and displayed on the computer monitor. Typing ALT+Q ends the program.

The Terminal Emulation program can be set to run on either COM 1 or COM 2. For convenience it is recommended that two separate programs be created, one for COM 1, one for COM 2.

If this program is started before power is applied to the BOP, the sign on message will be displayed on the computer monitor when the BOP is powered up:

KEPCO BOP BIT232 REV. 1.0
POWER SUPPLY Type = XX (BOP YY-Y)

where

XX = hex value representing setting of Power Supply Identifications switch S2 (see PAR. 2.2.2)

YY-Y = BOP Model No. (e.g., 20-5)

NOTE: If switch S2 is set to an illegal setting, the message will read:
UNDEFINED BOP TYPE
/********************************************************************
* RS232 interactive driver for BIT232
* use ROM BIOS INT 14 function calls
********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#define TRUE 1
#define FALSE 0
#define PORT 2     /* serial port */
/* function prototypes */
void open_port(int n);
void put_serc(int n, int c);
int get_serc(int n);
int in_ready(int n);
int SendWecho(int n, int c);
void SendCommand(char *send_cmmd);
void SendCmmdWready(char *send_cmmd);
void main(void)
{
    int c;
    clrscr();
    printf(" ** BIT232 Interactive Driver Program - Press ALT + Q to exit \n");
    /* open the serial port */
    open_port(PORT);
    /* main program loop */
    while (TRUE) {
        /* process keyboard presses */
        if (kbhit()) {
            c = getch();
            switch (c) {
                case 0: /* exit on Alt-X */
                    if (getch() == 16)
                        exit(0);
                    break;
                default:
                    SendWecho(PORT, c);
                    /* display it if one was available */
                    if (c != EOF)
                        putch(c);
                    if (c == '\r')
                        putch('\n');
            }
        }
        /* process remote characters */
        if (in_ready(PORT)) {
            /* get character from serial port*/
            c = get_serc(PORT);
            /* display it if one was available */
            if (c != EOF)
                putch(c);
        }
    }
}
APPENDIX E - RS232 COMMAND LOOP PROGRAM

E.1 INTRODUCTION

This appendix is a C language program used with an IBM-PC-compatible computer running a terminal emulation program to set up a command loop. Refer to Appendix F for all function prototypes. (See PAR. 1.4 to order Sample Programs diskette.)

To modify this program for computers other that IBM-PC-compatible (e.g., Macintosh), four ROM BIOS routines must be replaced with their equivalent (see Table E-1).

**TABLE E-1. ROM BIOS ROUTINES TO BE REPLACED WITH EQUIVALENT FOR NON-IBM-PC-COMPATIBLE PC’S**

<table>
<thead>
<tr>
<th>ROM BIOS ROUTINE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>open_port(n) n=0 for COM1 or n=1 for COM 2</td>
<td>Initialize the serial port.</td>
</tr>
<tr>
<td>put_serc(n,c)</td>
<td>Write the character c to COM 1 or COM 2 serial port.</td>
</tr>
<tr>
<td>get_serc(n)</td>
<td>Read a character from the serial port.</td>
</tr>
<tr>
<td>in_ready(n)</td>
<td>Fetch the current status of the serial port.</td>
</tr>
</tbody>
</table>

The Command Loop program performs the following functions

1. Interrogates the power supply (SCPI command VOLT?MAX) to determine the maximum voltage of the BOP (XXX).

2. Sets the voltage to full scale, current to 0.5A (VOLT XXX;CURR 0.5).

3. Reads the voltage and current (MEAS:VOLT?CURR?) and displays it on the computer monitor.

4. Sets voltage to minus full scale, current to 0.5A (VOLT –XXX;CURR 0.5).

5. Reads the voltage and current (MEAS:VOLT?CURR?) and displays it on the monitor.


7. Reads the voltage and current (MEAS:VOLT?CURR?) and displays it on the monitor.

8. Sends a test command (*TST?).

9. Reads the voltage and current (MEAS:VOLT?CURR?) and displays it on the monitor.

10.Checks for errors (SYST:ERR?); the message: 0. “No Error” is displayed on the monitor.

11. The program continuously cycles through steps 1 through 10 above; to exit the program press ALT+Q.

This program can be set to run on either COM 1 or COM 2; for convenience it is recommended that the user create two separate programs, one for COM1, one for COM 2.
/***************************************************************
* RS232 command loop program for BIT232
* use ROM BIOS  INT 14 function calls
***************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#define TRUE 1
#define FALSE 0
#define PORT 2     /* serial port */

/* function prototypes */
void open_port(int n);
void put_serc(int n, int c);
int get_serc(int n);
int in_ready(int n);
int SendWecho(int n, int c);
void SendCommand(char *send_cmmd);
void SendCmmdWready(char *send_cmmd);

void main(void)
{
    char string[80], stringm[80], wstr[80];
    char volt[4], *curr = ";curr 0.5";
    char *test = "*tst?";
    char *reset = "*rst";
    char *mvolt = "meas:volt?;curr?";
    char *sterror = "syst:err?";
    char *id = "*idn?";
    int j,k,c,vl;
    int on_line;
    double max;
    clrscr();
    printf(" ** BIT232 Command Loop Program - Press ALT + Q to exit \n");
    /* open the serial port */
    open_port(PORT);
    /* get power supply id */
    strcpy(string, id);
    SendCommand(string);
    on_line = FALSE;
    for (k=0; k < 10000; k++)
    {
        if (in_ready(PORT))
        {
            /* get character from serial port*/
            c = get_serc(PORT);
            /* display it if one was available */
            if (c != EOF)
            {
                putch(c);
                on_line = TRUE;
            }
        }
    }
    if (on_line == FALSE)
        exit(0);
    /* change the first \r in end of string */
    for (j = 0; j < strlen(string); j++)...
if (string[j] == \r)
    string[j] = \0;
}

/* get maximum voltage */
strcpy(string, "volt?max");
SendCmdWithReady(string);
/* change the first \r in end of string */
for (j = 0; j < strlen(string); j++)
    if (string[j] == \r)
        string[j] = \0;

max = atof(string);
if (max <= 10.0)
    vl=1;
else
    {
        if (max <= 100.0)
            vl=2;
        else
            vl=3;
    }

j=0;
volt[0]=string[j];
volt[1]=\0;
j ++;
if (vl > 1)
    if( string[j] == \. )
        {
            j ++;
            volt[1]=string[j];
            volt[2]=\0;
        }
    else
        {
            volt[1]=string[j];
            volt[2]=\0;
        }
if (vl > 2)
    if( string[j] == \. )
        {
            j ++;
            volt[2]=string[j];
            volt[3]=\0;
        }
    else
        {
            volt[2]=string[j];
            volt[3]=\0;
        }
strcpy(string, "volt ");
strcat(string, volt);
strcat(string, curr);
strcpy(stringm, "volt -");
strcat(stringm, volt);
strcat(stringm, curr);

/*/ main program loop */
while (TRUE) {
    /* process keyboard presses */
    if (kbhit()) {
    }
c = getch();
if (c == 0)
    /* exit on Alt-Q */
    if (getch() == 16)
        exit(0);
}
strcpy(wstr,string);
/* set voltage to full scale, current 0.5A */
SendCmdWready(wstr);
for (j = 0; j < 10000; j++)
    for (k = 0; k < 100; k++)
strcpy(wstr,msvolt);
/* read voltage and current */
SendCmdWready(wstr);
for (j = 0; j < 10000; j++)
strcpy(wstr,stringm);
/* set voltage to minus full scale, current 0.5A */
SendCmdWready(wstr);
for (j = 0; j < 10000; j++)
    for (k = 0; k < 100; k++)
strcpy(wstr,msvolt);
/* read voltage and current */
SendCmdWready(wstr);
for (j = 0; j < 10000; j++)
strcpy(wstr,reset);
/* send reset command - *rst */
SendCmdWready(wstr);
for (j = 0; j < 10000; j++)
    for (k = 0; k < 100; k++)
strcpy(wstr,msvolt);
/* read voltage and current */
SendCmdWready(wstr);
for (j = 0; j < 10000; j++)
strcpy(wstr,test);
/* send test command - *tst? */
SendCmdWready(wstr);
for (j = 0; j < 10000; j++)
    for (k = 0; k < 100; k++)
strcpy(wstr,msvolt);
/* read voltage and current */
SendCmdWready(wstr);
for (j = 0; j < 10000; j++)
strcpy(wstr,sterror);
/* read error output queue */
SendCmdWready(wstr);
for (j = 0; j < 10000; j++)
}
APPENDIX F - C LANGUAGE FUNCTIONS

F.1 INTRODUCTION

This appendix contains examples of C language functions which can be used as the building blocks for unique programs to interface the BIT Card and associated power supply with an IBM-PC-compatible computer. These functions are used in the sample programs shown in Appendices D and E.

F.2 OPEN SERIAL PORT USING ROM BIOS ROUTINE

This function opens the serial port selected by n, using the ROM BIOS routine; if the computer is not IBM-PC-compatible the ROM BIOS routine must be replaced by an equivalent.

/* open serial port using ROM BIOS routine */
void open_port(int n)
{
    union REGS regs;

    /* AH has the function code */
    regs.h.ah = 0x00;
    /* AL has the baud rate for 8 bits No parity 1 Stop bit */
    regs.h.al = 0xe3;
    /* DX has the port number */
    if (n == 1)
        regs.x.dx = 0;
    else
        regs.x.dx = 1;
    /* call the ROM BIOS routine */
    int86(0x14, &regs, &regs);
}

F.3 PUT CHARACTER TO SERIAL PORT

This function sends a character (c) to the serial port specified by (n) using the ROM BIOS routine; if the computer is not IBM-PC-compatible the ROM BIOS routine must be replaced by an equivalent.

/* put character to serial port */
void put_serc(int n, int c)
{
    union REGS regs;

    /* AH has the function code */
    regs.h.ah = 0x01;
    /* AL has the character */
    regs.h.al = c;
    /* DX has the port number */
    if (n == 1)
        regs.x.dx = 0;
    else
        regs.x.dx = 1;
    /* call the ROM BIOS routine */
    int86(0x14, &regs, &regs);
}
F.4 SEND, WAIT FOR ECHO

This function sends a character to the serial port specified by (n), allows a small delay (about 50 msec) for the character to be received by the BIT card, analyzed, put in a buffer and echoed back. If the BIT Card did not receive the character, or if the time-out expires, the BIT card returns the EOF character.

This command could be modified so that if the time-out expires, the last character is sent to the BIT Card again.

```c
int SendWecho(int n, int c)
{
    int j;
    /* send to the serial port */
    put_serc(n, c);

    /* timeout delay = 1000 - bigger for faster computers */
    for(j=1; j<1000; j++)
        if (in_ready(PORT)) {
            /* get character from serial port*/
            c = get_serc(PORT);
            return c;
        }
    /* if timeout - no character returned */
    c = EOF;
    return c;
}
```

F.5 GET CHARACTER FROM SERIAL PORT

This function accepts a character from the serial port specified by n using the ROM BIOS routine; if the computer is not IBM-PC-compatible the ROM BIOS routine must be replaced by an equivalent.

```c
/* get character from serial port */
int get_serc(int n)
{
    union REGS regs;

    /* AH has the function code */
    regs.h.ah = 0x02;
    /* DX has the port number */
    if (n == 1)
        regs.x.dx = 0;
    else
        regs.x.dx = 1;
    /* call the ROM BIOS routine */
    int86(0x14, &regs, &regs);

    /* return EOF if timed out */
    if (regs.h.ah & 0x80)
        return EOF;
    return regs.h.al;
}
```
F.6 CHECK TO SEE IF A CHARACTER WAS RECEIVED

This function checks to see if a character was received using the ROM BIOS routine; if the computer is not IBM-PC-compatible the ROM BIOS routine must be replaced by an equivalent.

```c
/* check to see if a character was received */
int in_ready(int n)
{
    union REGS regs;

    /* AH has the function code */
    regs.h.ah = 0x03;
    /* DX has the port number */
    if (n == 1)
        regs.x.dx = 0;
    else
        regs.x.dx = 1;
    /* call the ROM BIOS routine */
    int86(0x14, &regs, &regs);

    /* check for received data ready */
    if (regs.h.ah & 1)
        return TRUE;
    return FALSE;
}
```

F.7 SEND COMMAND

This function sends a character string to the BIT Card and sends the echoed characters to the monitor screen. The SendWecho function is used to send the command, byte by byte, to the BIT Card. After the last byte of the command is sent, the Carriage Return (CR) character is sent to the BIT Card to cause the parser to execute the command, and CR and Line Feed (LF) characters are sent to the monitor screen.

This function could be modified by comparing the character sent with the echoed character, then if they do not match, send the last character again.

```c
void SendCommand(char *send_cmmd)
{
    int chr;
    int (*send_cmmd != '\0') {
        /* output character here */
        chr = *send_cmmd ++;
        SendWecho(PORT,chr);
        /* display returned byte if available */
        if (chr != EOF)
            putch(chr);
    }

    /* output end command */
    chr = 'r';
    SendWecho(PORT,chr);
    /* display returned byte if available */
    if (chr != EOF)
        putch(chr);
    if (chr != '\n')
        putch('n');
}
**F.8  SEND COMMAND AND WAIT UNTIL READY**

This function uses the SendCommand function to send commands to the BIT Card and display echoed characters and response messages on the screen until the > character is received, indicating that parsing and execution of the command has been completed.

```c
void SendCmmdWready(char *send_cmmd)
{
    int not_endcmd;
    char recv, *st_buff;

    not_endcmd = TRUE;
    st_buff = send_cmmd;
    SendCommand(send_cmmd);
    send_cmmd = st_buff;
    while(not_endcmd)
    {
        if (in_ready(PORT)) {
            /* get character from serial port*/
            recv = get_serc(PORT);
            /* display it if one was available */
            if (recv != EOF)
            {  
                putch(recv);
                *send_cmmd = recv;
                send_cmmd ++;
                if (recv == '>')
                {
                    not_endcmd = FALSE;
                }
            }
        }
    }
    *send_cmmd = '\0';
}
```
APPENDIX G - BASIC LANGUAGE TERMINAL EMULATION

G.1 INTRODUCTION

This appendix presents a program written in BASIC language intended to aid the user in building BASIC language test programs using the BIT Card. (See PAR. 1.4 to order Sample Programs diskette.)

The BASIC Language Terminal Emulation program performs a series of commands, then allows the user to type commands and control a BOP via the BIT Card from a keyboard. The keyboard commands are displayed on the screen while the commands are executed and response messages are displayed on the screen.

SCPI commands are used, however the program may easily be modified to use CIIL commands by changing the applicable character strings.

The program is written to use COM port 2; to use COM port 1 change the following line:

```
OPEN "COM2:9600,N,8,1,RS,CS,DS" FOR RANDOM AS #1 LEN = 256
```

to:

```
OPEN "COM1:9600,N,8,1,RS,CS,DS" FOR RANDOM AS #1 LEN = 256
```

With the program loaded under a BASIC interpreter, press ALT+R to run the program or ALT+Q to exit.

The program performs the following functions

1. Request power supply identification (*IDN?).
2. Reset the power supply (*RST).
3. Test the power supply (*TST).
4. Check for errors (SYST:ERR)
5. Wait for keyboard commands. When keyboard commands are entered, the echoed commands and response messages are displayed.

```
DEFINT A-Z
DECLARE SUB SendWecho (SendChar$)
DECLARE SUB SendCommand (SendCmd$)
DECLARE SUB SendCmdWready (SendCmd$)
DECLARE SUB Filter (InString$)
CONST FALSE = 0, TRUE = NOT FALSE
COLOR 7, 1                      ' Set screen color.
CLS
Quit$ = CHR$(0) + CHR$(16)      ' Value returned by INKEY$      
   ' when ALT+q is pressed.

' Set up prompt on bottom line of screen and turn cursor on:
LOCATE 24, 1, 1
PRINT STRING$(80, "_"));
LOCATE 25, 1
PRINT TAB(30); "Press ALT+q to quit";
```
' Open communications (9600 baud, no parity, 8-bit data, 1 stop bit, 256-byte input buffer):
OPEN "COM2:9600,N,8,1,RS,CS,DS" FOR RANDOM AS #1 LEN = 256

' Check the RS 232 interface if characters are waiting (EOF(1) is true) and get them
IF NOT EOF(1) THEN
  LOC(1) gives the number of characters waiting:
  SendChar$ = INPUT$(LOC(1), #1)
END IF

ComString$ = "*idn?"
SendCmdWready ComString$

ComString$ = "*rst"
SendCmdWready ComString$

ComString$ = "*tst?"
SendCmdWready ComString$

ComString$ = "syst:err?"
SendCmdWready ComString$

DO ' Main communications loop.
  KeyInput$ = INKEY$ ' Check the keyboard.
  IF KeyInput$ = Quit$ THEN ' Exit the loop if the user pressed ALT+q.
    EXIT DO
  ELSEIF KeyInput$ <> "" THEN ' Otherwise, if the user has pressed a key
    SendWecho KeyInput$ ' character typed to the modem.
    PRINT KeyInput$; ' then print.
  END IF

  ' Check the modem. If characters are waiting (EOF(1) is true), get them and print them to the screen:
  IF NOT EOF(1) THEN
    LOC(1) gives the number of characters waiting:
    CharInput$ = INPUT$(LOC(1), #1)
    Filter CharInput$ ' Filter out line feeds and backspaces, then print.
    PRINT CharInput$;
  END IF

LOOP

CLOSE ' End communications.
CLS
END
' ========================= FILTER =========================
' Filters characters in an input string.
' ==========================================================================
'
SUB Filter (InString$) STATIC

' Look for backspace characters and recode them to
' CHR$(29) (the LEFT cursor key):
DO
  BackSpace = INSTR(InString$, CHR$(8))
  IF BackSpace THEN
    MID$(InString$, BackSpace) = CHR$(29)
  END IF
LOOP WHILE BackSpace

' Look for line-feed characters and remove any found:
DO
  LineFeed = INSTR(InString$, CHR$(10))
  IF LineFeed THEN
    InString$ = LEFT$(InString$, LineFeed - 1) + MID$(InString$, LineFeed + 1)
  END IF
LOOP WHILE LineFeed

END SUB
'
' ========================= Send Command =====================
' Send command line and wait for the command to be completed
' ==========================================================================
'
SUB SendCmdWready (SendCmd$) STATIC

SendCommand SendCmd$

Notendcm = TRUE

WHILE Notendcm
  ' Check the modem. If characters are waiting (EOF(1) is
  ' true), get them and print them to the screen:
  IF NOT EOF(1) THEN
    ' LOC(1) gives the number of characters waiting:
    CharInput$ = INPUT$(LOC(1), #1)
    IF CharInput$ = CHR$(62) THEN
      Notendcm = FALSE
    END IF
    Filter CharInput$
    PRINT CharInput$;        ' backspaces, then print.
  END IF
WEND
END SUB
'
' ========================= Send Command =====================
' Send command line using SendWecho
' ==================================================================
'
SUB SendCommand (SendCmd$) STATIC
FOR K = 1 TO LEN(SendCmd$)
    Istr$ = MID$(SendCmd$, K, 1)
    SendWecho Istr$
    PRINT Istr$; ' then print.
NEXT K

Istr$ = CHR$(13)
SendWecho Istr$
PRINT Istr$
END SUB
'
'
' ========================= Send with Echo ===================
' Send character and wait for the character to be echoed back
' ==================================================================
'
SUB SendWecho (SendChar$) STATIC
    PRINT #1, SendChar$; ' send the pressed key
FOR J = 1 TO 100
    ' Check the RS 232 interface  if characters are waiting (EOF(1) is
    ' true), get them and print them to the screen:
    IF NOT EOF(1) THEN
        SendChar$ = INPUT$(1, #1)
        Filter SendChar$ ' Filter out backspace
        EXIT FOR
    END IF
    ' If timeout expire return empty string
    SendChar$ = ""
NEXT J

END SUB