6.2 Troubleshooting

Condition	Possible Cause	Action
Does not run application software	Software problem (application software)	Contact software vendor to verify: • that software is IBM PC/AT-compatible. • that software does not require special "keys" to operate. If required, install special hardware or software as specified by vendor.
	Hardware failure (module)	Run module diagnostic program.
Does not communicate with PLC over I/O bus	Module not properly seated in base	Check that module is properly installed in base.
	Software problem (using TISOFT)	Verify that 386ATM.EXE is installed in the CONFIG.SYS. Start TISOFT by entering: TI505 CVU (if you are communicating via the I/O bus); or TI505 (if you are communicating via serial port 1); or TI505 P2 (if you are communicating via serial port 2).
	Software problem (using application program)	Refer to manual for application program. Check operating instructions. Verify that 386ATM.EXE is installed in the CONFIG.SYS. Verify that third-party I/O bus driver software (if used) will work. Contact software vendor. Verify that PCREAD, PCWRITE, IOREAD, and IOWRITE are properly formatted and have proper syntax in the application software. Refer to Chapter 5 and Appendix C.
Does not communicate through serial ports	Cabling problem	Check connections and cabling.
	Incompatible communication interface	Check interface. 386/ATM is DTE; devices attached to serial ports must be DCE, or must use appropriate crossover (e.g., null modem cable).
	Software problem (application program)	Verify software by running serial application on another machine/module.
	Hardware problem	Run module diagnostic program, using loop-back connector to check serial ports.
Does not communicate through parallel port	Cabling problem	Check connections and cabling.
	Printer problem	Check that printer is set for parallel communication. Verify printer operation.
	Hardware problem	Run module diagnostic program.
Video output not operating properly	Module not set correctly (switch 4)	Verify switch 4 is on. Refer to Chapter 2.
	Monitor not set correctly	If monitor requires setting switches for EGA/VGA or TTL/analog operation, verify that switches are set to VGA or analog.
	Interconnecting cable miswired or damaged	Verify wiring; see Appendix C.

Condition	Possible Cause	Action
Keyboard not operating properly	Keyboard failure	Replace keyboard.
Floppy disk drive does not work	Disk not set up properly	Run **SETUP** procedure and verify diskette drive is set up properly.
	Hardware problem	Run module diagnostics program.
Hard disk drive does not work	Disk not set up properly	Run **SETUP** procedure and verify hard drive is set up properly.
	Hardware problem	Run module diagnostics program.
Real time clock and disk setup data are lost after PLC power cycle	**SETUP** data not correctly entered and saved.	Verify **SETUP** procedures.
	Battery problem	Verify that module battery switch is on. Refer to Chapter 2. Replace module battery.
Seek and/or read/write errors occur during diskette access	Disk access during periods of high conducted or radiated electrical noise conditions	Use the diskette for startup, then operate with the hard drive.

Appendix A

387SX Math Coprocessor

A.1 Installing the 387SX Math Coprocessor A-2
Procedure A-3

A.1 Installing the 387SX Math Coprocessor

To enhance processing, the 386/ATM includes a socket for an optional CMOS 80C387SX math coprocessor (16 MHz or faster). See Figure A-1. Contact your local computer store to purchase a 387SX coprocessor.

Intel manufactures a 387SX coprocessor (Intel part number BOX387SX–16). Equivalent math coprocessor chips are available from other vendors. You can use one of these coprocessor chips, provided they are equivalent to the Intel coprocessor.

⚠CAUTION

You must install the 387 coprocessor correctly. To help avoid damage to the 386/ATM or to the 387 coprocessor, refer to installation instructions provided with the coprocessor.

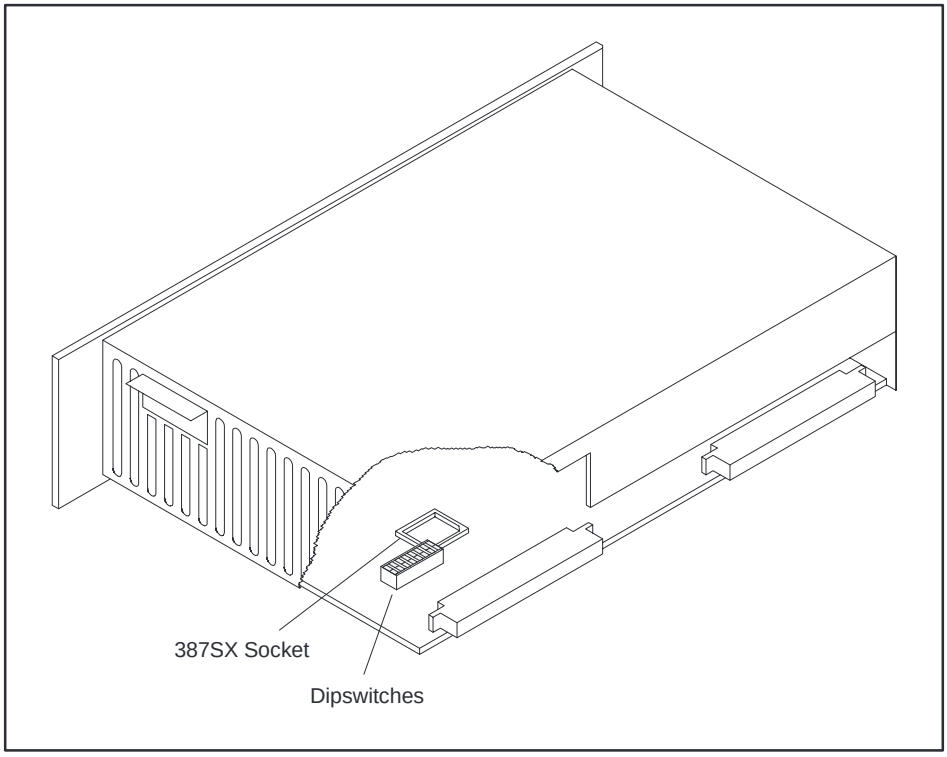

Figure A-1 387SX Socket Location

Procedure

Use the following procedure to install a 387 coprocessor in your 386/ATM. Refer to the installation instructions that accompany the 387SX coprocessor.

⚠ CAUTION

Both the 386/ATM and the 387 coprocessor can be damaged by electrostatic discharge. To help avoid potential damage, ground yourself and the 386/ATM before handling and installing the 387 coprocessor.

1. Place the 386/ATM on a flat surface, oriented as in Figure A-1 (with the printed circuit board down).
2. Orient the 387 coprocessor chip to correspond to the socket. See Figure A-2.
3. Refer to the installation instructions that accompany the 387SX coprocessor. Seat the 387 coprocessor into the socket by pressing firmly and evenly. Be careful that the pins are not bent or damaged and that the printed circuit board is not flexed.

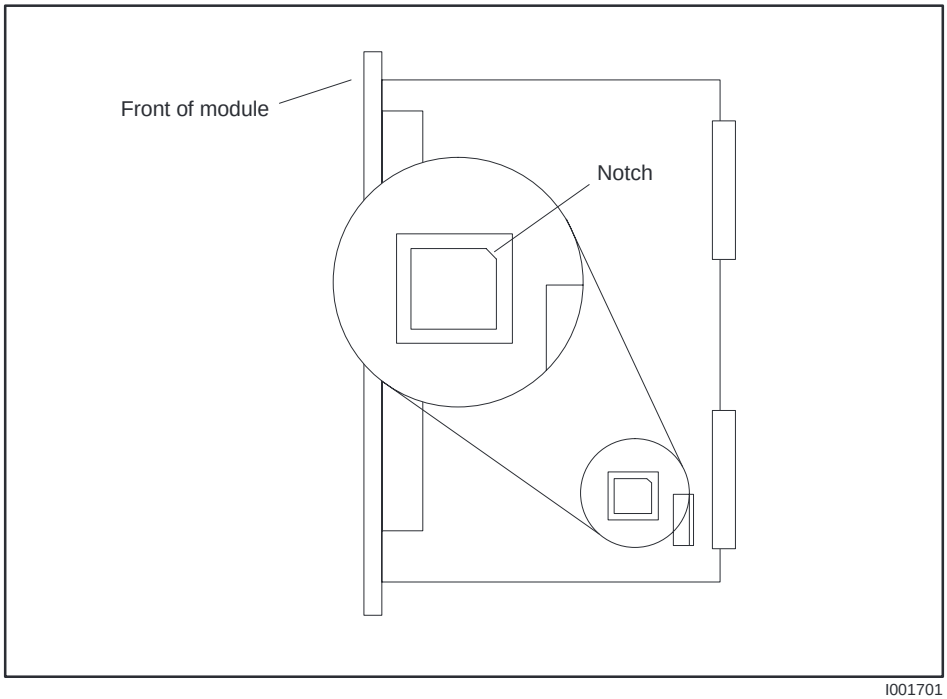

Figure A-2 387SX Socket Orientation (Top View)

Appendix B

Programming Examples

B.1 Overview B-2
PCCOMM Communication Examples B-2
C Programs B-2
QuickBASIC Programs B-2
GW-BASIC Programs B-2
B.2 C Program: IOREAD B-3
B.3 C Program: IOWRITE B-6
B.4 C Program: PCREAD B-9
B.5 C Program: PCWRITE B-13
B.6 QuickBASIC Program: IOREAD B-18
B.7 QuickBASIC Program: IOWRITE B-21
B.8 QuickBASIC Program: PCREAD B-24
B.9 QuickBASIC Program: PCWRITE B-28
B.10 GW-BASIC Program: IOREAD B-33
B.11 GW-BASIC Program: IOWRITE B-35
B.12 GW-BASIC Program: PCREAD B-38
B.13 GW-BASIC Program: PCWRITE B-42

B.1 Overview

PCCOMM Communication Examples

The example programs are provided to demonstrate using the PCCOMM communications service. There are three sets of example programs, one for each of the following languages: Microsoft GW-BASIC□, Microsoft QBasic (QuickBASIC), and C.

C Programs

The following C programs have been successfully compiled and linked with Microsoft C 5.1 and Turbo C□ 2.0.

- iord_c: Read the coprocessor module's WY values via IOREAD.
- iowr_c: Write to the coprocessor module's WX values via IOWRITE.
- pcrd_c: Read from V-memory, X registers and the WX points via PCREAD.
- pcwr_c: Write to V-memory, Y registers and the 386/ATM WY points via PCWRITE.

QuickBASIC Programs

Example programs for QuickBASIC are the following.

- iord_msb: Read the coprocessor module's WY values via IOREAD.
- iowr_msb: Write to the coprocessor module's WX values via IOWRITE.
- pcrd_msb: Read from V-memory, X registers and the 386/ATM WX points via PCREAD.
- pcwr_msb: Write to V-memory, Y registers and the 386/ATM WY points via PCWRITE.

GW-BASIC Programs

Example programs for GW-BASIC are the following.

- iord_gw: Read the coprocessor module's WY values via IOREAD.
- iowr_gw: Write to the coprocessor module's WX values via IOWRITE.
- pcrd_gw: Read from V-memory, X registers and the 386/ATM WX points via PCREAD.
- pcwr_gw: Write to V-memory, Y registers and the 386/ATM WY points via PCWRITE.

NOTE: GW-BASIC is not furnished with the 386/ATM Coprocessor module.

B.2 C Program: IOREAD

```
/*************************************************************************
*   iord_c:   Read the coprocessor module's WY values.                   *
*                                                                        *
*   Language:   Turbo C 2.0 or Microsoft C 5.1                           *
*   Date:       11/8/90                                                  *
*                                                                        *
*   Description:  This routine demonstrates the usage of the PCCOMM service   *
*       command IOREAD. The 4 local WY points will be read and displayed to   *
*       the screen.                                                      *
*                                                                        *
*   Suggestions:  You may want to run PCWR_C prior to execution of this  *
*       program to verify that real values are being read by this routine.    *
*       The last part of PCWR_C allows the user to write to the WY values.    *
*                                                                        *
*   Hardware Requirements:                                               *
*       Series 500/505 PLC                                               *
*       386/ATM COPROCESSOR                                              *
*                                                                        *
*   Software Requirements:                                               *
*       1.  Turbo C 2.0 or Microsoft C 5.1                               *
*                                                                        *
*   Warnings:                                                            *
*                                                                        *
*                                                                        *
*************************************************************************/
#include <stdio.h>
/*************************************************************************
* Data Declarations                                                      *
*************************************************************************/
                                    /*************************************
                                    * File pointer for the device driver.  *
                                    *************************************/
FILE *driver;
                                    /*************************************
                                    * Buffer for strings received from the *
                                    * device driver.                       *
                                    *************************************/
char buffer[200];
                                    /*************************************
                                    * The number of values read by the     *
                                    * device driver.  This value is parsed *
                                    * from the return string.              *
                                    *************************************/
int input_count;
                                    /*************************************
                                    * Pointer to token parsed from the     *
                                    * response string.                     *
                                    *************************************/
char *token;
                                    /*************************************
                                    * Tell the compiler that strtok (a C   *
                                    * library function) returns character  *
                                    * pointers.                            *
                                    *************************************/
char *strtok();
                                    /*************************************
                                    * Loop counter used for all FOR loops. *
                                    *************************************/
int loop_counter;
/*************************************************************************
* Program                                                                *
*************************************************************************/
int main(void)
{
```

```
                                    /*************************************
                                    * Display a message describing the     *
                                    * program.                             *
                                    *************************************/
printf("\n\n\n\n");
printf("IORD_C:   Example usage of the PCCOMM command IOREAD to read");
printf(" from the\n           module's WY points.\n");
printf("\nSee the file IORD_C.c for a more complete description of the");
printf("\noperation of this routine.\n");
                                    /*************************************
                                    * Open the device driver in update     *
                                    * mode (reading and writing).  If it   *
                                    * does not open correctly then exit    *
                                    * the program with an error message.   *
                                    *************************************/
if ((driver = fopen("PCCOMM", "r+")) == NULL)
  {
    printf("\nCould not open the device driver.\n");
    exit(1);
  }
                                    /*************************************
                                    * Write the request to the device      *
                                    * driver.                              *
                                    *************************************/
fprintf(driver, ":IR:5:4::\r");
                                    /*************************************
                                    * Flush the file buffer to ensure that *
                                    * the driver received the request.     *
                                    *************************************/
fflush(driver);
                                    /*************************************
                                    * The file pointer must be returned to *
                                    * the beginning of the file after each *
                                    * transaction with the device driver.  *
                                    *************************************/
fseek(driver,0L,SEEK_SET);
                                    /*************************************
                                    * Get a response from the device       *
                                    * driver.                              *
                                    *************************************/
fgets(buffer, 199, driver);
fseek(driver,0L,SEEK_SET);
                                    /*************************************
                                    * Skip to the 2nd token in the response*
                                    * string (it contains the number of    *
                                    * values read).                        *
                                    *************************************/
strtok(buffer, ":");
token = strtok(NULL, ":");
input_count = atoi(token);
                                    /*************************************
                                    * Print an error message if the number *
                                    * of values read does not equal 4.     *
                                    *************************************/
if (input_count != 4)
    printf("\nThe device driver was unable to read the 4 values!");
else
  {
```

```
                                        /************************************
                                        * Display the 4 values to the screen.  *
                                        ************************************/
          for (loop_counter = 0; loop_counter < 4; ++loop_counter)
            {
              token = strtok(NULL, ":");
              printf("\nLocation %d: %05u", loop_counter + 5, atoi(token));
            }
          printf("\n");
        }
}
```

B.3 C Program: IOWRITE

```
/*****************************************************************************
*   iowr_c:   Write to the coprocessor module's WX values.                   *
*                                                                            *
*   Language:   Turbo C 2.0 or Microsoft C 5.1                               *
*   Date:       11/8/90                                                      *
*                                                                            *
*   Description:  This routine demonstrates the usage of the PCCOMM service  *
*       command IOWRITE. The 4 local WX points will be written as specified  *
*       by the user.                                                         *
*                                                                            *
*   Suggestions:  You may want to run PCRD_C after this program to verify    *
*       that the values were written correctly by this routine.  The last    *
*       part of PCRD_C allows the user to read the WX values.                *
*                                                                            *
*   Hardware Requirements:                                                   *
*       Series 500/505 PLC                                                   *
*       386/ATM COPROCESSOR                                                  *
*                                                                            *
*   Software Requirements:                                                   *
*       1.  Turbo C 2.0 or Microsoft C 5.1                                   *
*                                                                            *
*   Warnings:                                                                *
*                                                                            *
*                                                                            *
*****************************************************************************/
#include <stdio.h>
/*****************************************************************************
* Data Declarations                                                          *
*****************************************************************************/
                                    /*************************************
                                    * File pointer for the device driver.  *
                                    *************************************/
FILE *driver;
                                    /*************************************
                                    * Buffer for strings received from the *
                                    * device driver.                       *
                                    *************************************/
char buffer[200];
                                    /*************************************
                                    * The number of values written by the  *
                                    * device driver.  This value is parsed *
                                    * from the return string.              *
                                    *************************************/
int input_count;
                                    /*************************************
                                    * Pointer to token parsed from the     *
                                    * response string.                     *
                                    *************************************/
char *token;
                                    /*************************************
                                    * Tell the compiler that strtok (a C   *
                                    * library function) returns character  *
                                    * pointers.                            *
                                    *************************************/
char *strtok();
                                    /*************************************
                                    * Loop counter used for all FOR loops. *
                                    *************************************/
int loop_counter;
                                    /*************************************
                                    * Storage for values that are to be    *
                                    * written to the 4 local WX points.    *
                                    *************************************/
int values[4];
```

```
/*************************************************************************
* Program                                                                *
*************************************************************************/
int main(void)
{
                                    /*************************************
                                    * Display a message describing the    *
                                    * program.                            *
                                    *************************************/
    printf("\n\n\n\n");
    printf("IOWR_C:    Example usage of the PCCOMM command IOWRITE to write ");
    printf("to the module's\n           WX points.\n");
    printf("\nSee the file IOWR_C.c for a more complete description of the");
    printf("\noperation of this routine.\n\n");
                                    /*************************************
                                    * Open the device driver in update    *
                                    * mode (reading and writing).  If it  *
                                    * does not open correctly then exit   *
                                    * the program with an error message.  *
                                    *************************************/
    if ((driver = fopen("PCCOMM", "r+")) == NULL)
      {
        printf("\nCould not open the device driver.\n");
        exit(1);
      }
                                    /*************************************
                                    * Prompt the user and accept entry of *
                                    * the 4 values that will be written to*
                                    * the WX points.                      *
                                    *************************************/
    for (loop_counter = 0; loop_counter < 4; ++loop_counter)
      {
        printf("Enter the value to write at location %d: ", loop_counter + 1);
        scanf("%d", &(values[loop_counter]));
      }
                                    /*************************************
                                    * Write the request to the device     *
                                    * driver.  Note that the values sent  *
                                    * to the device driver are unsigned.  *
                                    * This is because the device driver   *
                                    * does not accept values with a       *
                                    * negative sign in front of them.     *
                                    *************************************/
    fprintf(driver, ":IW:1:4:%u:%u:%u:%u::\r",
            values[0],
            values[1],
            values[2],
            values[3]);
                                    /*************************************
                                    * Flush the file buffer to ensure that*
                                    * the driver received the request.    *
                                    *************************************/
    fflush(driver);
                                    /*************************************
                                    * The file pointer must be returned to*
                                    * the beginning of the file after each*
                                    * transaction with the device driver. *
                                    *************************************/
    fseek(driver,0L,SEEK_SET);
                                    /*************************************
                                    * Get a response from the device      *
                                    * driver.                             *
                                    *************************************/
    fgets(buffer, 199, driver);
    fseek(driver,0L,SEEK_SET);
```

```
                                        /************************************
                                        * Skip to the 2nd token in the response*
                                        * string (it contains the number of    *
                                        * values written).                     *
                                        ************************************/
    strtok(buffer, ":");
    token = strtok(NULL, ":");
    input_count = atoi(token);
                                        /************************************
                                        * Print an error message if the number *
                                        * of values written does not equal 4.  *
                                        ************************************/
    if (input_count != 4)
        printf("\nThe device driver was unable to write the 4 values!");
}
```

B.4 C Program: PCREAD

```
/*****************************************************************************
*   pcrd_c:  Read from V-mem, X registers and the coprocessor module's        *
*            WX points via PCREAD.                                            *
*                                                                             *
*   Language:   Turbo C 2.0 or Microsoft C 5.1                                *
*   Date:       11/8/90                                                       *
*                                                                             *
*   Description:  This routine demonstrates the usage of the PCCOMM service   *
*       command PCREAD.  V-memory, X registers and the module's WX points     *
*       will be read and displayed to the screen.                             *
*           The first part of the program will let the user read from 8       *
*       consecutive V-memory locations.  The user is prompted to enter        *
*       an integer value which specifies the first V-memory location to read  *
*       from.  Then the 8 values are displayed to the screen.  An error       *
*       message will be displayed if the device driver was unable to read     *
*       the 8 values from the PLC.                                            *
*           The second part of the program will let the user read 8 discrete  *
*       X register values.  The user is prompted to enter an integer value    *
*       which specifies the first X register location of the 8 to read from.  *
*       Then the 8 values are read and displayed on the screen.               *
*           The final section of the program will allow the user to read from *
*       the module's 4 WX locations.  The user is prompted to enter an        *
*       integer value which specifies the first WX register location of the   *
*       module.  Then the values are displayed to the screen.                 *
*                                                                             *
*   Suggestions:  You may want to run the routines PCWR_C and IOWR_C          *
*       prior to execution of this routine to verify that real values are     *
*       being read back from the PLC.  PCWR_C can be used to write to         *
*       v-memory and discrete locations, and IOWR_C can be used to write to   *
*       the 4 WX values on the module.                                        *
*                                                                             *
*   Hardware Requirements:                                                    *
*       Series 500/505 PLC                                                    *
*       386/ATM COPROCESSOR                                                   *
*                                                                             *
*   Software Requirements:                                                    *
*       1.  Turbo C 2.0 or Microsoft C 5.1                                    *
*                                                                             *
*   Warnings:                                                                 *
*                                                                             *
*                                                                             *
*****************************************************************************/
#include <stdio.h>
/*****************************************************************************
* Data Declarations                                                          *
*****************************************************************************/
                                    /*************************************
                                    * File pointer for the device driver.  *
                                    *************************************/
FILE *driver;
                                    /*************************************
                                    * Buffer for strings received from the *
                                    * device driver.                       *
                                    *************************************/
char buffer[200];
                                    /*************************************
                                    * The number of values read by the     *
                                    * device driver.  This value is parsed *
                                    * from the return string.              *
                                    *************************************/
int input_count;
                                    /*************************************
                                    * The beginning location that the      *
                                    * series of data is to be read from.   *
                                    *************************************/
int start_point;
```

```
                                        /*************************************
                                        * Pointer to token parsed from the    *
                                        * response string.                    *
                                        *************************************/
char *token;
                                        /*************************************
                                        * Tell the compiler that strtok (a C  *
                                        * library function) returns character *
                                        * pointers.                           *
                                        *************************************/
char *strtok();
                                        /*************************************
                                        * Loop counter used for all FOR loops.*
                                        *************************************/
int loop_counter;
/*****************************************************************************
* Program                                                                    *
*****************************************************************************/
int main(void)
{
                                        /*************************************
                                        * Display a message describing the    *
                                        * program.                            *
                                        *************************************/
    printf("\n\n\n\n");
    printf("PCRD_C:    Example usage of the PCCOMM command PCREAD.\n");
    printf("\nSee the file PCRD_C.c for a more complete description of the");
    printf("\noperation of this routine.\n");
                                        /*************************************
                                        * Open the device driver in update    *
                                        * mode (reading and writing).  If it  *
                                        * does not open correctly then exit   *
                                        * the program with an error message.  *
                                        *************************************/
    if ((driver = fopen("PCCOMM", "r+")) == NULL)
      {
        printf("\nCould not open the device driver.\n");
        exit(1);
      }
                                        /*************************************
                                        * Prompt the user for input and read  *
                                        * the V-mem start point from the      *
                                        * keyboard.                           *
                                        *************************************/
    printf("\nEnter the address of the first V-memory point to read from: ");
    scanf("%d", &start_point);
                                        /*************************************
                                        * Write the request to the device     *
                                        * driver.                             *
                                        *************************************/
    fprintf(driver,":pr:VMEM:%d:8::\r", start_point);
                                        /*************************************
                                        * Flush the file buffer to ensure that*
                                        * the driver received the request.    *
                                        *************************************/
    fflush(driver);
                                        /*************************************
                                        * The file pointer must be returned to*
                                        * the beginning of the file after each*
                                        * transaction with the device driver. *
                                        *************************************/
    fseek(driver,0L,SEEK_SET);
```

```
                                    /*************************************
                                    * Get a response from the device      *
                                    * driver.                             *
                                    *************************************/
fgets(buffer, 199, driver);
fseek(driver,0L,SEEK_SET);
                                    /*************************************
                                    * Skip to the 2nd token in the response*
                                    * string (it contains the number of   *
                                    * values read).                       *
                                    *************************************/
strtok(buffer, ":");
token = strtok(NULL, ":");
input_count = atoi(token);
                                    /*************************************
                                    * Print an error message if the number *
                                    * of values read does not equal 8.    *
                                    *************************************/
if (input_count != 8)
    printf("\nThe device driver was unable to read the 8 values!");
else
  {
                                    /*************************************
                                    * Display the 8 values to the screen.  *
                                    *************************************/
    for (loop_counter = 0; loop_counter < 8; ++loop_counter)
      {
        token = strtok(NULL, ":");
        printf("\nV-mem location %d: %05u", start_point + loop_counter,
               atoi(token));
      }
  }
                                    /*************************************
                                    * Prompt the user for input and read   *
                                    * from the keyboard the location of    *
                                    * the first X register to read from.   *
                                    *************************************/
printf("\n\nEnter the address of the first X register to read: ");
scanf("%d", &start_point);
                                    /*************************************
                                    * Write the request to the device     *
                                    * driver.                             *
                                    *************************************/
fprintf(driver, ":pr:XREG:%d:8::\r", start_point);
fflush(driver);
fseek(driver, 0L,SEEK_SET);
                                    /*************************************
                                    * Get a response from the device      *
                                    * driver.                             *
                                    *************************************/
fgets(buffer, 199, driver);
fseek(driver, 0L, SEEK_SET);
strtok(buffer, ":");
token = strtok(NULL, ":");
input_count = atoi(token);
```

```
                                        /*************************************
                                        * Print an error message if the number *
                                        * of values read does not equal 8.     *
                                        *************************************/
    if (input_count != 8)
        printf("\nThe device driver was unable to read the 8 values!");
    else
      {
                                        /*************************************
                                        * Display the 8 values to the screen.  *
                                        *************************************/
        for (loop_counter = 0; loop_counter < 8; ++loop_counter)
          {
            token = strtok(NULL, ":");
            printf("\nX%d: %u", start_point + loop_counter, atoi(token));
          }
      }
                                        /*************************************
                                        * Prompt the user for input and read   *
                                        * from the keyboard the location of    *
                                        * the first WX to read from.           *
                                        *************************************/
    printf("\n\nEnter the address of the first WX register on the module: ");
    scanf("%d", &start_point);
                                        /*************************************
                                        * Write the request to the device      *
                                        * driver.                              *
                                        *************************************/
    fprintf(driver, ":pr:WX:%d:4::\r", start_point);
    fflush(driver);
    fseek(driver, 0L, SEEK_SET);
                                        /*************************************
                                        * Get a response from the device       *
                                        * driver.                              *
                                        *************************************/
    fgets(buffer, 199, driver);
    fseek(driver, 0L, SEEK_SET);
    strtok(buffer, ":");
    token = strtok(NULL, ":");
    input_count = atoi(token);
                                        /*************************************
                                        * Print an error message if the number *
                                        * of values read does not equal 4.     *
                                        *************************************/
    if (input_count != 4)
        printf("\nThe device driver was unable to read the 4 values!");
    else
      {
                                        /*************************************
                                        * Display the 4 values to the screen.  *
                                        *************************************/
        for (loop_counter = 0; loop_counter < 4; ++loop_counter)
          {
            token = strtok(NULL, ":");
            printf("\nWX%d: %05u", start_point + loop_counter, atoi(token));
          }
      }
    printf("\n");
}
```

B.5 C Program: PCWRITE

```
/*****************************************************************************
*   pcwr_c:  Write to V-mem, Y registers and the coprocessor module's         *
*             WY points via PCWRITE.                                          *
*                                                                             *
*   Language:   Turbo C 2.0 or Microsoft C 5.1                                *
*   Date:       11/8/90                                                       *
*                                                                             *
*   Description:  This routine demonstrates the usage of the PCCOMM service   *
*       command PCWRITE.  V-memory, Y registers and the module's WY points    *
*       will be written as specified by the user.                             *
*           The first part of the program will let the user write to 8        *
*       consecutive V-memory locations.  The user is prompted to enter        *
*       an integer value which specifies the first V-memory location to write *
*       to.  Then the user is prompted to enter 8 values which will be        *
*       written to consecutive V-memory locations starting with the location  *
*       previously specified.  An error message will be displayed if the      *
*       device driver was unable to write the 8 values to the PLC.            *
*           The second part of the program will let the user write 8 discrete *
*       Y register values.  The user is prompted to enter an integer value    *
*       which specifies the first Y register location of the 8 to write to.   *
*       Then the user is prompted to enter the 8 values which will be         *
*       written to 8 consecutive Y registers starting with the location       *
*       specified.  Any non-zero value will be written as a 1.                *
*           The final section of the program will allow the user to write to  *
*       the module's 4 WY locations.  The user is prompted to enter an        *
*       integer value which specifies the first WY register location of the   *
*       module.  Remember that the four WYs are located AFTER the 4 WXs.      *
*       Then the user is prompted to enter the 4 values which will            *
*       be written to the 4 consecutive WY registers on the module.           *
*                                                                             *
*   Suggestions:  Since this routine writes to various PLC memory locations   *
*       you may want a means of reading back the locations to verify that the *
*       values were in fact written.  One means of doing this would be to run *
*       the example programs PCRD_C and IORD_C.  PCRD_C can be used            *
*       to read the 8 v-memory and discrete locations, and IORD_C can be      *
*       used to read the 4 WY values (assuming that the module is installed   *
*       in the slot that you wrote the 4 WY values to).                       *
*                                                                             *
*   Hardware Requirements:                                                    *
*       Series 500/505 PLC                                                    *
*       386/ATM COPROCESSOR                                                   *
*                                                                             *
*   Software Requirements:                                                    *
*       1.  Turbo C 2.0 or Microsoft C 5.1                                    *
*                                                                             *
*   Warnings:                                                                 *
*       1.  THIS ROUTINE WRITES TO V-MEMORY AND I/O POINTS.                   *
*                                                                             *
*                                                                             *
*****************************************************************************/
#include <stdio.h>
/*****************************************************************************
* Data Declarations                                                          *
*****************************************************************************/
                                    /*************************************
                                    * File pointer for the device driver.  *
                                    *************************************/
FILE *driver;
                                    /*************************************
                                    * Buffer for strings received from the *
                                    * device driver.                       *
                                    *************************************/
char buffer[200];
```

```
                                    /*************************************
                                    * The number of values written by the  *
                                    * device driver.  This value is parsed *
                                    * from the return string.              *
                                    *************************************/
int input_count;
                                    /*************************************
                                    * The beginning location that the      *
                                    * series of data is to be written to.  *
                                    *************************************/
int start_point;
                                    /*************************************
                                    * Storage for 8 discrete or word       *
                                    * values that are to be written.       *
                                    *************************************/
int values[8];
                                    /*************************************
                                    * Pointer to token parsed from the     *
                                    * response string.                     *
                                    *************************************/
char *token;
                                    /*************************************
                                    * Tell the compiler that strtok (a C   *
                                    * library function) returns character  *
                                    * pointers.                            *
                                    *************************************/
char *strtok();
                                    /*************************************
                                    * Loop counter used for all FOR loops. *
                                    *************************************/
int loop_counter;
/*****************************************************************************
* Program                                                                    *
*****************************************************************************/
int main(void)
{
                                    /*************************************
                                    * Display a message describing the     *
                                    * program.                             *
                                    *************************************/
    printf("\n\n\n\n");
    printf("PCWR_C:    Example usage of the PCCOMM command PCWRITE.\n");
    printf("\nSee the file PCWR_C.c for a more complete description of the");
    printf("\noperation of this routine.\n");
                                    /*************************************
                                    * Print a warning message.             *
                                    *************************************/
    printf("\nWARNING: This program writes to V-memory and Y-registers!\n");
    printf("Hit <space> to continue and any other key to exit.\n\n");
    if (getch() != ' ')
        exit(1);
                                    /*************************************
                                    * Open the device driver in update     *
                                    * mode (reading and writing).  If it   *
                                    * does not open correctly then exit    *
                                    * the program with an error message.   *
                                    *************************************/
    if ((driver = fopen("PCCOMM", "r+")) == NULL)
      {
        printf("\nCould not open the device driver.\n\n");
        exit(1);
      }
```

```
                                    /*************************************
                                    * Prompt the user for input and read  *
                                    * the V-mem start point from the      *
                                    * keyboard.                           *
                                    *************************************/
printf("\nEnter the address of the first V-memory point to write to: ");
scanf("%d", &start_point);
                                    /*************************************
                                    * Allow the user to enter the 8 values *
                                    * at the keyboard.                    *
                                    *************************************/
for (loop_counter = 0; loop_counter < 8 ; ++loop_counter)
  {
    printf("Enter the value to write at location %d: ",
           loop_counter + start_point);
    scanf("%d", &(values[loop_counter]));
  }
                                    /*************************************
                                    * Write the request to the device     *
                                    * driver.  Note that the values sent  *
                                    * to the device driver are unsigned.  *
                                    * This is because the device driver   *
                                    * does not accept values with a       *
                                    * negative sign in front of them.     *
                                    *************************************/
fprintf(driver,":pw:VMEM:%d:8:%u:%u:%u:%u:%u:%u:%u:%u::\r",
        start_point,
        values[0],
        values[1],
        values[2],
        values[3],
        values[4],
        values[5],
        values[6],
        values[7]);
                                    /*************************************
                                    * Flush the file buffer to ensure that *
                                    * the driver received the request.    *
                                    *************************************/
fflush(driver);
                                    /*************************************
                                    * The file pointer must be returned to *
                                    * the beginning of the file after each *
                                    * transaction with the device driver.  *
                                    *************************************/
fseek(driver,0L,SEEK_SET);
                                    /*************************************
                                    * Get a response from the device      *
                                    * driver.                             *
                                    *************************************/
fgets(buffer, 199, driver);
fseek(driver,0L,SEEK_SET);
                                    /*************************************
                                    * Skip to the 2nd token in the response*
                                    * string (it contains the number of   *
                                    * values written).                    *
                                    *************************************/
strtok(buffer, ":");
token = strtok(NULL, ":");
input_count = atoi(token);
                                    /*************************************
                                    * Print an error message if the number *
                                    * of values written does not equal 8.  *
                                    *************************************/
if (input_count != 8)
    printf("\nERROR: The device driver was unable to write the 8 values\n");
```

```
                                    /*************************************
                                    * Prompt the user for input and read   *
                                    * from the keyboard the location of    *
                                    * the first Y register to write to.    *
                                    *************************************/
printf("\nEnter the address of the first Y register to write to: ");
scanf("%d", &start_point);
                                    /*************************************
                                    * Allow the user to enter the 8 values *
                                    * at the keyboard.  Any non-zero value *
                                    * is written as a 1.                   *
                                    *************************************/
for (loop_counter = 0; loop_counter < 8; ++loop_counter)
  {
    printf("Enter the value to write at Y%d: ",
           start_point + loop_counter);
    scanf("%d", &(values[loop_counter]));
  }
                                    /*************************************
                                    * Write the request to the device      *
                                    * driver.                              *
                                    *************************************/
fprintf(driver, ":pw:YREG:%d:8:%u:%u:%u:%u:%u:%u:%u:%u::\r",
        start_point,
        values[0],
        values[1],
        values[2],
        values[3],
        values[4],
        values[5],
        values[6],
        values[7]);
fflush(driver);
fseek(driver, 0L,SEEK_SET);
                                    /*************************************
                                    * Get a response from the device       *
                                    * driver.                              *
                                    *************************************/
fgets(buffer, 199, driver);
fseek(driver, 0L, SEEK_SET);
strtok(buffer, ":");
token = strtok(NULL, ":");
input_count = atoi(token);
                                    /*************************************
                                    * Print an error message if the number *
                                    * of values written does not equal 8.  *
                                    *************************************/
if (input_count != 8)
    printf("\nERROR: The device driver was unable to write the 8 values\n");
                                    /*************************************
                                    * Prompt the user for input and read   *
                                    * from the keyboard the location of    *
                                    * the first WY to write to.            *
                                    *************************************/
printf("\nEnter the address of the first WY register on the module: ");
scanf("%d", &start_point);
```

```
                                    /************************************
                                    * Allow the user to enter the 4 values *
                                    * that will be written to the module's *
                                    * 4 WY points.                         *
                                    ************************************/
    for (loop_counter = 0; loop_counter < 4; ++loop_counter)
      {
        printf("Enter the value to write at WY%d: ",
               start_point + loop_counter);
        scanf("%d", &(values[loop_counter]));
      }
                                    /************************************
                                    * Write the request to the device      *
                                    * driver.                              *
                                    ************************************/
    fprintf(driver, ":pw:WY:%d:4:%u:%u:%u:%u::\r",
            start_point,
            values[0],
            values[1],
            values[2],
            values[3]);
    fflush(driver);
    fseek(driver, 0L, SEEK_SET);
                                    /************************************
                                    * Get a response from the device       *
                                    * driver.                              *
                                    ************************************/
    fgets(buffer, 199, driver);
    fseek(driver, 0L, SEEK_SET);
    strtok(buffer, ":");
    token = strtok(NULL, ":");
    input_count = atoi(token);
                                    /************************************
                                    * Print an error message if the number *
                                    * of values written does not equal 4.  *
                                    ************************************/
    if (input_count != 4)
        printf("\nERROR: The device driver was unable to write the 4 values\n");
}
```

```
DECLARE FUNCTION GetToken$ (String1$, FirstTime%)
'*************************************************************************
'   iord_msb: Read the coprocessor module's WY values.                    *
'                                                                         *
'   Language:   Microsoft Quick Basic                                     *
'   Date:       11/13/90                                                  *
'                                                                         *
'   Description:  This routine demonstrates the usage of the PCCOMM service   *
'       command IOREAD.  The 4 local WY points will be read and displayed     *
'       to the screen.                                                    *
'                                                                         *
'   Suggestions:  You may want to run PCWR_MSB prior to execution of this     *
'       program to verify that real values are being read by this routine.    *
'       The last part of PCWR_MSB allows the user to write to the WY values.  *
'                                                                         *
'   Hardware Requirements:                                                *
'       Series 500/505 PLC                                                *
'       386/ATM COPROCESSOR                                               *
'                                                                         *
'   Software Requirements:                                                *
'       1.  Microsoft Quick Basic                                         *
'                                                                         *
'   Warnings:                                                             *
'                                                                         *
'                                                                         *
'*************************************************************************
    CONST FALSE = 0
    CONST TRUE = NOT FALSE
    DEFINT A-Z
    DIM Values(3)
                                    '***************************************
                                    ' Clear the screen and display a       *
                                    ' message describing the program.      *
                                    '***************************************
    CLS
    PRINT "IORD_MSB:  Example usage of the PCCOMM command IOREAD to read";
    PRINT " from the module's"
    PRINT "           WY points."
    PRINT
    PRINT "See the file IORD_MSB.bas for a more complete description of the"
    PRINT "operation of this routine."
    PRINT
                                    '***************************************
                                    ' Open the device driver for reading   *
                                    ' and writing.                         *
                                    '***************************************
    OPEN "PCCOMM" FOR OUTPUT AS #1
    OPEN "PCCOMM" FOR INPUT AS #2
                                    '***************************************
                                    ' Write the request to the device      *
                                    ' driver.                              *
                                    '***************************************
    PRINT #1, ":IR:5:4::"
                                    '***************************************
                                    ' Get the response from the device     *
                                    ' driver.                              *
                                    '***************************************
    LINE INPUT #2, ResponseString$
                                    '***************************************
                                    ' Skip to the 2nd token in the response*
                                    ' string (it contains the number of    *
                                    ' values read).                        *
                                    '***************************************
    Token$ = GetToken$(ResponseString$, TRUE)
    Token$ = GetToken$(ResponseString$, FALSE)
    InputCount = VAL(Token$)
```

```
                                        '************************************
                                        ' Print an error message if the number *
                                        ' of values read does not equal 4.     *
                                        '************************************
    IF InputCount <> 4 THEN
        PRINT
        PRINT "The device driver was unable to read the 4 values!";
    ELSE
                                        '************************************
                                        ' Display the 4 values to the screen.  *
                                        '************************************
        FOR LoopCounter = 0 TO 3
            Token$ = GetToken$(ResponseString$, FALSE)
            PRINT
            PRINT "Location "; LoopCounter + 5; ":"; VAL(Token$);
        NEXT
        PRINT
    END IF
END
'*****************************************************************************
' Function Name: GetToken$                                                   *
' Usage: Token$ = GetToken$(String1$, FirstTime)                             *
' Parameters:                                                                *
'        Token$: The token parsed from String1$ ("" if the end of the string  *
'                has been reached).                                          *
'        String1$: The string that is being parsed                           *
'        FirstTime: TRUE causes the function to begin parsing at the         *
'                   beginning of the string.                                 *
'                FALSE causes the function to parse the token following      *
'                   the token parsed on the previous call.                   *
'                                                                            *
' Description:  This function extracts a token from String1$.  To parse the  *
'       first token from a string, pass a value of TRUE for the FirstTime    *
'       parameter.  To parse subsequent tokens from the string pass a value of*
'       FALSE for the FirstTime parameter.  For the purposes of this routine *
'       a token is defined as a sequence of characters that have a preceding *
'       ':' character and a following ':' character.  The ':' characters are *
'       NOT returned with the token.                                         *
'                                                                            *
'*****************************************************************************
FUNCTION GetToken$ (String1$, FirstTime) STATIC
                                        '************************************
                                        ' If this is the first call for this   *
                                        ' particular string then set index to  *
                                        ' point to beginning of string and     *
                                        ' skip over the initial ':' character. *
                                        '************************************
    IF FirstTime = TRUE THEN
        I = 1
        I = INSTR(I, String1$, ":")
        I = I + 1
    END IF
```

```
                                        '************************************
                                        ' If I is greater than the length of   *
                                        ' the string then return "" as the     *
                                        ' token.  Otherwise parse the token    *
                                        ' from the string and update I to      *
                                        ' point to the beginning of the next   *
                                        ' token.                               *
                                        '************************************
    IF I > LEN(String1$) THEN
        GetToken$ = ""
    ELSE
        J = INSTR(I, String1$, ":")
        TokenLength = J - I
        GetToken$ = MID$(String1$, I, TokenLength)
        I = J + 1
    END IF
END FUNCTION
```

B.7 QuickBASIC Program: IOWRITE

```
DECLARE FUNCTION GetToken$ (String1$, FirstTime%)
'*************************************************************************
'   iowr_msb: Write to the coprocessor module's WX values.                *
'                                                                         *
'   Language:   Microsoft Quick Basic                                     *
'   Date:       11/13/90                                                  *
'                                                                         *
'   Description:  This routine demonstrates the usage of the PCCOMM service   *
'       command IOWRITE.  The 4 local WX points will be written as specified  *
'       by the user.                                                      *
'                                                                         *
'   Suggestions:  You may want to run PCRD_MSB after this program to verify   *
'       that the values were written correctly by this routine.  The last     *
'       part of PCRD_MSB allows the user to read the WX values.           *
'                                                                         *
'   Hardware Requirements:                                                *
'       Series 500/505 PLC                                                *
'       386/ATM COPROCESSOR                                               *
'                                                                         *
'   Software Requirements:                                                *
'       1.  Microsoft Quick Basic                                         *
'                                                                         *
'   Warnings:                                                             *
'                                                                         *
'                                                                         *
'*************************************************************************
    CONST FALSE = 0
    CONST TRUE = NOT FALSE
    DEFINT A-Z
    DIM Values(3)
                                    '************************************
                                    ' Clear the screen and display a    *
                                    ' message describing the program.   *
                                    '************************************
    CLS
    PRINT "IOWR_MSB:  Example usage of the PCCOMM command IOWRITE to write";
    PRINT " to the module's"
    PRINT "           WX points."
    PRINT
    PRINT "See the file IOWR_MSB.bas for a more complete description of the"
    PRINT "operation of this routine."
    PRINT
                                    '************************************
                                    ' Open the device driver for reading *
                                    ' and writing.                      *
                                    '************************************
    OPEN "PCCOMM" FOR OUTPUT AS #1
    OPEN "PCCOMM" FOR INPUT AS #2
                                    '************************************
                                    ' Prompt the user and accept entry of  *
                                    ' the 4 values that will be written to *
                                    ' the WX points.                    *
                                    '************************************
    FOR LoopCounter = 0 TO 3
        PRINT "Enter the value to write at location ";
        PRINT LoopCounter + 1; ": ";
        INPUT "", Values(LoopCounter)
    NEXT
```

```
                                    '*************************************
                                    ' Write the request to the device     *
                                    ' driver.  Note that leading blanks   *
                                    ' are removed from Values() via       *
                                    ' LTRIM$.                             *
                                    '*************************************
    RequestString$ = ":iw:1:4"
    FOR LoopCounter = 0 TO 3
        RequestString$ = RequestString$ + ":"
        RequestString$ = RequestString$ + LTRIM$(STR$(Values(LoopCounter)))
    NEXT
    RequestString$ = RequestString$ + "::"
    PRINT #1, RequestString$
                                    '*************************************
                                    ' Get a response from the device      *
                                    ' driver.                             *
                                    '*************************************
    LINE INPUT #2, ResponseString$
                                    '*************************************
                                    ' Skip to the 2nd token in the response*
                                    ' string (it contains the number of   *
                                    ' values written).                    *
                                    '*************************************
    Token$ = GetToken$(ResponseString$, TRUE)
    Token$ = GetToken$(ResponseString$, FALSE)
    InputCount = VAL(Token$)
                                    '*************************************
                                    ' Print an error message if the number *
                                    ' of values written does not equal 4.  *
                                    '*************************************
    IF InputCount <> 4 THEN
        PRINT
        PRINT "The device driver was unable to write the 4 values!";
    END IF
    PRINT
END
'*****************************************************************************
' Function Name: GetToken$                                                    *
' Usage: Token$ = GetToken$(String1$, FirstTime)                              *
' Parameters:                                                                 *
'        Token$: The token parsed from String1$ ("" if the end of the string  *
'                has been reached).                                           *
'        String1$: The string that is being parsed                            *
'        FirstTime: TRUE causes the function to begin parsing at the          *
'                   beginning of the string.                                  *
'                   FALSE causes the function to parse the token following    *
'                   the token parsed on the previous call.                    *
'                                                                             *
' Description:  This function extracts a token from String1$.  To parse the   *
'       first token from a string, pass a value of TRUE for the FirstTime     *
'       parameter.  To parse subsequent tokens from the string pass a value of*
'       FALSE for the FirstTime parameter.  For the purposes of this routine  *
'       a token is defined as a sequence of characters that have a preceding  *
'       ':' character and a following ':' character.  The ':' characters are  *
'       NOT returned with the token.                                          *
'                                                                             *
'*****************************************************************************
FUNCTION GetToken$ (String1$, FirstTime) STATIC
```

```
                                    '************************************
                                    ' If this is the first call for this   *
                                    ' particular string then set index to  *
                                    ' point to beginning of string and     *
                                    ' skip over the initial ':' character. *
                                    '************************************
    IF FirstTime = TRUE THEN
        I = 1
        I = INSTR(I, String1$, ":")
        I = I + 1
    END IF
                                    '************************************
                                    ' If I is greater than the length of   *
                                    ' the string then return "" as the     *
                                    ' token.  Otherwise parse the token    *
                                    ' from the string and update I to      *
                                    ' point to the beginning of the next   *
                                    ' token.                               *
                                    '************************************
    IF I > LEN(String1$) THEN
        GetToken$ = ""
    ELSE
        J = INSTR(I, String1$, ":")
        TokenLength = J - I
        GetToken$ = MID$(String1$, I, TokenLength)
        I = J + 1
    END IF
END FUNCTION
```

```
DECLARE FUNCTION GetToken$ (String1$, FirstTime%)
'*************************************************************************
'   pcrd_msb: Read from V-mem, X registers and the coprocessor module's       *
'             WX points via PCREAD.                                           *
'                                                                             *
'   Language:   Microsoft Quick Basic                                         *
'   Date:       11/12/90                                                      *
'                                                                             *
'   Description:  This routine demonstrates the usage of the PCCOMM service   *
'       command PCREAD.  V-memory, X registers and the module's WX points     *
'       will be read and displayed to the screen.                             *
'           The first part of the program will let the user read from 8       *
'       consecutive V-memory locations.  The user is prompted to enter an     *
'       integer value which specifies the first V-memory location to read     *
'       from.  Then the 8 values are displayed to the screen.  An error       *
'       message will be displayed if the device driver was unable to read the *
'       8 values from the PLC.                                                *
'           The second part of the program will let the user read 8 discrete  *
'       X register values.  The user is prompted to enter an integer value    *
'       which specifies the first X register location of the 8 to read from.  *
'       Then the 8 values are read and displayed on the screen.               *
'           The final section of the program will allow the user to read from *
'       the module's 4 WX locations.  The user is prompted to enter an        *
'       integer value which specifies the first WX register location of the   *
'       module.  Then the values are displayed to the screen.                 *
'                                                                             *
'   Suggestions:  You may want to run the routines PCWR_MSB and IOWR_MSB      *
'       prior to execution of this routine to verify that real values are     *
'       being read back from the PLC.  PCWR_MSB can be used to write to       *
'       v-memory and discrete locations, and IOWR_MSB can be used to write to *
'       the 4 WX values on the module.                                        *
'                                                                             *
'   Hardware Requirements:                                                    *
'       Series 500/505 PLC                                                    *
'       386/ATM COPROCESSOR                                                   *
'                                                                             *
'   Software Requirements:                                                    *
'       1.  Microsoft Quick Basic                                             *
'                                                                             *
'   Warnings:                                                                 *
'                                                                             *
'                                                                             *
'*************************************************************************
    CONST FALSE = 0
    CONST TRUE = NOT FALSE
    DEFINT A-Z
                                    '************************************
                                    ' Clear the screen and display a    *
                                    ' message describing the program.   *
                                    '************************************
    CLS
    PRINT "PCRD_MSB:   Example usage of the PCCOMM command PCREAD."
    PRINT
    PRINT "See the file PCRD_MSB.bas for a more complete description of the"
    PRINT "operation of this routine."
                                    '************************************
                                    ' Open the device driver for reading *
                                    ' and writing.                       *
                                    '************************************
    OPEN "PCCOMM" FOR OUTPUT AS #1
    OPEN "PCCOMM" FOR INPUT AS #2
                                    '************************************
                                    ' Prompt the user and read the V-mem *
                                    ' start point from the keyboard.    *
                                    '************************************
    PRINT
    PRINT "Enter the address of the first V-memory point to read from: ";
    INPUT "", StartPoint
```

```
                                    '*************************************
                                    ' Write the request to the device      *
                                    ' driver.  Note that leading blanks    *
                                    ' are removed from the StartPoint      *
                                    ' value via LTRIM$.                    *
                                    '*************************************
PRINT #1, ":pr:vmem:"; LTRIM$(STR$(StartPoint)); ":8::"
                                    '*************************************
                                    ' Get a response from the device       *
                                    ' driver.                              *
                                    '*************************************
LINE INPUT #2, ResponseString$
                                    '*************************************
                                    ' Skip to the 2nd token in the         *
                                    ' response string (it contains the     *
                                    ' number of values read).              *
                                    ' See GetToken$() at the bottom of     *
                                    ' this listing for more information.   *
                                    '*************************************
Token$ = GetToken$(ResponseString$, TRUE)
Token$ = GetToken$(ResponseString$, FALSE)
InputCount = VAL(Token$)
                                    '*************************************
                                    ' Print an error message if the number *
                                    ' of values read does not equal 8.     *
                                    '*************************************
IF InputCount <> 8 THEN
    PRINT
    PRINT "The device driver was unable to read the 8 values!";
ELSE
                                    '*************************************
                                    ' Display the 8 values to the screen.  *
                                    '*************************************
    FOR LoopCounter = 0 TO 7
        Token$ = GetToken$(ResponseString$, FALSE)
        PRINT
        PRINT "V-mem location"; StartPoint + LoopCounter; ":"; VAL(Token$);
    NEXT
END IF
                                    '*************************************
                                    ' Prompt the user for input and read   *
                                    ' from the keyboard the location of    *
                                    ' the first X register to read from.   *
                                    '*************************************
PRINT
PRINT
PRINT "Enter the address of the first X register to read: ";
INPUT "", StartPoint
                                    '*************************************
                                    ' Write the request to the device      *
                                    ' driver.                              *
                                    '*************************************
PRINT #1, ":pr:XREG:"; LTRIM$(STR$(StartPoint)); ":8::"
                                    '*************************************
                                    ' Get the response from the device     *
                                    ' driver and parse the count value     *
                                    ' from the response via GetToken.  See *
                                    ' GetToken$() at the bottom of this    *
                                    ' listing.                             *
                                    '*************************************
LINE INPUT #2, ResponseString$
Token$ = GetToken$(ResponseString$, TRUE)
Token$ = GetToken$(ResponseString$, FALSE)
InputCount = VAL(Token$)
```

```
                                        '*************************************
                                        ' Print an error message if the number *
                                        ' of values read does not equal 8.     *
                                        '*************************************
    IF InputCount <> 8 THEN
        PRINT
        PRINT "The device driver was unable to read the 8 values!";
    ELSE
                                        '*************************************
                                        ' Display the 8 values to the screen.  *
                                        '*************************************
        FOR LoopCounter = 0 TO 7
            Token$ = GetToken$(ResponseString$, FALSE)
            PRINT
            PRINT "X"; StartPoint + LoopCounter; ": "; VAL(Token$);
        NEXT
    END IF
                                        '*************************************
                                        ' Prompt the user for input and read   *
                                        ' from the keyboard the location of    *
                                        ' the first WX to read from.           *
                                        '*************************************
    PRINT
    PRINT
    PRINT "Enter the address of the first WX register on the module: ";
    INPUT "", StartPoint
                                        '*************************************
                                        ' Write the request to the device      *
                                        ' driver.                              *
                                        '*************************************
    PRINT #1, ":pr:WX:"; LTRIM$(STR$(StartPoint)); ":4::"
                                        '*************************************
                                        ' Get the response from the device     *
                                        ' driver and parse the count value     *
                                        ' from the response via GetToken.  See *
                                        ' GetToken$() at the bottom of this    *
                                        ' listing.                             *
                                        '*************************************
    LINE INPUT #2, ResponseString$
    Token$ = GetToken$(ResponseString$, TRUE)
    Token$ = GetToken$(ResponseString$, FALSE)
    InputCount = VAL(Token$)
                                        '*************************************
                                        ' Print an error message if the number *
                                        ' of values read does not equal 4.     *
                                        '*************************************
    IF InputCount <> 4 THEN
        PRINT
        PRINT "The device driver was unable to read the 4 values!";
    ELSE
                                        '*************************************
                                        ' Display the 4 values to the screen.  *
                                        '*************************************
        FOR LoopCounter = 0 TO 3
            Token$ = GetToken$(ResponseString$, FALSE)
            PRINT
            PRINT "WX"; StartPoint + LoopCounter; ": "; VAL(Token$);
        NEXT
    END IF
    PRINT
END
```

```
'************************************************************************
' Function Name: GetToken$                                              *
' Usage: Token$ = GetToken$(String1$, FirstTime)                        *
' Parameters:                                                           *
'        Token$: The token parsed from String1$ ("" if the end of the string  *
'                has been reached).                                     *
'        String1$: The string that is being parsed                      *
'        FirstTime: TRUE causes the function to begin parsing at the    *
'                   beginning of the string.                            *
'                  FALSE causes the function to parse the token following    *
'                   the token parsed on the previous call.              *
'                                                                       *
' Description:  This function extracts a token from String1$.  To parse the   *
'       first token from a string, pass a value of TRUE for the FirstTime     *
'       parameter.  To parse subsequent tokens from the string pass a value of*
'       FALSE for the FirstTime parameter.  For the purposes of this routine  *
'       a token is defined as a sequence of characters that have a preceding  *
'       ':' character and a following ':' character.  The ':' characters are  *
'       NOT returned with the token.                                    *
'                                                                       *
'************************************************************************
FUNCTION GetToken$(String1$, FirstTime) STATIC
                                     '************************************
                                     ' If this is the first call for this   *
                                     ' particular string then set index to  *
                                     ' point to beginning of string and     *
                                     ' skip over the initial ':' character. *
                                     '************************************
    IF FirstTime = TRUE THEN
        I = 1
        I = INSTR(I, String1$, ":")
        I = I + 1
    END IF
                                     '************************************
                                     ' If I is greater than the length of   *
                                     ' the string then return "" as the     *
                                     ' token.  Otherwise parse the token    *
                                     ' from the string and update I to      *
                                     ' point to the beginning of the next   *
                                     ' token.                               *
                                     '************************************
    IF I > LEN(String1$) THEN
        GetToken$ = ""
    ELSE
        J = INSTR(I, String1$, ":")
        TokenLength = J - I
        GetToken$ = MID$(String1$, I, TokenLength)
        I = J + 1
    END IF
END FUNCTION
```

```
DECLARE FUNCTION GetToken$ (String1$, FirstTime%)
'************************************************************************
'   pcwr_msb: Write to V-mem, Y registers and the coprocessor module's      *
'             WY points via PCWRITE.                                        *
'                                                                           *
'   Language:   Microsoft Quick Basic                                       *
'   Date:       11/12/90                                                    *
'                                                                           *
'   Description:  This routine demonstrates the usage of the PCCOMM service *
'       command PCWRITE.  V-memory, Y registers and the module's WY points  *
'       will be written as specified by the user.                           *
'           The first part of the program will let the user write to 8      *
'       consecutive V-memory locations.  The user is prompted to enter      *
'       an integer value which specifies the first V-memory location to write *
'       to.  Then the user is prompted to enter 8 values which will be      *
'       written to consecutive V-memory locations starting with the location *
'       previously specified.  An error message will be displayed if the    *
'       device driver was unable to write the 8 values to the PLC.          *
'           The second part of the program will let the user write 8 discrete *
'       Y register values.  The user is prompted to enter an integer value  *
'       which specifies the first Y register location of the 8 to write to. *
'       Then the user is prompted to enter the 8 values which will be written *
'       to 8 consecutive Y registers starting with the location specified.  *
'       Any non-zero value will be written as a 1.                          *
'           The final section of the program will allow the user to write to *
'       the module's 4 WY locations.  The user is prompted to enter an      *
'       integer value which specifies the first WY register location of the *
'       module.  Remember that the four WYs are located AFTER the 4 WXs.    *
'       Then the user is prompted to enter the 4 values which will be written *
'       to the 4 consecutive WY registers on the module.                    *
'                                                                           *
'   Suggestions:    Since this routine writes to various PLC memory locations *
'       you may want a means of reading back the locations to verify that the *
'       values were in fact written.  One means of doing this would be to run *
'       the example programs PCRD_MSB and IORD_MSB.  PCRD_MSB can be used to *
'       read the 8 v-memory and discrete locations, and IORD_MSB can be used *
'       to read the 4 WY values (assuming that the module is installed in the *
'       slot that you wrote the 4 WY values to).                            *
'                                                                           *
'   Hardware Requirements:                                                  *
'       Series 500/505 PLC                                                  *
'       386/ATM COPROCESSOR                                                 *
'                                                                           *
'   Software Requirements:                                                  *
'       1.  Microsoft Quick Basic                                           *
'                                                                           *
'   Warnings:                                                               *
'       1.  THIS ROUTINE WRITES TO V-MEMORY AND I/O POINTS.                 *
'                                                                           *
'                                                                           *
'************************************************************************
    CONST FALSE = 0
    CONST TRUE = NOT FALSE
    DEFINT A-Z
    DIM Values(7)
                                    '************************************
                                    ' Clear the screen and display a    *
                                    ' message describing the program.   *
                                    '************************************
    CLS
    PRINT "PCWR_MSB:   Example usage of the PCCOMM command PCWRITE."
    PRINT
    PRINT "See the file PCWR_MSB.bas for a more complete description of the"
    PRINT "operation of this routine."
```

```
                                    '*************************************
                                    ' Print a warning message.           *
                                    '*************************************
PRINT
PRINT "WARNING: This program writes to V-memory and Y-registers!"
PRINT "Hit <space> to continue and any other key to exit."
PRINT
KeyHit$ = INKEY$
WHILE KeyHit$ = ""
    KeyHit$ = INKEY$
WEND
IF KeyHit$ <> " " THEN
    END
END IF
                                    '*************************************
                                    ' Open the device driver for reading  *
                                    ' and writing.                       *
                                    '*************************************
OPEN "PCCOMM" FOR OUTPUT AS #1
OPEN "PCCOMM" FOR INPUT AS #2
                                    '*************************************
                                    ' Prompt the user and read the V-mem  *
                                    ' start point from the keyboard.     *
                                    '*************************************
PRINT
PRINT "Enter the address of the first V-memory point to write to: ";
INPUT "", StartPoint
                                    '*************************************
                                    ' Allow the user to enter the 8 values *
                                    ' at the keyboard.                   *
                                    '*************************************
FOR LoopCounter = 0 TO 7
    PRINT "Enter the value to write at location ";
    PRINT LoopCounter + StartPoint; ": ";
    INPUT "", Values(LoopCounter)
NEXT
                                    '*************************************
                                    ' Write the request to the device    *
                                    ' driver.  Note that leading blanks  *
                                    ' are removed from the StartPoint    *
                                    ' and Values() via LTRIM$.           *
                                    '*************************************
RequestString$ = ":pw:vmem:" + LTRIM$(STR$(StartPoint)) + ":8"
FOR LoopCounter = 0 TO 7
    RequestString$ = RequestString$ + ":"
    RequestString$ = RequestString$ + LTRIM$(STR$(Values(LoopCounter)))
NEXT
RequestString$ = RequestString$ + "::"
PRINT #1, RequestString$
                                    '*************************************
                                    ' Get a response from the device     *
                                    ' driver.                            *
                                    '*************************************
LINE INPUT #2, ResponseString$
                                    '*************************************
                                    ' Skip to the 2nd token in the       *
                                    ' response string (it contains the   *
                                    ' number of values written).         *
                                    ' See GetToken$() at the bottom of   *
                                    ' this listing for more information. *
                                    '*************************************
Token$ = GetToken$(ResponseString$, TRUE)
Token$ = GetToken$(ResponseString$, FALSE)
InputCount = VAL(Token$)
```

```
                                    '*************************************
                                    ' Print an error message if the number *
                                    ' of values written does not equal 8.  *
                                    '*************************************
IF InputCount <> 8 THEN
    PRINT
    PRINT "The device driver was unable to write the 8 values!";
    PRINT
END IF
                                    '*************************************
                                    ' Prompt the user for input and read   *
                                    ' from the keyboard the location of    *
                                    ' the first Y register to write to.    *
                                    '*************************************
PRINT
PRINT "Enter the address of the first Y register to write to: ";
INPUT "", StartPoint
                                    '*************************************
                                    ' Allow the user to enter the 8 values *
                                    ' at the keyboard.  Any non-zero value *
                                    ' is written as a 1.                   *
                                    '*************************************
FOR LoopCounter = 0 TO 7
    PRINT "Enter the value to write at Y";
    PRINT LoopCounter + StartPoint; ": ";
    INPUT "", Values(LoopCounter)
NEXT
                                    '*************************************
                                    ' Write the request to the device      *
                                    ' driver.  Note that leading blanks    *
                                    ' are removed from the StartPoint      *
                                    ' and Values() via LTRIM$.             *
                                    '*************************************
RequestString$ = ":pw:YREG:" + LTRIM$(STR$(StartPoint)) + ":8"
FOR LoopCounter = 0 TO 7
    RequestString$ = RequestString$ + ":"
    RequestString$ = RequestString$ + LTRIM$(STR$(Values(LoopCounter)))
NEXT
RequestString$ = RequestString$ + "::"
PRINT #1, RequestString$
                                    '*************************************
                                    ' Get the response from the device     *
                                    ' driver and parse the count value     *
                                    ' from the response via GetToken.  See *
                                    ' GetToken$() at the bottom of this    *
                                    ' listing.                             *
                                    '*************************************
LINE INPUT #2, ResponseString$
Token$ = GetToken$(ResponseString$, TRUE)
Token$ = GetToken$(ResponseString$, FALSE)
InputCount = VAL(Token$)
                                    '*************************************
                                    ' Print an error message if the number *
                                    ' of values written does not equal 8.  *
                                    '*************************************
IF InputCount <> 8 THEN
    PRINT
    PRINT "The device driver was unable to write the 8 values!";
    PRINT
END IF
                                    '*************************************
                                    ' Prompt the user for input and read   *
                                    ' from the keyboard the location of    *
                                    ' the first WY to write to.            *
                                    '*************************************
PRINT
PRINT "Enter the address of the first WY register on the module: ";
INPUT "", StartPoint
```

```
                                    '************************************
                                    ' Allow the user to enter the 4 values *
                                    ' at the keyboard.                    *
                                    '************************************
    FOR LoopCounter = 0 TO 3
        PRINT "Enter the value to write at WY";
        PRINT LoopCounter + StartPoint; ": ";
        INPUT "", Values(LoopCounter)
    NEXT
                                    '************************************
                                    ' Write the request to the device     *
                                    ' driver.  Note that leading blanks   *
                                    ' are removed from the StartPoint     *
                                    ' and Values() via LTRIM$.            *
                                    '************************************
    RequestString$ = ":pw:WY:" + LTRIM$(STR$(StartPoint)) + ":4"
    FOR LoopCounter = 0 TO 3
        RequestString$ = RequestString$ + ":"
        RequestString$ = RequestString$ + LTRIM$(STR$(Values(LoopCounter)))
    NEXT
    RequestString$ = RequestString$ + "::"
    PRINT #1, RequestString$
                                    '************************************
                                    ' Get the response from the device    *
                                    ' driver and parse the count value    *
                                    ' from the response via GetToken.  See *
                                    ' GetToken$() at the bottom of this   *
                                    ' listing.                            *
                                    '************************************
    LINE INPUT #2, ResponseString$
    Token$ = GetToken$(ResponseString$, TRUE)
    Token$ = GetToken$(ResponseString$, FALSE)
    InputCount = VAL(Token$)
                                    '************************************
                                    ' Print an error message if the number *
                                    ' of values written does not equal 4.  *
                                    '************************************
    IF InputCount <> 4 THEN
        PRINT
        PRINT "The device driver was unable to write the 4 values!";
        PRINT
    END IF
    PRINT
END
```

```
'*************************************************************************
' Function Name: GetToken$                                               *
' Usage: Token$ = GetToken$(String1$, FirstTime)                         *
' Parameters:                                                            *
'        Token$: The token parsed from String1$ ("" if the end of the string  *
'                has been reached).                                      *
'        String1$: The string that is being parsed                       *
'        FirstTime: TRUE causes the function to begin parsing at the     *
'                   beginning of the string.                             *
'                  FALSE causes the function to parse the token following    *
'                   the token parsed on the previous call.               *
'                                                                        *
' Description:  This function extracts a token from String1$.  To parse the   *
'       first token from a string, pass a value of TRUE for the FirstTime     *
'       parameter.  To parse subsequent tokens from the string pass a value of*
'       FALSE for the FirstTime parameter.  For the purposes of this routine  *
'       a token is defined as a sequence of characters that have a preceding  *
'       ':' character and a following ':' character.  The ':' characters are  *
'       NOT returned with the token.                                     *
'                                                                        *
'*************************************************************************
FUNCTION GetToken$ (String1$, FirstTime) STATIC
                                    '************************************
                                    ' If this is the first call for this   *
                                    ' particular string then set index to  *
                                    ' point to beginning of string and     *
                                    ' skip over the initial ':' character. *
                                    '************************************
    IF FirstTime = TRUE THEN
        I = 1
        I = INSTR(I, String1$, ":")
        I = I + 1
    END IF
                                    '************************************
                                    ' If I is greater than the length of   *
                                    ' the string then return "" as the     *
                                    ' token.  Otherwise parse the token    *
                                    ' from the string and update I to      *
                                    ' point to the beginning of the next   *
                                    ' token.                               *
                                    '************************************
    IF I > LEN(String1$) THEN
        GetToken$ = ""
    ELSE
        J = INSTR(I, String1$, ":")
        TokenLength = J - I
        GetToken$ = MID$(String1$, I, TokenLength)
        I = J + 1
    END IF
END FUNCTION
```

B.10 GW-BASIC Program: IOREAD

```
1   '************************************************************************
2   ' iord_gw: Read the coprocessor module's WY values.                      *
3   '                                                                        *
5   ' Language:   Microsoft GW-Basic                                         *
6   ' Date:       11/13/90                                                   *
7   '                                                                        *
8   ' Description:  This routine demonstrates the usage of the PCCOMM service  *
9   '     command IOREAD.  The 4 local WY points will be read and displayed  *
10  '     to the screen.                                                     *
11  '                                                                        *
12  ' Suggestions:  You may want to run PCWR_GW prior to execution of this   *
13  '     program to verify that real values are being read by this routine. *
14  '     The last part of PCWR_GW allows the user to write to the WY values. *
15  '                                                                        *
16  ' Hardware Requirements:                                                 *
17  '     Series 500/505 PLC                                                 *
18  '     386/ATM COPROCESSOR                                                *
19  '                                                                        *
20  ' Software Requirements:                                                 *
21  '     1.  Microsoft GW-BASIC                                             *
22  '                                                                        *
23  ' Warnings:                                                              *
24  '                                                                        *
25  '                                                                        *
26  '************************************************************************
40   DEFINT A-Z
50   FALSE = 0
60   TRUE = NOT FALSE
70   DIM VALUES(3)
90                                      '***************************************
110                                     ' Clear the screen and display a       *
130                                     ' message describing the program.      *
140                                     '***************************************
150  CLS
160  PRINT "IORD_GW:  Example usage of the PCCOMM command IOREAD to read";
170  PRINT " from the module's"
180  PRINT "           WY points."
190  PRINT
200  PRINT "See the file IORD_GW.bas for a more complete description of the"
210  PRINT "operation of this routine."
220  PRINT
230                                     '***************************************
240                                     ' Open the device driver for reading   *
250                                     ' and writing.                         *
260                                     '***************************************
270  OPEN "PCCOMM" FOR OUTPUT AS #1
280  OPEN "PCCOMM" FOR INPUT AS #2
290                                     '***************************************
300                                     ' Write the request to the device      *
310                                     ' driver.                              *
320                                     '***************************************
330  PRINT #1, ":IR:5:4::"
340                                     '***************************************
350                                     ' Get the response from the device     *
360                                     ' driver.                              *
370                                     '***************************************
380  LINE INPUT #2, RESPONSESTRING$
390                                     '***************************************
400                                     ' Skip to the 2nd token in the response*
410                                     ' string (it contains the number of    *
420                                     ' values read).  See GetToken          *
425                                     ' subroutine (line 700).               *
430                                     '***************************************
440  STRING1$ = RESPONSESTRING$
450  FIRSTTIME = TRUE
460  GOSUB 700
470  FIRSTTIME = FALSE
480  GOSUB 700
```

```
490  INPUTCOUNT = VAL(TOKEN$)
500                                   '*************************************
510                                   ' Print an error message if the number *
520                                   ' of values read does not equal 4.     *
530                                   '*************************************
540  IF INPUTCOUNT = 4 THEN GOTO 610
550  PRINT
560  PRINT "The device driver was unable to read the 4 values!";
570  GOTO 690
580                                   '*************************************
590                                   ' Display the 4 values to the screen.  *
600                                   '*************************************
610     FOR LOOPCOUNTER = 0 TO 3
620         STRING1$ = RESPONSESTRING$
630         FIRSTTIME = FALSE
640         GOSUB 700
650         PRINT
660         PRINT "Location "; LOOPCOUNTER + 5; ":"; VAL(TOKEN$);
670     NEXT
680  PRINT
690  END
700  '************************************************************************
701  ' Subroutine Name: GetToken
702  ' Global Parameters:
703  '      String1$: The string that is being parsed
704  '      FirstTime: TRUE causes the subroutine to begin parsing at the
705  '                  beginning of the string.
706  '                 FALSE causes the subroutine to parse the token
707  '                  following the token parsed on the previous call.
708  '      Token$: The token parsed from String1$ ("" if the end of the string
709  '              has been reached).
710  ' Description:  This routine extracts a token from String1$ and places
711  '      it in Token$.  To parse the first token from a string, pass a
712  '      value of TRUE for the FirstTime parameter.  To parse subsequent
713  '      tokens from the string pass a value of FALSE.  For the purposes
714  '      of this routine a token is defined as a sequence of characters
715  '      that have a preceding ':' character and a following ':' character.
716  '      The ':' characters are NOT returned with the token.
717  '
718  '  Assumptions:
719  '      1. No program lines use the variable 'I' except this routine.
720  '
721  '************************************************************************
730                                   '*************************************
731                                   ' If this is the first call for this   *
732                                   ' particular string then set index to  *
733                                   ' point to beginning of string and     *
734                                   ' skip over the initial ':' character. *
735                                   '*************************************
760  IF FIRSTTIME = TRUE THEN GOTO 770 ELSE GOTO 800
770  I = 1
780  I = INSTR(I, STRING1$, ":")
790  I = I + 1
800                                   '*************************************
810                                   ' If I is greater than the length of   *
820                                   ' the string then return "" as the     *
830                                   ' token.  Otherwise parse the token    *
840                                   ' from the string and update I to      *
850                                   ' point to the beginning of the next   *
860                                   ' token.                               *
870                                   '*************************************
880  IF I > LEN(STRING1$) THEN TOKEN$ = "":GOTO 930
890  J = INSTR(I, STRING1$, ":")
900  TOKENLENGTH = J - I
910  TOKEN$ = MID$(STRING1$, I, TOKENLENGTH)
920  I = J + 1
930  RETURN
```

B.11 GW-BASIC Program: IOWRITE

```
'************************************************************************
' iowr_gw:  Write to the coprocessor module's WX values.                 *
'                                                                        *
' Language:   Microsoft GW-Basic                                         *
' Date:       11/13/90                                                   *
'                                                                        *
' Description:  This routine demonstrates the usage of the PCCOMM service  *
'     command IOWRITE.  The 4 local WX points will be written as specified *
'     by the user.                                                       *
'                                                                        *
' Suggestions:  You may want to run PCRD_GW after this program to verify   *
'     that the values were written correctly by this routine.  The last    *
'     part of PCRD_GW allows the user to read the WX values.             *
'                                                                        *
' Hardware Requirements:                                                 *
'     Series 500/505 PLC                                                 *
'     386/ATM COPROCESSOR                                                *
'                                                                        *
' Software Requirements:                                                 *
'     1.  Microsoft GW-BASIC                                             *
'                                                                        *
' Warnings:                                                              *
'                                                                        *
'                                                                        *
'************************************************************************
DEFINT A-Z
FALSE = 0
TRUE = NOT FALSE
DIM Values(3)
                                    '*************************************
                                    ' Clear the screen and display a     *
                                    ' message describing the program.    *
                                    '*************************************
CLS
PRINT "IOWR_GW:  Example usage of the PCCOMM command IOWRITE to write";
PRINT " to the module's"
PRINT "           WX points."
PRINT
PRINT "See the file IOWR_GW.bas for a more complete description of the"
PRINT "operation of this routine."
PRINT
                                    '*************************************
                                    ' Open the device driver for reading  *
                                    ' and writing.                       *
                                    '*************************************
OPEN "PCCOMM" FOR OUTPUT AS #1
OPEN "PCCOMM" FOR INPUT AS #2
                                    '*************************************
                                    ' Prompt the user and accept entry of *
                                    ' the 4 values that will be written to *
                                    ' the WX points.                     *
                                    '*************************************
FOR LoopCounter = 0 TO 3
    PRINT "Enter the value to write at location ";
    PRINT LoopCounter + 1; ": ";
    INPUT "", Values(LoopCounter)
NEXT
                                    '*************************************
                                    ' Write the request to the device     *
                                    ' driver.  Note that leading blanks   *
                                    ' are removed from Values() via       *
                                    ' RemoveBlanks subroutine (line 2000). *
                                    '*************************************
RequestString$ = ":iw:1:4"
FOR LoopCounter = 0 TO 3
    RequestString$ = RequestString$ + ":"
    String2$ = STR$(Values(LoopCounter))
    GOSUB 2000
```

```
530     RequestString$ = RequestString$ + String2$
540 NEXT
550 RequestString$ = RequestString$ + "::"
560 PRINT #1, RequestString$
570                                     '*************************************
580                                     ' Get a response from the device       *
590                                     ' driver.                              *
600                                     '*************************************
610 LINE INPUT #2, ResponseString$
620                                     '*************************************
630                                     ' Skip to the 2nd token in the response*
640                                     ' string (it contains the number of    *
650                                     ' values written).  See GetToken       *
655                                     ' subroutine (line 800).               *
660                                     '*************************************
670 String1$ = ResponseString$: FirstTime = TRUE: GOSUB 800
680 FirstTime = FALSE: GOSUB 800
690 InputCount = VAL(Token$)
700                                     '*************************************
710                                     ' Print an error message if the number *
720                                     ' of values written does not equal 4.  *
730                                     '*************************************
740 IF InputCount = 4 THEN GOTO 770
750 PRINT
760 PRINT "The device driver was unable to write the 4 values!";
770 PRINT
780 END
800  '*************************************************************************
801  ' Subroutine Name: GetToken
802  ' Global Parameters:
803  '      String1$: The string that is being parsed
804  '      FirstTime: TRUE causes the subroutine to begin parsing at the
805  '                 beginning of the string.
806  '                FALSE causes the subroutine to parse the token
807  '                 following the token parsed on the previous call.
808  '      Token$: The token parsed from String1$ ("" if the end of the string
809  '              has been reached).
810  ' Description:  This routine extracts a token from String1$ and places
811  '      it in Token$.  To parse the first token from a string, pass a
812  '      value of TRUE for the FirstTime parameter.  To parse subsequent
813  '      tokens from the string pass a value of FALSE.  For the purposes
814  '      of this routine a token is defined as a sequence of characters
815  '      that have a preceding ':' character and a following ':' character.
816  '      The ':' characters are NOT returned with the token.
817  '
818  '  Assumptions:
819  '      1. No program lines use the variable 'I' except this routine.
820  '
821  '*************************************************************************
829                                     '*************************************
830                                     ' If this is the first call for this   *
835                                     ' particular string then set index to  *
840                                     ' point to beginning of string and     *
845                                     ' skip over the initial ':' character. *
850                                     '*************************************
860 IF FIRSTTIME = TRUE THEN GOTO 870 ELSE GOTO 900
870     I = 1
880     I = INSTR(I, STRING1$, ":")
890     I = I + 1
900                                     '*************************************
910                                     ' If I is greater than the length of   *
920                                     ' the string then return "" as the     *
930                                     ' token.  Otherwise parse the token    *
940                                     ' from the string and update I to      *
950                                     ' point to the beginning of the next   *
960                                     ' token.                               *
970                                     '*************************************
980 IF I > LEN(STRING1$) THEN TOKEN$ = "":GOTO 1030
```

```
990 J = INSTR(I, STRING1$, ":")
1000 TOKENLENGTH = J - I
1010 TOKEN$ = MID$(STRING1$, I, TOKENLENGTH)
1020 I = J + 1
1030 RETURN
2000 '*************************************************************************
2001 '  Subroutine Name: RemoveBlanks
2002 '  Global Parameters:
2003 '       String2$: The string that leading blanks are removed from
2004 '  Description:  This routine removes leading blanks from String2$.
2005 '
2006 '  Assumptions:
2007 '       1.  No program lines use the variable 'I2' except this routine.
2008 '
2009 '*************************************************************************
2030 I2 = LEN(String2$)
2040 FirstChar$ = MID$(String2$, 1, 1)
2050 WHILE FirstChar$ = " "
2060    I2 = I2 - 1
2070    String2$ = RIGHT$(String2$, I2)
2085    FirstChar$ = MID$(String2$, 1, 1)
2090 WEND
2100 RETURN
```

B.12 GW-BASIC Program: PCREAD

```
1  '************************************************************************
2  ' pcrd_gw:  Read from V-mem, X registers and the coprocessor module's      *
3  '           WX points via PCREAD.                                          *
4  '                                                                          *
5  ' Language:   Microsoft GW-BASIC                                          *
6  ' Date:       11/12/90                                                     *
7  '                                                                          *
8  ' Description:  This routine demonstrates the usage of the PCCOMM service  *
9  '    command PCREAD.  V-memory, X registers and the module's WX points     *
10 '    will be read and displayed to the screen.                             *
11 '        The first part of the program will let the user read from 8       *
12 '    consecutive V-memory locations.  The user is prompted to enter an     *
13 '    integer value which specifies the first V-memory location to read     *
14 '    from.  Then the 8 values are displayed to the screen.  An error       *
15 '    message will be displayed if the device driver was unable to read the *
16 '    8 values from the PLC.                                                *
17 '        The second part of the program will let the user read 8 discrete  *
18 '    X register values.  The user is prompted to enter an integer value    *
19 '    which specifies the first X register location of the 8 to read from.  *
20 '    Then the 8 values are read and displayed on the screen.               *
21 '        The final section of the program will allow the user to read from *
22 '    the module's 4 WX locations.  The user is prompted to enter an        *
23 '    integer value which specifies the first WX register location of the   *
24 '    module.  Then the values are displayed to the screen.                 *
25 '                                                                          *
26 ' Suggestions:  You may want to run the routines PCWR_GW and IOWR_GW       *
27 '     prior to execution of this routine to verify that real values are    *
28 '     being read back from the PLC.  PCWR_GW can be used to write to       *
29 '     v-memory and discrete locations, and IOWR_GW can be used to write to *
30 '     the 4 WX values on the module.                                       *
31 '                                                                          *
32 ' Hardware Requirements:                                                  *
33 '     Series 500/505 PLC                                                   *
34 '     386/ATM COPROCESSOR                                                  *
35 '                                                                          *
36 ' Software Requirements:                                                  *
37 '     1.  Microsoft GW-BASIC                                               *
38 '                                                                          *
39 ' Warnings:                                                               *
40 '                                                                          *
41 '                                                                          *
42 '************************************************************************
48  DEFINT A-Z
49  FALSE = 0
50  TRUE = NOT FALSE
55                                    '*************************************
60                                    ' Clear the screen and display a      *
65                                    ' message describing the program.     *
70                                    '*************************************
80  CLS
90  PRINT "PCRD_GW:   Example usage of the PCCOMM command PCREAD."
100 PRINT
110 PRINT "See the file PCRD_GW.bas for a more complete description of the"
120 PRINT "operation of this routine."
130                                   '*************************************
140                                   ' Open the device driver for reading  *
150                                   ' and writing.                        *
160                                   '*************************************
170 OPEN "PCCOMM" FOR OUTPUT AS #1
180 OPEN "PCCOMM" FOR INPUT AS #2
190                                   '*************************************
200                                   ' Prompt the user and read the V-mem  *
210                                   ' start point from the keyboard.      *
220                                   '*************************************
230 PRINT
240 PRINT "Enter the address of the first V-memory point to read from: ";
250 INPUT "", StartPoint
```

```
260                                     '*************************************
270                                     ' Write the request to the device      *
280                                     ' driver.  Note that leading blanks    *
290                                     ' are removed from the StartPoint      *
300                                     ' value via RemoveBlanks subroutine    *
305                                     ' (line number 1950).                  *
310                                     '*************************************
320 String2$ = STR$(StartPoint)
330 GOSUB 1950
340 PRINT #1, ":pr:vmem:"; String2$; ":8::"
350                                     '*************************************
360                                     ' Get a response from the device       *
370                                     ' driver.                              *
380                                     '*************************************
390 LINE INPUT #2, ResponseString$
400                                     '*************************************
410                                     ' Skip to the 2nd token in the         *
420                                     ' response string (it contains the     *
430                                     ' number of values read).              *
440                                     ' See GetToken subroutine (line 1700). *
460                                     '*************************************
470 String1$ = ResponseString$: FirstTime = TRUE
480 GOSUB 1700
490 FirstTime = FALSE
500 GOSUB 1700
510 InputCount = VAL(Token$)
520                                     '*************************************
530                                     ' Print an error message if the number *
540                                     ' of values read does not equal 8.     *
550                                     '*************************************
560 IF InputCount = 8 THEN GOTO 630
570 PRINT
580 PRINT "The device driver was unable to read the 8 values!";
590 GOTO 740
600                                     '*************************************
610                                     ' Display the 8 values to the screen.  *
620                                     '*************************************
630 FOR LoopCounter = 0 TO 7
640     String1$ = ResponseString$: FirstTime = FALSE: GOSUB 1700
650     PRINT
660     PRINT "V-mem location"; StartPoint + LoopCounter; ":"; VAL(Token$);
670 NEXT
690                                     '*************************************
700                                     ' Prompt the user for input and read   *
710                                     ' from the keyboard the location of    *
720                                     ' the first X register to read from.   *
730                                     '*************************************
740 PRINT
750 PRINT
760 PRINT "Enter the address of the first X register to read: ";
770 INPUT "", StartPoint
860                                     '*************************************
870                                     ' Write the request to the device      *
880                                     ' driver.  Then get the response and   *
885                                     ' parse the count value from it.       *
890                                     '*************************************
900 String2$ = STR$(StartPoint): GOSUB 1950
910 PRINT #1, ":pr:XREG:"; String2$; ":8::"
930 LINE INPUT #2, ResponseString$
950 String1$ = ResponseString$: FirstTime = TRUE
960 GOSUB 1700
970 FirstTime = FALSE
980 GOSUB 1700
990 InputCount = VAL(Token$)
```

```
1000                                    '*************************************
1010                                    ' Print an error message if the number *
1020                                    ' of values read does not equal 8.     *
1030                                    '*************************************
1040 IF InputCount = 8 THEN GOTO 1110
1050 PRINT
1060 PRINT "The device driver was unable to read the 8 values!";
1070 GOTO 1160
1080                                    '*************************************
1090                                    ' Display the 8 values to the screen.  *
1100                                    '*************************************
1110 FOR LoopCounter = 0 TO 7
1120     String1$ = ResponseString$: FirstTime = FALSE: GOSUB 1700
1130     PRINT
1140     PRINT "X"; StartPoint + LoopCounter; ": "; VAL(Token$);
1150 NEXT
1160                                    '*************************************
1170                                    ' Prompt the user for input and read   *
1180                                    ' from the keyboard the location of    *
1190                                    ' the first WX to read from.           *
1200                                    '*************************************
1210 PRINT
1220 PRINT
1230 PRINT "Enter the address of the first WX register on the module: ";
1240 INPUT "", StartPoint
1330                                    '*************************************
1340                                    ' Write the request to the device      *
1350                                    ' driver.  Then get response and parse *
1355                                    ' the count value from it.             *
1360                                    '*************************************
1370 String2$ = STR$(StartPoint)
1380 GOSUB 1950
1390 PRINT #1, ":pr:WX:"; String2$; ":4::"
1410 LINE INPUT #2, ResponseString$
1430 String1$ = ResponseString$: FirstTime = TRUE
1440 GOSUB 1700
1450 FirstTime = FALSE
1460 GOSUB 1700
1470 InputCount = VAL(Token$)
1480                                    '*************************************
1490                                    ' Print an error message if the number *
1500                                    ' of values read does not equal 4.     *
1510                                    '*************************************
1520 IF InputCount = 4 THEN GOTO 1590
1530 PRINT
1540 PRINT "The device driver was unable to read the 4 values!";
1550 GOTO 1650
1560                                    '*************************************
1570                                    ' Display the 4 values to the screen.  *
1580                                    '*************************************
1590 FOR LoopCounter = 0 TO 3
1600     String1$ = ResponseString$: FirstTime = FALSE: GOSUB 1700
1610     PRINT
1620     PRINT "WX"; StartPoint + LoopCounter; ": "; VAL(Token$);
1630 NEXT
1650 PRINT
1670 END
```

```
1700 '************************************************************************
1701 ' Subroutine Name: GetToken
1702 ' Global Parameters:
1703 '      String1$: The string that is being parsed
1704 '      FirstTime: TRUE causes the subroutine to begin parsing at the
1705 '                 beginning of the string.
1706 '                FALSE causes the subroutine to parse the token
1707 '                 following the token parsed on the previous call.
1708 '      Token$: The token parsed from String1$ ("" if the end of the string
1709 '              has been reached).
1710 ' Description:  This routine extracts a token from String1$ and places
1711 '      it in Token$.  To parse the first token from a string, pass a
1712 '      value of TRUE for the FirstTime parameter.  To parse subsequent
1713 '      tokens from the string pass a value of FALSE.  For the purposes
1714 '      of this routine a token is defined as a sequence of characters
1715 '      that have a preceding ':' character and a following ':' character.
1716 '      The ':' characters are NOT returned with the token.
1717 '
1718 '  Assumptions:
1719 '      1. No program lines use the variable 'I' except this routine.
1720 '
1721 '************************************************************************
1730                                  '*************************************
1731                                  ' If this is the first call for this   *
1732                                  ' particular string then set index to  *
1740                                  ' point to beginning of string and     *
1750                                  ' skip over the initial ':' character. *
1760                                  '*************************************
1770 IF FIRSTTIME = TRUE THEN GOTO 1780 ELSE GOTO 1890
1780     I = 1
1790     I = INSTR(I, STRING1$, ":")
1800     I = I + 1
1810                                  '*************************************
1820                                  ' If I is greater than the length of   *
1830                                  ' the string then return "" as the     *
1840                                  ' token.  Otherwise parse the token    *
1850                                  ' from the string and update I to      *
1860                                  ' point to the beginning of the next   *
1870                                  ' token.                               *
1880                                  '*************************************
1890 IF I > LEN(STRING1$) THEN TOKEN$ = "":GOTO 1940
1900 J = INSTR(I, STRING1$, ":")
1910 TOKENLENGTH = J - I
1920 TOKEN$ = MID$(STRING1$, I, TOKENLENGTH)
1930 I = J + 1
1940 RETURN
1950 '************************************************************************
1951 '  Subroutine Name: RemoveBlanks
1952 '  Global Parameters:
1953 '      String2$: The string that leading blanks are removed from
1954 '  Description:  This routine removes leading blanks from String2$.
1955 '
1956 '  Assumptions:
1957 '      1.  No program lines use the variable 'I2' except this routine.
1958 '
1959 '************************************************************************
1970 I2 = LEN(String2$)
1980 FirstChar$ = MID$(String2$, 1, 1)
1990 WHILE FirstChar$ = " "
2000    I2 = I2 - 1
2010    String2$ = RIGHT$(String2$, I2)
2020    FirstChar$ = MID$(String2$, 1, 1)
2030 WEND
2040 RETURN
```

```
1   '************************************************************************
2   ' pcwr_gw: Write to V-mem, Y registers and the coprocessor module's        *
3   '          WY points via PCWRITE.                                         *
4   '                                                                         *
5   ' Language:   Microsoft GW-BASIC                                          *
6   ' Date:       11/12/90                                                    *
7   '                                                                         *
8   ' Description:  This routine demonstrates the usage of the PCCOMM service  *
9   '    command PCWRITE.  V-memory, Y registers and the module's WY points    *
10  '    will be written as specified by the user.                            *
11  '        The first part of the program will let the user write to 8       *
12  '    consecutive V-memory locations.  The user is prompted to enter       *
13  '    an integer value which specifies the first V-memory location to write *
14  '    to.  Then the user is prompted to enter 8 values which will be       *
15  '    written to consecutive V-memory locations starting with the location  *
16  '    previously specified.  An error message will be displayed if the     *
17  '    device driver was unable to write the 8 values to the PLC.           *
18  '        The second part of the program will let the user write 8 discrete *
19  '    Y register values.  The user is prompted to enter an integer value   *
20  '    which specifies the first Y register location of the 8 to write to.  *
21  '    Then the user is prompted to enter the 8 values which will be written *
22  '    to 8 consecutive Y registers starting with the location specified.   *
23  '    Any non-zero value will be written as a 1.                           *
24  '        The final section of the program will allow the user to write to *
25  '    the module's 4 WY locations.  The user is prompted to enter an       *
26  '    integer value which specifies the first WY register location of the  *
27  '    module.  Remember that the four WYs are located AFTER the 4 WXs.     *
28  '    Then the user is prompted to enter the 4 values which will be written *
29  '    to the 4 consecutive WY registers on the module.                     *
30  '                                                                         *
31  ' Suggestions:    Since this routine writes to various PLC memory locations*
32  '     you may want a means of reading back the locations to verify that the*
33  '     values were in fact written.  One means of doing this would be to run*
34  '     the example programs PCRD_GW and IORD_GW.  PCRD_GW can be used to    *
35  '     read the 8 v-memory and discrete locations, and IORD_GW can be used  *
36  '     to read the 4 WY values (assuming that the module is installed in the*
37  '     slot that you wrote the 4 WY values to).                            *
38  '                                                                         *
39  ' Hardware Requirements:                                                  *
40  '     Series 500/505 PLC                                                  *
41  '     386/ATM COPROCESSOR                                                 *
42  '                                                                         *
43  ' Software Requirements:                                                  *
44  '     1.  Microsoft GW-BASIC                                              *
45  '                                                                         *
46  ' Warnings:                                                               *
47  '     1.  THIS ROUTINE WRITES TO V-MEMORY AND I/O POINTS.                 *
48  '                                                                         *
49  '                                                                         *
50  '************************************************************************
54   DEFINT A-Z
55   FALSE = 0
60   TRUE = NOT FALSE
65   DIM Values(7)
70                                      '***************************************
75                                      ' Clear the screen and display a       *
80                                      ' message describing the program.      *
85                                      '***************************************
90   CLS
100  PRINT "PCWR_GW:   Example usage of the PCCOMM command PCWRITE."
110  PRINT
120  PRINT "See the file PCWR_GW.bas for a more complete description of the"
130  PRINT "operation of this routine."
131                                     '***************************************
132                                     ' Print a warning message.             *
133                                     '***************************************
134  PRINT
135  PRINT "WARNING: This program writes to V-memory and Y-registers!"
```

```
136 PRINT "Hit <space> to continue and any other key to exit."
137 PRINT
138 KeyHit$ = INKEY$
139 WHILE KeyHit$ = ""
140     KeyHit$ = INKEY$
141 WEND
142 IF KeyHit$ <> " " THEN END
177                                     '***************************************
178                                     ' Open the device driver for reading   *
179                                     ' and writing.                         *
180                                     '***************************************
190 OPEN "PCCOMM" FOR OUTPUT AS #1
200 OPEN "PCCOMM" FOR INPUT AS #2
220                                     '***************************************
230                                     ' Prompt the user and read the V-mem   *
240                                     ' start point from the keyboard.       *
250                                     '***************************************
260 PRINT
270 PRINT "Enter the address of the first V-memory point to write to: ";
280 INPUT "", StartPoint
300                                     '***************************************
310                                     ' Allow the user to enter the 8 values *
320                                     ' at the keyboard.                     *
350                                     '***************************************
360 FOR LoopCounter = 0 TO 7
370     PRINT "Enter the value to write at location ";
380     PRINT LoopCounter + StartPoint; ": ";
390     INPUT "", Values(LoopCounter)
400 NEXT
420                                     '***************************************
430                                     ' Write the request to the device      *
440                                     ' driver.  Note that leading blanks    *
450                                     ' are removed from the StartPoint      *
460                                     ' and Values() via RemoveBlanks        *
465                                     ' subroutine (line number 2950).       *
470                                     '***************************************
475 String2$ = STR$(StartPoint): GOSUB 2950
480 RequestString$ = ":pw:vmem:" + String2$ + ":8"
490 FOR LoopCounter = 0 TO 7
500     RequestString$ = RequestString$ + ":"
505     String2$ = STR$(Values(LoopCounter)): GOSUB 2950
510     RequestString$ = RequestString$ + String2$
520 NEXT
530 RequestString$ = RequestString$ + "::"
540 PRINT #1, RequestString$
560                                     '***************************************
570                                     ' Get a response from the device       *
580                                     ' driver.                              *
590                                     '***************************************
600 LINE INPUT #2, ResponseString$
620                                     '***************************************
630                                     ' Skip to the 2nd token in the         *
640                                     ' response string (it contains the     *
650                                     ' number of values written).  See      *
660                                     ' GetToken subroutine (line 2700)      *
680                                     '***************************************
690 String1$ = ResponseString$: FirstTime = TRUE: GOSUB 2700
700 FirstTime = FALSE: GOSUB 2700
710 InputCount = VAL(Token$)
730                                     '***************************************
740                                     ' Print an error message if the number *
750                                     ' of values written does not equal 8.  *
760                                     '***************************************
770 IF InputCount = 8 THEN GOTO 880
780 PRINT
790 PRINT "The device driver was unable to write the 8 values!";
800 PRINT
```

```
                                    '*************************************
                                    ' Prompt the user for input and read   *
                                    ' from the keyboard the location of    *
                                    ' the first Y register to write to.    *
                                    '*************************************
PRINT
PRINT "Enter the address of the first Y register to write to: ";
INPUT "", StartPoint
                                    '*************************************
                                    ' Allow the user to enter the 8 values *
                                    ' at the keyboard.  Any non-zero value *
                                    ' is written as a 1.                   *
                                    '*************************************
FOR LoopCounter = 0 TO 7
    PRINT "Enter the value to write at Y";
    PRINT LoopCounter + StartPoint; ": ";
    INPUT "", Values(LoopCounter)
NEXT
                                    '*************************************
                                    ' Write the request to the device      *
                                    ' driver.  Note that leading blanks    *
                                    ' are removed from the StartPoint      *
                                    ' and Values() via RemoveBlanks        *
                                    ' subroutine (line number 2950).       *
                                    '*************************************
String2$ = STR$(StartPoint): GOSUB 2950
RequestString$ = ":pw:YREG:" + String2$ + ":8"
FOR LoopCounter = 0 TO 7
    RequestString$ = RequestString$ + ":"
    String2$ = STR$(Values(LoopCounter)): GOSUB 2950
    RequestString$ = RequestString$ + String2$
NEXT
RequestString$ = RequestString$ + "::"
PRINT #1, RequestString$
                                    '*************************************
                                    ' Get response from device driver and  *
                                    ' parse the count value from reponse   *
                                    ' via GetToken subroutine (line 2700). *
                                    '*************************************
LINE INPUT #2, ResponseString$
String1$ = ResponseString$: FirstTime = TRUE: GOSUB 2700
FirstTime = FALSE: GOSUB 2700
InputCount = VAL(Token$)
                                    '*************************************
                                    ' Print an error message if the number *
                                    ' of values written does not equal 8.  *
                                    '*************************************
IF InputCount = 8 THEN GOTO 1490
PRINT
PRINT "The device driver was unable to write the 8 values!";
PRINT
                                    '*************************************
                                    ' Prompt the user for input and read   *
                                    ' from the keyboard the location of    *
                                    ' the first WY to write to.            *
                                    '*************************************
PRINT
PRINT "Enter the address of the first WY register on the module: ";
INPUT "", StartPoint
                                    '*************************************
                                    ' Allow the user to enter the 4 values *
                                    ' at the keyboard.                     *
                                    '*************************************
FOR LoopCounter = 0 TO 3
    PRINT "Enter the value to write at WY";
    PRINT LoopCounter + StartPoint; ": ";
    INPUT "", Values(LoopCounter)
NEXT
```

```
1740                                    '*************************************
1750                                    ' Write the request to the device      *
1760                                    ' driver.  Note that leading blanks    *
1770                                    ' are removed from the StartPoint      *
1780                                    ' and Values() via RemoveBlanks        *
1785                                    ' subroutine (line number 2950).       *
1790                                    '*************************************
1795 String2$ = STR$(StartPoint): GOSUB 2950
1800 RequestString$ = ":pw:WY:" + String2$ + ":4"
1810 FOR LoopCounter = 0 TO 3
1820     RequestString$ = RequestString$ + ":"
1825     String2$ = STR$(Values(LoopCounter)): GOSUB 2950
1830     RequestString$ = RequestString$ + String2$
1840 NEXT
1850 RequestString$ = RequestString$ + "::"
1860 PRINT #1, RequestString$
1862                                    '*************************************
1865                                    ' Get response from device driver and  *
1867                                    ' parse the count value from reponse   *
1870                                    ' via GetToken subroutine (line 2700). *
1875                                    '*************************************
1880 LINE INPUT #2, ResponseString$
1900 String1$ = ResponseString$: FirstTime = TRUE: GOSUB 2700
1910 FirstTime = FALSE: GOSUB 2700
1920 InputCount = VAL(Token$)
1940                                    '*************************************
1950                                    ' Print an error message if the number *
1960                                    ' of values written does not equal 4.  *
1970                                    '*************************************
1980 IF InputCount = 4 THEN GOTO 2040
1990     PRINT
2000     PRINT "The device driver was unable to write the 4 values!";
2010     PRINT
2040 PRINT
2060 END
2699 '****************************************************************************
2700 ' Subroutine Name: GetToken
2701 ' Global Parameters:
2702 '       String1$: The string that is being parsed
2703 '       FirstTime: TRUE causes the subroutine to begin parsing at the
2704 '                   beginning of the string.
2705 '                  FALSE causes the subroutine to parse the token
2706 '                   following the token parsed on the previous call.
2707 '       Token$: The token parsed from String1$ ("" if the end of the string
2708 '               has been reached).
2709 ' Description:  This routine extracts a token from String1$ and places
2710 '       it in Token$.  To parse the first token from a string, pass a
2711 '       value of TRUE for the FirstTime parameter.  To parse subsequent
2712 '       tokens from the string pass a value of FALSE.  For the purposes
2713 '       of this routine a token is defined as a sequence of characters
2714 '       that have a preceding ':' character and a following ':' character.
2715 '       The ':' characters are NOT returned with the token.
2716 '
2717 '  Assumptions:
2718 '       1. No program lines use the variable 'I' except this routine.
2719 '
2720 '****************************************************************************
2725                                    '*************************************
2727                                    ' If this is the first call for this   *
2730                                    ' particular string then set index to  *
2740                                    ' point to beginning of string and     *
2750                                    ' skip over the initial ':' character. *
2760                                    '*************************************
2770 IF FIRSTTIME = TRUE THEN GOTO 2780 ELSE GOTO 2890
2780     I = 1
2790     I = INSTR(I, STRING1$, ":")
2800     I = I + 1
```

```
2810                                    '************************************
2820                                    ' If I is greater than the length of   *
2830                                    ' the string then return "" as the     *
2840                                    ' token.  Otherwise parse the token    *
2850                                    ' from the string and update I to      *
2860                                    ' point to the beginning of the next   *
2870                                    ' token.                               *
2880                                    '************************************
2890 IF I > LEN(STRING1$) THEN TOKEN$ = "":GOTO 2940
2900 J = INSTR(I, STRING1$, ":")
2910 TOKENLENGTH = J - I
2920 TOKEN$ = MID$(STRING1$, I, TOKENLENGTH)
2930 I = J + 1
2940 RETURN
2950 '*****************************************************************************
2951 '  Subroutine Name: RemoveBlanks
2952 '  Global Parameters:
2953 '       String2$: The string that leading blanks are removed from
2954 '  Description:  This routine removes leading blanks from String2$.
2955 '
2956 '  Assumptions:
2957 '       1.  No program lines use the variable 'I2' except this routine.
2958 '
2959 '*****************************************************************************
2975 I2 = LEN(String2$)
2980 FirstChar$ = MID$(String2$, 1, 1)
2990 WHILE FirstChar$ = " "
3000    I2 = I2 - 1
3010    String2$ = RIGHT$(String2$, I2)
3020    FirstChar$ = MID$(String2$, 1, 1)
3030 WEND
3040 RETURN
```

Appendix C
Pinouts

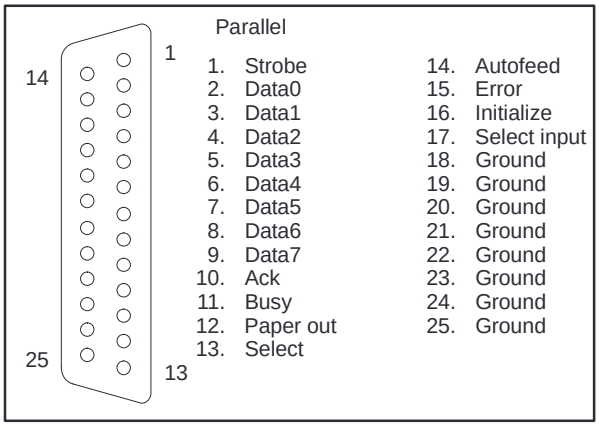

Figure C-1 Parallel Port Pinout

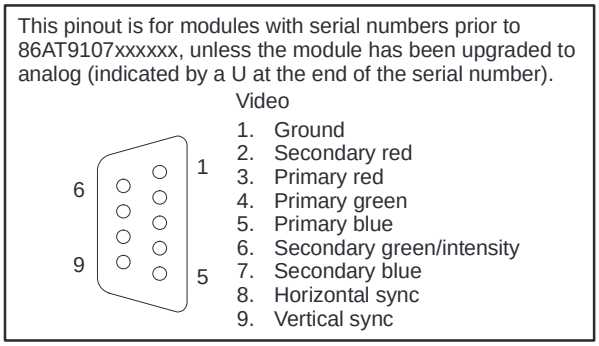

Figure C-2 TTL VGA Port Pinout

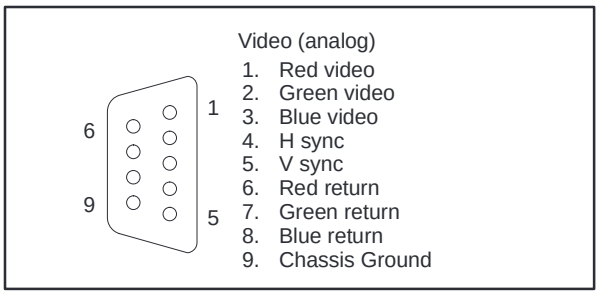

Figure C-3 Analog VGA Port Pinout

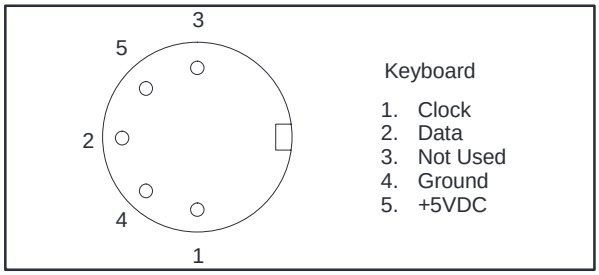

Figure C-4 Keyboard Connector Pinout

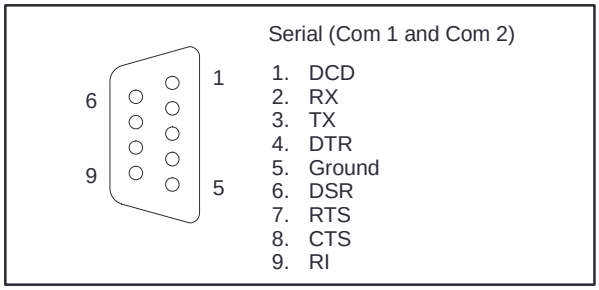

Figure C-5 Serial Port 1 and 2 Pinout

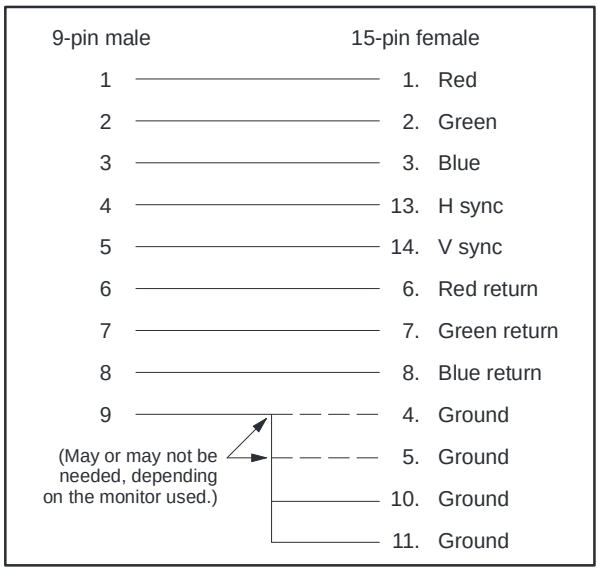

Figure C-6 9-pin Analog VGA to 15-Pin VGA Adapter Cable Pinout

Appendix D
Specifications

CPU	80C386SX (socket for 80C387SX math coprocessor)		
Memory DRAM w/parity Hard Disk	*Model –0220* 2M byte 20M byte type 2	*Model –0440* 4M byte 40M byte type 17	*Model –4120* 4M byte 120M byte type 41
Diskette Drive	3.5″ 720K byte/1.44M byte		
Operating System	Microsoft MS-DOS 5.0		
Communication Ports	2 Serial ports (RS-232/423); rates: 110 to 57600 baud (non-standard 5-volt operation) 1 Parallel printer port		
Other Ports	1 analog VGA port 1 IBM PC/AT-compatible keyboard port		
I/O Bus Communication	Integrated interface to the PLC I/O Bus		
Channels per Module	8 Analog I/O (4 WX, 4 WY)		
Data Communication Rate over PLC I/O Bus (maximum per PLC scan)	2048 bits + 8 analog I/O *or* 480 16-bit words + 8 analog I/O *or* combinations of both		
Power Consumption (Typical)	11 watts @ 5 VDC 0.2 watts @ –5 VDC		
Diagnostic	Internal diagnostic on power-up; Continuous DRAM parity check		
Operating Temperature	5° to 45°C (41° to 113°F)		
Storage Temperature	–40° to 60°C (–10° to 140°F)		
Relative Humidity	20% to 80%, non-condensing		
EMI	Meets MIL STD 461b RS01 and RS02–2		
Size	Triple-wide Series 505 I/O module		
Weight	1.6 Kg (3.3 lb.)		
Agency Certification Approvals	UL; CSA; FM Class I, Div. 2		

NOTE: During periods of high conducted or radiated electrical noise conditions, diskette access may cause seek and/or read/write errors. These errors do not affect the operation of other parts of either the 386/ATM or the programmable controller system. It is recommended that you start up with the diskette and then switch to the hard drive for operation. If you experience a seek or read/write error during a diskette access, please try the operation a second time. If the problem continues, wait for quiescent periods before performing diskette operations.