PC700/900/1100 Communications Interface

Application Note Revision 1.2

December 18, 1985

Westinghouse Electric Corporation

V1.2 18Dec85
V1.1 14Dec85
V1.0 12Dec85

Table of Contents

Section 1: Westinghouse Communications Protocol Introduction

The following discussion will help to describe the protocol necessary to communicate with a PC using one of the following methods:

- Westnet II Data Highway
- PC1100 Data Sidewalk
- Direct to Program Loader port

The Westnet II Data Highway is a token passing network that can interconnect up to 254 programmable controllers, color CRT's, computers, or mass storage devices on one 3 mile coaxial cable.

The PC1100 data sidewalk is built-in network of the PC1100 that permits up to 127 programmable controllers with one supervisory computer on one RS-232 based network.

Direct communications with a programmable controller is possible simply by connecting a computer to the program loader port found on the PC700, PC900 or PC1100.

A computer can communicate with a programmable controller by sending certain messages from the computer's serial port to the PC though the PC's program loader port. The programmable controller will examine the message and return the requested data to the computer. The program loader port is a standard RS-232 serial port. The computer can connect directly to the serial port or could be connected via modems or a network like Westnet II. Appendix E shows how to build your own cables for connecting your computer to the programmable controller, modem or Westnet II Programmable Controller Interface (PCI).

Messages sent to the programmable controller (PC) must conform to the proper format. If a message is improperly constructed, the remote PC will ignore it and return an error response.

If the computer is communicating to the programmable controller via the Westnet II data highway or the PC1100 data sidewalk, only the addressed PC will respond to a message. Communications via these networks is discussed in sections 3 (PC1100 data sidewalk) and section 7 (Westnet II Data Highway).

Most messages sent to the PC are six bytes long. The remote PC will wait until all six bytes are received before issuing a response. Since it is possible that noise on the communications line may insert or delete parts of the message, error detection is built into this communications protocol. This protocol is called "6 byte".

The error detection includes:

- parity checking of each byte
- checksum of each message
- allowable range checking of each byte
- byte counting
- acknowledgment of message

A remote PC expects most messages to contain six bytes. If the remote PC already has a byte or two in its receive buffer due to noise, then it only requires the number of bytes necessary to complete the six byte message. The other bytes from your computer will be discarded. Since there is no way to know how many bytes are in the receive buffer at any instant, a method must be available to clear the remote PC's communications buffer. This method is called "Synchronizing".

Under normal operating conditions, synchronization is only necessary if the remote PC returns error codes or does not return any data in response to a message.

Synchronization is not necessary when communicating via the Westnet II data highway since the programmable controller interface (PCI) attempts to synchronize with its connected programmable controllers.

See "Section 2 : Communication/Synchronization with PC700/900/1100 processors" for a description of the method of communicating directly with a PC from a computer.

See "Section 3 : Communication/Synchronization of PC1100 Multi-Drop Network" for a description of the method of communicating via the PC1100 data sidewalk.

See "Section 6 : PC700/900/1100 Communications Commands" for a description of the valid "6 byte" messages.

Section 2: Communication/Synchronization with PC700/900/1100 processors

To communicate from the serial port of your computer to the serial port of the PC:

1. Transmit from your computer's serial port to the programmable controller a valid "6 byte" message.
2. Wait for a response from the programmable controller. If no response is heard within 2 seconds or an error code is returned (see Section 6 : PC700/900/1100 Communications Commands), attempt to "synchronize" with the programmable controller.

3. Synchronize with PC by sending a null byte (00 Hex) while continuously looking for a response from the PC.
4. If no response detected after 1 second, send another null byte.
5. Continue in this manner until a response is received. If no response is received after 6 nulls have been transmitted, signal that a communications failure has occurred.

6. If a response from the programmable controller is detected, the return message will look like one of the following:

   6 bytes returned:

   XX XX XX XX XX XX

   where:

   XX - undefined bytes (value not important at this time)

   This is the normal response message from the programmable controller if no errors were detected. The fact that you received 6 bytes means two things:

   a. baud rates on the computer and programmable controller match.
   b. number of data bits, type of parity and number of stop bits are probably correct. (Certain mismatches may still occur in data bit selection, parity selection or stop bit selection that will still allow a correct response to be returned from the programmable controller. Transmit various valid 6 byte messages to the remote PC and observe the responses. If each message returns the correct response, you are synchronized with the PC.)

   2 bytes returned:

   XX XX

   This response message means that the programmable controller detected an invalid message sent to it. Your program may examine (parse) this returned error message. This message contains information as to the source of the error. See "Section 6 : PC700/900/1100 Communications Commands", or refer to the

Westinghouse Communications Manual; Catalog Number NLAM-B58.

5 bytes returned:

XX XX XX XX XX

This response message indicates that your computer was looking for a message length longer than that sent by the programmable controller. If the programmable controller is configured for 8 data bits and no parity, while your computer is looking for 8 data bits and parity, the sixth byte transmitted by the programmable controller will not include enough bits to complete a byte. Your computer will think that only 5 bytes have been transmitted.

1 byte returned:

XX

This response means that the baud rate of the remote PC is higher than the baud rate of the computer. All of the data bits returned only seemed like one byte to your computer. Typical problem when remote PC is set to 9600 baud, while the computer is communicating at 1200.

More than 6 bytes returned:

XX XX XX .....

This response occurs when the remote baud rate is slower than the local computer baud rate. Since one character transmitted to the remote PC is interpreted as less than one character by the remote PC, it may take several attempted messages to the remote PC before ANY response is heard. When the response finally returns, the returned message is seemingly very large since one character at the remote PC is transmitted slower (longer message).

****
Note: If system is configured as a polled multi-drop configuration utilizing the Unit Address function, the synchronization procedure is different. Refer to "Section 3 : Synchronization of PC1100 Multi-Drop Network".
****

Section 3 : Communication/Synchronization of PC1100 Multi-Drop Network

The unit address function of the PC1100 series programmable controller allows up to 128 processors to share a common Master-Slave network.

There can only be one master on the network with up to 127 slaves. The master can be either a computer or a PC1100 programmable controller. If a PC1100 is the master, the PC must be supplied with the Port Transmit (PT) function. All the PC1100 slaves must be supplied with the Unit Address (UA) function. A PC1100 with software revision 2.1 or later meets both these criteria. If the multi-drop network is to be connected via modems, software revision 2.3 or later is recommended.

PC1100's, V2.1 or later are supplied with two serial channels, Channel A and Channel B. Channel A cannot be used to connect to the multi-drop network. Channel A can be connected to the Westnet II data highway or to a computer/program loader directly. Channel B can be used to connect to the multi-drop network as either a Master or as a Slave or it can be configured as another standard serial port for the Westnet II data highway or computer/program loader.

If Channel B is configured for multi-drop operation and the PC has the proper Unit Address function programmed in ladder, the port will only respond to a properly addressed message. The standard program loaders will not be able to communicate with the PC over channel B since they do not send the proper "address code" to the port while channel B is configured for multi-drop.

If it is desired to program over the PC1100 "Data Sidewalk", an address code must be transmitted over the multi-drop network before attempting to communicate with that PC. For a description of a program that runs on the IBM PC that can be used to address a PC on a multi-drop network, refer to Appendix A. This program can be used to allow the IBM PC remotely program a PC using the standard program loader software packages "WESTAPL" or "DOC782".

Channel B is provided on two separate electrical interfaces, RS-232 and RS-485. RS-232 can be connected directly to a computer but is not suitable for a multi-drop configuration without using an external modem. RS-485 is suitable for a multi-drop configuration, permitting up to 32 PC's to be connected over 3 twisted shielded pairs.

Channel B, RS-232 can be connected via modems or multi-point line drivers to achieve the full 127 slave PC maximum. Refer to Section 5, "PC1100 Network using Modems" and to Appendix E, "Serial Port Wiring Diagrams".

The following diagram shows the physical placement of the equipment.

PC 1100 Data Sidewalk

```
--------------+-------------------+---------//---------+----
              !                   !                   !
        +-----+-----+       +-----+-----+       +-----+-----+
        !   Modem   !       !   Modem   !       !   Modem   !
        +-----+-----+       +-----+-----+       +-----+-----+
              !                   !                   !
        +-----+-----+       +-----+-----+       +-----+-----+
        !  PC 1100  !       !  PC 1100  !       !  PC 1100  !
        !           !       !           !       !           !
        !   PC1     !       !   PC2     !       !   PC3     !
        +-----------+       +-----------+       +-----------+

           Master              Slave 1             Slave N
```

1. Communication with a group of remote PC via the Unit Address Network follows a polled multi-drop scheme. The station initiating commands is called the master. All other stations are called slaves. Only one station may conveniently be the master in a network.

2. The synchronization method is different on the PC1100 data sidewalk. To synchronize all remote PC's (clear remote PC's communications buffers) it is necessary to drop the "clear buffer line" of the each PC. This line is located on the slave PC1100's serial ports. The master station should drop these lines before attempting to communicate with any of the slave stations. Once the line has been dropped, the next command that must be sent is the "Unit Address" command. This command will be acknowledged only by the addressed slave. If the addressed slave does not exist, no acknowledgment will be heard by the master. See "Appendix E : Serial Port Wiring Diagrams" for a description of the wiring used to connect PC1100's together in a data sidewalk.

3. The line to drop at the slave PC to clear its buffer depends on the software version of the slave PC.

   Clear Buffer Line

   V2.1 - CTS Pin 5
   V2.3 (and later) - DSR Pin 6

   This line must be pulled low for a minimum of 100 uS. If this line is pulled low for at least 100 uS, that PC's communication buffer will be cleared. Appendix E shows a method of wiring multiple PC1100's together via modems so that a computer can clear all slave PC's communications buffers simultaneously.

4. If using modems, a convenient method of dropping the "clear buffer line" is to tie the modem's carrier detect line (Pin 8 or pin 6 of the PC1100) to the appropriate "clear buffer line" (either pin 5 or pin 6). If the modem loses the carrier, it will

drop this carrier detect line. By wiring this line to the "Clear buffer line" of the PC, anytime the carrier is removed from the communications line, the slave PC's communications buffer will be cleared.

5. The master station must be able to control the transmit carrier through software. This is important since the removal of the carrier will clear each slave PC's communications buffers and thereby synchronize the network. The master only drops the transmit carrier if and only if the next command to be sent is the "Unit Address" command. If the master inadvertently drops the carrier during normal communications, a "Unit Address" command must be sent to re-address the remote PC. See Section 6 "PC700/900/1100 Communications Commands" for a discussion of the Unit Address Message.

6. The PC1100 does not support the Unit Address Function opcode on port A.

7. Modems can be used to extend the distance between the PC1100's located on the data sidewalk. If a PC1100 is used as a master, it must be programmed with the Port Transmit Function. Only the master is programmed with this function. Only the slaves are programmed with the Unit Address function. Both master and slave can be programmed with the Configure Port function if it is desired to communicate over the network at other than Numa-Logic standard format (9600 baud, 8 data bits, odd parity, 2 stop bits). It is not necessary to program a Configure Port command unless a different communications format is desired. The Configure Port command can also be used to modify the RTS/TxD time delay (see the comment 9).

8. If a modem is used on a Port Transmit master, special care must be used to wire the modem's RTS lead. The RTS lead is usually the pin that turns on and off the carrier. As described above (see comment 4), the switching of the master modem carrier is what synchronizes the network. Wire the modem RTS lead to a 24 VDC output card. A program must be written to pulse low this lead just before the port transmit statement is activated.

   The reason for doing this is to synchronize the network before the port transmit function activates. This 24VDC output card is not necessary if the RS-485 loop is used. Refer to the NL-1075 module Instruction Leaflet (I.L. 15753) for information on the correct wiring of the RS-485 port.

   Refer to Appendix E for a description of the method of wiring modems to the Port Transmit master using a 24 VDC output module.

9. Note that V2.3 and later requires CTS to be high before communications can proceed. This allows the use of modem with a longer RTS/CTS turnaround. Both versions (2.1 and 2.3+) raise RTS just before transmitting and lower RTS when the message has been sent. Version 2.1 does not require CTS to be high before transmitting. Both versions require DSR to be high before a

message can be transmitted. Both versions maintain DTR high during normal communications as a remote. Both versions pulse DTR low when that processor is used as a Port Transmit master. Version 2.3 also provides an expanded Configure Port (CP) function block in its instruction set. This new function block is programmed as follows:

```
!                                      +-------------+        !
! Enable                               !             ! CP0000 !
!--] [--------------------------------+ PORT        +--( )---+
!                                      !    0003     !        !
!                                      !             !        !
!                                      !             !        !
!                                      !             !        !
!                                      ! CONFIGURE   !        !
!                                      !  HR0009     !        !
!                                      !             !        !
!                                      !             !        !
!                                      +-------------+        !
```

Normally, the Configure Port command is used to set the baud rate, parity, data bit, and stop bit selection of either of the two serial ports. Note that is the configure port is set to "port 3", the configure register becomes a two register pair. The register shown retains the same function as before. The register immediately preceding that register contains a value from 1 to 128. This value is the number of program scans the processor waits before transmitting data AFTER the RTS line is raised high.

HR0008 Number of scans to delay
HR0009 Configuration register. (See PC1100 Advanced Programming Manual)

10. Once a Unit Address or Configure Port function is placed in memory and the enable line is pulsed on, the function can be deleted from memory and the port will continue to act as if the functions exist in ladder. If power is cycled or the unit is "Retested", the port will revert to the default conditions.

    If power is cycled and the functions exist in memory, the processor will find both functions and configure the port(s) appropriately. The functions do not need to be re-enabled after a power failure. Just the fact that the functions exist in the ladder program means that the processor will find them and configure the ports properly.

11. If a PC1100 fails to communicate with any device, verify that pin 20 of the serial port is high. If not:
    1. Remove power from the unit
    2. Disconnect the battery from the processor and allow to sit for several minutes.
    3. Power up the processor and verify that pin 20 of the RS-232 port is high. If it is, communications should proceed normally.
    4. Reconnect the battery and initialize and reload the PC.

Section 4a: Summary of PC1100 RS-232 Signals

Pin	Name	Description (PC1100 significance)
2	Transmitted Data	Data sent from PC to computer
3	Received Data	Data received from computer
4	Request to Send	Raised just before a response/message is transmitted from the PC
5	Clear to Send	V2.1 - If pulsed low for more than 100uS, communication buffer is cleared. V2.3+- Must be high before information is allowed to be transmitted from the PC
6	Data Set Ready	V2.1 - Must be high for data to be transmitted from the PC V2.3 - If pulsed low for more than 100uS, communication buffer is cleared.
7	Signal Ground	Signal common for all leads
8	Data Carrier Detect	No connection on port B
20	Data Terminal Ready	Master : Pulsed low by the PC for approximately 100 uS just before the Unit Address command is sent to the remote multi-drop network Remotes : Always held high

Refer to Section 4b "Summary of PC700/900 RS-232 Signals" for a description of the method of connecting PC1100's via radio modems.

Section 4b: Summary of PC700/900 RS-232 Signals

Pin	Name	Description
2	Transmitted Data	Data sent from PC to computer
3	Received Data	Data received from computer
4	Request to Send	Raised just before a response/message is transmitted from the PC
5	Clear to Send	Must be high for PC to transmit data
6	Data Set Ready	Must be high for PC to transmit data
7	Signal Ground	Signal common for all leads
8	Data Carrier Detect	No connection
20	Data Terminal Ready	Raised when PC is powered up.

If either pin 5 (CTS) or pin 6 (DSR) are dropped low, the PC will still receive (buffer) up to 6 bytes of data (up to 135 bytes for block write), but will not issue a response until both lines are high. Any messages sent to the PC before a previous message is acknowledged are discarded.

For radio modem applications, pin 4 (RTS) can be connected to the transmitter push to talk (PTT) relay. The transmitter must be equipped with a signal that says "transmitter enabled". This signal is wired onto pin 5 (CTS) to prevent the PC from transmitting data until the transmitter is enabled. If the radio does not have a "transmitter enabled" line, an external solid state timer can be used to delay raising the CTS lead. If spare DC I/O on the PC is available, the RTS lead could be wired to a spare DC input module and the CTS lead wired to a spare DC output module. See Appendix E, "Serial Port Wiring Diagrams" for more information.

If a radio modem is used with a PC1100 processor with V2.3 or later firmware, this delay can be performed internally using the "Configure Port" command.

Section 5: PC1100 Network using modems

```
   +---------+           +--------+            Four wire/Full Duplex
   !         !           !        ! Tx +
   ! Master  +2---------2+ Modem  +----------+
   !   PC    +3---------3+        ! Tx -     !
   !         +20--------4+ Full   +---------------+
   !         +5---------5+Duplex  ! Rx +     !    !
   !         +6---------8+        +----------------------+
   !         +7---------7+ Bell   ! Rx -     !    !      !
   !         !           ! 202    +---------------------------+
   !         !           !        !          !    !      !    !
   +---------+           +--------+          !    !      !    !
                                             !    !      !    !
   +---------+           +--------+          !    !      !    !
   !         !           !        ! Rx +     !    !      !    !
   ! Slave   +2---------2+ Modem  +----------+    !      !    !
   !         +3---------3+        ! Rx -     !    !      !    !
   !         +4---------4+        +---------------+      !    !
   !         +5---------5+        ! Tx +     !    !      !    !
   !         +6---------8+        +----------------------+    !
   !         +7---------7+        ! Tx -     !    !      !    !
   !         !           !        +---------------------------+
   !         !           !        !          !    !      !    !
   +---------+           +--------+          !    !      !    !
                                             !    !      !    !
                                             !    !      !    !
                                             !    !      !    !

                                         To additional slave PC's
```

Note that the connection of the slave PC's to the modems is shown for V2.3 or later software.

V2.1 software is not recommended for modem applications.*

Once a programmable controller has been synchronized, normal "6 byte" communications can take place. The term "6 byte" is used since all commands to the PC (except block write) are exactly six bytes in length.

Note that the master PC's modem Tx+/Tx- pair is connected to each of the slave PC modem's Rx+/Rx- pair. Each of the slave modem's Tx+/Tx- pairs is connected to the master modem's Rx+/Rx- pair.

The master PC does not have to be physically placed at one end of the network but can be located anywhere along the bus.

* Version 2.1 software will work with modems (or multi-point line drivers) that have extremely fast RTS/CTS turnaround time only. The main problem with V2.1 is that the V2.1 PC will transmit regardless of the state of the CTS lead. Once the V2.1 PC has raised RTS signaling it wants to transmit data, there is no convenient method of telling the PC that the modem is not ready. Version 2.3 and later correct this problem by reassigning the "clear buffer pin" to pin 6 (DSR), and using pin 5 of the PC to interrupt data transmission from the PC to a modem.

Section 6: PC700/900/1100 Communication Commands

The PC700/900/1100 programmable controllers all recognize and responds to, "6 byte" messages. The name "6 byte" comes from the fact that most messages sent to the programmable controller are six bytes long.

The messages include a byte telling the PC what this message is supposed to do. This byte is called the "opcode". The message must also tell the programmable controller where in the memory of the PC this function is supposed to be performed. Since the programmable controller has up to 64K of addressable space, the address portion of this message must contain 16 bits, or two bytes. These bytes are called the "address". If the function is writing data into the PC's memory, part of the message must tell the PC what value is to be written into memory. Again since memory locations (registers) are 16 bits wide, 16 bits or two bytes must be set aside in our message to send this data. These bytes are called "data".

Also included is an error byte. This sixth byte is transmitted at the end of the message. The purpose of this byte is to help the PC and computer distinguish a valid message from a message corrupted by noise.

The sixth error detecting byte is simply a sum of the previous five bytes.

If the message is somehow corrupted by noise and a few bits are lost, the checksum calculated by the receiving device will not match the checksum transmitted. It should be pointed out that a checksum is not designed to detect transposed digits or inserted nulls. Transposed digits are detected by comparing the response with the transmitted message. Transposed digits occur from programming errors and are not likely to occur as a result of noise. Inserted nulls are detected by the odd parity detection and by byte counting.

This is an example of a typical message:

```
     1          2          3          4          5          6
+----------+----------+----------+----------+----------+----------+
!  opcode  !   16 bit address    !   16 bit data       ! checksum !
+----------+----------+----------+----------+----------+----------+
```

The valid opcodes defined at this time are:

- 0000 0000 read a word
- 0000 0001 write a word
- 0000 0010 search for bit pattern
- 0000 0011 bit write
- 0000 0100 start monitor
- 0000 0101 block read
- 0000 0110 open memory bump
- 0000 0111 block write
- 0000 1000 close memory bump
- 0000 1001 highway command
- 0000 1010 unit address

These opcodes are described in greater detail in the "Communications Manual", catalog number NLAM-B58.

The address block of the message supports an addressing range of 0 to 65535. A description of the processors memory map is included later in this document to describe all valid addresses within each processor.

The data block contains information that further defines the opcode. In those cases where no further information is needed, these 16 bit words are described as a dummy bytes.

The checksum is a simple addition of the previous 5 bytes, discarding any carries past the eighth bit.

After this 6 byte command is transmitted to a synchronized PC, the PC will respond in one of two ways, with a response indicating message received, understood, and executed or with a response indicating some error condition was detected.

The following examples show the proper format of some typical messages sent to a programmable controller.

Read a word from the PC

Example : read location 8081 Hex from the PC memory

The only information required of a "read word" command is the opcode and the address inside the PC to be read.

Transmit the following sequence to the PC:

1st byte	0000 0000	op code (read word)
2nd byte	1000 0001	lower byte of address in PC memory
3rd byte	1000 0000	upper byte of address in PC memory
4th byte	0000 0000	dummy byte
5th byte	0000 0000	dummy byte
6th byte	0000 0001	checksum

If the 6 byte sequence is received without error (framing, overrun, parity, etc. See description of error codes later in this section), then the system will echo back the following:

1st byte	0000 0000	opcode
2nd byte	1000 0001	lower byte of address in PC memory
3rd byte	1000 0000	upper byte of address in PC memory
4th byte	XXXX XXXX	lower byte of data found at that location
5th byte	XXXX XXXX	upper byte of data found at that location
6th byte	XXXX XXXX	checksum

Write a word

A "write word" message requires the opcode, the location to write the data, as well as the data to be written. These three pieces of

information are all transmitted in the same message.

Example : write the value C030 at memory location 80A0 Hex

Transmit the following sequence to the PC:

1st byte	0000 0001	op code
2nd byte	1010 0000	lower byte of address in PC memory
3rd byte	1000 0000	upper byte of address in PC memory
4th byte	0011 0000	lower byte of data to be loaded
5th byte	1100 0000	upper byte of data to be loaded
6th byte	0001 0001	checksum

The processor will echo a write statement back to the sending unit exactly as received. This allows the sending unit to error check the data received to confirm that indeed the correct address and data were received.

Users may write programs for their computer to emulate this protocol. A program would load these values into a buffer, calculate a checksum, then transmit the 6 byte string out the serial port. A few milliseconds later a response from the remote PC should be heard. The program should continuously test the receive buffer for characters (1 second timeout), confirm checksum integrity on the received string, and unpack the embedded information found in the received data.

If a PC detects an error, rather than echoing back what was received (6 bytes), it will instead send back only two bytes:

1st byte XXXX YYYY
2nd byte (one's complement of 1st byte)

where:

XXXX - error code
YYYY - received OP code

The second byte is the one's complement of the first byte (all bits are inverted referenced to the first byte). If the processor sends two bytes in response to a message, but the two bytes are not the one's complement of each other, one of the following situations may have occurred:

1. The number of data, parity, or stop bits do not match between the computer and the programmable controller.
2. Baud rates do not match.
3. Severe noise on the line is corrupting the error response.

The following error codes are defined at this time:

0000	command implemented with no errors
0001	attempt to write ladder or parameter table data with key in program protect position
0010	invalid command OP code (verify that processor being used

supports the requested opcode. Some processors do not support block read, block write or unit address)

0011 checksum error (check program framing)

0100 command overrun (check program framing, PC was not able to execute previous command before next one received. This error can occur if part of the next message is transmitted before the previous message was acknowledged.

0101 command aborted

0110 UART overrun (check noise, matching of both clocks on block read/write command)

0111 invalid address (an attempt was made to write to memory that does not exist or is otherwise protected.)

1000 UART framing error (check noise, data formats, parity selection, baud rates)

1001 UART parity error (check noise, data formats, parity selection, baud rate selection

1111 Data Highway error (execute a "read highway status" command to determine cause of error). Data highway communications is discussed in Section 7, "Westnet II Communications Commands".

If an error code is received, try sending the command one more time. If the error persists, attempt a synchronization procedure. If communications cannot be verified following the synchronization procedure, a "communications failure" message or flag should be transferred to the calling program. Be sure and transfer the upper nybble of the first byte received (shows if remote PC detected the error) or some other descriptive information from your program (if local error) to assist in troubleshooting the communications link.

The standard asynchronous Numa-Logic data format is transmission with 1 start bit, 8 data bits, 1 odd parity bit, and 2 stop bits at 9600 baud. This can be changed via jumpers to fit other system requirements though. The PC1100 supports a ladder function (configure port) that permits changing these parameters through software.

Summary of Point to Point Communications Sequences

The following list gives a "6 byte" summary of the most commonly used messages sent to a programmable controller.

Command

READ(WORD)	Tx to PC:	00 LA HA DM DM CK
	Rx from PC:	00 LA HA LD HD CK
WRITE(WORD)	Tx to PC:	01 LA HA LD HD CK
	Rx from PC:	01 LA HA LD HD CK
WRITE(BIT)	Tx to PC:	03 LA HA BD DM CK
	Rx from PC:	03 LA HA BD DM CK
READ(BLOCK)	Tx to PC:	05 LA HA NU DM CK
	Rx from PC:	05 LA HA NU DM CK LD1 HD1 LD2 HD2 ... CD
WRITE(BLOCK)	Tx to PC:	07 LA HA NU DM CK LD1 HD1 LD2 HD2 ... CD
	Rx from PC:	07 LA HA NU DM CK
UNIT ADDRESS	Tx to PC:	0A AD 00 00 00 CK
	Rx from PC:	0A AD KY FL FH CK

Where:

LA - Low byte of address in PC
HA - High byte of address in PC
LD - Low byte of data transmitted to PC (write)
received from PC (read)
HD - High byte of data transmitted to PC (write)
received from PC (read)
CK - Checksum of previous 5 bytes
NU - Number of words minus one to be read from PC
LDx- Low byte of returned data byte number 'x'
transmitted data byte number 'x'
HDx- High byte of returned data byte number 'x'
transmitted data byte number 'x'
KY - Keyswitch position of addressed PC. See page 7 of Comm. Manual
FL - Low byte of addressed PC fault register
FH - High byte of addressed PC fault register
CD - Checksum of data bytes only (does not include command bytes)
BD - Bit Data. Configuration byte for "bit write" command

```
8   7   6   5   4   3   2   1
0   0   0   X   Y   Y   Y   Y
            !   +---+-+-+---+
            !         !
            !         +----  Binary number of bit
            !                number (0 - F)
            +--------------  0 - write zero to location
                             1 - write one to location
```

DM - Dummy variable (value not significant)

Section 7: Westnet II Communications Commands

The Westnet II data highway permits up to 254 programmable controllers, computers, and WDPF drops to share a common data base. This means that information can be passed easily among each other drop. Information is available two ways from this network. One, the intelligent device emulates a program loader, while the second method involves the intelligent device emulating a programmable controller.

Emulating a program loader is easier and more straight forward and is the method discussed here.

Hardware Overview

Each Westnet II Programmable Controller Interface (PCI) designed for the PC 700/900/1100 series processor has the following connections:

```
                          Westnet II Data Highway
      ------------+----------------------+--------------------
                  ! Drop 1               ! Drop 2
                  !                      !
          +-------+-------+        +-------+-------+
          !               !        !               !
          ! Westnet II    !        ! Westnet II    !
          !     PCI       !        !     PCI       !
          !               !        !               !
          +---------------+        +---------------+
            !     !     !            !     !     !
            !     !     !            !     !     !
Program     !     !     !            !     !     !       Program
Loader -----+     !     !            !     !     +----- Loader
                  !     !            !     !
PC #1  -----------+     !            !     +----------- PC #3 (Sub-
(Sub-Drop 0)            !            !                  Drop 1)
                        !            !
PC #2  -----------------+            +----------------- PC #4 (Sub-
(Sub-Drop 1)                                            Drop 0)
```

The above diagram show the topology of a typical Westnet system. All stations communicate via a coaxial cable. Up to 2 PC's and one computer or program loader can be connected to each PCI.

The steps required to communicate with a programmable controller via the Westnet II data highway are:

1. Address a drop and sub-drop that communications is to be directed to by sending a "Set Highway Address" command from your computer to the PCI. The PCI (not the PC) will acknowledge this message. Note that the PCI will acknowledge this message even if the requested drop or subdrop does not exist. Although the PCI acknowledges this message within a few milliseconds, the PCI has not attempted to link up with the requested drop yet. For this reason, your computer should pause for 100 mS (maximum) before

attempting to read data or session status from the addressed PC.

2. Read the Session Status of that drop to determine if any other drop is communicating with that particular drop. If the drop or subdrop does not exist, then no response will be heard from any session command. The computer should interpret no response as no drop or subdrop exists at that address. If the drop exists but another program loader or computer is already in session with the same addressed drop, your computer will receive a response giving the drop and subdrop address of that other computer or program loader. See the summary of Westnet II commands to show how you can tell from the response message which other drop is in session with the drop you have addressed.

3. (Optional) Open a session with that drop to lock out other stations that may attempt to communicate simultaneously with the same drop. The programmer must make the decision on whether simultaneous communications with a drop is to be permitted or prohibited. In a data acquisition system, multiple stations reading the same information at very nearly the same time may be permitted or even required, while it may be undesirable to have two stations downloading ladder to the same processor simultaneously.

4. Communicate with the drop using conventional "6 byte" protocol. All of the above commands (except unit address and data highway commands) are transmitted over the program loader port of the data highway exactly as if you were connected to the PC directly. Once a drop is sucessfully addressed, the PCI performs all tasks required to open a transparent link from your computer to the address drop. As far as your computer knows, you are connected directly to the addressed programmable controller.

5. (Optional) If your program "Opens a session" with a drop, all other devices are prohibited from communicating with that drop. If the programmer wishes other devices to have access with this drop again, the session must be "closed".

6. Repeat with step 1.

7. A PCI continuously polls the connected programmable controller, reading and writing to various memory locations within the PC.

8. If pin 5 or pin 6 of the PCI drops low, the PCI stops communicating with the PC.

9. If pin 5 and pin 6 remain high, but the PC does not respond to any messages from the PCI, the PCI continues trying to communicate for approximately 45 seconds. If after 45 seconds no response is heard from the PC, the PCI will stop trying. The PCI reset button (located on the MPI card) must be pressed to restart communications.

Summary of Westnet II Commands

Command

SET HWY ADDRESS	Tx to PC:	09 00 HW DR SB CK
	Rx from PC:	09 00 HW DR SB CK
READ REMOTE ADD	Tx to PC:	09 01 00 00 00 CK
	Rx from PC:	09 01 HW DR SB CK
READ HWY STATUS	Tx to PC:	09 02 00 00 00 CK
	Rx from PC:	09 02 EC 00 00 CK
OPEN SESSION	Tx to PC:	09 03 00 00 00 CK
	Rx from PC:	09 03 00 00 00 CK
CLOSE SESSION	Tx to PC:	09 04 00 00 00 CK
	Rx from PC:	09 04 00 00 00 CK
READ SESSION	Tx to PC:	09 05 00 00 00 CK
	Rx from PC:	09 05 SS DR SB CK
READ LOCAL ADD	Tx to PC:	09 06 00 00 00 CK
	Rx from PC:	09 06 HW DR SB CK

Where:

HW - Highway number (0 is all that is presently supported)
DR - Drop number (1 - 254)
SB - Sub-Drop number (0 or 1)
CK - Checksum

SS - Session Status
- 80 - no drop in session with this PC
- XX - highway number if drop in session with another

EC - Error Code
- 0 - no error found
- 1 - message buffer full
- 2 - invalid addressed drop (255 is invalid, for example)
- 4 - message size invalid (128 byte or 64 HR's is maximum allowed)
- 12 - PC in session with another drop. You are locked out.

Description of Highway Command codes

Set Address:

This code is used to set up a transparent channel from one drop of the highway to another. Once the "Set Address" command is acknowledged by the highway, conventional "6 byte" commands may be sent to the programmable controller. Communications proceed as if the computer were connected directly to the programmable controller.

Read Remote Address:
This function returns the address of the drop addressed on the highway.

Read Highway Status:
If a highway command is acknowledged with a 2 byte error response, execute this command to determine the cause of the highway error.

Open Session:
If the programmer wishes to lock out other stations during the time that his computer is communicating with the addressed drop, a "Session" may be opened.

Close Session:
A session must be closed with a drop to allow other locations the ability to communicate with that drop.

Read Session Status:
This command returns the address of the location in session with the addressed drop.

Read Local Address:
Returns the address of the drop your computer is plugged into.

Section 8: PC 700/900/1100 Memory Map

The PC700/900/1100 Memory Map is composed of five main areas:

- Holding Registers
- Ladder Memory (Program Memory)
- I/O Image
- Parameter Table
- Monitor Table

Holding Registers are usually used by the Programmable Controller for storage of numerical data. Since all numerical data is stored in binary form, certain applications may use Holding Registers to store groups of bits. A discussion later in this document shows some techniques on dealing with binary data in a packed digital form.

Program Memory starts at the Start of Program location and continues to just above the Highest Holding Register Used (HRRU) location (see memory map listed later). Each 16 bit memory location is referred to as a "word". Since one word of memory is used for each element in a PC's ladder diagram, each word address contains the information for completely describing a ladder element. Refer to Westinghouse for a description of the PC700/900/1100 data base used to store ladder. Ladder contact logic is stored in memory using the following format:

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
!16 !15 !14 !13 !12 !11 !10 ! 9 ! 8 ! 7 ! 6 ! 5 ! 4 ! 3 ! 2 ! 1 !
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
  !   !   !   !   !   +            11 bit address             +
  !   !   !   !   !
  !   !   !   !   +--------------- Return attribute if 1
  !   !   !   !
  !   !   !   +------------------- Up attribute if 1
  !   !   !
  !   !   +----------------------- Open attribute if 1
  !   !
  !   +--------------------------- N.C. contact if 1
  !
  +------------------------------- Contact is a S.F. if 1
```

For a discussion of attributes, refer to Westinghouse.

Note that the PC 700/900/1100 family of programmable controllers do not use user memory for the I/O image table. Separate memory is available and can be accessed through the RS-232 port. This memory is not keyswitch protected, thus permitting forcing with the key-switch in the RUN/PROGRAM PROTECT position. The I/O image table's position within the memory map varies from the PC 700 to the PC 900 and 1100. The exact location for each element (IR's, IG's, OR's, and OG's) can be determined through the processor Parameter Table. Refer to the Westinghouse "Communications Manual" for a more complete description of this table as well as other memory locations.

Description of I/O Image Memory

The programmable controller solves ladder rung, communicates with its various serial ports, and also communicates with it I/O racks. Part of the processors scan time is devoted to reading the data from input modules and placing this data into memory called the "input image table". The processor also reads a portion of memory called the "output image table" and uses this information to command the various output modules on or off.

The Westinghouse PC700/900/1100 processors conveniently group these image tables into groups. These groups are called Input Groups and Output Groups. A group is 16 bits of I/O data. Group numbering starts at one.

When the processor is communicating with analog or register I/O the concept is similar. The processor reads a block of data (a register) from one of these analog or register input cards and places this data into memory. This memory location is called an Input Register. The processor also reads a location in memory called an Output Register and writes this information to an analog or register output module.

OG - Output Group. OG1 represents outputs 1 through 16. OG2 represents outputs 17 through 32, etc. . Simply a group of 16 outputs all available in one register. This register, and those listed below, are located in the processors I/O image memory. This table's location can be found through the processor parameter table (described later).

IG - Input Group. IG1 represents the inputs 1 through 16, etc.

OR - Output Register. A register as defined by the PC is a 16 bit location in memory typically used for storing numerical data, although it can be used for discrete data. Output registers are used by the PC to transmit numerical data to analog or register (LED/LCD display) modules

IR - Input Register. Similar to an Output Register except the data received from analog or register input modules is placed in this location for use by other parts of the program.

All processors have a similar memory map. The major differences in locations between the three processors is the location of the I/O image table. Location of Holding registers, for example, always starts at absolute address 0000H and extends to the "Highest Holding Register Used" (HRRU) pointer (value of this pointer can be found by reading absolute memory address 8201H). The start of ladder always starts at the "Top of Memory" (dependent on the amount of memory available in the particular processor) and extends DOWNWARD to the bottom of memory. Note that HR's start at address 0000H and work up, while ladder starts at the top of memory and works down. The value of (EOP)-(HRRU) gives the amount of free memory available.

In general, the PC700/900/1100 programmable controllers have the following memory organization:

```
------ 16 bits wide -----

+-----------------------+ 0000H
! Holding Registers     !
!                       !
+-----------------------+ Highest Holding Register Used (HRRU)
:                       :   :       :
:                       : Unused Memory
:                       :   :       :
+-----------------------+ End of Program (EOP)
! Ladder Program        !
!                       !
+-----------------------+ Start of Program (varies with memory)
: * Memory Void *       :
+-----------------------+ 8000H
! I/O Image Table       !
!                       !
+-----------------------+ 80BFH (varies with processor type)
: * Memory Void *       :
+-----------------------+ 8200H
! Parameter Table       !
!                       !
+-----------------------+ 821FH
: * Memory Void *       :
+-----------------------+ 84B8H
! Monitor Table         !
!                       !
+-----------------------+ 84FFH
: * Memory Void *       :
+-----------------------+ FFFFH
```

To better describe this memory mapping, let's examine the PC700 in greater detail.

The PC700 processor has the following memory organization. Each memory location is 16 bits wide. The memory maps of the PC900 and PC1100 are similar except for the location of the I/O image (IG's, OG's, IR's, OR's). A quick reading of the PC's parameter table locations 8213H, 8212H, 820EH, and 820DH, respectively, would show their exact location (see description of PC parameter table listed below).

Absolute Memory Addresses (PC700)

Address				Description
Hex	Decimal	2-byte	Decimal	
0000	0000	00	00	HR0001
0001	0001	00	01	HR0002
0002	0002	00	02	HR0003

XXXX	XXXX	XX	XX	Highest Holding Register used (HRRU)	
--	--	--	--	------	
--	--	--	--	------	
--	--	--	--	Unused Memory (Available for Ladder or HR use)	
--	--	--	--	------	
--	--	--	--	------	
YYYY	YYYY	YY	YY	End of program (EOP) *	
7FFF	32767	127	255	Largest memory permitted.	
--	--	--	--	------	
8000	32768	128	00	IG0001	
8001	32769	128	01	IG0002	Discrete Inputs
8002	32770	128	02	IG0003	
800F	32783	128	15	IG0016	
--	--	--	--	------	
8040	32832	128	64	OG0001	
8041	32833	128	65	OG0002	Discrete Outputs
8042	32834	128	66	OG0003	
805F	32863	128	95	OG0032	
--	--	--	--	------	
8080	32896	128	128	IR0001	
8081	32897	128	129	IR0002	Register Inputs
8082	32898	128	130	IR0003	
809F	32927	128	159	IR0032	
--	--	--	--	------	
80A0	32928	128	160	OR0001	
80A1	32929	128	161	OR0002	Register Outputs
80A2	32930	128	162	OR0003	
80BF	32959	128	191	OR0032	
--	--	--	--	------	
8200	33280	130	00	**********	
				Parameter Table	
821F	33311	130	31	**********	
--	--	--	--	------	
FFFF	65535	255	255	End of Memory	

* Ladder program is written from the Top of Memory, down to the End of Program address. As the ladder program grows, the End of Program address approaches the Highest Holding Register Used pointer.
In other words, the processor continually optimizes the available memory between Registers and Ladder Logic. As the ladder requirements grow, the available Holding Registers decrease to allow for the expansion. On the other hand, if only a few registers are needed for the program, the rest can be converted into "Ladder memory".

Binary to "2 byte" Decimal

If only one bit of a byte is on, it is relatively easy to recognize which bit is set. Each bit contributes a value to the decimal number equal to 2 raised to the power of the bit number minus one, if the least significant bit (LSB) is considered bit 1. For example, if the 2nd bit of the byte is on, the decimal number representing this binary bit is 2. If 16 bit data is to be dealt with, the procedure is not changed except that the conversion must be done with both the upper and lower bytes. The following table describes this concept in more detail.

Summary of Binary / "2 byte" decimal Conversions

Following a word read, the programmable controller will return 6 bytes of data. The fourth and fifth bytes contain the lower and upper bytes found a the requested memory location inside the PC.

In the following table, the fourth byte is represented by "L" and the fifth byte as "H" (for low byte and high byte).

		Bit Number															
H	L	Upper Byte								Lower Byte							
		16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	8	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	16	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	32	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	64	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	128	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
64	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
128	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

If only one bit was energized in the register, the fourth and fifth data bytes returned would look like those shown in the table.

The procedure of reading from, or writing to, a specific bit in a PC representing a control relay (CR), is similar to that used with inputs. In this case, however, output groups rather than input groups are used. Also, since it is desirable to write to just one output at a time, the "Bit Write" opcode should be used.

Bit Number

	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
OG01	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
OG02	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17
OG06	96	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81

These and other addresses are more fully discussed in the Westinghouse "Communications Manual", catalog #NLAM-B58.

Section 9: PC 700/900/1100 Parameter Table

The PC700/900/1100 all set aside a 32 word (16 bit wide) memory block that contains various pieces of information used by the processor. This table is always located at absolute memory address 8200H and extends to 821FH. A user's program may access this table or even write data to it. The following description gives an idea of the type of information stored at these locations. If the processor keyswitch is in the "Run/Program Protect" position, you cannot write to the parameter table.

8200H	End of Program Location
8201H	Highest Holding Register Used (HRRU)
8202H	Low Ladder Checksum
8203H	High Ladder Checksum
8204H	Register Checksum
8205H	H:Mode Register L:Flag Register
8206H	Error Register
8207H	Bits 16 - 3 Reserved Bit 2: Ladder Checksum override Bit 1: Keyswitch override
8208H- 820BH	Reserved
820CH	Monitor Table Address
820DH	Output Register Address
820EH	Input Register Address
820FH	H:Number of Input Registers L:Number of Output Registers
8210H	Discrete Output Force Table Address
8211H	Discrete Input Force Table Address
8212H	Output Status Table Address
8213H	Discrete Input Status Table Address
8214H	H:Number of Inputs/8 L:Number of Outputs/8
8215H	H:Memory Size/256 L:Maximum Discrete Coils/8
8216H	H:Product Line Code/Product Line Modifier L:Software Version Number
8217- 821FH	Special Functions Allowed (see page 5 of "Communications Manual" for description of this table.)

Section 10: IBM PC Communications via BASICA

The IBM Personal Computer DOS diskette includes an advanced BASIC interpreter called BASICA. This language includes some powerful serial communications commands that make the programmers job easier.

Refer to Appendix E for an example of the cable necessary to connect between your computer and the programmable controller, modem or Westnet II PCI.

Configuring the Serial Port

You need to determine the following parameters before you can configure a serial port for asynchronous communications. These parameters are:

1. Baud Rate
2. Number of bits per character
3. Type of parity
4. Number of stop bits

Numa-Logic programmable controller default to 9600 baud, 8 data bits, odd parity, and 2 stop bits. Let's use these parameters in our BASICA program.

BASICA uses the statement 'OPEN "com1:....' to configure the serial port. Using the Numa-Logic default parameters:

```
OPEN "com1:9600,n,8,2" as #2
```

This statement opens up the COM1 serial port, assigns it to file #2 and raises RTS and DTR. One problem with this statement is that BASICA is unable to OPEN a communications file using 8 data bits and any parity other than none. We can code around this by inserting the following statements after the OPEN statement:

```
T%=INP(1019) ' read the COM1 line control register
T%=T% OR 8   ' set the 4th bit (enable parity)
OUT 1019,T%  ' send new line control register to 8250 UART
```

If you are using COM2 instead of COM1, the address of the line control register is 763 rather than 1019.

The IBM BASICA is now configured for 8 data bits AND odd parity.

Transmitting Characters

Let's assume that we want to transmit a "read HR0001" message to the PC. Referring back to "Section 6 : PC700/900/1100 Communications Commands" we note that the correct six byte message is:

```
00 00 00 00 00 00
```

Simple enough. Now how do we transmit this out the serial port?

With the "PRINT #n" command, that's how. Using the above example, we code the following statement:

```
PRINT#2, CHR$(0);CHR$(0);CHR$(0);CHR(0);CHR$(0);CHR$(0);
```

Note the use of the semicolon. The semicolon suppresses the carriage return/line feed (0D, 0A Hex) normally sent at the end of a print statement. The "CHR$" is used to convert the ASCII "0" to a binary zero. For more flexibility we may wish to replace the "0's" with variable names so that we may change the message just by changing the variable value. For example, let's define each of the six bytes with a descriptive name:

Byte	Variable	Description
1	OP%	The opcode describes what the message does
2	LA%	The next byte is the low byte of the address
3	HA%	The next is the high byte of the address
4	LD%	This is the low byte of the data sent
5	HD%	And the high byte of data sent to the PC
6	CK%	The checksum of the previous five bytes

```
PRINT#2, CHR$(OP%);CHR$(LA%);CHR$(HA%);CHR$(LD%);CHR$(HD%);
         CHR$(CK%)
```

The calculation of the checksum is simple enough:

```
CK%=OP%+LA%+HA%+LD%+HD% 'sum up previous 5 bytes
CK%=CK% AND 255         'discard data past the 8th bit
```

To conclude, let's put all of these statements together.

```
 10 ' Transmit a "Read HR0001 message out the COM1 serial port
 20 '
 30 OPEN "com1:9600,n,8,2" as #2
 40 T%=INP(1019)
 50 T%=T% OR 8
 60 OUT 1019,T%
 70 '
 80 OP%=0:LA%=0:HA%=0
 90 '
100 GOSUB 200 ' calculate checksum
110 '
120 PRINT#2, CHR$(OP%);CHR%(LA%);CHR$(HA%);CHR$(LD%);CHR%(HD%);
    CHR$(CK%);
130 END
200 '
210 'calculate the checksum
220 CK%=OP%+LA%+HA%+LD%+HD
230 CK%=CK% AND 255
240 RETURN
```

Read Data from the Serial Port

IBM BASICA makes it easy to read data from a serial port since the language will buffer up to 256 characters before being overrun. If more buffer capacity is needed, BASICA can be invoked with the /C: option. Using this option, up to 32K of buffer space can be set aside. Refer to your BASICA manual for more details.

To receive those characters loaded into the communications buffer, the BASICA INPUT$(x,y) is used, where x is the number of characters to be removed from the buffer during each read and y is the file buffer to read characters from. For example, read one character from file #2:

```
Z$=INPUT$(1,2)
```

When this statement is executed, one character from file #2 will be loaded into the string variable Z$. For convenience, let's use an array variable to receive the data from the buffer:

```
Z$(I)=INPUT$(1,2)
```

To read all the data from the buffer, we need to know when the buffer has data to be read. The BASICA EOF(n) statement performs that task. For example:

```
WHILE EOF(2)=0
Z$(I)=INPUT$(1,2):Y%(I)=ASC(Z$(I)):I=I+1
WEND
```

The WHILE/WEND loop will continue to read characters from the receive buffer as long as EOF(2) equals zero (more data available). Once EOF(2) equals negative one (signaling no more data in buffer), the program drops out of the WHILE/WEND loop.

In summary, a simple and concise program to read data returned from a programmable controller could be:

```
300 'Routine to read data from the file buffer #2 (COM1)
310 '
320 I=0
330 WHILE EOF(2)=0
340 Z$(I)=INPUT$(1,2):Y%(ASC(Z$(I)):I=I+1
350 WEND
```

The above programming examples assume that the computer will always initiate messages and will know when to expect data (poll/response). If the programmer must write a program that will expect data at any time, the built-in interrupt features of BASICA can be used.

If it is desired to input a value from the keyboard to be loaded to the programmable controller, and additional routine to convert an integer value into two, 8 bit bytes must be shown. Note carefully the following program, especially line 310, which shows the statement necessary to perform the conversion. This same statement

can be used to convert any 16 bit integer into two, 8 bit bytes. This is handy for converting both data and address words sent to the PC that have been entered from the keyboard.

```
 10 ' Configure serial port for 9600, 8 data, odd parity, 2 stop
 20 '
 30 OPEN "com1:9600,n,8,2" as #2
 40 T%=INP(1019)
 50 T%=T% OR 8
 60 OUT 1019,T%
 70 '

100 ' Program to prompt for address and data in PC
110 '
120 INPUT"Read or write (R/W) ";A$
130 IF A$="r" OR A$="R" THEN OP%=0:GOTO 160
140 IF A$="w" OR A$="W" THEN OP%=1:GOTO 160
150 GOTO 120
160 INPUT"Enter address ";X%
170 GOSUB 300 :' CONVERT TO TWO 8 BIT BYTES
180 LA%=L%:HA%=H%
190 IF OP%=1 THEN 230
200 INPUT"Enter data ";X%
210 GOSUB 300 :' CONVERT TO TWO 8 BIT BYTES
220 LD%=L%:HD%=H%
230 GOSUB 400 :' TRANSMIT DATA OUT THE SERIAL PORT
240 GOSUB 500 :' GET RETURNED DATA FROM RCV BUFFER
250 PRINT"Returned opcode   = ";OP%
260 PRINT"Returned address  = ";AD%
270 PRINT"Returned data     = ";DA%
280 GOTO 120

300 ' Convert 16 bit integer to two 8 bit bytes
310 H%=X% 256:L%=X%-(H%*256)
320 RETURN

400 '
410 GOSUB 450 ' calculate checksum
420 '
430 PRINT#2, CHR$(OP%);CHR%(LA%);CHR$(HA%);CHR$(LD%);CHR%(HD%);
    CHR$(CK%);
440 RETURN
450 '
460 'calculate the checksum
470 CK%=OP%+LA%+HA%+LD%+HD
480 CK%=CK% AND 255
490 RETURN

500 'Routine to read data from the file buffer #2 (COM1)
510 '
520 I=0
530 WHILE EOF(2)=0
540 Z$(I)=INPUT$(1,2):Y%(ASC(Z$(I)):I=I+1
550 WEND
560 RETURN
```

Interrupt Based Communications

IBM BASICA has additional functions that permit the communications task to exist somewhat in the background of the main program. These functions allow, among other things:

- The program to execute lines of code while characters are being received or transmitted via the serial port

- Maintain communications over two serial ports "simultaneously" without the necessity of waiting until one remote station is finished downloading data to your computer.

BASICA provides two statements that permit activity on one of the serial ports to "interrupt" the BASICA program. The handler will buffer the characters, then transfer control (GOSUB) to another portion of the program for receive data processing. These BASICA statements are:

- ON COM(n) : Enable interrupt

- ON COM(n) GOSUB xxxx : Branch to line xxxx on receive data

These statements are not used extensively for Numa-Logic interface routines since the programmable controller is nominally an interrogate-only device. The PC will not send data to you unless it is asked for.

Refer to the IBM (or Micro-Soft) BASICA (or GWBASIC) manual for more information on interrupt driven communications.

Refer to Appendices A,B and C for examples of programs written using IBM (MicroSoft) BASICA.

Section 11: Apple][+ Communications via Super Serial Card

At the time this was written, APPLESOFT BASIC did not support any high level commands that permitted easy reading or writing through the serial port provided on the "Apple Super Serial Card" (catalog # A2L0044). The user could use INPUT and PRINT statements for communications, but these impose severe limitations on the programmer who needs to communicate with a binary protocol.

The only solution was to program the communications handlers in assembly language and use the APPLESOFT CALL statement to transfer control to these handlers when necessary. Due to the complexity of the assembly code, and the fact that an understanding of the code is not necessary to use the functions, Appendix H was set aside for an in-depth listing of all the assembly handlers used.

The program makes the following assumptions about the users computer:

1. 48K memory or more
2. Super Serial Card located in slot 1

Transmitting a 6 byte sequence to the Programmable Controller

The APPLESOFT programmer has an easy way to send a six byte command to the programmable controller connected to the Apple][+ Super Serial Port using an APPLESOFT CALL statement. For our example, assume that we want to send the command "read HR0001" to the programmable controller. We remember (from section 6) that the six byte message is:

Read HR0001: 00 00 00 00 00 00

The programmer must load, via the POKE statement each of the six bytes to be transmitted. The following table gives the names and POKE addresses that are used:

Byte	POKE Address	Description
1	28928 (7100h)	opcode
2	28929 (7101h)	low byte of PC address
3	28930 (7102h)	high byte of PC address
4	28931 (7103h)	low byte of data to be sent to PC
5	28932 (7104h)	high byte of data to be sent to PC
6	28933 (7105h)	checksum

These variables may be loaded with the APPLESOFT POKE statement.

```
100  POKE 28928,OP : REM  load the opcode
110  POKE 28929,LA : REM  load the low byte of the address
120  POKE 28930,HA : REM  load the high byte of the address
130  POKE 28931,LD : REM  load the low byte of data sent
140  POKE 28932,HD : REM  load the high byte of data sent out
```

Note that it is not necessary to load the value of the checksum. Another CALL statement has been provided to do that. To calculate the checksum of the five bytes loaded at 28928 through 28932 inclusive, simply CALL 24880. This routine calculates the proper checksum and loads the value at 28933 (7105h).

Once the data is loaded into the proper memory locations with the POKE statements, it is a simple matter to execute a CALL statement to transmit this data out the serial port. This statement is:

```
CALL 26112 : REM  transmit 6 byte sequence out the serial port
```

The assembly language routine will send the data out the serial port and wait for a six byte response from the PC. If six bytes are not returned, the program gets any data that was returned and sets an error flag. The returned data is loaded into these memory locations:

Byte	PEEK Address	Description
1	28934 (7106h)	returned opcode
2	28935 (7107h)	returned low byte of address
3	28936 (7108h)	returned high byte of address
4	28937 (7109h)	returned low byte of PC data
5	28938 (710Ah)	returned high byte of PC data
6	28939 (710Bh)	returned checksum

Execute a CALL 24912 to verify that the returned checksum matches the data received. If it doesn't, an error flag will be set that can be tested by a APPLESOFT PEEK statement.

	28940 (710Ch)	error flag. should be zero

Using these statements in a program to communicate with a Numa-Logic PC700/900/1100 PC is a simple matter. For example:

```
 50      REM  PROGRAM TO READ HR0001 AND PRINT RESULT
 60      REM
 70      REM
 80      REM  SYNC WITH PC FIRST
 90      CALL 26368
100      IF PEEK(28942)=0 THEN 120
110      PRINT"UNABLE TO SYNC":GOTO 90
120      OP=0:LA=0:HA=0 : REM   PRESET OPCODE AND ADDRESS
130      POKE 28928,OP  : REM   LOAD OPCODE
140      POKE 28929,LA  : REM   LOAD LOW BYTE OF PC ADDRESS
150      POKE 28930,HA  : REM   LOAD HIGH BYTE OF PC ADDRESS
160      CALL 24880     : REM   CALCULATE CHECKSUM
170      CALL 26112     : REM   SEND MESSAGE AND WAIT FOR RESPONSE
180      CALL 24912     : REM   TEST RETURNED DATA INTEGRITY
190      IF PEEK(28940)=0 THEN GOTO 210
200      PRINT"RETURNED DATA INVALID"
210      PRINT"RETURNED VALUE OF HR0001 = ";(256*PEEK(28932))+
         PEEK(28931)
```

Section 12: Radio Shack TRS-80 Model II Communications

The Model II (as well as the Model 12 and 16 operating in the Model II mode), operating under the TRSDOS 2.0b operating system and using Model II BASIC, does not provide a convenient method of communicating with the serial ports. Again assembly language provided the needed link with BASIC via the USR statement (Model II BASIC has no CALL statement).

Configuring the system

The programmer must first configure the serial port via the SETCOM command from the TRSDOS prompt:

```
SETCOM B=(9600,8,O,2)
```

This statement configures the 'B' channel serial port for 9600 baud, 8 data bits, odd parity and two stop bits. Channel 'B' was configured since the driver "COMLINK" uses channel 'B' supervisory calls. Refer to your TRSDOS 2.0b/4.2 manual for more information on the use of supervisory calls.

Once the serial port is configured, the assembly language driver COMLINK is loaded:

```
COMLINK
```

Now BASIC can be loaded. Note that the high memory boundary was set to 57340. This is done to prevent BASIC program statements from over-writing the assembly language driver COMLINK. Additionally, the number of files is set to one (to maximize the memory available for program) and the program TEST is loaded and executed.

```
BASIC TEST -F:1 -M:57340
```

Configuring BASIC

COMLINK loads two assembly language drivers into memory that can be accessed from BASIC. One driver is located at memory address E013h. This driver is used to transfer a byte from BASIC and then transmit this byte out the serial port. The other driver is located at memory address E000h. This other driver is used to remove a byte from the receive buffer and return it to BASIC. To use these drivers, the following BASIC statements are necessary:

```
100  DEFUSR2=&HE013 : ' point to memory location of Tx USR stmt.
110  DEFUSR1=&HE000 : ' point to memory location of Rx USR stmt.
```

Transmit a byte

Simply load the variable to be transmitted into the variable T% and transfer control to the following statement. The variable X% is a dummy variable.

```
110  X%=USR2(T%)     : ' transmit the variable T% out the port
```

Receive a byte

One nice thing the the assembly language receive driver does, is to buffer up to 16 characters received at the serial port. Since most responses from the PC will be only six bytes in length, this does not pose a problem. To transfer a byte from the receive buffer to a variable Y%(I) in BASIC, the following statement is used. The variable R% is a dummy variable.

```
200  Y%(I)=USR1(R%)
```

These two USR statements can be used to communicate with the programmable controller.

```
 85  DEFUSR1=&HE000 :' SET UP POINTER TO MEMORY FOR RCV ROUTINE
 90  DEFUSR2=&HE013 :' SET UP POINTER TO MEMORY FOR TX ROUTINE
 95  GOTO 130

100  GOSUB 105:T%=OP%:GOSUB 120:T%=LA%:GOSUB 120:T%=HA%:GOSUB 120:
     T%=LD%:GOSUB 120:T%=HD%:GOSUB 120:T%=CK%:GOSUB 120:GOSUB 110:
     RETURN
105  CK%=OP%+LA%+HA%+LA%+HA%:RETURN
110  FOR I=1 TO 50:NEXT:FOR I=1 TO 6:GOSUB 125:NEXT:IF Y%(1)=OP%
     THEN RETURN ELSE PRINT"COMMUNICATIONS FAILURE"

115  '
120  X%=USR2(T%):RETURN
125  Y%(I)=USR1(R%):RETURN
130  FOR I=0 TO 15  :' RCV BUFFER CLEAR ROUTINE
135  GOSUB 125
140  NEXT I
150  PRINT"Attempting Sync with PC"
155  CO=0:T%=0:R%=0 :' PRESET SYNC COUNTER AND TX DATA
160  CO=CO+1        :' INCREMENT SYNC TRIES COUNTER
165  IF CO=20 THEN 170 ELSE 185
170  PRINT"Unable to Sync"
175  PRINT"Error code - ";HEX$(Y%(1))
180  INPUT"Hit ENTER to reattempt sync procedure ";A$:GOTO 130
185  GOSUB 120      :'CLEAR RCV BUFFER AFTER SYNC ATTEMPT
190  FOR K=1 TO 50:NEXT :'PAUSE
195  I=1
200  GOSUB 125
205  IF Y%(I)>255 THEN 210 ELSE 215:' TEST FOR RCV DATA
210  IF Y%(I)=256 THEN 160 :'IF =256, NO DATA IN RCV BUFFER
215  IF Y%(I)>10 THEN 160 :' IF GREATER, THEN PC DETECTED ERROR
```

```
 220  FOR I=2 TO 6:GOSUB 125:NEXT :'CLEAR RCV BUFFER

 225  '
 230  'READ HR0001 FROM PC
 235  OP%=0:LA%=0:HA%=0 :'PRESET OPCODE AND PC ADDRESS
 240  GOSUB 100 :' TRANSMIT MESSAGE
 245  FOR I=1 TO 50:NEXT :' PAUSE
 250  GOSUB 1565 :'EXTRACT RETURNED DATA
 255  PRINT "HR0001 contains - ";VL
 260  END

1560  'CALCULATE RETURNED DATA
1565  VL=(256*Y%(5))+Y%(4):RETURN
```

Appendix A : IBM PC Communications Sample Program - PCTEST.BAS

This program is designed to communicate with a programmable controller using the COM1 serial port.

The program prompts the user for a type of command to be sent to a PC directly or via the Westnet II data highway. The response from the PC or PCI is parsed and descriptive error messages (if any) are given.

```
10 DIM Z$(200),Y%(200)
20 ' ON ERROR GOTO 2850
30 CLS:KEY OFF
40 INPUT"Do you need help (Y/N) ";A$
50 IF A$="y" OR A$="Y" THEN GOSUB 2960
60 OPEN "com1:9600,n,8,2" AS #2
70 CLS:PRINT"Baud Rate Selection Menu"
80 PRINT
90 PRINT:PRINT"1.  150"
100 PRINT"2.  300":
                PRINT"3.  600":
                PRINT"4. 1200":
110 PRINT"5. 2400":
                PRINT"6. 4800":
                PRINT"7. 9600"
120 T%=INP(1019)
130 PRINT:INPUT"Enter your choice ";B$
140 IF B$="1" THEN LB%=&H0:HB%=&H3
150 IF B$="2" THEN LB%=&H80:HB%=&H1
160 IF B$="3" THEN LB%=&HC0:HB%=&H0
170 IF B$="4" THEN LB%=&H60:HB%=0
180 IF B$="5" THEN LB%=&H30:HB%=0
190 IF B$="6" THEN LB%=&H18:HB%=0
200 IF B$="7" THEN LB%=&HC:HB%=0
210 OUT 1019,&H80:' set UART line control register up for Baud rate
change
220 OUT 1016,LB% :' baud rate divisor LSB
230 OUT 1017,HB% :' baud rate divisor MSB
240 OUT 1019,15  :' reset line control register
250 COM(2) ON    :' activate communications interrupt handler
260 CLS
270 PRINT" Westnet II/Direct Connect Logic Analyzer
Program";TAB(70)"V1.3a"
290 LOCATE 3,1:PRINT"Messages:"
300 IF STRY=0 THEN LOCATE 5,1:PRINT"Sync - Not Attempted":GOTO 320
310 LOCATE 5,1:IF SFAIL=1 THEN PRINT"Sync - Failed    " ELSE PRINT
"Sync - OK      "
320 LOCATE 11,1
325 PRINT"Data Highway Commands"
330 PRINT"1.  Set Highway Address"
340 PRINT"2.  Open Session"
350 PRINT"3.  Close Session"
360 PRINT"4.  Read Session Status"
370 PRINT"5.  Read Local Address"
```

```
380 PRINT"6.  Read Remote Address"
390 LOCATE 11,30:PRINT"Direct Connect Commands"
400 LOCATE 12,30:PRINT"7.  Word Read"
410 LOCATE 13,30:PRINT"8.  Word Write"
420 LOCATE 14,30:PRINT"9.  Block Read"
430 LOCATE 15,30:PRINT"10. Block Write"
440 LOCATE 11,55:PRINT"Status Utility Functions"
450 LOCATE 12,55:PRINT"11. Read Highway Status"
460 LOCATE 13,55:PRINT"12. Message editor"
470 LOCATE 14,55:PRINT"13. Quit"
480 LOCATE 15,55:PRINT"14. Select Baud Rate"
490 LOCATE 16,55:PRINT"15. Synchronize with PC"
500 LOCATE 22,1:INPUT "Enter your choice ";CH%
510 IF CH%=99 THEN GOTO 2660
520 IF CH%=13 THEN GOTO 3060
530 ON CH% GOTO 1090, 1220, 1330, 1440, 1560, 1670, 1780, 1930,
    2090, 2230, 2530, 540, 70, 70, 3160
540 CLS
550 INPUT"Reset Westnet ";W$
560 IF W$="y" OR W$="Y" THEN GOSUB 990
570 INPUT"Format 6 bytes or 3 words (6/3) ";F$
580 IF F$="6" THEN 930
590 GOSUB 820:INPUT"Enter Opcode  ";OP%
600 INPUT"Enter Address ";AD%
610 INPUT"Enter Data    ";DA%
620 GOSUB 780
630 CK%=(OP%+LA%+HA%+LD%+HD%)AND 255
640 PRINT #2, CHR$(OP%); CHR$(LA%); CHR$(HA%); CHR$(LD%); CHR$(HD%);
              CHR$(CK%);
650 'pause, then receive any returned data
660 FOR I=0 TO 200:NEXT
670 'start of data receive routine
680 I=0
690 WHILE EOF(2)=0
700 Z$(I)=INPUT$(1,2):Y%(I)=ASC(Z$(I)):I=I+1
710 WEND
720 'print results of unit address request
730 FOR J=0 TO I-1:
                PRINT "y%(";J;") = "Y%(J);SPC(5);HEX$(Y%(J)):
    NEXT
740 PRINT "Value = ";(256*(Y%(4)))+Y%(3):PRINT
750 IF F$="6" THEN 930
760 IF W$="Y"OR W$="y" THEN 550
770 GOTO 590
780 'calculate address and data bytes from 16 bit integer values
790 HA%=AD%\256:LA%=AD%-(HA%*256)
800 HD%=DA%\256:LD%=DA%-(HD%*256)
810 RETURN
820 'prompt if RTS is to be dropped low
830 INPUT"Return to main menu ";M$
840 IF M$="y" OR M$="Y" THEN 260
850 INPUT"Drop RTS line (Y/N) ";M$
860 IF M$ = "y" OR M$ ="Y" THEN 880
870 RETURN
880 CLOSE:PRINT"DROPPING RTS"
```

```
890 FOR I= 0 TO 2000:NEXT
900 OPEN "com1:9600,n,8,2,cs,ds,cd" AS #2
910 OUT 1019,15:COM(2) ON
920 RETURN
930 INPUT"1st byte ";OP%
940 INPUT"2nd byte ";LA%
950 INPUT"3rd byte ";HA%
960 INPUT"4th byte ";LD%
970 INPUT"5th byte ";HD%
980 GOTO 630
990 OP%=9:LA%=0:HA%=0:LD%=0:HD%=0
1000 GOTO 630
1010 FOR I=0 TO 2000:NEXT
1020 'start of data receive routine
1030 I=0
1040 WHILE EOF(2)=0
1050 Z$(I)=INPUT$(1,2):Y%(I)=ASC(Z$(I)):I=I+1
1060 WEND:RETURN
1070 PRINT #2, CHR$(OP%); CHR$(LA%); CHR$(HA%); CHR$(LD%);
               CHR$(HD%); CHR$(CK%);: RETURN
1080 CK%=(OP%+LA%+HA%+LD%+HD%)AND 255:RETURN
1090 '***** Set hiway add ****
1100 '
1110 OP%=9:LA%=0:HA%=0
1120 CLS:PRINT"Set Highway address":PRINT:PRINT:INPUT"Enter drop
     ";LD%
1130 INPUT"Enter subdrop ";HD%
1140 GOSUB 1080:' calculate checksum
1150 GOSUB 1070:' transmit 6 byte sequence
1160 GOSUB 1010:' get returned data
1170 FOR J=0 TO I-1:
                 PRINT "y%(";J;") = "Y%(J);SPC(5);HEX$(Y%(J)):
     NEXT
1180 IF I>3 THEN PRINT"Highway address set to drop -  ";Y%(3);"
,subdrop -  ";Y%(4) ELSE PRINT"Highway command failed"
1190 GOSUB 3140
1200 LOCATE 25,1:INPUT"Press Enter to continue";A$
1210 GOTO 260
1220 '***** OPEN SESSION *****
1230 '
1240 CLS:PRINT"Open Session": PRINT: PRINT: OP%=9: LA%=3: HA%=0:
         LD%=0: HD%=0
1250 GOSUB 1080
1260 GOSUB 1070
1270 GOSUB 1010
1280 FOR J=0 TO I-1:
                 PRINT "y%(";J;") = "Y%(J);SPC(5);HEX$(Y%(J)):
     NEXT
1290 IF I>3 THEN PRINT"Open Session successful" ELSE PRINT"Command
     Failed"
1300 GOSUB 3140
1310 LOCATE 25,1:INPUT"Press Enter to continue";A$
1320 GOTO 260
1330 '***** CLOSE SESSION *****
1340 '
```

```
1350 CLS:PRINT"Close
         Session":PRINT:PRINT:OP%=9:LA%=4:HA%=0:LD%=0:HD%=0
1360 GOSUB 1080
1370 GOSUB 1070
1380 GOSUB 1010
1390 FOR J=0 TO I-1:
                 PRINT "y%(";J;") = "Y%(J);SPC(5);HEX$(Y%(J)):
     NEXT
1400 IF I>3 THEN PRINT"Close Session successful" ELSE PRINT"Command
     Failed"
1410 GOSUB 3140
1420 LOCATE 25,1:INPUT"Press Enter to continue";A$
1430 GOTO 260
1440 '***** READ SESSION STATUS *****
1450 '
1460 CLS:PRINT"Read Session Status":PRINT:PRINT: OP%=9: LA%=5:
         HA%=0: LD%=0: HD%=0
1470 GOSUB 1080
1480 GOSUB 1070
1490 GOSUB 1010
1500 FOR J=0 TO I-1:
                 PRINT "y%(";J;") = "Y%(J);SPC(5);HEX$(Y%(J)):
     NEXT
1510 IF I>3 AND Y%(2)=128 THEN PRINT"Drop not in session":GOTO 1540
1520 IF I>3 THEN PRINT"Drop in session with drop -  ";Y%(3);
         " ,subdrop -  "; Y%(4)
1530 GOSUB 3140
1540 LOCATE 25,1:INPUT"Press Enter to continue";A$
1550 GOTO 260
1560 '***** READ LOCAL ADDRESS *****
1570 '
1580 CLS:PRINT"Read Local Address":PRINT:PRINT: OP%=9: LA%=6: HA%=0:
         LD%=0: HD%=0
1590 GOSUB 1080
1600 GOSUB 1070
1610 GOSUB 1010
1620 FOR J=0 TO I-1:
                 PRINT "y%(";J;") = "Y%(J);SPC(5);HEX$(Y%(J)):
     NEXT
1630 IF I>3 THEN PRINT"Local address: drop -  ";Y%(3);" ,subdrop -
";Y%(4)
1640 GOSUB 3140
1650 LOCATE 25,1:INPUT"Press Enter to continue";A$
1660 GOTO 260
1670 '***** READ REMOTE ADDRESS *****
1680 '
1690 CLS:PRINT"Read Remote Address":PRINT:PRINT: OP%=9: LA%=1:
         HA%=0: LD%=0: HD%=0
1700 GOSUB 1080
1710 GOSUB 1070
1720 GOSUB 1010
1730 FOR J=0 TO I-1:
                 PRINT "y%(";J;") = "Y%(J);SPC(5);HEX$(Y%(J)):
     NEXT
1740 IF I>3 THEN PRINT"Presently addressed to drop -  ";Y%(3);"
```

```
,subdrop -  ";Y%(4)
1750 GOSUB 3140
1760 LOCATE 25,1:INPUT"Press Enter to continue";A$
1770 GOTO 260
1780 '***** WORD READ *****
1790 '
1800 CLS:PRINT"Word Read":PRINT:PRINT:OP%=0
1810 INPUT"Enter address to read ";AD%
1820 GOSUB 780
1830 GOSUB 1080
1840 GOSUB 1070
1850 GOSUB 1010
1860 PRINT"Transmitted to PC:"
1870 PRINT HEX$(OP%);" ";HEX$(LA%);" ";HEX$(HA%);" ";HEX$(LD%);"
";HEX$(HD%);" ";HEX$(CK%):PRINT
1880 PRINT"Received from PC:"
1890 FOR J=0 TO I-1:PRINT HEX$(Y%(J));" ";:NEXT
1900 PRINT:PRINT:PRINT "Value = ";(256*(Y%(4)))+Y%(3):PRINT
1910 LOCATE 25,1:INPUT"Press Enter to continue";A$
1920 GOTO 260
1930 '***** WORD WRITE *****
1940 '
1950 CLS:PRINT"Word Write":PRINT:PRINT:OP%=1
1960 INPUT"Enter address to write ";AD%
1970 INPUT"Enter data to write at that address ";DA%
1980 GOSUB 780
1990 GOSUB 1080
2000 GOSUB 1070
2010 GOSUB 1010
2020 PRINT"Transmitted to PC:"
2030 PRINT HEX$(OP%);" ";HEX$(LA%);" ";HEX$(HA%);" ";HEX$(LD%);"
";HEX$(HD%);" ";HEX$(CK%):PRINT
2040 PRINT"Received from PC:"
2050 FOR J=0 TO I-1:PRINT HEX$(Y%(J));" ";:NEXT
2060 PRINT:PRINT:PRINT "Value = ";(256*(Y%(4)))+Y%(3):PRINT
2070 LOCATE 25,1:INPUT"Press Enter to continue";A$
2080 GOTO 260
2090 '***** block read *****
2100 '
2110 CLS:PRINT"Block Read":PRINT:PRINT:OP%=5
2120 INPUT"Enter starting address ";AD%
2130 INPUT"Enter number of words to read (64 max.) ";DA%
2140 IF DA%<>0 THEN DA%=DA%-1
2150 GOSUB 780
2160 GOSUB 1080
2170 GOSUB 1070
2180 GOSUB 1010
2190 FOR J=0 TO I-1:
                 PRINT "y%(";J;") = "Y%(J);SPC(5);HEX$(Y%(J)):
     NEXT
2200 PRINT "Value = ";(256*(Y%(4)))+Y%(3):PRINT
2210 LOCATE 25,1:INPUT"Press Enter to continue";A$
2220 GOTO 260
2230 '***** block write *****
2240 '
```

```
2250 CLS:PRINT"Block Write":PRINT:PRINT:OP%=7
2260 INPUT"Enter starting address ";AD%
2270 INPUT"Enter number of words to write ";DA%
2280 '
2290 FOR K=0 TO DA%-1
2300 PRINT"Enter word number ";K+1;" ";:INPUT WO%(K)
2310 NEXT K
2320 '
2330 IF DA%<>0 THEN DA%=DA%-1
2340 GOSUB 780
2350 GOSUB 1080:'calculate checksum
2360 GOSUB 1070:'tx first 6 command bytes
2370 '
2380 CK%=0
2390 FOR J=0 TO K-1
2400 GOSUB 2510
2410 CK%=CK%+WO%(J)
2420 PRINT #2,CHR$(LA%);CHR$(HA%);
2430 NEXT J
2440 PRINT #2,CHR$(CK%);
2450 GOSUB 1010:'get data
2460 FOR J=0 TO I-1:
                 PRINT "y%(";J;") = "Y%(J);SPC(5);HEX$(Y%(J)):
     NEXT
2470 PRINT "Value = ";(256*(Y%(4)))+Y%(3):PRINT
2480 LOCATE 25,1
2490 INPUT"Press Enter to continue ";A$
2500 GOTO 260
2510 '
2520 HA%=WO%(J)\256:LA%=WO%(J)-(HA%*256):RETURN
2530 '***** Read Highway Status *****
2540 '
2550 CLS:PRINT"Read Highway Status":PRINT:PRINT:OP%=9:LA%=2
2560 GOSUB 1080
2570 GOSUB 1070
2580 GOSUB 1010
2590 FOR J=0 TO I-1:
                 PRINT "y%(";J;") = "Y%(J);SPC(5);HEX$(Y%(J)):
     NEXT
2600 IF I>3 AND Y%(2)>0 THEN PRINT"Error code -  ";Y%(2);"
detected"
2610 IF I<3 THEN PRINT"Command Failed"
2620 IF I>3 AND Y%(2)=0 THEN PRINT"No highway errors found"
2630 GOSUB 3140
2640 LOCATE 25,1:INPUT"Press Enter to continue";A$
2650 GOTO 260
2860 ON ERROR GOTO 2900
2870 CLS:KEY OFF
2880 INPUT"Do you need help (Y/N) ";A$
2890 IF A$="y" OR A$="Y" THEN GOSUB 2960
2900 PRINT"BASIC has trapped error number ";ERR;" in line ";ERL
2910 PRINT:PRINT"Consult your IBM BASIC manual's appendix A for an
explination"
2920 PRINT"of this error number"
2930 PRINT:PRINT"Note that improper operation may occur"
```

```
2940 INPUT"Press Enter to restart program ",A$
2950 RUN
2960 OPEN "comhelp.txt" FOR INPUT AS #1
2970 C=0
2980 IF EOF(1) THEN 3020
2990 LINE INPUT #1,A$
3000 PRINT A$
3010 C=C+1:IF C>20 THEN 3040 ELSE 2980
3020 INPUT"Press Enter to continue ",A$:LOCATE 23,1:PRINT
     SPC(30):LOCATE 23,1
3030 RETURN
3040 C=0:INPUT"Press Enter to continue ",A$:LOCATE 23,1:PRINT
     SPC(30):LOCATE 23,1
3050 GOTO 2980
3060 CLS
3070 PRINT"1. Quit - Return to DOS (return to menu choices)"
3080 PRINT"2. Quit - Stay in BASIC (modify program)"
3090 PRINT"3. Restart program"
3100 PRINT:INPUT"Enter your choice (1-3) ";A$
3110 IF A$="1" THEN INPUT"Are your sure (Y/N) ";Q$:IF Q$="y" OR
     Q$="Y" THEN SYSTEM
3120 IF A$="2" THEN END
3130 GOTO 260
3140 X=Y%(0) AND 32:IF X= 32 THEN PRINT"Not connected to Westnet "
3150 RETURN
3160 ' --- Synchronization point to point ---
3170 '
3180 STRY=1
3190 PRINT #2,CHR$(0);
3195 LOCATE 6,1:PRINT "Transmitting Sync byte - ";STRY;SPC(10);
3200 GOSUB 1010
3210 IF I>1 THEN SFAIL=0:GOTO 260
3220 STRY=STRY+1:IF STRY>8 THEN SFAIL=1:GOTO 260 ELSE 3190
```

Appendix B : IBM PC Communications Sample Program

This program is designed to work with the AST CC-232 Advanced Communications Card. The program communicates with a PC connected to port B (J3) of the CC-232 card. The AST handlers are written in assembly language and are accessed via the BASICA CALL statement.

```
10 ON ERROR GOTO 990
20 CLS:KEY OFF
30 INPUT"Do you need help (Y/N) ";A$
40 IF A$="y" OR A$="Y" THEN GOSUB 1050
50 ' This program uses the AST-232 card's port B for communications
60 ' Purpose:
70 ' To allow the user to remotely start and stop a PC 700/900/1100
80 ' by overriding the keyswitch position.
90 '
100 ' The processor used must support the keyswitch override command
and
110 ' must have its keyswitch in the Run/Modify Position.  This
program
120 ' can, from the Run/Modify position change the processor from
130 ' Run/Modify to Stop/Program mode and back.
140 '
150 CLS
160 KEY OFF
170 LOCATE 12,20:PRINT"Loading Assembly Drivers"
180 DEF SEG=&H3F54
190 BLOAD "COMINTP.BAS",0
200 CLS:PRINT"Keyswitch Override";TAB(70);"V1.3a"
210 PRINT:PRINT" Westinghouse Electric Corporation - 1985"
220 LOCATE 10,1
230 PRINT"0.   300"
240 PRINT"1.  1200"
250 PRINT"2.  2400"
260 PRINT"3.  4800"
270 PRINT"4.  9600"
280 PRINT"5. 19200"
285 PRINT"6. Quit - Return to DOS (return to menu choices) "
287 PRINT"7. Quit - Stay in BASIC (for program modifications) "
290 PRINT:INPUT"Enter Baud rate ";A%
292 IF A%=6 THEN INPUT"Are you sure (Y/N) ";Q$:IF Q$="y" OR Q$="Y"
THEN SYSTEM ELSE 200
294 IF A%=7 THEN END
300 FLAG=1
310 B%=0:C%=2
320 LET COMINTP=&H311
330 ' *** Load Baud Rate to AST CC-232 card ***
340 CALL COMINTP(A%,B%,C%)
350 '*** Reset pointer to Assembly language program for serial
communications
360 COMINTP=0:C%=0
370 A%=0:B%=-32251
380 GOSUB 910
390 CALL COMINTP(A%,B%,C%)
```

```
400 GOSUB 930
410 ' *** check keyswitch position ***
420 X= C% AND 256:IF X>0 THEN KY$="stop/program "
430 X= C% AND 512:IF X>0 THEN KY$="run/program protect"
440 X= C% AND 1024:IF X>0 THEN KY$="run/modify"
450 IF A%=255 THEN KY$="unable to read"
460 IF A%=255 THEN STAT$="comm fail"
470 T=C% AND 2048:IF T>0 THEN STAT$="fault" ELSE STAT$="Ok"
475 IF A%=255 THEN 620
477 GOTO 620
480 '
490 ' *** Check if keyswitch is presently overriden ***
500 '
510 B%=-32249
520 GOSUB 910
530 CALL COMINTP(A%,B%,C%)
550 GOSUB 930
570 X= C% AND 1:IF X>0 THEN OV=1 ELSE OV=0
580 X= C% AND 2:IF X>0 THEN RP=1 ELSE RP=0
600 IF OV=0 THEN MODE$="Override OFF":ELSE MODE$="Override ON"
605 IF RP=1 THEN STAT$="Warning - CPU retest pending - Turn
keyswitch to stop to clear"
620 CLS:PRINT"Keyswitch Status Display":LOCATE 5,1
630 PRINT"Status    - ";STAT$
640 PRINT"Keyswitch - ";KY$
650 PRINT"Mode      - ";MODE$;:IF MF=1 THEN PRINT " - ";C% ELSE
PRINT
660 PRINT:PRINT"1. Stop Processor":PRINT"2. Start
Processor":PRINT"3. Re-read Keyswitch position":PRINT"4. Quit -
Return to DOS"
670 PRINT"5. Quit - Return to BASIC"
680 PRINT"6. Change BAUD rate"
690 PRINT"7. Retest CPU"
700 PRINT:INPUT"Enter your choice ";CHOICE
710 IF CHOICE=5 THEN END
720 ON CHOICE GOTO 730,790,370,870,900,200,1150
730 '*** Stop Processor ***
740 A%=3:B%=-32249:C%=16
750 GOSUB 910
760 CALL COMINTP(A%,B%,C%)
770 GOSUB 930
780 GOTO 370
790 '*** Start Processor ***
800 A%=3:B%=-32249:C%=0
810 GOSUB 910
820 CALL COMINTP(A%,B%,C%)
830 GOSUB 930
840 LOCATE 22,1:PRINT"Pausing to allow remote PC to complete retest
cycle"
850 FOR I=0 TO 4000:NEXT
860 GOTO 370
870 INPUT"Are you sure (Y/N) ";Q$
880 IF Q$="Y" OR Q$="y" THEN SYSTEM
890 GOTO 370
900 END
```

```
910 LOCATE 25,30:PRINT"COMMUNICATING";
920 RETURN
930 LOCATE 25,30:PRINT SPC(15);
940 RETURN
990 PRINT"BASIC has trapped error number ";ERR;" in line ";ERL
1000 PRINT:PRINT"Consult your IBM BASIC manual's appendix A for an
explination"
1010 PRINT"of this error number"
1020 PRINT:PRINT"Note that improper operation may occur"
1030 INPUT"Press Enter to restart program ",A$
1040 RUN
1050 OPEN "keyhelp.txt" FOR INPUT AS #1
1060 C=0
1070 IF EOF(1) THEN 1110
1080 LINE INPUT #1,A$
1090 PRINT A$
1100 C=C+1:IF C>20 THEN 1130 ELSE 1070
1110 INPUT"Press Enter to continue ",A$
1120 RETURN
1130 C=0:INPUT"Press Enter to continue ",A$
1140 GOTO 1070
1150 '
1160 ' *** Retest CPU Routine ***
1170 '
1180 'set the 2nd bit of the low byte of address 8205H
1190 '
1200 A%=3:B%=-32251:C%=17
1210 GOSUB 910
1220 CALL COMINTP(A%,B%,C%)
1230 GOSUB 930
1240 LOCATE 22,1:PRINT"Pausing to allow remote PC to complete retest
cycle"
1250 FOR I=0 TO 4000:NEXT
1260 GOTO 370
```

Appendix C : IBM PC Communications Sample Program - MESGEN.BAS

This program is designed to operate with the AST CC-232 Advanced Communications Card. This program will allow the user to enter ASCII messages from the keyboard, edit them and up/download these messages directly to the PC memory. The program will convert the ASCII characters, then format a proper message to the PC.

```
10 CLS
20 KEY OFF
30 FOR I=1 TO 10
40 KEY I,""
50 NEXT I
60 DIM B1$(256)
70 DIM E%(256)
80 DIM E$(256)
90 DEF SEG=&H3F54
100 BLOAD "COMINTP.BAS",0
110 LET CONFIG1=&H311
120 A%=4
130 B%=0
140 C%=2
150 CALL CONFIG1(A%,B%,C%)
160 CLS
170 SCREEN 2
180 PRINT:PRINT: PRINT ,,"WESTINGHOUSE ELECTRIC "
190 PRINT ,,"     NUMA LOGIC"
200 PRINT:PRINT ,,"   PC-700/900/1100"
210 PRINT ,  "           MESSAGE GENERATION PROGRAM"
220  LOCATE 10,17,0,7,7
230 PRINT "                    MAIN MENU"
240 PRINT:PRINT:PRINT ,,"1 ------  NEW MESSAGE"
250 PRINT:PRINT ,,"2 ------  REVIEW AND EDIT MESSAGE"
260 PRINT:PRINT ,,"3 ------  DIRECTIONS"
265 PRINT:PRINT,,"4 ------  QUIT"
270 O$=INKEY$:IF O$="1" THEN 290
275 IF O$="3" THEN 2300
277 IF O$="4" THEN SYSTEM
280 IF O$="2" THEN 1510 ELSE 270
290 SCREEN 0 :COLOR 0,7:LOCATE 1,30,0,7,7:PRINT   " NEW MESSAGE
MODE":COLOR 7,0
300 PRINT:PRINT ;"ENTER MESSAGE TO BE DISPLAYED BY ASCII TRANSMIT
FUNCTION"
310 PRINT ; "(HIT [ENTER] TO RETURN TO MAIN MENU)"
320 COLOR 0,7
330 PRINT ,,,, "    "
340 PRINT ,,,, "    "
350 PRINT ,,,, "    "
360 COLOR 7,0
370 IF T2$="Y"  THEN 1560
380 LOCATE 11,30,0,7,7:PRINT "BCODES":PRINT
390 PRINT "B0   --- Return from subroutine"
400 PRINT "B1XN --- Print character X, N times  (N<255)"
410 PRINT "B2X  --- Print lower two digits of HR pointed to by
```

```
pointer X"
420 PRINT "B3X  --- Print  +/- five digits of HR pointed to by
pointer X"
430 PRINT "B4X  --- Print lower four digits of HR pointed to by
pointer X"
440 PRINT "B5X  --- Print five digits of HR pointed to by pointer X"
450 PRINT "B6X  --- Print five digits (ldng zeros blanked) of HR
pointed to by pointer X"
460 PRINT "B7X  --- Jumped to HR pointed to by pointer X"
470 PRINT "B8   --- Message end"
480 IF O$="2" THEN 1580
490 IF T3$<>"Y" THEN 550
500 LOCATE 5,1,0,7,7:COLOR 0,7
510 PRINT ,,,,,:PRINT ,,,,,:PRINT ,,,,,
520 COLOR 7,0:LOCATE 9,1,0,7,7:PRINT "
            "
530 LOCATE 22,1,0,7,7:PRINT"
          "
540 PRINT "
                "
550 COLOR 0,7
560 LOCATE 5,1,1,7,7:A$="":P=1:R=CSRLIN:C=POS(N):HP=1
570 T$=INKEY$:IF T$<>"" THEN T=ASC(T$) ELSE 570
580 IF F=1 THEN COLOR 7,0:LOCATE 8,1,0,7,7:PRINT "
                "  ELSE 600
590 COLOR 0,7:LOCATE R,C,0,7,7:F=0
600 IF P>HP THEN HP=P
610 IF T>20 AND T<93 THEN PRINT T$;:B1$(P)=T$:P=P+1:GOTO 690
620 IF T=13 AND P=1 AND O$="1" THEN 170
630 IF T=13 AND P<3 AND O$="1" THEN 570
640 IF T=13 AND O$="2" THEN 900
650 IF T=13 AND O$="1" THEN I=0:GOTO 860
660 IF LEN(T$)=2 THEN T=ASC(RIGHT$(T$,1))
670 IF T=77 THEN P=P+1:GOTO 770
680 IF T=75 THEN P=P-1:GOTO 810 ELSE 570
690 IF P<=2 THEN 770
700 TP$="":TP$=B1$(P-2)+B1$(P-1)
710 IF FL=1 THEN 730
720 IF TP$<"B2" OR  TP$>"B7" THEN 770 ELSE FL=1
730 IF RIGHT$(TP$,1)>="0" AND RIGHT$(TP$,1)<="9" THEN 770
740 IF RIGHT$(TP$,1)=" " THEN FL=0:GOTO 770
750 B1$(P)=B1$(P-1):B1$(P-1)=" ":PRINT CHR$(29);B1$(P-1);B1$(P);
760 C=C+1:P=P+1:FL=0
770 C=C+1:IF C=61 AND R=5 THEN R=6:C=1
780 IF R=6 AND C=61 THEN R=7:C=1
790 IF R=7 AND C=61 THEN C=60:P=255
800 GOTO 840
810 C=C-1:IF C=0 THEN C=1:P=0
820 IF R=6 AND C=1 THEN R=5:C=60
830 IF R=7 AND C=1 THEN R=6:C=60
840 LOCATE R,C,1,7,7
850 GOTO 570
860 IF HP<2 THEN 950
870 IF B1$(HP-2)="B" AND B1$(HP-1)="8" THEN 950
871 K4=0:P2=HP-1
```

```
873 WHILE ASC(B1$(P2))=32:P2=P2-1:K4=K4+1:WEND
875 IF B1$(P2)="8" AND B1$(P2-1)="B" THEN HP=P2:GOTO 950
877 C=C-K4:P=P-K4:GOTO 880
880 BEEP:COLOR 7,0:LOCATE 8,1,0,7,7:PRINT "A [B8] CODE MUST BE USED
FOR END OF MESSAGE"
890 COLOR 0,7:F=1:GOTO 840
900 IF HP<2 THEN 950
905 IF HP<HPR-2 THEN 950
910 IF HP>HPR-2 AND B1$(HP-1)="B" AND B1$(HP)="8" THEN 950
920 IF HP>HPR AND B1$(HP-2)="B" AND B1$(HP-1)="8" THEN 950
921 P2=HP-1:K4=0
922 WHILE ASC(B1$(P2))=32 :P2=P2-1:K4=K4+1:WEND
924 IF B1$(P2)="8" AND B1$(P2-1)="B" THEN 950
925 C=C-K4:P=P-K4
940 GOTO 880
950 IF O$="2" AND HP>HPR THEN H=HP ELSE H=HPR
960 IF O$="1" THEN H=HP
970 FOR I= 1 TO H
980 A$=A$ + B1$(I)
990 NEXT I
1000 COLOR 7,0
1010 IF O$="2" THEN 1060
1020 LOCATE  9,1,0,7,7
1030 INPUT "AT WHAT HR REF NO. DO YOU WANT MESSAGE TO START";A
1040 IF A >0 AND A<5000 THEN 1060 ELSE LOCATE 5,1,0,7,7
1050 GOTO 1030
1060 K=0:I=0
1070 WHILE E%(I) <> -18432
1080 K=K+1:I=I+1
1090 C$=MID$(A$,K,1)
1100 IF D1$="B" AND C$<"9" AND C$>="0" THEN E%(I-1)=E%(I-1) AND -
256:T$=D1$+C$:GOTO 1190
1110 C%=256*ASC(C$)
1120 K=K+1:D$=MID$(A$,K,1)
1130 IF C$="B" AND D$<"9" AND D$>="0" THEN T$=C$+D$:GOTO 1190
1140 IF D$="B" THEN D1$=D$
1150 D%=ASC(D$)
1160 E%(I)=C% + D%
1170 WEND
1180 GOTO 1350
1190 IF T$="B0" THEN E%(I)=-20480:GOTO 1070
1200 IF T$="B1" THEN K=K+1:E%(I)=  - (20224 + ASC(MID$(A$,K,1)))
:GOTO 1280
1210 IF T$="B2" THEN E%(I)=-19968:GOTO 1280
1220 IF T$="B3" THEN E%(I)=-19712:GOTO 1280
1230 IF T$="B4" THEN E%(I)=-19456:GOTO 1280
1240 IF T$="B5" THEN E%(I)=-19200:GOTO 1280
1250 IF T$="B6" THEN E%(I)=-18944:GOTO 1280
1260 IF T$="B7" THEN E%(I)=-18688:GOTO 1280
1270 IF T$="B8" THEN E%(I)=-18432:GOTO 1070
1280 TB$=""
1290 WHILE MID$(A$,K,1) <> " "
1300 K=K+1
1310 TB$=TB$+MID$(A$,K,1)
1320 WEND
```

```
1330 I=I+1
1340 E%(I)=VAL(TB$):GOTO 1070
1350 L=I:K=0
1360 FOR I= A TO A+L-1
1370 K=K+1
1380 A%=1
1390 B%=I-1
1400 C%=E%(K)
1410 COMMPROC=0
1420 CALL COMMPROC(A%,B%,C%)
1430 NEXT I
1440 LOCATE 22,1,0,7,7:PRINT "MESSAGE MEMORY LOCATIONS  HR";A;" ---
HR";A+L-1
1450 IF O$="2" THEN PRINT "DO YOU WANT TO REVIEW ANOTHER MESSAGE
(Y/N)?":GOTO 1490
1460 PRINT "DO YOU WANT TO INPUT ANOTHER MESSAGE (Y/N)?"
1470 T3$=INKEY$:IF T3$="Y" THEN 490
1480 IF T3$="N" THEN 160 ELSE 1470
1490 T2$=INKEY$:IF T2$="Y" THEN 1550
1500 IF T2$="N" THEN 160 ELSE 1490
1510 SCREEN 0 :P1=0
1520 LOCATE 2,25,0,7,7
1530 COLOR 0,7
1540 PRINT "REVIEW AND EDIT MESSAGE MODE"
1550 LOCATE 5,1,0,7,7:GOTO 320
1560 LOCATE 22,1,0,7,7:PRINT "
                    "
1570 PRINT "
                "
1580 LOCATE 4,1,0,7,7
1590 INPUT "ENTER MESSAGE HR STARTING POINT (0 - MAIN MENU)";T
1600 IF T=0 THEN 170
1610 IF T<1  OR T>5000  THEN LOCATE 4,1,0,7,7:GOTO 1590
1620 I=0:A=T:C%=0
1630 WHILE C%<>-18432
1640 I=I+1
1645 IF I>=256 THEN 2100
1650 A%=0:B%=T-1
1660 CALL COMMPROC(A%,B%,C%)
1670 IF C%<0 THEN 1790
1680 L%=C%/256
1690 L$=CHR$(L%)
1700 B1$(I)=L$
1710 I=I+1
1720 J%=255
1730 H%=C% AND J%
1740 H$=CHR$(H%)
1750 B1$(I)=H$
1760 T=T+1
1770 WEND
1780 GOTO 1930
1790 IF C%=-20480 THEN B1$(I)="B":I=I+1:B1$(I)="0": GOTO 1760
1800 IF C%=-19968 THEN B1$(I)="B":I=I+1:B1$(I)="2":GOTO 1880
1810 IF C%=-19712 THEN B1$(I)="B":I=I+1:B1$(I)="3":GOTO 1880
1820 IF C%=-19456 THEN B1$(I)="B":I=I+1:B1$(I)="4":GOTO 1880
```

```
1830 IF C%=-19200 THEN B1$(I)="B":I=I+1:B1$(I)="5":GOTO 1880
1840 IF C%=-18944 THEN B1$(I)="B":I=I+1:B1$(I)="6":GOTO 1880
1850 IF C%=-18688 THEN B1$(I)="B":I=I+1:B1$(I)="7":GOTO 1880
1860 IF C%=-18432 THEN B1$(I)="B":I=I+1:B1$(I)="8":GOTO 1760
1870 B1$(I)="B":I=I+1:B1$(I)="1":I=I+1:B1$(I)= CHR$(C% AND J%)
1880 T=T+1:B%=T-1:TEMP$=""
1890 CALL COMMPROC(A%,B%,C%)
1900 TEMP$=STR$(C%)+" ":Z=LEN(TEMP$):TEMP$=RIGHT$(TEMP$,Z-1):N=1
1910 WHILE B1$(I)<>" ":I=I+1:B1$(I)=MID$(TEMP$,N,1)
1920 N=N+1:WEND:T=T+1:GOTO 1630
1930 LOCATE 5,1,0,7,7:COLOR 0,7:R=5:C=1
1940 A$="":L1=I
1950 FOR I=1 TO L1
1960 PRINT B1$(I);
1970 IF ASC( B1$(I))=0  THEN PRINT CHR$(29);:C=C-1
1980 C=C+1
1990 IF R=5 AND C=61 THEN R=6:C=1
2000 IF R=6 AND C=61 THEN R=7:C=1
2010 LOCATE R,C,0,7,7
2020 A$=A$+B1$(I)
2030 NEXT I
2040 HPR=LEN(A$):GOTO 560
2100 BEEP:LOCATE 23,1,0,7,7:PRINT "B8 CODE NOT FOUND --- TRY AGAIN
(Y/N)?"
2110 G1$=INKEY$:IF G1$="N" THEN 170
2120 IF G1$><"Y" THEN 2110
2130 LOCATE 23,1,0,7,7:PRINT "
"
2140 GOTO 1580
2300 SCREEN 0:COLOR 0,7
2310 LOCATE 2,23,0,7,7:PRINT "MESSAGE GENERATION PROGRAM"
2320 COLOR 7,0:LOCATE 4,1,0,7,7
2330 PRINT "      This ASCII transmit program is a first
from";:COLOR 16,7:PRINT" EDWEA (weed in pig latin) Soft";
2340 PRINT "ware Inc";:COLOR 7,0:PRINT". The software products
company that brings you close to quality soft- ";
2350 PRINT "ware at extremely reasonable prices";:COLOR 16,7:PRINT "
(FREE)";:COLOR 7,0:PRINT ". Yes, you wont find our software copy";
2360 PRINT "protected -- WE DON'T HAVE TOO."
2370 PRINT "     This program enhances the ASCII transmit function
(PC-700/900/1100) thru   ";
2380 PRINT "facilitating message entry. It allows a message to be
typed in directly (as a   ";
2390 PRINT "typists would) instead of entering ASCII characters two
at a time into PC hold- ";
2400 PRINT "ing registers. The string is then analyized, including
any B codes, formated and";
2410 PRINT " placed within PC memory starting at any user desired
location. There is also a ";
2420 PRINT "review and edit mode, allowing existing messages to be
extracted,displayed, ed- ";
2450 PRINT "ited and then returned to the programmable controller. "
2460 PRINT "     Some notes on message entry, all end of messages
require a B8 code and when";
2470 PRINT " using B2-B7 codes a space must be inserted after
```

```
pointer. Format (pointer in   ";
2480 PRINT "brackets) is as follows:     Bx[xxx][space]   Program
prompts user if these two ";
2490 PRINT "items are left out. These and other safegaurds are
included. However,not all    ";
2500 PRINT "protective measures may have been taken. If program
fails, rerun, no damage will";
2510 PRINT " have occured because all messages are maintained within
programmable contoller ";
2520 PRINT "memory."
2530 PRINT "    Obviously, the more messages the more beneficial
this program. Its intent is";
2540 PRINT " to service those applications requiring much production
and management report-";
2550 PRINT "ing. WE HOPE YOU LIKE IT !!!!!";
2560 PRINT "             HIT [ENTER] TO RETURN TO MAIN MENU"
2570 G3$=INKEY$:IF G3$<>"" THEN G3=ASC(G3$) ELSE 2570
2580 IF G3=13 THEN 160 ELSE 2570
```

Appendix D: Assembly Language Calls from IBM BASICA

This section describes how to communicate with a programmable controller using a communications driver written in assembly language.

The programmer should be familiar with IBM BASICA.

If the programmer wishes to modify the assembly language program, they should also be familiar with Micro-Soft MACRO Assembler and Micro-Soft MS-LINK.

General

The IBM BASICA interpreter permits subprograms to be "called" from within the BASICA environment. The syntax is:

```
CALL programname(variable#1,variable#2,...,variable#N)
```

Before BASICA can transfer control to "programname", BASICA must know where "programname" is in memory. Two BASICA statements are used to define the location of a subprogram, "DEF SEG=", and "programname =".

"DEF SEG=" will define the segment (8088 term) while "programname=" defines the offset within that segment.

Numa-Logic Communication Subprogram "DATACOM"

To use this program, set up the segments and offsets as:

```
DEF SEG=&H3F54
PCLINK =&H230
CHGPARA=&H32A
```

Example of use of "DATACOM"

```
10 DEF SEG =&H3F54
20 KEY OFF
30 CLS
35 ' ** load the file DATACOM.BAS **
40 PRINT"Loading Assembly Language Drivers"
50 BLOAD "DATACOM.BAS",0
60 CLS
70 INPUT"BAUD RATE 1-1200 4-9600 ";A%
120 ' ** set default parity to odd **
130 B%=0
135 ' ** transfer control to DATACOM.BAS **
140 CHGPARA=&H32A
149 ' ** test to see if it really is there! **
150 IF PEEK(CHGPARA)<>85 THEN PRINT"PROGRAM MALFUNCTION":STOP
```

```
155 ' ** if so, initialize for parity, and baud rate **
160 CALL CHGPARA(A%,B%)
165 ' ** if A% is not zero, invalid baud or parity selected **
170 PRINT A%,B%
200 '
300 INPUT"opcode   ";OP%
310 INPUT"address  ";AD%
320 INPUT"data     ";DA%
325 '** entry point of communications handler **
330 PCLINK=&H230
335 IF PEEK(PCLINK)<>85 THEN PRINT"PROGRAM MALFUNCTION":STOP
340 CALL PCLINK(OP%,AD%,DA%)
350 PRINT HEX$(OP%),AD%,DA%
400 INPUT "CONTINUE (Y/N) ";A$
410 IF A$="Y" OR A$="y" THEN 500 ELSE 300
500 '***** START OF BLOCK DUMP *****
505 OP%=0:CLS
510 INPUT"ENTER STARTING ADDRESS ";ST%
520 INPUT"ENTER NUMBER OF WORDS  ";NU%
530 FOR J%=ST% TO ST%+NU%-1
537 OP%=0
540 CALL PCLINK(OP%,J%,DA%)
550 PRINT"ADDRESS ";J%;" ";DA%,HEX$(DA%)
560 NEXT
570 GOTO 400
```

Appendix E : Serial Port Wiring Diagrams

This section will describe the cables necessary to connect together various pieces of Numa-Logic Hardware to computers and modems including:

- Computer to PC direct
- Computer to PC via modem (point to point)
- Computer to Westnet II PCI
- PC1100 slave to modem (pin 8 carrier detect)
- PC1100 slave to modem (pin 6 carrier detect)
- PC1100 port transmit master to modem (pin 4 carrier enable)

Before we cover the actual cables necessary, let's cover some of the basics of RS-232 cabling:

1. RS-232 cables are usually terminated by 25 pin connectors that have pins or sockets. Connectors with pins are called "male" and connectors with sockets are called "female". A male connector always plugs into a female connector and vice versa.

2. Although you can always physically connect a male RS-232 connector to a female RS-232 connector, each of the two connectors must also be opposite in electrical connections too. The two types of electrical connections available for RS-232 ports are described as:

   - Data Terminal Equipment (DTE) configuration
   - Data Communications Equipment (DCE) configuration

   DTE devices can connect only to DCE devices. The only difference between the two is how the 25 pin connectors are wired. You can easily reconfigure a port to its opposite configuration by plugging in a "Null Modem" adapter or cable. A "Null Modem" adapter cable is described later.

2. Although the connector may have 25 terminal points, rarely are they all used. Most typical RS-232 connectors use 9 or less. These most common points are:

Name	DTE pin	DCE pin	Common Abbreviation
Transmit data	2	3	TxD
Receive data	3	2	RxD
Request to send	4	5	RTS
Clear to send	5	4	CTS
Data Set Ready	6	20	DSR
Data Carrier Detect	-	8	DCD
Data Terminal Rdy	20	6	DTR

   Note that "Data Carrier Detect" is not shown as having a pin on a DTE configured device. This is because a carrier detect line implies a modem, and as such, should always be wired as DCE.

Computer to PC direct (Standard Null Modem connection)

IBM PC/Apple][+ to PC700/900/1100

```
Computer          Programmable Controller
  2 ------------------ 3
  3 ------------------ 2
  4 +                + 4
    !                !
  5 +                + 5
  6 ------------------ 20
  7 ------------------ 7
 20 ------------------ 6
```

Since both the computer and the programmable controller are wired for DTE, a null modem cable is necessary for connection. When using certain software packages, the IBM PC may need for its pin 8 to be pulled high in order to operate. If this is the case, use the cable configuration shown next instead of the one above.

Radio Shack Model II to PC700/900/1100

```
Computer          Programmable Controller
  2 ------------------ 3
  3 ------------------ 2

  4 +                + 4
    !                !
  5 +                + 5

  6 +----------------- 20
    !
  8 +

  7 ------------------ 7

 20 ------------------ 6
```

As was the case with the IBM PC and Apple, a null modem cable is necessary to connect two DTE devices together. The Radio Shack system also requires that its DCD line be pulled high. Since pin 20 of the PC will always be high (if the PC is functional), then we can meet this criteria be connecting the programmable controllers pin 20 to the Radio Shack computer's pin 8 (also pin 6).

Computer to Modem (point to point)

IBM / Apple][+ / Radio Shack II,12,16

```
Computer          Modem                          Modem              PC

  2 ----------------- 2                           2 ----------------- 2
  3 ----------------- 3   Tx + ----- Rx+          3 ----------------- 3
  4 ----------------- 4   Tx - ----- Rx-          4 ----------------- 4
  5 ----------------- 5                           5 ----------------- 5
  6 ----------------- 6   Rx + ----- Tx+          6 ----------------- 6
  7 ----------------- 7   Rx - ----- Tx-          7 ----------------- 7
  8 ----------------- 8                           8 ----------------- 8
 20 ----------------- 20                         20 ----------------- 20
```

Since the modem is configured as DCE and the PC and computer are DTE, a simple straight through cable is all that is required to connect both together. The modem shown is assumed to be a 4 wire, full duplex type. Full duplex modems are easier to use, but are not necessary as long as the designer follows these restrictions:

1. If a half duplex modem is being used, data cannot travel both directions simultaneously. With Westinghouse programmable controllers (PC700/900/1100), half duplex communications media is acceptable if:
   - both modems have adjustable RTS/CTS turnaround times.
   - that the PC's being used support modem controls. If your PC700 or PC900 was built after 1982 (executive V3.x on the PC700 or V4.x on the PC900), you are probably OK. If your PC1100 has executive software V2.3 or later, you are OK.
2. The modem's RTS and CTS lines are wired to the programmable controller's RTS and CTS. In effect, when the PC receives a message from the communications line, it will be prevented from responding until its CTS line is brought high. This is known as line turnaround. Full duplex modems and line drivers (both 2 wire and 4 wire type) do not require modem control signals.

Computer to Westnet II PCI

IBM / Apple][+ / Radio Shack II,12,16

```
Computer          PCI Loader Port

  2 ----------------- 3
  3 ----------------- 2
  4 ----------------- 5
  5 ----------------- 4
  6 ----------------- 20
  7 ----------------- 7
 20 ----------------- 6
```

Both the computer and PCI loader port are configured as DTE, so a null modem cable connection is necessary.

Programmable Controller to Westnet II PCI

```
PC                PCI PC port

  2 ----------------- 2
  3 ----------------- 3
  4 ----------------- 4
  5 ----------------- 5
  6 ----------------- 6
  7 ----------------- 7
 20 ----------------- 20
```

The PCI PC port is designed for direct connection to a PC using only a straight through cable.

Programmable controller to Westnet II PCI via modem

```
Westnet PCI      Modem                        Modem              PC

  2 ----------------- 3                       2 ----------------- 2
  3 ----------------- 2  Tx + ----- Rx+        3 ----------------- 3
  4 ----------------- 5  Tx - ----- Rx-        4 ----------------- 4
  5 ----------------- 4                       5 ----------------- 5
  6 ----------------- 8  Rx + ----- Tx+        6 ----------------- 6
  7 ----------------- 7  Rx - ----- Tx-        7 ----------------- 7
  8                   20                       8 ----------------- 8
 20 ----------------- 6                      20 ----------------- 20
```

If the PC is to be located more than 50 feet from the PCI, a modem or line driver should be used. Note that the connection from the PCI PC port to the modem utilizes a null modem configuration, while the PC to modem connection is a simple straight through cable.

This configuration can be used to extend the distance of the Westnet II data highway beyond the 6km limitation. Multiple drops can be located remotely from the central coaxial based highway when channel splitters and statistical multiplexers are used. The following wiring diagram shows how two drops may be remotely located from the PCI.

Multiple Remote drops off the Westnet II data highway

```
+---------+              +-------------+    +-------+
! PCI     +Subdrop 0 ---+Asynchronous !    !       !  to remote
!         !              !Channel      +----+ Modem +--station  --+
!         +Subdrop 1 ---+Splitter     !    !       !              !
+---------+              +-------------+    +-------+              !
                                                                   !
+-------+                +-------------+    +-------+              !
!PC 1   +----------------+Asynchronous !    !       !              !
+-------+   +------+     !Channel      +----+ Modem +-------------+
            !PC 2  +----+Splitter     !    !       !
            +------+     +-------------+    +-------+
```

PC1100 Slave to modem (assumes pin 8 as the carrier detect)

Version 2.1

```
PC1100 slave     Modem

  2 ----------------- 2
  3 ----------------- 3
  4 ----------------- 4
  5 ----------------- 8
  6 ----------------- 6
  7 ----------------- 7
 20 ----------------- 20
```

Version 2.3

```
PC1100 slave     Modem

  2 ----------------- 2
  3 ----------------- 3
  4 ----------------- 4
  5 ----------------- 5
  6 ----------------- 8
  7 ----------------- 7
 20 ----------------- 20
```

PC1100 Slave to modem (assumes pin 6 as the carrier detect)

Version 2.1

```
PC1100 slave     Modem

  2 ----------------- 2
  3 ----------------- 3
  4 ----------------- 4
  5 -+---------------- 6
     !
  6 -+

  7 ----------------- 7
 20 ----------------- 20
```

Version 2.3

```
PC1100 slave     Modem

  2 ----------------- 2
  3 ----------------- 3
  4 ----------------- 4
  5 ----------------- 5
  6 ----------------- 6
  7 ----------------- 7
 20 ----------------- 20
```

PC1100 Port Transmit Master using modems

PC1100 master Modem

2 ---------- 2
3 ---------- 3
5 ---------- 5
6 ---------- 6
7 ---------- 7

4- ~~from 24 VDC output module~~ ------- 4

This configuration must be used since the master modem must remove the carrier just before the Port Transmit function is enabled. This wiring diagram assumes that pin 4 of the modem is used to switch the transmit carrier on and off. Read the modem's operating manual to verify that this is the case. ~~In any event, wire the 24 VDC output to the appropriate carrier control line of the modem for proper operation.~~

~~This configuration is necessary since the "sync" or "clear buffer" pulse from the master PC's pin 20 is too short to be detected by a modem's RTS line. Refer to section 5 for more information on this configuration.~~

V2.6 & LATER, OK TO USE RTS OF PLC TO SWITCH CARRIER.

Appendix H: Apple][+ Assembly Language Calls

Apple Computer Assembly Drivers
for
Communication with
Westinghouse PC700/900/1100 Family

Description: These assembly language drivers are designed to interface a user program written in Applesoft BASIC to a Westinghouse PC700/900/1100 processor. The various routines are described individually.

Hardware Overview and I/O Addressing (Super Serial Card)

This program was designed to allow the Apple's 6502 microprocessor access serial data arriving from an Apple Super Serial Card (SSC) located in slot 1 of the computer. The routines also permit the Apple to transmit data out of the Super Serial Card. To assist in the understanding of these routines, the following memory address of the Apple and SSC are explained.

Address (Hex)/(Decimal)		Bit(s)	Interpretation
C091/49297	DIPSW1	0	Dip Switch 1 - 6 is off when 1, on when 0
		1	Dip Switch 1 - 5 is off when 1, on when 0
		4-7	same as above for 1-4 through 1-1
C092/49298	DIPSW2	0	CTS (pin 5 of serial port) true (-) when 0
		1-3	same as above for 2-5 through 2-3
		5,7	same as above for 2-2 through 2-1
C098/49304	TDREG	0-7	ACIA transmit register
	RDREG	0-7	ACIA receive register Note that both registers occupy the same address space in the Apple's memory map.
C099/49305	STATUS	0	parity error if 1
		1	framing error if 1
		2	overrun error if 1
		3	ACIA receive register has character when 1
		4	ACIA transmit register empty when 1
		5	DCD (pin 8 of serial port) true when 0
		6	DSR (pin 6 of serial port) true when 0
		7	Interrupt (IRQ) has occurred when 1. Note: interrupts are not used by these routines.
C09A/49306	COMMAND	0	1. Enable DTR (pin 20 of serial port) if 1

			2. Enable receiver if 1 3. Enable all interrupts if 1		
		1	If 1, allow bit 3 of STATUS register to cause interrupt.		
		2,3	Controls RTS, transmitter enable, and transmitter interrupt. Note: This program sets bit 3 and clears bit 2 whenever the SYNC1 routine is called. These bits are not controlled by any other routine or program. If both bits 2 and 3 are turned on simultaneously, a BREAK signal is sent for the duration of the time that bits 2 and 3 are set.		
		4	Echo mode if 1. Note: This program resets this bit.		
		5-7	Controls parity	0,2,4,6	none
				1	odd
				3	even
				5	mark
				7	space
C09B/49307	CONTROL	0-3	Controls Baud Rate	0	16 * ext. clk.
				1	50
				2	75
				3	109.92
				4	134.58
				5	150
				6	300
				7	600
				8	1200
				9	1800
				A	2400
				B	3600
				C	4800
				D	7200
				E	9600
				F	19200
		4	When 1 use internal baud rate generator. When 0 use external clock.		
		5,6	Number of Data Bits	5 6	Number
				0 0	8
				1 0	7
				0 1	6
				1 1	5
		7	Number of Stop Bits	Bit	Stop Bits
				0	1
			5 data bits	1	1.5
			8 data + parity		1
			all other		2

Buffer Locations used by Program

Address (Hex/Decimal)	Name	Description
7100/28928	Tx OP	Start of transmit buffer. Opcode
7101/28929	Tx LA	Low byte of transmitted address
7102/28930	Tx HA	High byte of transmitted address
7103/28931	Tx LD	Low byte of transmitted data
7104/28932	Tx HD	High byte of transmitted data
7105/28933	Tx CK	Transmitted checksum
7106/28934	Rx OP	1st byte received from PC. Opcode
7107/28935	Rx LA	2nd byte received from PC. Low address
7108/28936	Rx HA	3rd byte received from PC. High address
7109/28937	Rx LD	4th byte received from PC. Low data
710A/28938	Rx HD	5th byte received from PC. High data
710B/28939	Rx CK	6th byte received from PC. Checksum
710C/28940		(Tx CK) - (Rx CK). Should be zero
710D/28941		Bytes received pointer
710E/28942		Error code received from SYNC1. Should be zero.
710F/28943		Temporary register used by GSTA
7110/28944		Low byte of starting address used by FCLEAR
7111/28945		High byte of starting address used by FCLEAR
7112/28946		Low byte of ending address used by FCLEAR
7113/28947		High byte of ending address used by FCLEAR
7114/28948		Low byte of EOP. (Temp location)
7115/28949		High byte of EOP. (Temp location)
7116/28950		Low byte of HRRU. (Temp location)
7117/28951		High byte of HRRU. (Temp location)
7118/28951		Low byte of Low Ladder Checksum. (Temp loc.)
7119/28952		Hi byte of Low Ladder Checksum. (Temp loc.)
711A/28953		Low byte of Hi Ladder Checksum. (Temp loc.)
711B/28953		Hi byte of Hi Ladder Checksum. (Temp loc.)
711C/28956		Low byte of PC starting address
711D/28957		High byte of PC starting address
711E/28958		Low byte of PC ending address

711F/28959	High byte of PC ending address
7120/28960	Low byte decimal data used by HEXCONV
7121/28961	High byte decimal data used by HEXCONV
7122/28962	Low byte of PC starting address. (FSTORE)
7123/28963	High byte of PC starting address. (FSTORE)

Entry points of Assembly Calls

Name	Address	Description
Sync1	26368(6700h)	Synchronizes with programmable controller
Chxsum	24912(6150h)	Error check returned data
Hexconv	25152(6240h)	Compute Hex value from decimal
Xsum	24880(6130h)	Compute checksum
Fclear	26624(6800h)	Clears a group of PC addresses
Incadr	24992(61A0h)	Increment PC address
Fdump	26464(6760h)	Offload PC memory
Zero	24864(6120h)	Zero 7103h and 7104h, load 7100 with 1
Fstore	26548(67B4h)	Store data to PC
DataIO	26112(6600h)	Transmit buffer, receive reply
Bufinc	26976(6960h)	Increment Apple buffer pointer
Gsta	25088(6200h)	Get PC parameter table
Ptrsave	26720(6860h)	Save buffer pointers
Bufclr	25184(6260h)	Clear Apple program buffer
Decadr	25172(6254h)	Decrement PC address
Comlink	29440(7300h)	Receives data from and prints hex on screen

Description of Apple Assembly Communications Routines

Name : Comlink

Description: Receives characters from the serial port, converts them to hex codes, and displays on the CRT screen. Baud rate can be changed by writing to location 730DH/29453D, while parity can be controlled by writing to 7308H/29448D.

```
7300-   A2 00       LDX   #$00
7302-   A9 00       LDA   #$00
7304-   8D 99 C0    STA   $C099   ;clear status register
7307-   A9 29       LDA   #$29    ;odd parity, enable transmitter
7309-   8D 9A C0    STA   $C09A   ;load to command register
730C-   A9 9E       LDA   #$9E    ;9600 baud, 2 stops, 8 data bits
730E-   8D 9B C0    STA   $C09B   ;load to control register
7311-   AD 99 C0    LDA   $C099   ;get status register
7314-   29 08       AND   #$08    ;if bit 3 is set, character has
                                  ; been recv'd
7316-   F0 0E       BEQ   $7326   ;if not, go check keyboard for
                                  ; data
7318-   AD 98 C0    LDA   $C098   ;otherwise, get character from
                                  ; serial port
731B-   20 DA FD    JSR   $FDDA   ;display accumulator as hex byte
                                  ; on CRT
731E-   A2 01       LDX   #$01    ;load space factor = 1
7320-   20 4A F9    JSR   $F94A   ;advance cursor by space factor
7323-   18          CLC           ;set up unconditional jump
7324-   90 EB       BCC   $7311   ;unconditional jump to 7311H
7326-   AC 00 C0    LDY   $C000   ;get Keyboard Data Input
7329-   30 06       BMI   $7331   ;MSB set? If so, keyboard data
                                  ; received
732B-   AE 10 C0    LDX   $C010   ;if not, clear keyboard strobe
732E-   18          CLC           ;set up unconditional jump
732F-   90 E0       BCC   $7311   ;unconditional jump to 7311H
7331-   A9 92       LDA   #$92    ;load ASCII code for "control R"
7333-   CD 00 C0    CMP   $C000   ;was ^R pushed?
7336-   F0 0D       BEQ   $7345   ;if so, then prepare to quit
7338-   20 58 FC    JSR   $FC58   ;if not, clear screen
733B-   AD 10 C0    LDA   $C010   ;clear keyboard strobe
733E-   18          CLC           ;set up unconditional jump
733F-   90 D0       BCC   $7311   ;unconditional jump to 7311H

7345-   AD 10 30    LDA   $C010   ;clear keyboard strobe
7348-   60          RTS           ;return to calling program
```

Name : CHXSUM

Description: Error checks received data by computing a checksum of the first 5 bytes and comparing this checksum with the 6th byte. If the checksum does not match, the location 710CH will not equal zero. To detect an invalid checksum in the returned data, simply call this

routine, then test 710C for a non-zero value. If the value at this location is not equal to zero, the received data checksum was invalid.

```
6150-   A2 00       LDX   #$00     ;clear pointer
6152-   AD 06 71    LDA   $7106    ;get first byte of received data
6155-   18          CLC            ;clear carry
6156-   7D 07 71    ADC   $7107,X  ;add to next byte
6159-   E8          INX            ;increment to next byte
615A-   E0 04       CPX   #$04     ;have all 5 bytes been added
                                   ; together?
615C-   D0 F7       BNE   $6155    ;if not, get next byte
615E-   38          SEC            ;set carry
615F-   ED 0B 71    SBC   $710B    ;compare computed checksum with
                                   ; calculated
6162-   8D 0C 71    STA   $710C    ;store this difference at 710CH
6165-   60          RTS            ;return to calling routine
```

Name : HEXCONV

Description: Routine to convert decimal data to hexadecimal. Uses Apple Monitor call F941H. To use, place low byte of data at address 7120H/28960D and high byte of data at 7121H/28961D.

```
6240-   AD 21 71    LDA   $7121
6243-   AE 20 71    LDX   $7120
6246-   20 41 F9    JSR   $F941
6249-   60          RTS
```

Name : stored with HEXCONV

Description: Entry point for a routine to simply take the received data from the PC, convert the 4th and 5th byte to Hex, and display on screen.

```
624A-   AD 0A 71    LDA   $710A
624D-   AE 09 71    LDX   $7109
6250-   20 41 F9    JSR   $F941
6253-   60          RTS
```

Name : DECADR

Description: Decrements address pointer used to communicate with PC.

```
6254-   18          CLC
6255-   CE 01 71    DEC   $7101    ;decrement low byte of address
                                   ; pointer
6258-   F0 01       BEQ   $625B    ;new value = 0? If so, continue
625A-   60          RTS            ;otherwise, return to calling
                                   ; routine
625B-   CE 02 71    DEC   $7102    ;decrement high byte of address
```

```
                                        ; pointer
625E-   60          RTS                 ;return to calling program
```

Name : BUFCLR

Description: Clears the received data buffer of the Apple. This buffer is only used when a block of data is offloaded from the PC or downloaded to the PC. This block is located at address 4000H in the Apple memory map.

```
6260-   A9 00       LDA   #$00    ;
6262-   8D 01 71    STA   $7101
6265-   A9 40       LDA   #$40
6267-   8D 02 71    STA   $7102
626A-   A9 00       LDA   #$00
626C-   85 E0       STA   $E0
626E-   A9 40       LDA   #$40
6270-   85 E1       STA   $E1
6272-   A9 00       LDA   #$00
6274-   8D 1E 71    STA   $711E
6277-   A9 50       LDA   #$50
6279-   8D 1F 71    STA   $711F
627C-   A0 00       LDY   #$00
627E-   A9 00       LDA   #$00
6280-   91 E0       STA   ($E0),Y
6282-   C8          INY
6283-   91 E0       STA   ($E0),Y
6285-   20 60 69    JSR   $6960
6288-   B0 06       BCS   $6290
628A-   20 A0 61    JSR   $61A0
628D-   18          CLC
628E-   90 EC       BCC   $627C
6290-   60          RTS
```

Name : XSUM

Description: Computes checksum and loads this value in the transmit message buffer.

```
6130-   A2 00       LDX   #$00    ;clear pointer
6132-   AD 00 71    LDA   $7100   ;get 1st byte of transmit message
                                  ; buffer
6135-   18          CLC           ;clear carry
6136-   7D 01 71    ADC   $7101,X ;add to next byte of buffer
6139-   E8          INX           ;get next byte
613A-   E0 04       CPX   #$04    ;got all 5 bytes?
613C-   D0 F7       BNE   $6135   ;if not, get next byte
613E-   8D 05 71    STA   $7105   ;if so, store computed checksum at
                                  ; 7105H
6141-   60          RTS           ;return to calling program
```

Name : FCLEAR

Description: Clears a block of addresses in the programmable controller.

```
6800-   AD 10 71    LDA   $7110   ;get low byte of starting address
6803-   8D 01 71    STA   $7101   ;put this at low byte of address
                                  ; pointer
6806-   AD 11 71    LDA   $7111   ;get high byte of starting address
6809-   8D 02 71    STA   $7102   ;put this at high byte of address
                                  ; pointer
680C-   A9 01       LDA   #$01    ;get "write word" opcode
680E-   8D 00 71    STA   $7100   ;store this opcode at first byte
                                  ; of msg buf.
6811-   A9 00       LDA   #$00    ;load "zero" as data....
6813-   8D 03 71    STA   $7103   ;first to the low byte of address
                                  ; pointer,
6816-   8D 04 71    STA   $7104   ;then to the high byte
6819-   EA          NOP
681A-   EA          NOP
681B-   EA          NOP
681C-   20 30 61    JSR   $6130   ;calculate transmitted checksum
681F-   EA          NOP
6820-   EA          NOP
6821-   EA          NOP
6822-   20 00 66    JSR   $6600   ;transmit 6 byte string out serial
                                  ; port
6825-   EA          NOP
6826-   EA          NOP
6827-   EA          NOP
6828-   AD 0C 71    LDA   $710C   ;load "return data error" register
682B-   D0 1B       BNE   $6848   ;if error, then quit
682D-   EA          NOP
682E-   EA          NOP
682F-   EA          NOP
6830-   AD 02 71    LDA   $7102   ;otherwise, get high byte of add.
                                  ; pointer
6833-   CD 13 71    CMP   $7113   ;compare this with high byte of
                                  ; end pointer
6836-   D0 11       BNE   $6849   ;if not the same, then continue
6838-   AD 01 71    LDA   $7101   ;otherwise, get low byte of add.
                                  ; pointer
683B-   CD 12 71    CMP   $7112   ;compare this with low byte of
                                  ; end. pointer
683E-   D0 09       BNE   $6849   ;if not the same, then continue
6840-   EA          NOP
6841-   EA          NOP
6842-   EA          NOP
6843-   EA          NOP
6844-   EA          NOP
6845-   EA          NOP
6846-   EA          NOP
6847-   EA          NOP
6848-   60          RTS           ;otherwise it is time to quit
6849-   20 A0 61    JSR   $61A0   ;increment address pointer
684C-   A9 00       LDA   #$00    ;set up unconditional jump
684E-   F0 CC       BEQ   $681C   ;unconditionally jump to 681CH
```

Name : PTRSAVE

Description: Saves buffer pointers

```
6860-   AD 1C 71    LDA   $711C
6863-   8D 24 71    STA   $7124
6866-   AD 1D 71    LDA   $711D
6869-   8D 25 71    STA   $7125
686C-   AD 1E 71    LDA   $711E
686F-   8D 26 71    STA   $7126
6872-   AD 1F 71    LDA   $711F
6875-   8D 27 71    STA   $7127
6878-   60          RTS
```

Name : INCADR

Description: Increments PC address pointer

```
61A0-   18          CLC
61A1-   EE 01 71    INC   $7101
61A4-   F0 01       BEQ   $61A7
61A6-   60          RTS
61A7-   EE 02 71    INC   $7102
61AA-   60          RTS
```

Name : FDUMP

Description: Offloads a block of memory from the programmable controller. This data from the PC is placed in a 16K block of memory in the Apple][+ starting at address 4000h.

```
Start of Program     : 26464/6760
Start of Buffer      : 16384/4000
PC Starting Address  : 28956/711C
                       28957/711D
PC Ending Address    : 28958/711E
                       28959/711F
Buffer Pointer       :   224/00E0  same as FSTORE
                         225/00E1   "    "    "
```

```
6760-   18          CLC
6761-   EA          NOP
6762-   AD 1C 71    LDA   $711C
6765-   8D 01 71    STA   $7101
6768-   AD 1D 71    LDA   $711D
676B-   8D 02 71    STA   $7102
676E-   A9 00       LDA   #$00
6770-   85 E0       STA   $E0
6772-   8D 00 71    STA   $7100
```

```
6775-   A9 40       LDA   #$40
6777-   85 E1       STA   $E1
6779-   EA          NOP
677A-   EA          NOP
677B-   EA          NOP
677C-   EA          NOP
677D-   EA          NOP
677E-   EA          NOP
677F-   20 30 61    JSR   $6130
6782-   20 00 66    JSR   $6600
6785-   20 50 61    JSR   $6150
6788-   EA          NOP
6789-   EA          NOP
678A-   EA          NOP
678B-   EA          NOP
678C-   EA          NOP
678D-   EA          NOP
678E-   AD 0C 71    LDA   $710C
6791-   D0 20       BNE   $67B3
6793-   A0 00       LDY   #$00
6795-   EA          NOP
6796-   EA          NOP
6797-   EA          NOP
6798-   EA          NOP
6799-   AD 09 71    LDA   $7109
679C-   91 E0       STA   ($E0),Y
679E-   C8          INY
679F-   AD 0A 71    LDA   $710A
67A2-   91 E0       STA   ($E0),Y
67A4-   20 60 69    JSR   $6960
67A7-   EA          NOP
67A8-   EA          NOP
67A9-   EA          NOP
67AA-   EA          NOP
67AB-   B0 06       BCS   $67B3
67AD-   20 A0 61    JSR   $61A0
67B0-   18          CLC
67B1-   90 CC       BCC   $677F
67B3-   60          RTS
```

Name : FSTORE

Description: An assembly language program that copies a block of memory from the Apple][+ computer and transmits this block to a Numa-Logic PC700/900/1100 programmable controller.

```
Start of Program     : 26548/67B4
Start of Buffer      : 16384/4000
PC Starting Address  : 28962/7122
                       28963/7123
PC Ending Address    : 28958/711E
                       28959/711F
Buffer Pointer       :   224/00E0  same as FSTORE
```

```
                                        225/00E1    "     "     "
                Buffer End Pointer  :   228/00E4
                                        229/00E5

67B4-   A5 E0       LDA   $E0
67B6-   85 E4       STA   $E4
67B8-   A5 E1       LDA   $E1
67BA-   85 E5       STA   $E5
67BC-   A9 01       LDA   #$01
67BE-   8D 00 71    STA   $7100
67C1-   AD 22 71    LDA   $7122
67C4-   8D 01 71    STA   $7101
67C7-   AD 23 71    LDA   $7123
67CA-   8D 02 71    STA   $7102
67CD-   A9 00       LDA   #$00
67CF-   85 E0       STA   $E0
67D1-   A9 40       LDA   #$40
67D3-   85 E1       STA   $E1
67D5-   A0 00       LDY   #$00
67D7-   B1 E0       LDA   ($E0),Y
67D9-   8D 03 71    STA   $7103
67DC-   C8          INY
67DD-   B1 E0       LDA   ($E0),Y
67DF-   8D 04 71    STA   $7104
67E2-   20 30 61    JSR   $6130
67E5-   20 00 66    JSR   $6600
67E8-   20 50 61    JSR   $6150
67EB-   AD 0C 71    LDA   $710C
67EE-   D0 0B       BNE   $67FB
67F0-   20 60 69    JSR   $6960
67F3-   B0 06       BCS   $67FB
67F5-   20 A0 61    JSR   $61A0
67F8-   18          CLC
67F9-   90 DA       BCC   $67D5
67FB-   60          RTS
```

Name : ZERO

Description: Loads zero into Apple memory locations 7103H and 7104H. This program also loads the opcode register (7100H) with the value one in preparation for a write command.

```
6120-   A9 00       LDA   #$00
6122-   8D 03 71    STA   $7103
6125-   8D 04 71    STA   $7104
6128-   A9 01       LDA   #$01
612A-   8D 00 71    STA   $7100
612D-   60          RTS
```

Name : XSUM

Description: Calculates the proper checksum from the values stored in Apple memory locations 7100H through 7104H, inclusive. The calculated checksum is loaded into Apple memory location 7105H.

```
6130-   A2 00       LDX   #$00
6132-   AD 00 71    LDA   $7100
6135-   18          CLC
6136-   7D 01 71    ADC   $7101,X
6139-   E8          INX
613A-   E0 04       CPX   #$04
613C-   D0 F7       BNE   $6135
613E-   8D 05 71    STA   $7105
6141-   60          RTS
```

Name : DATAIO

Description: Apple program to transmit a 6 byte message to a programmable controller and wait for a response.

```
6600-   A2 00       LDX   #$00
6602-   A0 00       LDY   #$00
6604-   EA          NOP
6605-   EA          NOP
6606-   EA          NOP
6607-   EA          NOP
6608-   EA          NOP
6609-   8E 00 70    STX   $7000
660C-   AD 98 C0    LDA   $C098
660F-   EA          NOP
6610-   EA          NOP
6611-   BD 00 71    LDA   $7100,X
6614-   8D 98 C0    STA   $C098
6617-   E8          INX
6618-   E0 06       CPX   #$06
661A-   F0 09       BEQ   $6625
661C-   AD 99 C0    LDA   $C099
661F-   29 10       AND   #$10
6621-   F0 F9       BEQ   $661C
6623-   D0 EC       BNE   $6611
6625-   A2 00       LDX   #$00
6627-   EA          NOP
6628-   EA          NOP
6629-   EA          NOP
662A-   AD 99 C0    LDA   $C099
662D-   29 08       AND   #$08
662F-   D0 0F       BNE   $6640
6631-   EA          NOP
6632-   C8          INY
6633-   D0 F2       BNE   $6627
6635-   EE 00 70    INC   $7000
6638-   D0 ED       BNE   $6627
```

```
663A-   A9 01       LDA   #$01
663C-   0D 0C 71    ORA   $710C
663F-   60          RTS
6640-   A0 00       LDY   #$00
6642-   8C 00 70    STY   $7000
6645-   AD 98 C0    LDA   $C098
6648-   9D 06 71    STA   $7106,X
664B-   E8          INX
664C-   E0 06       CPX   #$06
664E-   D0 D7       BNE   $6627
6650-   EA          NOP
6651-   AD 98 C0    LDA   $C098
6654-   8E 0D 71    STX   $710D
6657-   60          RTS
```

Name : BUFINC

Description: This program increments the buffer pointer memory locations in the Apple memory.

```
6960-   18          CLC
6961-   E6 E0       INC   $E0
6963-   E6 E0       INC   $E0
6965-   EA          NOP
6966-   EA          NOP
6967-   EA          NOP
6968-   EA          NOP
6969-   A5 E0       LDA   $E0
696B-   C9 00       CMP   #$00
696D-   F0 0F       BEQ   $697E
696F-   AD 1E 71    LDA   $711E
6972-   CD 01 71    CMP   $7101
6975-   F0 09       BEQ   $6980
6977-   18          CLC
6978-   60          RTS
```

Name : GSTA

Description: This program offloads the programmable controller parameter table and loads this information into the following Apple memory locations:

25856	6500	End of program pointer
57	01	
25858	6502	Highest Holding Register Used
59	03	
25860	6504	Low ladder checksum
61	05	
25862	6506	High ladder checksum
63	07	

25864	6508	Register checksum
65	09	
25866	650A	Flag register
67	0B	Mode register
25868	650C	Error register
69	0D	
25870	650E	RESERVED
:	:	:
25879	6517	RESERVED
25880	6518	Monitor Table Address
81	19	
25882	651A	Output Register Address
83	1B	
25884	651C	Input Register Address
85	1D	
25886	651E	OR's allowed
87	1F	IR's allowed
25888	6520	Force table address - outputs
89	21	
25890	6522	Force table address - inputs
91	23	
25892	6524	Output image table address
93	25	
25894	6526	Input image table address
95	27	
25896	6528	Number of outputs divided by 8
97	29	Number of inputs divided by 8
25898	652A	Maximum discrete coils allowed
99	2B	Memory size divided by 256
25900	652C	Software version
01	2D	Executive code (upper 4 bits-product code) (lower 4 bits-modifier)
25902	652E	Special Functions Allowed
:	:	:
25919	653F	Special Functions Allowed

6200-	EA	NOP
6201-	EA	NOP
6202-	EA	NOP

```
6203-   A9 00       LDA   #$00
6205-   8D 0F 71    STA   $710F
6208-   8D 00 71    STA   $7100
620B-   8D 01 71    STA   $7101
620E-   A9 82       LDA   #$82
6210-   8D 02 71    STA   $7102
6213-   20 30 61    JSR   $6130
6216-   20 00 66    JSR   $6600
6219-   EA          NOP
621A-   EA          NOP
621B-   EA          NOP
621C-   AE 0F 71    LDX   $710F
621F-   AD 09 71    LDA   $7109
6222-   9D 00 65    STA   $6500,X
6225-   AD 0A 71    LDA   $710A
6228-   9D 01 65    STA   $6501,X
622B-   EE 0F 71    INC   $710F
622E-   EE 0F 71    INC   $710F
6231-   20 A0 61    JSR   $61A0
6234-   AD 0F 71    LDA   $710F
6237-   C9 40       CMP   #$40
6239-   D0 D8       BNE   $6213
623B-   60          RTS
```

Name : SYNC1

Description: Routine to synchronize with the programmable controller by sending null (00H) bytes, while continuously looking for a response.

```
6700-   A2 00       LDX   #$00
6702-   A9 00       LDA   #$00
6704-   8D 99 C0    STA   $C099
6707-   8D 00 70    STA   $7000
670A-   A9 29       LDA   #$29
670C-   8D 9A C0    STA   $C09A
670F-   A9 9E       LDA   #$9E
6711-   8D 9B C0    STA   $C09B
6714-   EA          NOP
6715-   EA          NOP
6716-   EA          NOP
6717-   EA          NOP
6718-   EA          NOP
6719-   EA          NOP
671A-   A2 00       LDX   #$00
671C-   A9 00       LDA   #$00
671E-   8D 98 C0    STA   $C098
6721-   A9 FF       LDA   #$FF
6723-   EA          NOP
6724-   20 A8 FC    JSR   $FCA8
6727-   AD 99 C0    LDA   $C099
672A-   29 08       AND   #$08
672C-   F0 0E       BEQ   $673C
```

```
672E-   A9 00       LDA   #$00
6730-   CD 98 C0    CMP   $C098
6733-   D0 07       BNE   $673C
6735-   60          RTS
6736-   A9 10       LDA   #$10
6738-   20 00 FC    JSR   $FC00
673B-   EA          NOP
673C-   EE 00 70    INC   $7000
673F-   A9 10       LDA   #$10
6741-   CD 00 70    CMP   $7000
6744-   D0 D4       BNE   $671A
6746-   A9 01       LDA   #$01
6748-   0D 0E 71    ORA   $710E
674B-   8D 0E 71    STA   $710E
674E-   60          RTS
```

Appendix J: Radio Shack Assembly Calls (TRSDOS)

The Radio Shack Model II computer has a convenient method of communicating through the serial port using assembly language. That method is called "Supervisory Calls (SVC)". For more information on the TRSDOS SVC, consult your TRSDOS manual.

The program was written in Z80 assembly language. The reader should be familiar with this language to analyze the program.

The following program is named "COMLINK" and is loaded into high memory just before transferring control to BASIC. Some abbreviations include:

LB	Lower Byte
UB	Upper Byte
SVC	Supervisory Call
B/TX	Channel B transmit SVC
B/RCV	Channel B receive SVC
ASA	Argument Storage Area (Consult your Model II BASIC manual)

```
                  00000
                   00100          ENTRY RCV
0000'              00200          ASEG
                   00300          ORG  0E0000H    ;START OF PROGRAM
E000   3E 62       00400     RCV: LD   A,62H      ;LOAD B/RCV OP CODE
E002   CF          00500          RST  8          ;EXECUTE SVC
E003   C2 E008     00600          JP   NZ,NOCHR   ;CHAR. FOUND?
E006   70          00700          LD   (HL),B     ;LOAD CHR TO LB ASA
E007   C9          00800          RET             ;RETURN TO BASIC
E008   E5          00900   NOCHR: PUSH HL         ;PUSH ADD OF LB ASA
E009   DD E1       00950          POP  IX         ;MOVE HL INTO IX
E00B   DD 23       01000          INC  IX         ;INC UPPER BYTE ADD.
E00D   C6 01       01025          ADD  A,01H      ;ADD 256 TO ERR REG.
E00F   DD 77 00    01050          LD   (IX),A     ;LOAD STATUS TO U.B.
E012   C9          01100          RET             ;RETURN TO BASIC
E013   46          01300      TX: LD   B,(HL)     ;LOAD CHR TO BE SENT
E014   3E 63       01400          LD   A,63H      ;LOAD B/TX OP CODE
E016   CF          01500          RST  8          ;EXECUTE SVC
E017   C2 E01B     01600          JP   NZ,NTX     ;IF NOT TX'ED, JMP
E01A   C9          01700          RET             ;RETURN TO BASIC
E01B   E5          01800     NTX: PUSH HL         ;PUSH L.B. OF ASA
E01C   DD E1       01850          POP  IX         ;MOVE HL INTO IX
E01E   DD 23       01900          INC  IX         ;INC U.B. ADD.
E020   DD 77 00    01950          LD   (IX),A     ;LOAD STATUS TO U.B.
E023   C9          02000          RET             ;RETURN TO BASIC
                   02200          END
```