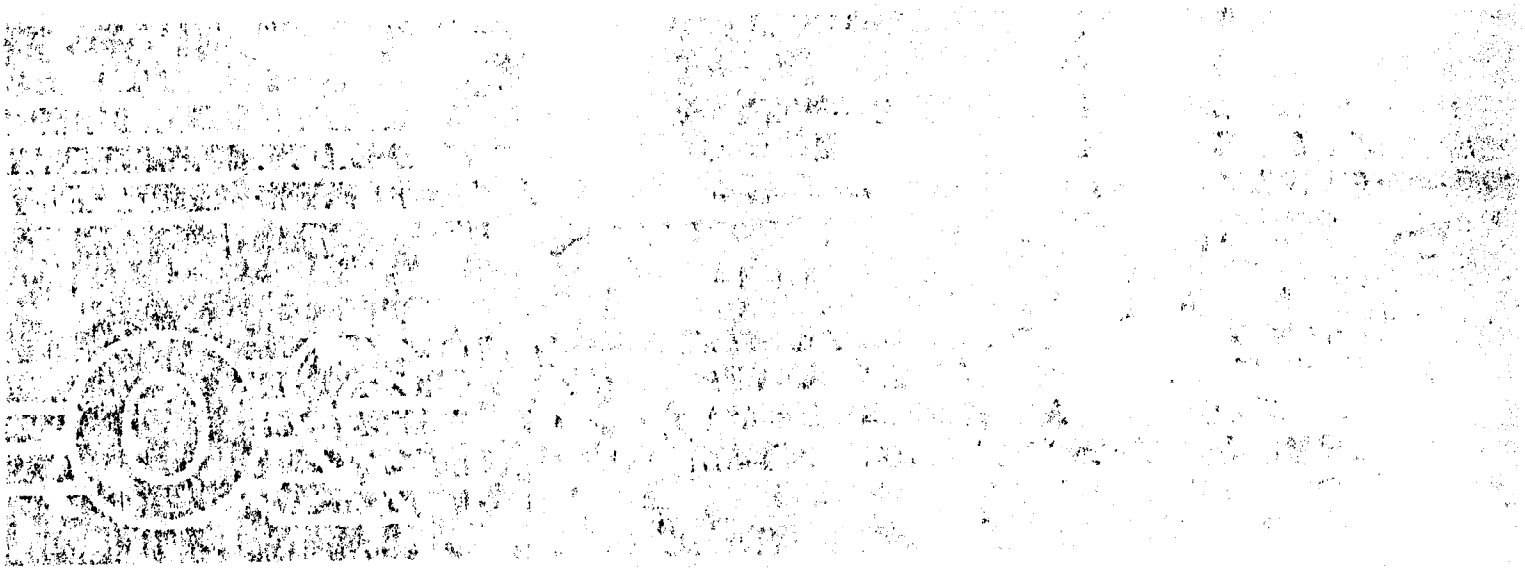
M20

I/O with External Peripherals User Guide

olivetti L1



M20

I/O with External Peripherals User Guide

olivetti L1

PREFACE

This M20 manual describes the RS-232-C (serial) and IEEE 488 (parallel) communications interfaces and their use with external peripherals. The first chapter introduces the reader to the concepts used to describe the interfaces discussed in the two latter parts: part one covers RS-232-C and part two covers IEEE 488. Each part consists of three chapters. The first is on specific interface related concepts: hardware, software, and programming. The second is on the commands and statements required to program the interface. The last chapter describes some typical examples of the use of the interface and includes actual programs.

The reader is assumed to be familiar with the corresponding interface Standard and to have had experience with programming interfaces for such peripherals as well as experience of using PCOS and M20 BASIC.

Some parts of this manual are derived directly (or partially modified) from the RS-232-C and IEEE 488 Standards (see Appendix A).

REFERENCES:

PCOS (Professional Computer Operating System) User Guide.

BASIC Language Reference Guide.

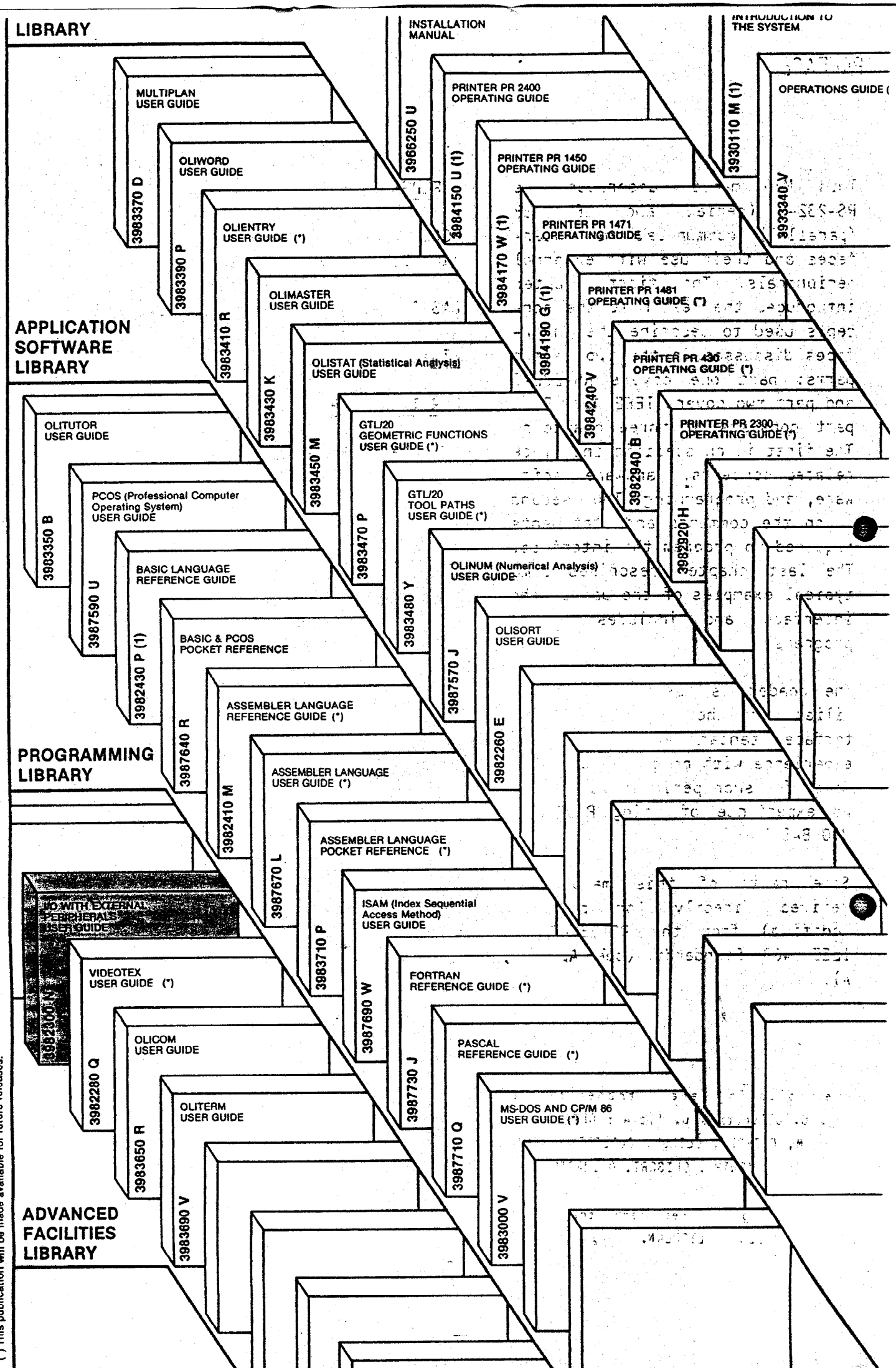
DISTRIBUTION: General (G)

FIRST EDITION: October 1982

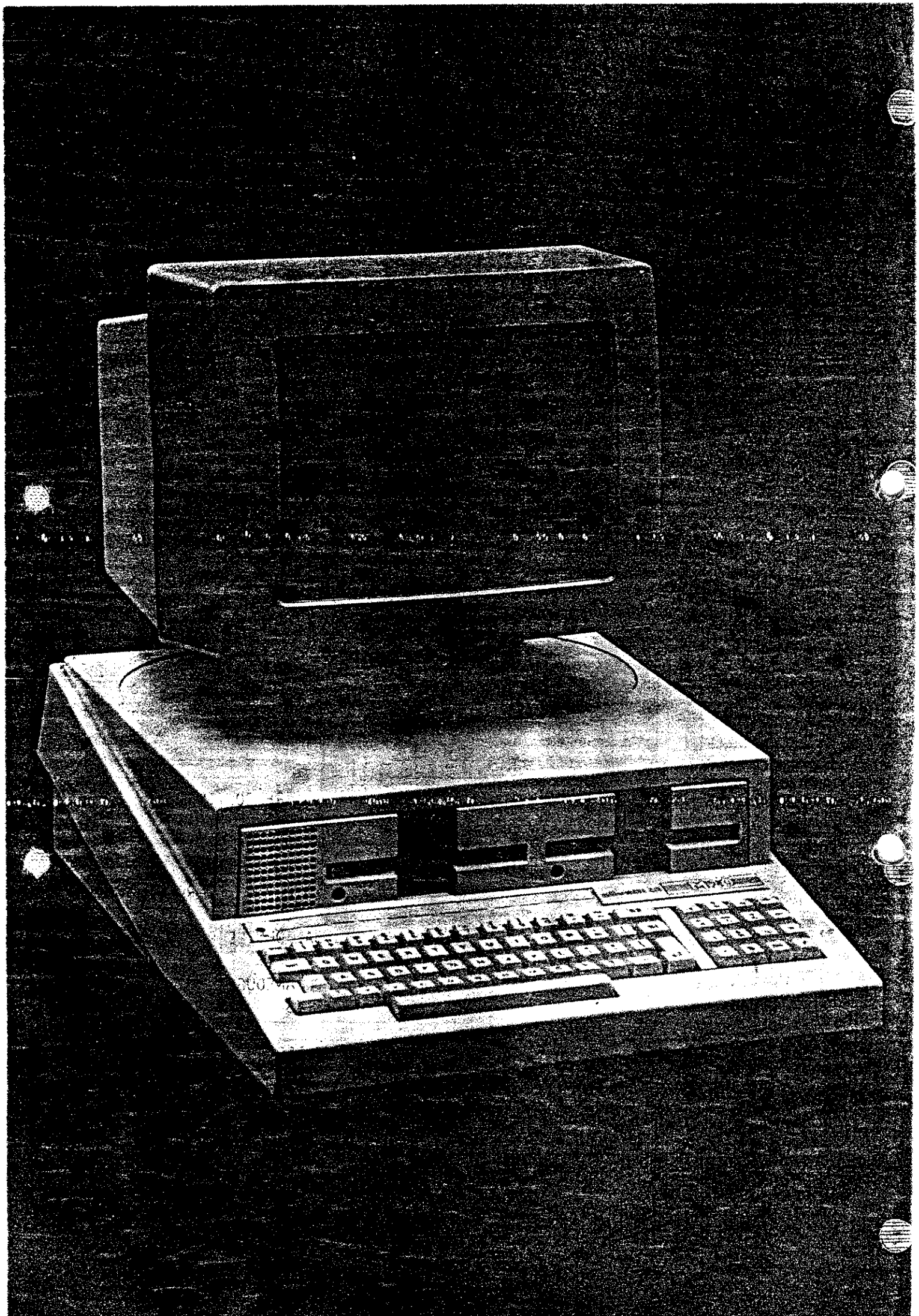
RELEASE: 1.3 Onwards

PUBLICATION ISSUED BY:

Ing. C. Olivetti & C., S.p.A.
Servizio Centrale Documentazione
77, Via Jervis-10015 IVREA (Italy)



(*) This publication will be made available for future releases.



CONTENTS

CONTENTS

1. GENERAL INTRODUCTION

ABOUT THIS CHAPTER

This chapter introduces the reader to the RS-232-C (serial) and IEEE 488 (parallel) interfaces and the concepts used in the remainder of the manual to describe them. In addition the major classes of connectable peripherals are also described.

CONTENTS

GENERAL INTRODUCTION

THE TWO COMMUNICATIONS INTERFACES

This M20 manual describes the two communications interfaces: the RS-232-C and IEEE 488 for connecting serial and parallel peripherals respectively.

The serial interface, RS-232-C, is available in two versions:

- Single RS-232-C which is built into the M20.

- Dual RS-232-C which is available as an option.

The parallel interface, IEEE 488, is only available as an option.

These two interfaces handle digital data transmission asynchronously, character by character (byte by byte). The transmission may be half duplex (i.e. in either direction - in or out of the M20 - but only one way at a time) or full duplex (both ways at the same time). Transmission can be:

- serial (bit by bit) on the SERIAL interface, or

- parallel (all bits together) on the PARALLEL interface

through a communication port. For serial transmission, the transmitted bits are output in sequence, whereas in parallel transmission the characters are output serially but the bits representing each character are output in parallel. See Figure 1-1.

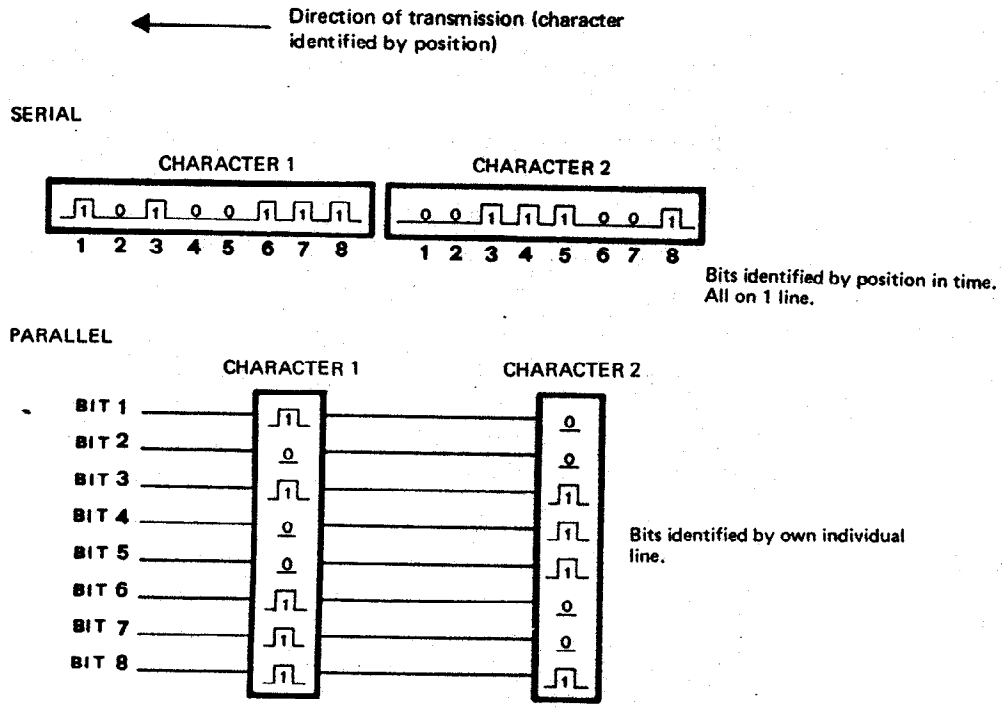


Figure 1-1 Serial and Parallel Transmission

SERIAL TRANSMISSION

Serial transmission is used for slower speed (under 2ØØØ baud), long distance (over 6Ø feet/2Ø m) M20-to-modem to communications equipment transfers. However, serial transmission is also extensively used for high-speed (at 96ØØ baud) direct wire connection of M20-to-terminals where the connection distance is less than 6Ø feet/2Ø m.

The serial interface is designed according to the Recommended Standard 232 C of the Electronic Industries Association of Washington, DC, USA and hence is referred to as EIA RS-232-C or by its short name of RS-232-C. (The European equivalent is Recommendation V.24 of the International Consultative Committee for Telephone and Telegraph, which is better known under its French abbreviation as CCITT V.24. See also Appendix A.)

PARALLEL TRANSMISSION

Parallel transmission is used extensively in M20-to-peripheral transfers at higher speed (over 5ØØØ baud), over short distances (less than 6Ø feet/2Ø m) but rarely for long distances due to the high cost of the

additional circuits required. The IEEE interface is used typically to connect instrumentation.

The parallel interface is designed according to the Standard 488 of the Institute of Electrical and Electronics Engineers of New York, NY, USA and hence its short name of IEEE 488. (See also Appendix A.)

CONCEPTS

In this manual, no attempt is made to differentiate between the various types of equipment which may be connected together using these interfaces.

A device is a generic term for a piece of equipment, and may be an instrument, a peripheral or a machine. An instrument is a device used as an implement or tool for scientific purposes (e.g. to measure temperature). A peripheral is generally a device which can be connected externally to a computer (e.g. a badge reader). A machine is a device which performs a particular function via a logic interface (e.g. a numerically controlled lathe) while a computer has calculation and logic capabilities (e.g. the M20).

Since this manual deals with the interfaces of a particular computer, the M20, and no particular device, the terms device, peripheral, instrument and machine can be thought of as interchangeable, unless otherwise specified.

An interface is a generalised means of communication between the M20 and some external input or output device. Hence the standards specify the necessary conventions for use, as well as electrical and mechanical requirements. (On the M20 Olivetti has provided the necessary hardware and software to support peripherals conforming to the operational requirements of these standards in terms of device function and logic.)

The M20 provides a serial interface that allows it to be connected in a wide variety of configurations to a number of peripheral units, either on-line or in a network.

Figure 1-2 shows an M20 configuration that allows for the handling of peripherals either directly connected or via modem with a connection to another computer.

Generally, any device that conforms to the EIA Standard RS-232-C or a 20 mA Current Loop interface can be used.

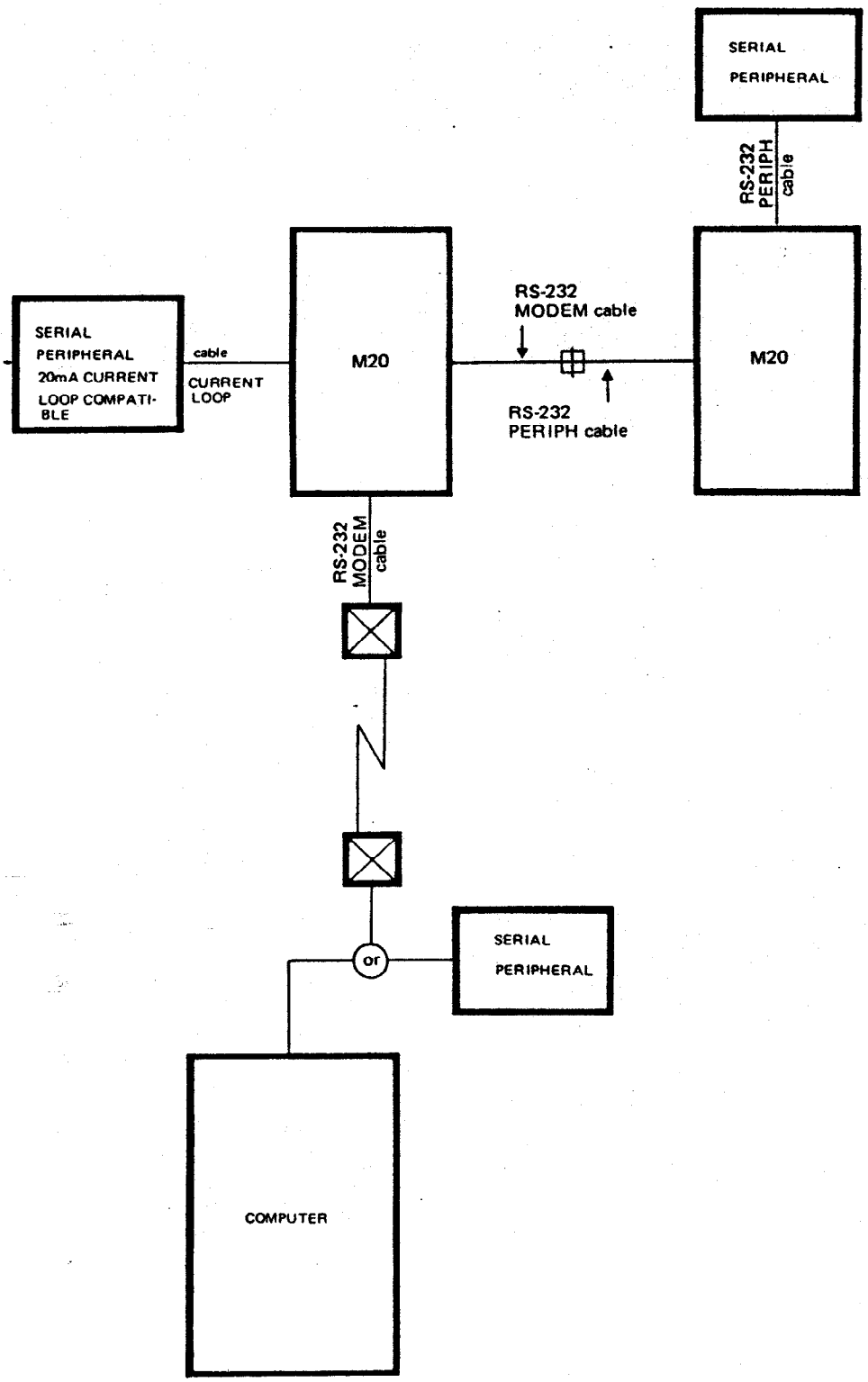


Figure 1-2 M20 System Configuration Example

The EIA RS-232-C Interface

The use of this interface allows the M20 to implement all configurations that come under the standard release by the Electronic Industries Association (EIA) witht he name RS-232-C.

The EIA RS-232-C standard interface for the connection of two units is shown below.

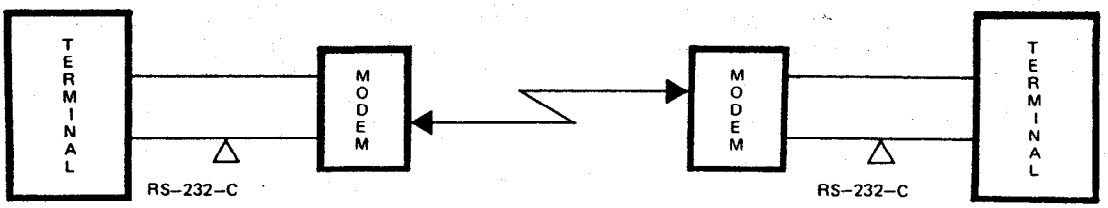


Figure 1-3 Schematic Diagram of the RS-232-C Interface

The connection between terminal and modem is standardized. The side of the interface that ends at the terminal is defined as the DTE (Data Terminal Equipement) side; the side that ends at the modem is defined as the DCE (Data Circuit-terminating Equipment) side.

The M20 can be used on:

1. The DTE side of the RS-232-C, using the RS-232-C MODEM cable with which the M20 can be connected to an asynchronous modem or a serial peripheral used as the DCE side of the interface.

2. The DCE side of the RS-232-C, using the RS-232-C PERIPH cable with which the M20 can be connected to terminals used as the DTE side of the interface.

The Current Loop 20mA Interface

The M20 system, (with the RS-232-C Twin board) and the CURRENT LOOP cable, implements a CURRENT LOOP 20mA interface for data exchange with compatible peripherals.

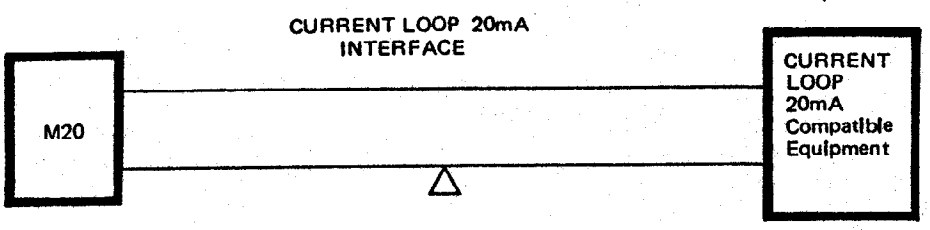


Figure 1-4 Schematic Diagram of the CURRENT LOOP Interface

IEEE 488 Standard

This interface defines the methods by which a "Controller" (usually a processor such as a computer) can control a "Talker" (usually a measurement instrument such as a voltmeter, etc.) and a "Listener" or "Listeners" (usually recording devices such as a printer and a digital tape recorder, etc.).

Figure 1-5 summarizes one possible system configuration in which the M20 can be used.

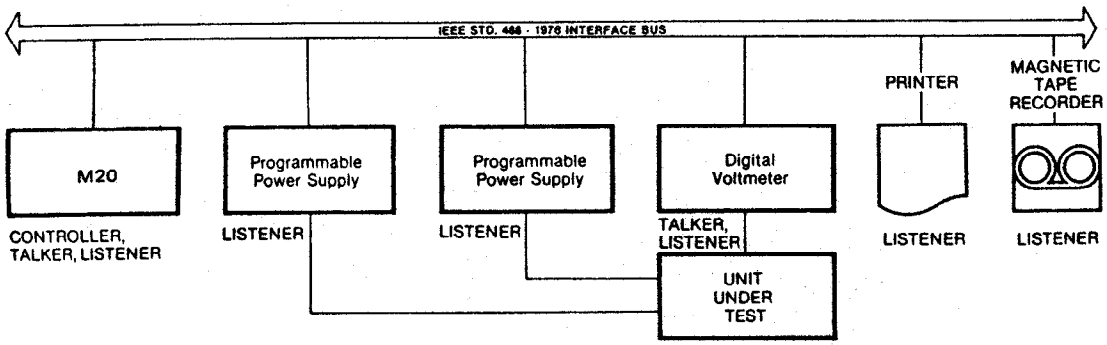


Figure 1-5 An IEEE Std. 488 Interface System with M20

Here is a list of types of devices (made by various manufacturers) compatible with the IEEE Standard 488 Interface System and therefore controllable by the M20:

- Stimulation

  . Microwave Frequency Synthesizer
  . Word Generator

GENERAL INTRODUCTION

- . Signal Generator
- . Timing Generator
- . Power Supply Programmer

- Measurement

- . Digital Voltmeter
- . Electronic Counter
- . Automatic Capacitance Bridge

- Display

- . Numeric Display
- . Plotter

PART I - RS-232-C SERIAL INTERFACE

2. THE RS-232-C INTERFACE

ABOUT THIS CHAPTER

This chapter describes the RS-232-C interface hardware in terms of its principal integrated circuits (ICs or chips), the pins and signals used on the connector as well as configuration. The interface software is described in terms of the input/output ports and status information that the user needs to program in order to obtain the necessary baud rate and the other transmission characteristics according to the requirements of the application. (The commands needed to do this are described in Chapter 3.)

CONTENTS

THE RS-232-C INTERFACE

THE INTERFACE HARDWARE

Inside the M20 the RS-232-C interface hardware makes use of two Intel 8259A Programmable Interrupt Controllers. The master is linked to the external Single, and the slave to the Dual RS-232-C edge connector and connector sockets shown in Figure 2-1.

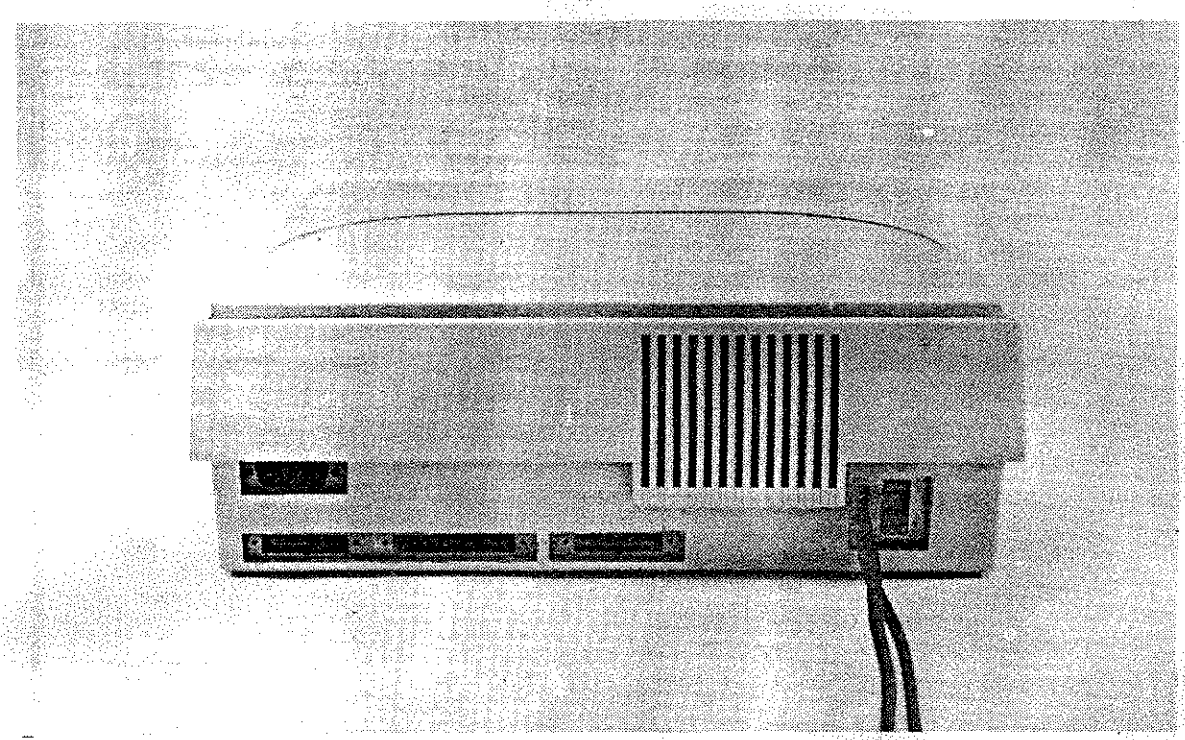


Figure 2-1 Rear View of M20 showing cable connectors

BUILT-IN RS-232-C INTERFACE

Mounted on the motherboard are the principal integrated circuits (ICs) forming the built-in Single RS-232-C Interface for communication port Ø. In corner A there is the Intel 8251A Universal Synchronous/ Asynchronous Receiver/Transmitter (USART). The 8251A is programmed by the use of the PCOS commands described in Chapter 3 to respond to the desired transmission characteristics (baud rate, start/stop bits, parity and data bits). In corner B is the Intel 8253 Programmable Interval Timer. The 8253 master in conjunction with the 8259A Programmable Interrupt Controller slave is used to "clock" the baud rate.

These characteristics must be programmed by the user via the RS-232-C interface, set communication and communication interface commands. See the section on Interface software and subsequent chapters.

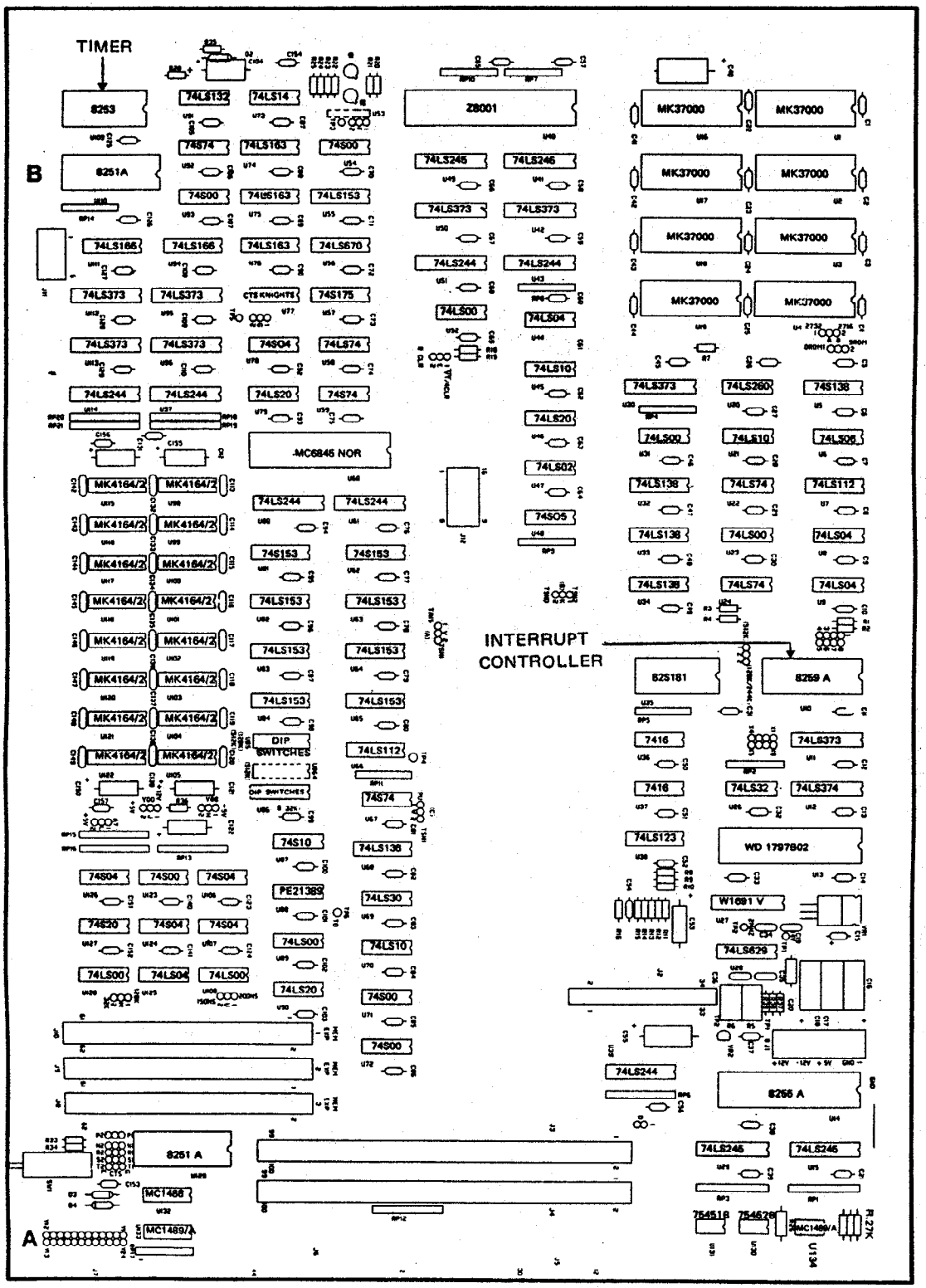


Figure 2-2 RS-232-C Interface Areas of Motherboard

OPTIONAL RS-232-C INTERFACE

Also mounted near corner A is the miniboard bearing the ICs forming the optional Dual RS-232-C Interface for communication ports 1 and 2. Each port has its own 8251A and shares one 8253 and slave 8259A PIC with function and programming as for the built-in Single RS-232-C Interface.

The hardware (via the corresponding connector cables) automatically isolates the M20 via optical-isolators from the 20mA Current Loop equipment when present.

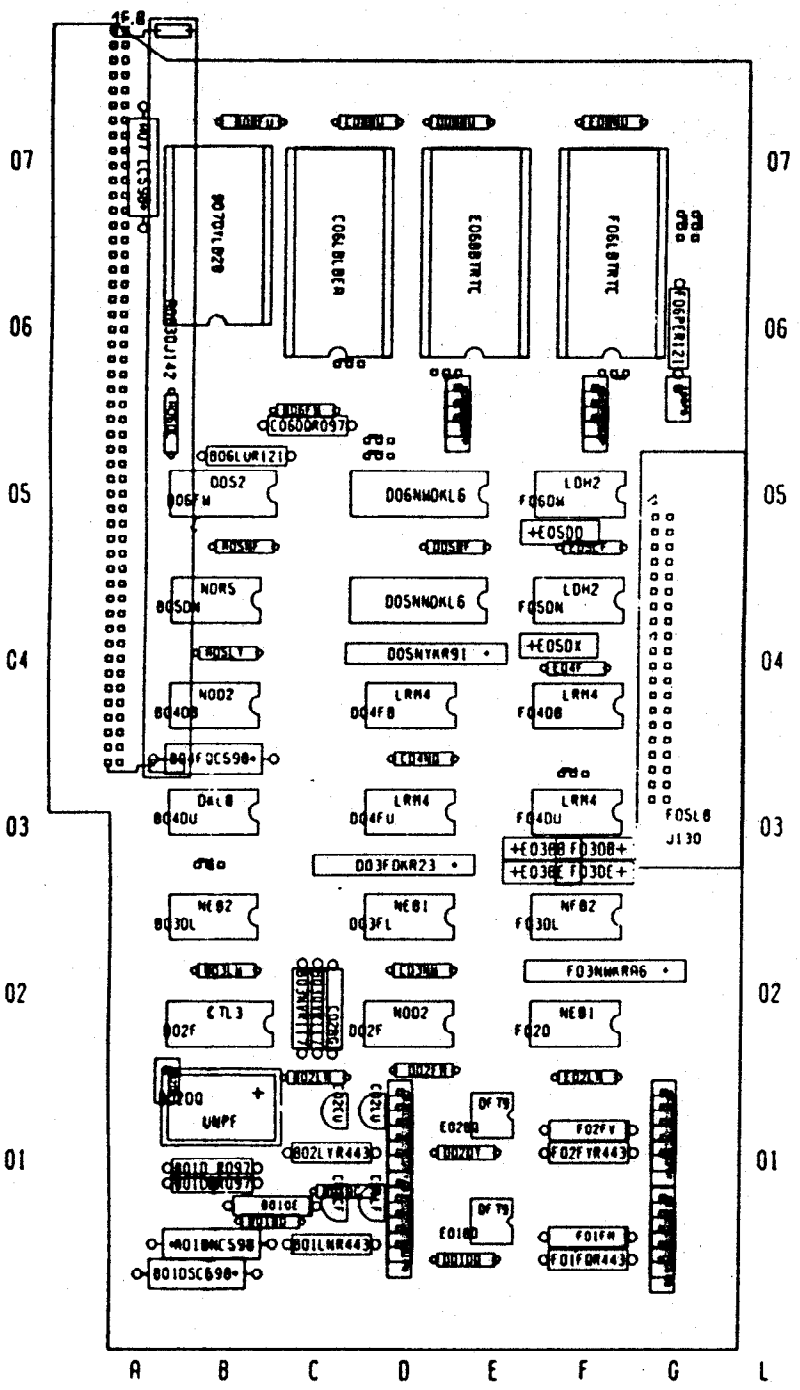


Figure 2-3 RS-232-C Dual Interface Mini Board

THE RS-232-C INTERFACE

M20 TO MODEM

The signals that the M20 transmits to and receives from a modem, on the same cable, conform to the EIA RS-232-C standard and to recommendation V.24 of the CCITT.

The interface signals and their functions are described in Figure 2-4 and are explained in the text that follows. The names of the signals conform to the EIA RS-232-C standard. The standard pin assignments apply to both single and dual RS-232-C Interface plugs as shown in Figure 2-4. See also Appendix B and the section on Configuration.

Signals are classified in three categories: ground, data, and control. In the explanatory text, references will be found to "Condition ON" and "Condition OFF". These relate to the following voltage levels:

Condition ON 3 volts
Condition OFF -3 volts

Ground Signals

Protective Ground: (Circuits: CCITT 101, EIA AA): is a conductor that is electrically bonded to the machine or equipment frame. It may be further connected to external grounds as required.

Signal Ground/Common Return: (Circuits: CCITT 102, EIA AB): is a conductor that establishes the signal common reference potential for the signals described in this chapter. Within the DCE, it is fixed at one point from which it can be connected to the PROTECTIVE GROUND by a metallic strap within the equipment. The strap can be removed to reduce electronic disturbances.

Data Signals

Transmitted Data: (Circuits: CCITT 103, EIA BA): is a binary signal that represents the data to be transmitted on-line from the M20 to the DCE.

Received Data: (Circuits: CCITT 104, EIA BB): is a binary signal transmitted as data from the DCE to the M20.

Control Signals

Request To Send: (Circuits: CCITT 1Ø5, EIA CA): signals on this circuit, sent from the M20, control the data channel transmit function. The ON condition causes the DCE to assume the data channel transmit mode. The OFF condition causes the DCE to assume the data channel non-transmit mode, when all data transferred on circuit 1Ø3 (TRANSMITTED DATA) has been transmitted.

Clear To Send: (Circuits: CCITT 1Ø6, EIA CB): signals on this circuit, sent from the DCE, indicate whether the DCE is conditioned to receive data on the data channel. The ON condition indicates that the DCE is conditioned to receive data on the data channel. The OFF condition indicates that the DCE is not prepared to receive data on the data channel.

Data Set Ready: (Circuits: CCITT 1Ø7, EIA CC): signals on this circuit, sent from the DCE, indicate whether the DCE is ready to operate. The ON condition indicates that the signal-conversion or similar equipment is connected to the line and the DCE is ready to exchange further control signals with the M20 to initiate the exchange of data. The OFF condition indicates that the DCE is not ready to operate. Note that this signal is not always significant and, in some cases, may arrive at the control unit inverted.

Data Terminal Ready: (Circuits: CCITT 1Ø8/2, EIA CD): signals on this circuit, sent by the M20, control switching of the signal-conversion or similar equipment to or from the line. The ON condition, indicating that the M20 is ready to operate, prepares the DCE to connect the signal conversion or similar equipment to the line and maintains this connection after it has been established by supplementary means. The M20 is permitted to present the ON condition on the circuit whenever it is ready to transmit or receive data. The OFF condition authorises the DCE to return to rest status.

Received Line Signal Detector: (Circuits: CCITT 1Ø9, EIA CF): signals on this circuit, sent by the DCE, indicate whether the received data channel line signal is within appropriate limits for data transfer. The ON condition indicates that the received signal is within appropriate limits. The OFF condition indicates that the received signal is not within appropriate limits.

Transmit Clock: this signal controls the rate at which the character will be transmitted.

Receive Clock: this signal controls the rate at which the character will be received.

Ring Indicator: (Circuits: CCITT 125, EIA CE): signals on this circuit, sent by the DCE, indicate whether a calling indicator is being received by the DCE. The ON condition indicates that a calling signal is being received. The OFF condition indicates that no calling signal is being received.

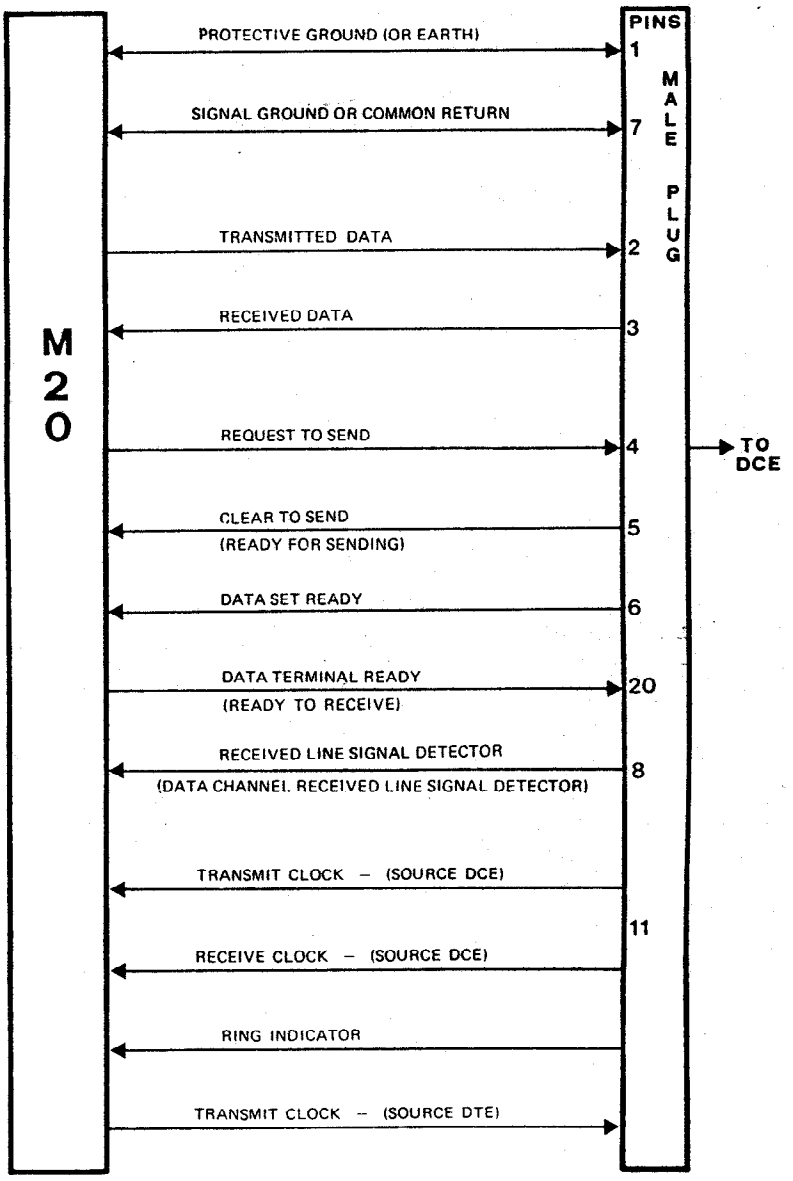


Figure 2-4 Signals Transmitted on the RS-232-C MODEM Cable

M20 TO PERIPHERAL

The signals that the M20 transmits to and receives from a serial peripheral conform to the EIA RS-232-C standard.

The interface signals and their functions are described in Figure 2-5 and are explained in the text that follows. The names of the signals conform to the EIA RS-232-C standard. These pin assignments apply to both Single and Dual RS-232-C Interface plugs as shown in Figure 2-5. See also Appendix B and the section on Configuration. As the electronic interpretation of each signal may vary according to the equipment connected to the M20, it is suggested that the manufacturer's equipment handbook be consulted for additional explanation of the signals.

Signals are classified in three categories: ground, data, and control. In the explanatory text, references will be found to "condition ON" and "condition OFF". These relate to the following voltage levels:

Condition ON 3 volts
Condition OFF -3 volts

Ground Signals

Protective Ground: (Circuits: CCITT 101, EIA AA): is a conductor that is electrically bonded to the machine or equipment frame. It may be further connected to external grounds as required.

Signal Ground/Common Return: (Circuits: CCITT 102, EIA AB): is a conductor that establishes the signal common reference potential for the signals described in this chapter. Within the peripheral unit, it is fixed at one point from which it can be connected to the PROTECTIVE GROUND by a metallic strap within the unit. The strap can be removed to reduce electronic disturbances.

Data Signals

Transmitted Data: (Circuits: CCITT 103, EIA BA): is a binary signal that represents the data to be transmitted on-line from the peripheral unit to the M20.

Received Data: (Circuits: CCITT 1Ø4, EIA BB): is a binary signal transmitted as data from the M20 to the peripheral unit.

Control Signals

Request To Send: (Circuits: CCITT 1Ø5, EIA CA): signals on this circuit, sent from the peripheral unit, control the data channel transmit function. The ON condition notifies the M20 that the peripheral is ready to transmit data. The OFF condition indicates that the peripheral has no data to send. (This signal is not always significant. Its use depends on the peripheral.)

Clear To Send: (Circuits: CCITT 1Ø6, EIA CB): signals on this circuit, sent from the M20, indicate whether the M20 is conditioned to transmit data on the data channel. The ON condition indicates that the M20 is conditioned to transmit data on the data channel. The OFF condition indicates that the M20 is not prepared to transmit data on the data channel.

Data Set Ready: (Circuits: CCITT 1Ø7, EIA CC): signals on this circuit, sent from the M20, indicate whether the M20 is ready to operate. The ON condition indicates that the M20 is ready. The OFF condition indicates that the M20 is not ready.

Received Line Signal Detector: (Circuits: CCITT 1Ø9, EIA CF): signals on this circuit, sent from the M20, indicate whether the M20 is ready to transmit data to the peripheral unit. The ON condition indicates that the M20 is ready. The OFF condition indicates that the M20 is not ready.

Data Terminal Ready: (Circuits: CCITT 1Ø8/2, EIA CD): signals on the circuit, sent from the peripheral unit, indicate whether the peripheral unit is ready to exchange information with the M20. The ON condition indicates the unit is ready. The OFF condition indicates the unit is not ready. (This signal is not always significant. Its use depends on the peripheral.)

Reverse Channel: (non-standard circuit): the significance of this signal, which is sent from a peripheral unit, depends on the peripheral connected to the M20. It is sometimes used to transmit a "busy" or an "anomaly" status to the M20.

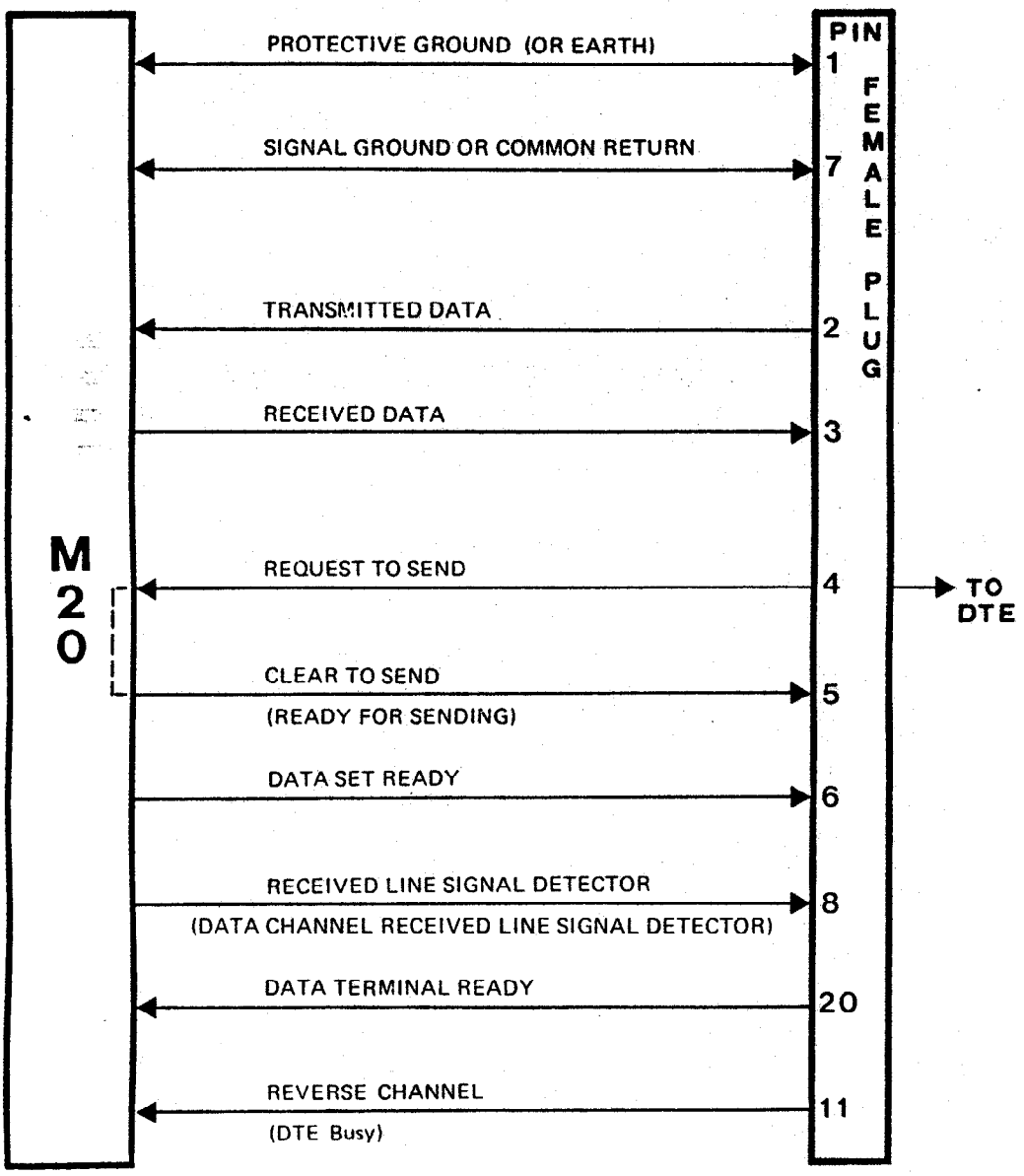


Figure 2-5 Signals Transmitted on the RS-232-C PERIPHERAL Cable

M20 TO M20

As shown in previous sections, the RS-232-C can support an M20 on either side of the interface: on the DTE side, with an RS 232 MODEM cable; on the DCE side, with an RS-232 PERIPH cable. The text that follows describes the connection of an M20 on the DTE side with an M20 on the DCE side. The connection is schematically shown in Figure 2-6.

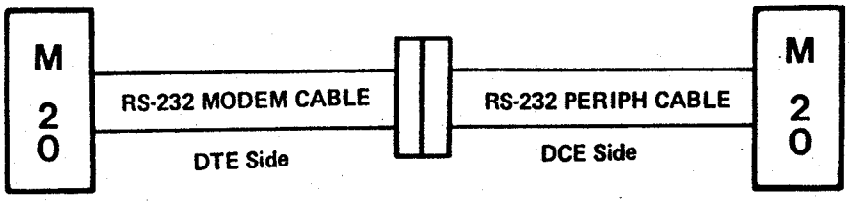


Figure 2-6 M20 to M20 Connection

The interface signals used are described in Figure 2-7 and are explained in the previous sections. The names of the signals conform to the EIA RS-232-C standard. The standard pin assignments apply to both Single and Dual RS-232-C Interface plugs as shown in Figure 2-7. See also Appendix B and section on Configuration.

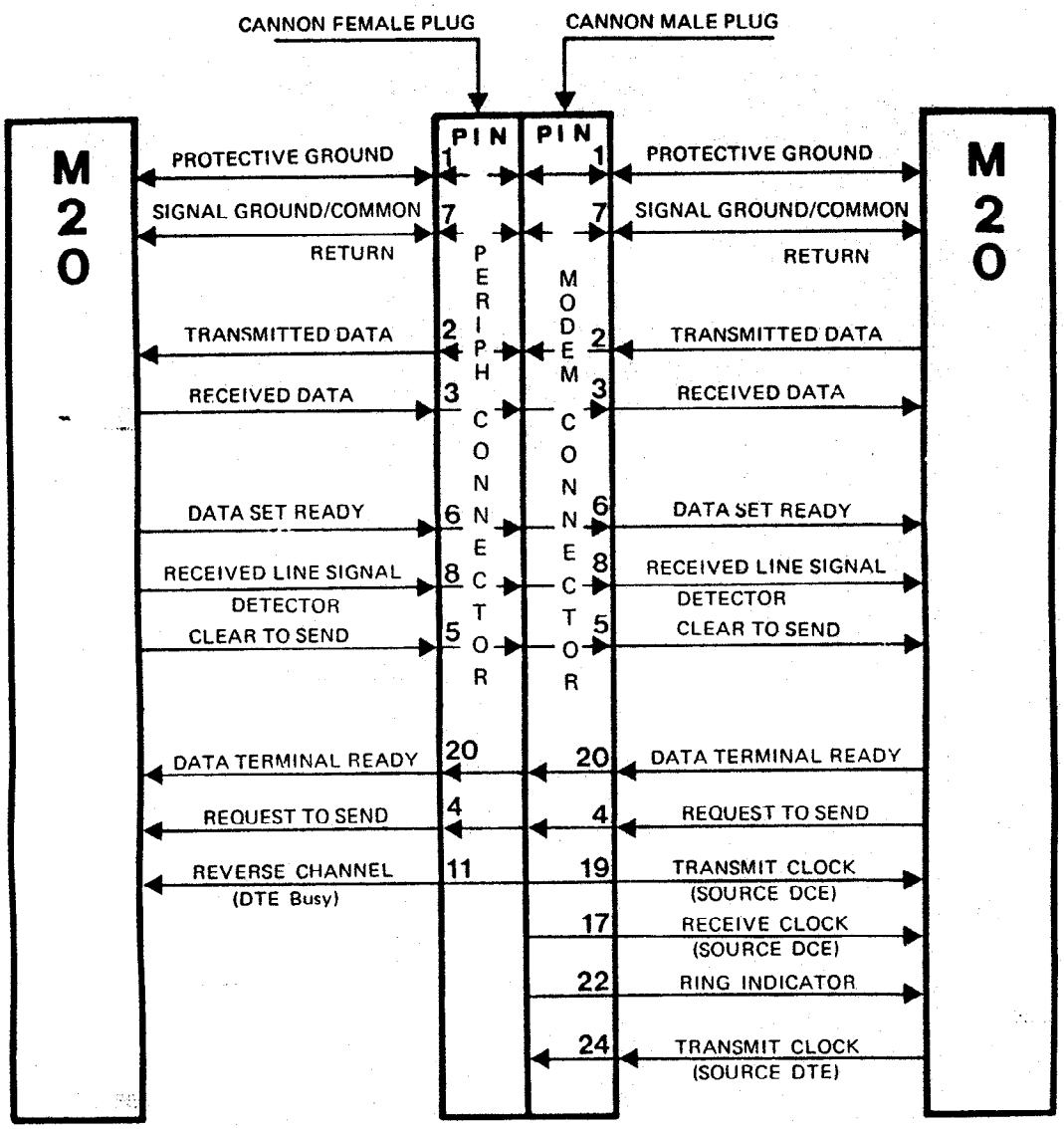


Figure 2-7 Signals Exchanged Between Two M20s

CURRENT LOOP

As shown in Figure 2-8, a Current Loop cable contains four wires without a connector, two assigned to the transmitter and two to the receiver. The wires must be connected as shown; the significance of the individual signals depends on the type of peripheral unit connected.

Note

All manufacturers use the coloured wires and terms specified for Current Loop 20 mA compatible equipment. The current source can be supplied by the M20 ("active" usage) or by the connected equipment ("passive" usage). Refer to manufacturer's publications for full details.

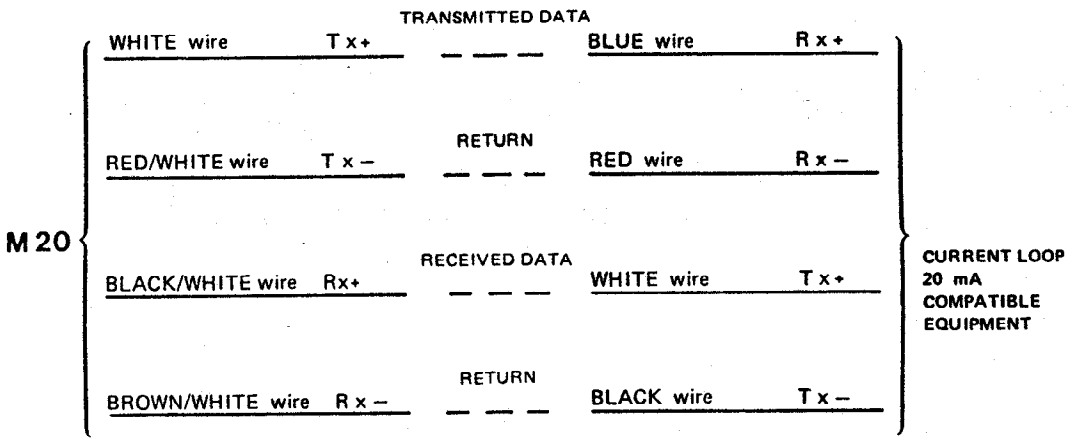


Figure 2-8 Signals Transmitted on Wires of a Current Loop Cable

The Dual RS-232-C Interface can be connected to one or two Current Loop compatible peripherals. Such circuits may not be in parallel. See the section on Configuration.

M20 As Receiver

The usual connection between the M20 as receiver and the compatible equipment as transmitter is the serial circuit of Figure 2-9.

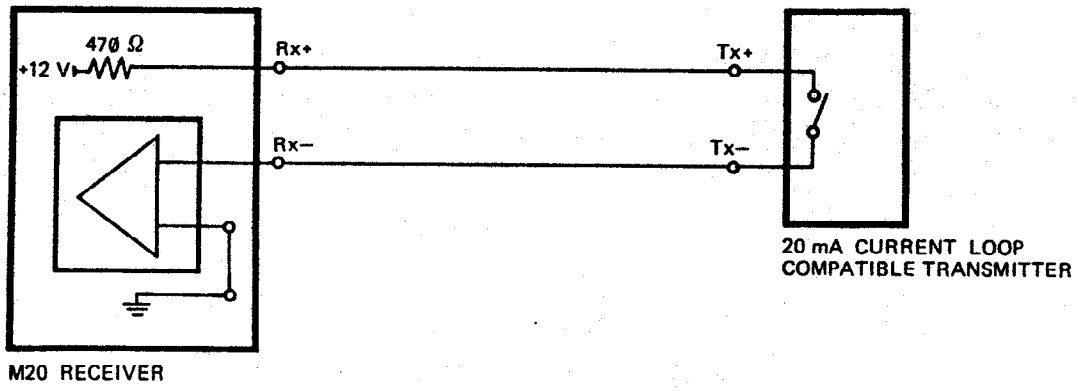


Figure 2-9 Serial Connection With M20 as Current Loop Receiver

M20 As Transmitter

The usual connection between the M20 as transmitter and the compatible equipment as receiver is the serial circuit of Figure 2-1Ø.

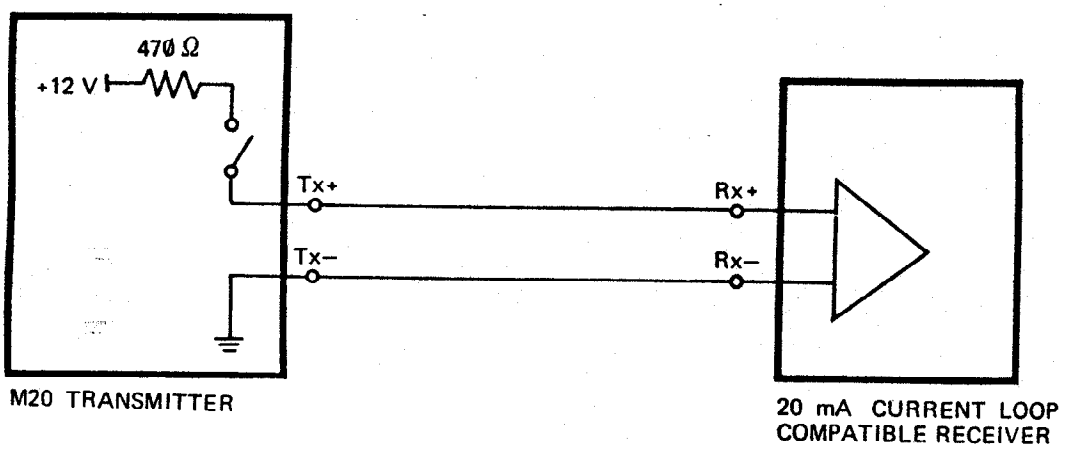


Figure 2-1Ø Serial Connection with M20 as Current Loop Transmitter

MOUNTING A CONNECTOR

Olivetti supplies an RS-232-C conversion cable to link the connector at the back of the M20:

- for the Single RS-232-C interface to an RS-232-C compatible connector:

  . a male Cannon plug for Modem connection, or

. a female Cannon plug for Peripheral connection;

- for the Dual RS-232-C interface to:

. a pair of:

* female Cannon plugs for connection to 2 Peripherals, or

* Current Loop cables for connection to 2 Peripherals;

or

. one female Cannon plug and one Current Loop cable for connection to 2 Peripherals, or

. one male Cannon plug for connection to a Modem and one Universal cable (to be wired by the user) for connection to:

* a female Cannon plug for a Peripheral, or

* a male Cannon plug for a Modem, or

* a Current Loop peripheral.

CONFIGURATION

Each RS-232-C Interface port can be connected to only one RS-232-C compatible device at a time. But each port can be used independently as can the Single and Dual RS-232-C Interfaces.

The valid combinations of connections on the three ports are as shown in Table 2-1, where capital letters indicate pre-wired cables and small letters user wired connections (M and m for Modem, P and p for Peripheral and c for Current Loop).

SINGLE RS-232-C INTERFACE	DUAL RS-232-C INTERFACE	
PORT Ø	PORT 1	PORT 2
M	M	m p c
	P	p c
	c	c
P	M	m p c
	P	p c
	c	c

Table 2-1 M20 RS-232-C Interface Port Connections

THE INTERFACE SOFTWARE

The interface software has to program the two RS-232-C ICs for each port. This is achieved by:

- the user loading the RS-232-C driver with the PCOS RS232 command;
- setting the transmission characteristics of individual ports with the PCOS SCOMM (set communication port) command, where parameter values are selected by the user or the system defaults are used;
- the user programming the processing at individual ports with a series of BASIC "ci" calls to the communications interface;
- and possibly, the user redefining the names of any port with the PCOS SDEVICE (set device) command and redirecting input/output with the PCOS '+' and '-' redirection extension.

RS-232-C INTERFACE DRIVER

The RS-232-C Interface driver is a general purpose asynchronous communication utility, which is invoked by use of the PCOS RS232 command. Via the PCOS SCOMM command, the user may specify the baud rate, parity, stop bits, and data bits for the communication line. In addition, the SCOMM command can specify the standard XON/XOFF handshake, a variable-length input buffer, and either character echoing or no character echoing at input.

The RS-232-C Interface driver makes use of the PCOS byte stream interface.

The receive mechanism is interrupt driven and maintains an input ring buffer. The output routine is not interrupt driven.

DC1: set on-line when the M20 is ready to communicate.

SCOMM Defaults

The default values for a serial port are: baud rate = 96ØØ, parity = NONE, stop bits = 2, data bits = 8, duplex = HALF, handshake = ON, buffer size = 128.

Handshake

Handshaking can be enabled or disabled by using SCOMM. When the handshaking is enabled, the RS-232-C Interface driver implements the standard XON/XOFF serial handshake. When the input buffer is three-quarters full the receiving routine will send an XOFF character (DC3 = 13 hexadecimal) to the transmitting device (setting bit 5 of table 2-2 to 1). When the buffer becomes less than half full an XON character (DC1 = 11 hexadecimal) is sent to the sending device (and bit 5 is cleared to Ø).

When handshaking is enabled, the receive interrupt routine scans incoming characters for either the XON or XOFF character, and will set or reset a flag to indicate the receiving bit 3 of the handshake status (see bit 3 of Table 2-2). The transmitting routine will look at this flag prior to transmitting a character and will wait until the XON character has been received before sending the character.

Device Parameter Table

A table of values called the Device Parameter Table (DPT) is used by the driver to control the RS-232-C Interface (serial I/O) ports. This table contains the port status word, all of the I/O port addresses and device commands, and the receive buffer control parameters. The SCOMM command will use the DPT to set the port parameters.

Handling Input Errors

Errors that occur when a character is input (a ring buffer overflow, or a hardware parity, overflow, or framing error) will cause flags to be set in the driver (see Table 2-2). The first operation that performs a read from the input buffer will return the error code for Disk I/O Error (PCOS error code 57) and will clear the error bits. (The error bits can be read and not cleared with the "ci" Status Read command, or can be cleared selectively with the "ci" Status Write command.)

This will allow any program running to know that an error has occurred, but cannot identify which is the erroneous character as the input buffer neither stores nor identifies the status for each character.

COMMUNICATIONS INTERFACE

The Communications Interface is the interface from the BASIC interpreter to the RS-232-C Interface ports. A relocatable utility called "ci.sav" is provided as a part of PCOS, which can be called from BASIC.

If the RS-232-C Interface driver "rs232.sav" has not been loaded prior to the execution of "ci.sav", then a PCOS error will be returned in the Error variable of the PCOS "ci" command.

The general format for the calls is:

call "ci" (Port number, "I/O Command",@ Error, Parameter list)

where

- Port number identifies the specific serial I/O channel. Port numbers are defined as follows:

  . Ø = Main RS-232-C port (on Single RS-232-C Interface)

  . 1 = First expansion port

  . 2 = Second expansion port (on Dual RS-232-C Interface)

- I/O Command specifies which operation is to be performed. The following are valid operations:

  . "o" - Open the port

  . "c" - Close the port

  . "w" - Write to port

  . "r" - Read from port

  . "sr" - Status read from port

  . "sw" - Status write to port

- @ Error is a value returned in an integer variable (which must be preceded by an "at" (@) sign). A non-zero value indicates unsuccessful completion of the command.

- Parameter list, which is not required for Open or Close, includes any parameters passed to, or returned from, the utility.

STATUS OF THE DRIVER

The status of the RS-232-C Driver can be ascertained from reading Driver Status word associated with the port identified in the PCOS Status Read function call (see Chapter 3, "STATUS READ (sr)" Command). The format is to be interpreted as shown in Table 2-2.

STATUS	BIT POSITION	LEGAL VALUE*	MEANING*
Duplex mode	15	1	full echoing of all input
		Ø	No echoing of input
(reserved)	14	Ø	(not used)
Framing error	13	1	a valid stop bit has not been detected at the end of each character. (Reported from 8251A - Table 2-3)
		Ø	No Framing Error

Table 2-2 Driver Hardware/Software Status Word - Driver State Flag (cont.)

STATUS	BIT POSITION	LEGAL VALUE*	MEANING*
Overrun Error	12	1	a character has not been read before the next one becomes available. (Reported from 8251A - Table 2-3)
		Ø	No Overrun Error
Parity Error	11	1	a change in parity value has been detected. (Reported from 8251A - Table 2-3)
		Ø	No Parity Error
Timeout Error	1Ø	1	a timeout has occurred while waiting for the Transmit Ready line on the 8251A
		Ø	No Timeout Error
Memory Error	9	1	driver failed to open buffer - no Open Port call or insufficient memory.
		Ø	No Memory Error
Buffer Error	8	1	interrupt routine tried to overwrite the buffer.
		Ø	No Buffer Error
(reserved)	7	Ø	(not used)
Free-running protocol	6	1	free-running protocol,
		Ø	Handshake protocol using XON/XOFF
XOFF/XON Flag (N20 previously acted as transmitter)	5	1	XOFF character, sent in previous transmission
		Ø	XON character sent in previous transmission

Table 2-2 Driver Hardware/Software Status Word - Driver State Flag (cont.)

STATE	BIT POSITION	LEGAL VALUE*	MEANING*
			state 1 - Buffer is 75% full. XOFF is sent from M20 i.e. other sender should stop.
			state Ø - Input buffer is ready to receive characters (default state). XON is sent from M20 i.e. other sender should start again.
Hardware state	4	1	hardware present and 8259A passed interrupt mask test.
		Ø	No hardware or failed test
XOFF/XON Flag (M20 acts as receiver)	3	1	XOFF character, } detected in current reception
		Ø	XON character }
			state 1 - XOFF character is received from outside. No characters will be transmitted.
			state Ø - XON received from outside. Characters will be transmitted (default state).

Table 2-2 Driver Hardware/Software Status Word - Driver State Flag (cont.)

STATE	BIT POSITION	LEGAL VALUE*	MEANING*
(reserved)	2	Ø	(not used)
(reserved)	1	Ø	(not used)
(reserved)	Ø	Ø	(not used)

* with respect to M20 RS-232-C interface

Table 2-2 Driver Hardware/Software Status Word - Driver State Flag

Correspondingly the Driver Status word can be changed by using the ci "sw" status write command.

STATUS OF THE PERIPHERAL

The status of the 8251A for each port can be ascertained reading the status byte of the port address identified in the PCOS Status Read function call. In this case the format is to be interpreted as shown in Table 2-3.

STATUS	BIT POSITION	LEGAL VALUE*	MEANING*
Data Set Ready	7	1 Ø	data Set Ready signal ON OFF
SYN detected	6	Ø	asynchronous transmission
Framing Error	5	1 Ø 	a valid stop bit has not been detected at the end of each character. No Framing Error. This bit is reset by Error Reset bit in Table 2-4
Overrun Error	4	1 Ø 	a character has not been read before the next one becomes available. No Overrun Error. This bit is reset by Error Reset bit in Table 2-4

Table 2-3 Peripheral Hardware Status - 8251A Status Byte (cont.)

STATE	BIT POSITION	LEGAL VALUE*	MEANING
Parity Error	3	1	a change in parity value has been detected.
		Ø	No Parity Error.
			This bit is reset by Error Reset bit in Table 2-4
Transmitter empty	2	1	the 8251A has no characters to send.
		Ø	Transmitter is not empty
Receiver ready	1	1	a character received by 8251A is ready to be input to M20.
		Ø	Receiver not ready
Transmitter ready	Ø	1	the 8251A is ready to receive a character for transmission.
		Ø	Transmitter not ready

* with respect to M20 RS-232-C interface

Table 2-3 Peripheral Hardware Status - 8251A Status Byte

Correspondingly this status can be changed by writing the command byte on the port address using the ci "sw" command, which sends the hardware command directly to the 8251A. See Table 2-4.

FUNCTION	BIT POSITION	LEGAL VALUE*	MEANING*
Enter Hunt mode	7	Ø	asynchronous transmission
Internal Reset	6	Ø	no internal reset
Request to Send	5	1	activate Request To Send line
Error reset	4	1	reset parity, overrun and framing errors

Table 2-4 Peripheral Hardware Status - 8251A Command Byte (cont.)

FUNCTION	BIT POSITION	LEGAL VALUE*	MEANING*
Send Break character	3	Ø	no break character
Receive enable	2	1	enable Hardware Receive state
Data Terminal Ready	1	1	activate Data Terminal Ready line
Transmit enable	Ø	1	enable Hardware Transmit state

* with respect to M20 RS-232-C Interface

Table 2-4 Peripheral Hardware Status - 8251A Command Byte

PROGRAMMING THE INTERFACE

The RS-232-C Interface is programmed for each port. The user:

- loads the RS-232-C driver using the PCOS command "rs232";
- sets the transmission characteristics of individual ports with the PCOS SCOMM (set communication port) command, where parameter values are selected by the user or the system defaults are used;
- programs the processing at individual ports with a series of PCOS "ci" commands (a set of BASIC calls to the communications interface);
- possibly, redefines the names of any port with the PCOS SDEVICE (set device) command and redirecting input/output with the PCOS '+' and '-' redirection extension.

All these commands are described in Chapter 3.

3. RS-232-C COMMANDS

ABOUT THIS CHAPTER

This chapter describes all those PCOS commands directly usable with the RS-232-C Interface. These are RS232 to invoke the RS-232-C Driver, SCOMM to specify the port data characteristics, SDEVICE to define the port device names, the "ci" calls from BASIC to program the functions needed on each port, and the "+" or "-" command indicators for redirection of input or output.

CONTENTS

RS-232-C COMMANDS

INTRODUCTION

This chapter deals with the PCOS command CI and other PCOS features specifically related to the use of the RS-232-C interface. The examples contain M20 BASIC statements and may cite other PCOS commands, of which a working knowledge is assumed. The BASIC statements used are not described here but are fully illustrated in the "BASIC Language Reference Guide". More details on PCOS and its commands are to be found in the "Professional Computer Operating System (PCOS) User Guide".

SCOMM

Sets various environment parameters for an RS-232-C communications port.

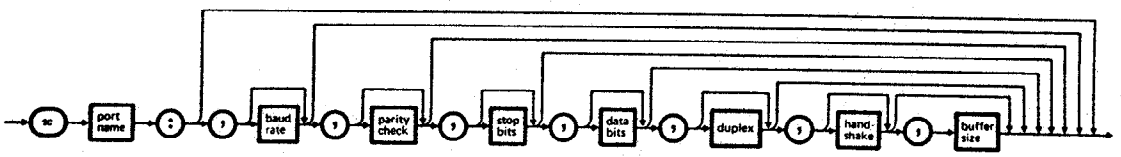


Figure 3-1 SCOMM

Where

SYNTAX ELEMENT	MEANING
port name	the string line name specified by the SDEVICE command (or a default name, such as COM or COM2).
baud rate	an integer that may be either 5Ø, 11Ø, 3ØØ, 6ØØ, 12ØØ, 18ØØ, 24ØØ, 48ØØ, 96ØØ.
parity check	a string that may be either ODD, EVEN, or NONE.
stop bits	an integer that may be either Ø,1, or 2. (Ø = 1 stop bit; 1 = 1.5 stop bits; 2 = 2 stop bits)

SYNTAX ELEMENT	MEANING
data bits	an integer that may be either 5, 6, 7, or 8.
duplex	a string that is either HALF or FULL. When FULL characters are echoed as they are received. When HALF no echoing occurs.
handshake	a string that is either ON or OFF. The ON indicates that the driver will carry on the standard XON/XOFF handshake. The OFF indicates that no handshake should be done.
buffer size	the integer that indicates the size of the input buffer which is allocated on the heap.

Characteristics

In particular, the user may specify the baud rate, parity check, number of stop bits, number of data bits, duplex, handshake, and size of input buffer.

If RS232, the communications driver RS232.SAV, has not been loaded prior to execution of SCOMM, a device not found error will be returned.

If no parameters other than the port name are specified the current values for that port will be displayed. If a null parameter is specified that particular parameter will remain unchanged.

Example

IF you enter...	THEN...
SC COM:,,,2,,,, FULL CR	on port Ø (com:) the number of stop bits is changed to 2, characters are echoed as received and all other parameters are left as before

Remarks

The default values for a port are: baud = 96ØØ, parity = NONE, stop bits = 2, data bits = 8, duplex = HALF, handshake = ON, buffer size = 128.

Changes made using the SCOMM command can be made permanent by using the PCOS PSAVE command.

SDEVICE

Displays the device table and optionally renames a device.

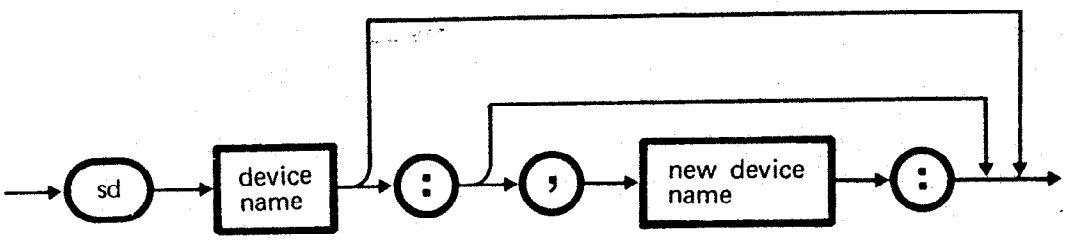


Figure 3-2 SDEVICE

Where

SYNTAX ELEMENT	MEANING
device name	the current name of any device (that exists in the device table) consisting of at most 13 printable characters followed by a colon. This can be either the name assigned in the last SDEVICE command (in the current working session) or, in the absence of a preceding SDEVICE command, the default device name, or, if a customized PCOS is being used, the PSAVEd device name.
new device name	the name to be assigned to the device in question. The name must be at most 13 characters long and must be followed by a colon (:)

Default device names

prt: = PCOS Printer Driver
cons: = PCOS Console Driver
(video and keyboard)

Com: = Standard RS-232-C
communication port

Com1: = First expansion RS-232-C
communication port

Com2: = Second expansion RS-232-C
communication port

Characteristics

Default usage of SDEVICE will display the device table for those devices available. Each device is described in terms of R(ead) or W(rite) to indicate capability for input, output or both.

RS232

Causes the RS-232-C Communications Interface device driver to be loaded.

Figure 3-3 RS232

Characteristics

This device driver is a general purpose asynchronous communication utility.

By using the SCOMM command the user can specify the baud rate, parity, stop bits, and data bits for the communication line. In addition, the standard XON/XOFF handshake, a variable-length input buffer, and (no) character echoing (DUPLEX) is supported.

The interface for the device driver is a PCOS byte-stream interface.

All driver parameters are specified in the Device Parameter Table so that accessing the parameters is simplified. The receive mechanism is interrupt driven and uses an input ring buffer, while the output routine is executed without using a buffer.

The default receive buffer size can be increased manually using a "ci" parameter. Such an increase does not alter the user memory space.

CI

Allows the user to program I/O operations on the Communications Interface, provided the RS-232-C Driver has already been invoked (see RS232).

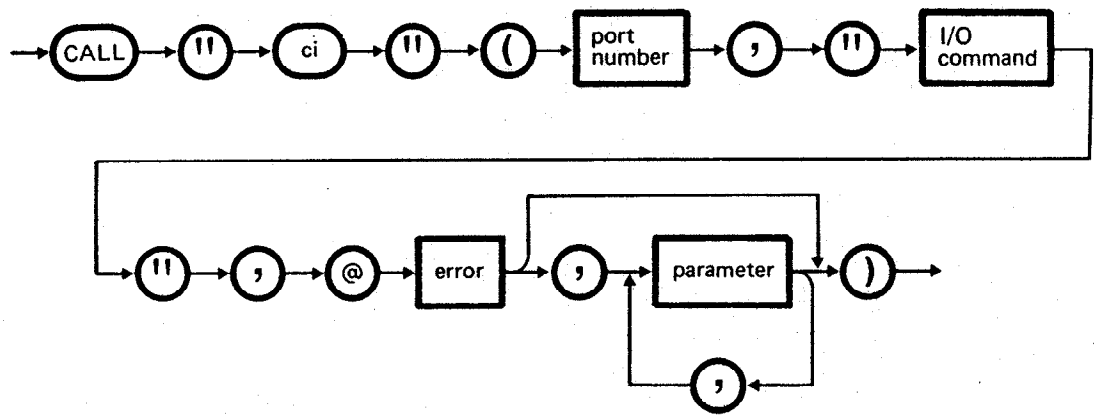


Figure 3-4 CI

Where

SYNTAX ELEMENT	MEANING
port number	any valid RS-232-C interface port number: Ø - built-in port 1 - first expansion port 2 - second expansion port

SYNTAX ELEMENT	MEANING
I/O command	any valid CI command mnemonic: o Open the port c Close the port r Read from the port w Write to the port sr Status Read from the port sw Status Write to the port
error	a valid integer variable name used to receive the returned error value: Meaning Ø correct functioning of the operation 1 invalid parameters 2 incompatible port status where higher values have meanings according to the function of the CI function call
parameter	a parameter chosen according to the function of the CI function call

Characteristics

Any CI command must be called from BASIC using CALL statements. The first such call will cause CI to be loaded and made memory resident.

If the Communications Driver has not been loaded prior to the execution of any CI command, the error integer variable will return the value of the PCOS error code corresponding to "command not found" (92).

The RS-232-C Communications Driver, being held in RS232.SAV, only needs to be loaded once by use of the PCOS RS232 command (or an EXEC statement) prior to using any CI command.

Example

DISPLAY	COMMENTS
LIST . . . . 1Ø PN% = Ø: E% = Ø 2Ø EXEC "rs" 3Ø CALL "ci" (PN%, "o", @ E%) . . .	Define initial values for port (built-in one) and error variables. Load RS232 Communications Driver Open port Ø (built-in on mother-board)

Remarks

The error is returned in an integer variable (which must be preceded by an "at" (@) sign). A zero return value always indicates no error has occurred in the command. PCOS error 92 will be returned in the error variable, if the driver command file, RS232.SAV, has not been executed beforehand.

The parameter list must include at least three parameters (port number, command, error). If one of these is missing, PCOS parameter error 9Ø is returned.

OPEN CALL "o"

Causes the RS-232-C Communications Interface input buffer space to be allocated and the hardware to be initialised.

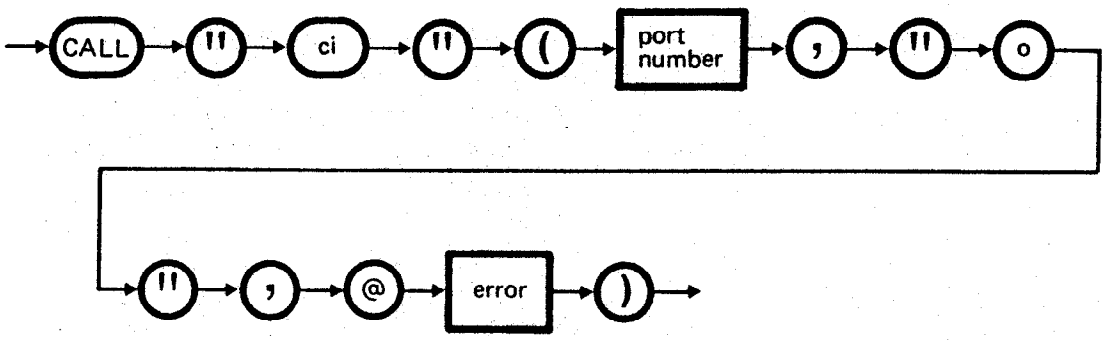


Figure 3-5 CI "o"

Where

SYNTAX ELEMENT	MEANING
port number	any valid RS-232-C interface port number: Ø - built-in port 1 - first expansion port 2 - second expansion port
o	the valid CI command mnemoric: o open the port
error	a valid integer variable name used to receive the returned error value: Meaning Ø correct functioning of the operation 1 invalid parameters 2 port already open 3 input buffer too big for heap 4 time out error on initial XON (only in handshake mode)

Example

DISPLAY	COMMENTS
LIST	
.	
.	
.	
.	
1Ø PN% = Ø: E% = Ø	Open port Ø (built-in on mother-
3Ø CALL "ci" (PN%,"o",@E%)	board)
.	
.	
.	

CLOSE CALL "c"

Causes the RS-232-C Communications Interface buffer space to be deallocated and the hardware interrupts to be disabled.

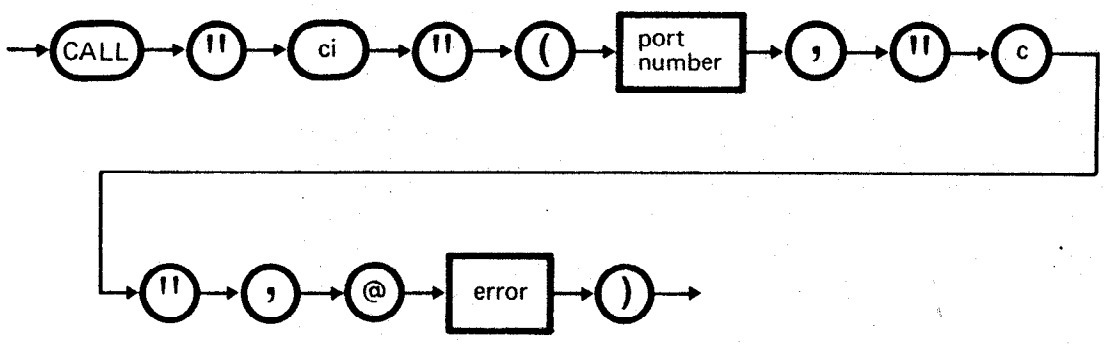


Figure 3-6 Cl "c"

Where

SYNTAX ELEMENT	MEANING
port number	any valid RS-232-C interface port number: Ø - built-in port 1 - first expansion port 2 - second expansion port
c	the valid CI command mnemonic: c close the port
error	a valid integer variable name used to receive the returned error value: Meaning Ø correct functioning of the operation 1 invalid parameters 2 port not open 3 input buffer was never allocated

Characteristics

Any CI command must be called from BASIC using CALL statements. The first such call will cause CI to be loaded and made memory resident.

If the Communications Driver has not been loaded prior to the execution of any CI command, the error integer variable will return the value of the PCOS error code corresponding to "command not found" (92).

The RS-232-C Communications Driver, being held in RS232.SAV, only needs to be executed once by use of the PCOS RS232 command (or an EXEC statement) prior to using any CI command.

Example

DISPLAY	COMMENTS
LIST	
.	
.	
.	
.	
1Ø PN% = Ø: E% = Ø	
.	
.	
.	
19Ø CALL "ci" (PN%, "c", @E%)	close port Ø (built-in on mother-board)
.	
.	
.	

READ CALL "r"

Causes the string variable specified to be filled with the data received from the serial port on the RS-232-C Communications Interface.

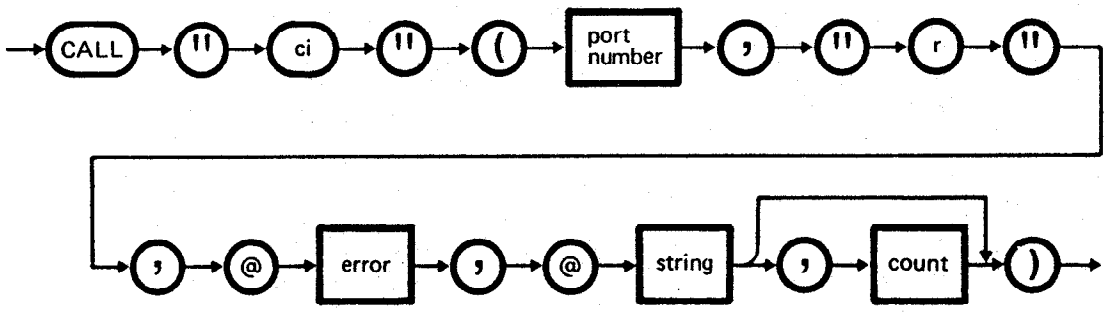


Figure 3-7 CI "r"

Where

SYNTAX ELEMENT	MEANING
port number	any valid RS-232-C interface port number: Ø - built-in port 1 - first expansion port 2 - second expansion port
r	the valid CI command mnemonic: r Read from the port
error	a valid integer variable name used to receive the returned error value: Meaning Ø correct functioning of the operation 1 invalid parameters 2 port not open 3 time-out error on XON handshake 4 character (parity, overflow, framing) error or buffer overflow error
string	a valid name for the string variable to receive data read from the port
count	a valid integer variable name for the optional maximum number of characters to be read in from the port

Characteristics

If the optional count is not specified, either the string is filled with the length of the string, or, if a separator character is encountered in the input stream before the string is full, the string is filled up to, but not including, the separator character. (The separator character is specified with the Status Write "sw" command. The default separator character is a carriage return.)

An optional count can be specified which specifies the number of characters to be read from the port. This causes the string to be filled either up to its length or up to the count, whichever is least. (Note: the separator character will not terminate the command in this case.)

In either case, the length of the string is set equal to the number of characters received. Also, in either case, the command will wait to receive characters until the termination condition is satisfied. (Note: in order to detect the presence of received characters and to avoid the wait, the Status Read CALL "sr" may be used to determine the number of characters received in the input buffer.)

Example

DISPLAY	COMMENTS
LIST	
.	
.	
.	
1Ø PN% = Ø: E% = Ø	statements to use default port characteristics of 96ØØ baud; 7-bit characters with no parity, 2 stop bits and no echo; etc. (see SCOMM).
15 C$ = SPACE $(1)	Define initial values for port (built-in one), error and string variables. Line 15 defines a space C$ which will be filled by received data.
.	
.	
.	
12Ø CALL "ci" (PN%,"r",@E%,@C$,1)	read one character (string length = 1)
.	
.	
.	

Remarks

The data read in is returned in a string variable (which must be preceded by an "at" (@) sign). Space for the string variable must be allocated before the call is made.

A PCOS parameter error will be returned, if a Read CALL is used without the string variable.

STATUS READ CALL "sr"

Allows the BASIC program to read the current status of the hardware and the RS-232-C Communications Interface device driver.

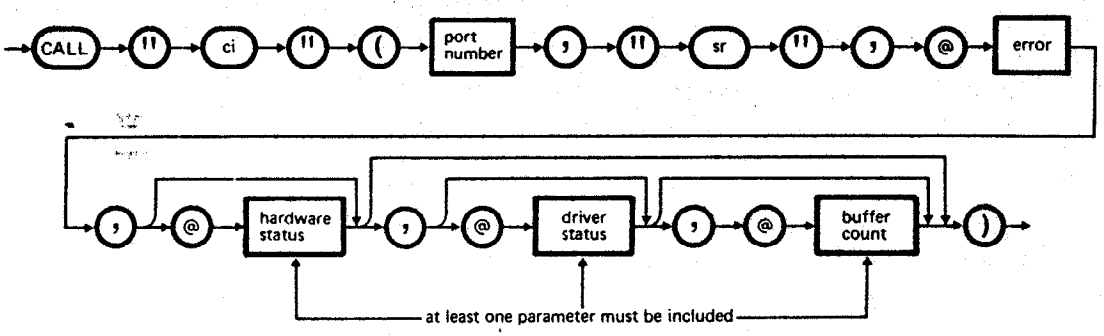


Figure 3-8 CI "sr"

Where

SYNTAX ELEMENT	MEANING
port number	any valid RS-232-C interface port number: Ø - built-in port 1 - first expansion port 2 - second expansion port
sr	the valid CI command mnemonic: sr Status Read from the port

SYNTAX ELEMENT	MEANING
error	a valid integer variable name used to receive the returned error value: Meaning Ø correct functioning of the operation 1 invalid parameters 2 port not open
hardware status	a valid integer variable name used to receive the returned hardware status value. This is the value of the byte defined in Table 2-3
driver status	a valid integer variable name used to receive the returned driver status value. This is the value of the word defined in Table 2-2
buffer count	a valid integer variable name used to receive the returned buffer count. This is the count of the characters in the input buffer

Characteristics

Up to three integers are returned which indicate the hardware status, the RS-232-C device driver status and the current buffer count. Examining the buffer count before a Read CALL "r", permits the BASIC program to determine whether a character has been received before actually committing to reading a character.

The driver status word accessed via a Status Read CALL "sr" and the driver control word (accessed via a Status Write CALL "sw", below) are identical.

The hardware status parameter is defined in Table 2-3 and corresponds to the 8251A Programmable Communication Interface (USART) STATUS byte.

The RS-232-C driver status parameter is defined in Table 2-2 and corresponds to the RS232.SAV driver internal STATE FLAG word.

For example, for the RS232.SAV driver:

hex ØØ1Ø — default internal driver state after Open Command:

Half Duplex, Handshake Enabled, XON, no errors;
Buffer allocated and Hardware exists.

```
                DISPLAY                                COMMENTS

LIST
  .
  .
  .
1Ø PN% = Ø: E% = Ø
14 HS% = Ø:DS% = Ø:BC% = Ø
  .
  .
  .
1ØØ CALL "ci" (PN%,"sr",@E%,          Get status and buffer count
   @HS%,@DS%,@BC%)
  .
  .
  .
```

STATUS WRITE CALL "sw"

Allows the BASIC program to write control information directly to the hardware and the RS-232-C device driver.

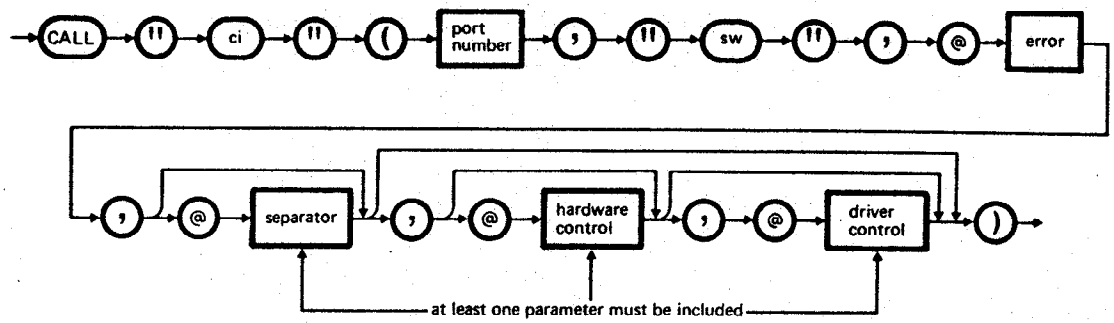


Figure 3-9 CI "sw"

Where

SYNTAX ELEMENT	MEANING
port number	any valid RS-232-C interface port number Ø - built-in port 1 - first expansion port 2 - second expansion port
sw	the valid CI command mnemonic: sw Status Write to the port
error	a valid integer variable name used to receive the returned error value: Meaning Ø correct functioning of the operation 1 invalid parameters 2 port not open

SYNTAX ELEMENT	MEANING
separator	a valid integer name for the character used to separate items read by Read command. The default character is a carriage return.
hardware control	a valid integer variable name used to define the hardware control byte. See Table 2-4
driver control	a valid integer variable name used to define the driver control word. See Table 2-2

The driver control word accessed via the Status Write CALL "sw" and the driver status word in Table 2-2 are identical.

Characteristics

The first parameter is an integer and specifies the separator character to be used in separating input for the Read CALL "r" command.

The driver control word accessed via the Status Write CALL "sw" and the driver status word in Table 2-2 are identical.

The hardware control parameter is a byte defined in Table 2-4 and corresponds to the 8251A Programmable Communication Interface (USART) COMMAND byte.

For example, for the RS232.SAV driver:

hex 37 — default command byte:
parity, overflow and framing error bit is reset; Request to Send and Data Terminal Ready signals are set to Ø, receive and transmit are enabled.

hex 15 — clear error command byte:
parity, overflow and framing error bit is reset; Request to Send and Data Terminal signals are unchanged; receive and transmit are enabled.

Example

```
                DISPLAY                                  COMMENTS

LIST
  .
  .
  .
1Ø PN% = Ø: E% = Ø
14 HS% = Ø: DS% = Ø: BC = Ø
  .
  .
  .
61Ø CALL "ci" (PN%, "sw",@E%,,HS)      set hardware status
  .
  .
  .
```

Remarks

In order to reconfigure the baudrate, parity, stop bits, data bits, duplex, handshake enable, or input buffer size of the RS-232-C driver, the PCOS set communications command SCOMM should be executed. If the hardware control byte is specified, the hardware status bits and the error bits of the driver control word may change.

WRITE CALL "w"

Causes the parameters specified in the list (a mix of integers and strings) to be transmitted from the specified port on the RS-232-C Communications Interface.

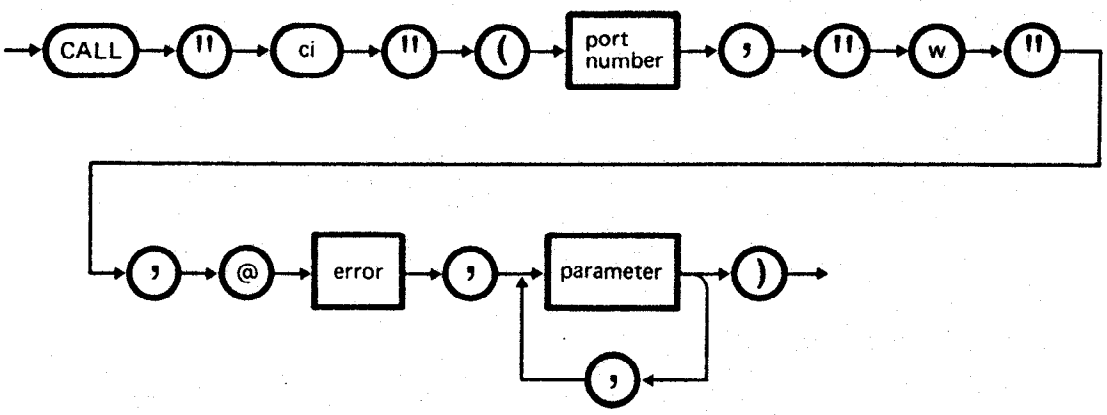


Figure 3-10 CI "W"

Where

SYNTAX ELEMENT	MEANING
port number	any valid RS-232-C interface port number: Ø - built-in port 1 -first expansion port 2 - second expansion port
w	the valid CI command mnemonic: w write to the port
error	a valid integer variable name used to receive the returned error value: Meaning Ø correct functioning of the operation 1 invalid parameters 2 port not open 3 time-out error on character transmission
parameter	an integer or a string of any length to be transmitted via the specified port.

Example

```
                DISPLAY                              COMMENTS

LIST
  .
  .
  .
  .
1Ø PN% = Ø: E% = Ø
  .
  .
  .
6Ø LINE INPUT "enter string",A$          receive string from keyboard
7Ø CALL "ci" (PN%, "w",@E%,A$,13)        write string (followed by carriage
  .                                      return) to the port
  .
```

Remarks

The parameter list includes at least one parameter which is to be passed via the CI command. A PCOS parameter error will be returned, if no parameters are used in the parameter list.

The parameter list may contain any number and mix of integers and strings. Integers are used to send control or binary information to the connected device. The RS-232-C driver sends only the least significant byte of the integer. Strings can be of any length and are sent as a string of bytes.

DEVICE RE-ROUTING

INTRODUCTION

Normal operation of the M20 system involves entering information via the keyboard and receiving responses on the video. In some cases however the user may wish to use additional input and/or output devices; for example a printer can be used to get a print of what is displayed on the video; this can be done on the M20 using "Device re-routing parameters".

Device re-routing can be specified on two levels: Local and Global. These two cases are dealt with in detail below.

LOCAL DEVICE RE-ROUTING

Re-routes input and output of the PCOS command being executed.

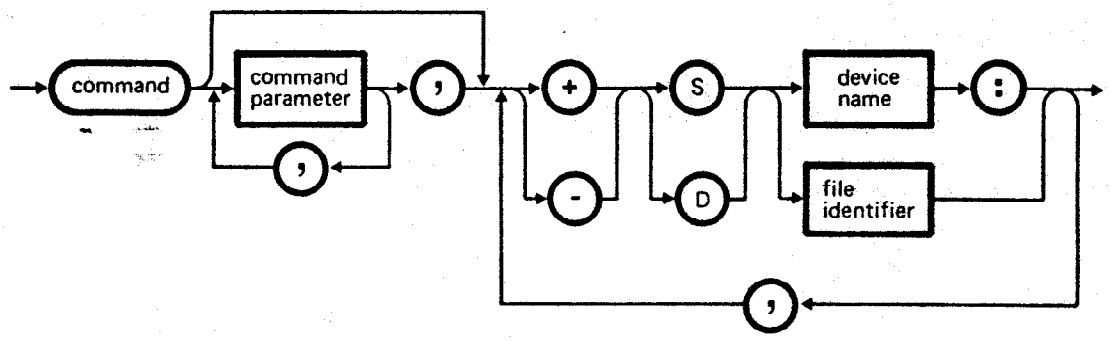


Figure 3-11 Local device re-routing

Where

SYNTAX ELEMENT	MEANING
command	any PCOS command to be executed via the devices specified in the device re-routing parameters
command parameter	a parameter to be passed to the PCOS command in question
+	the device or file specified is to be allocated
-	the device or file specified is to be cancelled
S	specifies the device to be a source (for input into the M20). It can be entered in upper or lower case.
D	specifies the device to be a destination (for output from the M20). It can be entered in upper or lower case.

SYNTAX ELEMENT	MEANING
device name	a string of up to 13 printable ASCII characters the first of which must not be a digit, denoting the device in question. This can be the device default name or any other name assigned to the device using the SDEVICE command. Note: A device name must be followed by a colon ":"
file identifier	a file identifier complying with the conditions described in the PCOS (Professional Computer Operating System) User Guide. If the file specified does not exist it will be created on either the specified disk or the disk inserted in the last selected drive.

Note: + or -, S or D and Device name or File name follow each other without any spaces in between. The command parameters may also follow the device re-routing parameter block.

Local device re-routing can only be implemented from BASIC via the EXEC statement. Furthermore, it can only be terminated either by returning control to PCOS or by explicitly deactivating it via an EXEC statement (see example).

For reasons of compatibility with previous software releases, the printer can be specified either by +DPRT: or simply +PRT (without a colon). See Figure 3-12.

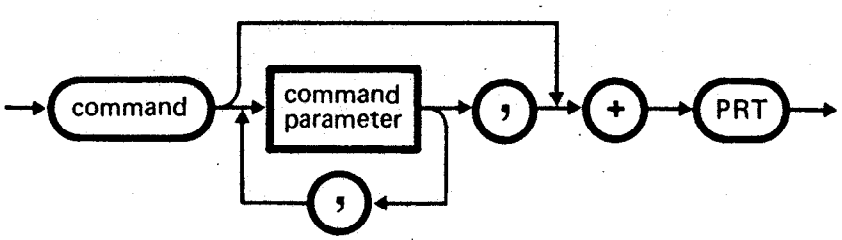


Figure 3-12 Re-routing local output to printer

Example

IF you enter...	THEN...
ba + STEXT CR	the M20 will go into the BASIC environment and take in commands from the TEXT file and keyboard; but the local effect to accept commands from TEXT file will cease when environment returns to PCOS.
ba CR EXEC "vl 1:,+d1:out" CR . . . EXEC "-d" CR	M20 enters the BASIC environment, then the first EXEC statement lists the files contained on the disk inserted in drive 1, but re-routes the output to the file "out". All subsequent output is routed to this file until the second EXEC statement cancels the re-routing.
ba CR EXEC "vl 1:,+dprt:" CR SYSTEM CR	M20 enters the BASIC environment, then the EXEC statement lists the files contained on the disk mounted in drive 1 and re-routes the output to the printer. +dprt: is then cancelled by returning to PCOS.

GLOBAL DEVICE RE-ROUTING

Re-routes input and output of all PCOS commands executed thereafter in operation until otherwise specified or until the system is reset.

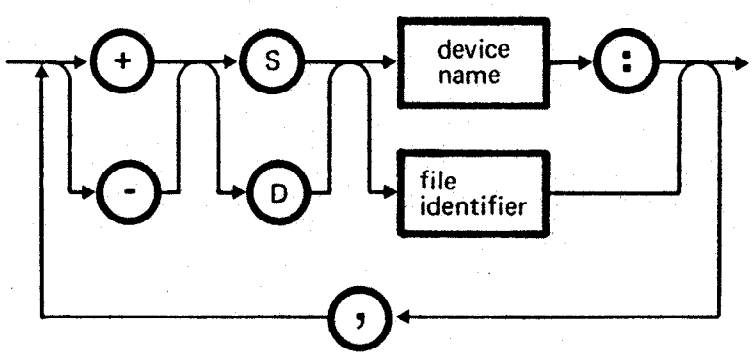


Figure 3-13 Global device re-routing

Global device re-routing will occur if the parameters described in the syntax diagram for local device re-routing (see Figure 3-11) are specified in the absence of a command and command parameters.

Example

IF you enter...	THEN...
+SCOM:,+dcom1: CR	the M20 will accept input from both the keyboard and the standard built-in RS-232-C communication port; output will be displayed on the video and directed to the extension RS-232-C communication port 1

SIMULTANEOUSLY ACTIVE DEVICES

The keyboard is not automatically disabled when other input devices are specified. In general, all specified input and output devices will be simultaneously active. This requires the user to exercise a degree of caution when re-routing input, in order to avoid data from several devices becoming intertwined.

The local keyboard can be disabled by specifying "-SCONS:". However if this is done on a Global level, control cannot be regained unless an external active device issues a "+SCONS:" or the system is reset.

NO INTERACTION FLAG

If the flag is set, the display of any parameters and/or related error messages is suppressed.

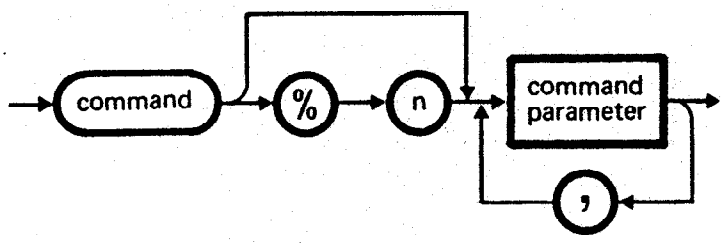


Figure 3-14 No interaction flag

Where

SYNTAX ELEMENT	MEANING
command	any PCOS command to be executed
%	string flag indicator
n	suppresses the display of the parameters and/or related error messages. It can be entered in upper or lower case
command parameter	a parameter to be passed to the PCOS command in question

Example

IF you enter...	THEN...
vf %N 1: CR	the M20 will format the disk in drive 1 without giving any indication of any user intervention required, the track being formatted or any error condition that may arise.

4. RS-232-C SAMPLE PROGRAMS

ABOUT THIS CHAPTER

This chapter illustrates with the use of example programs, two typical uses for this interface. These are the transfer of data out of the M20 via the RS-232-C interface and receipt of data into the M20 (with and without status checking). There is one sample program for each corresponding example.

CONTENTS

RS-232-C SAMPLE PROGRAMS

INTRODUCTION

This chapter describes two very simple, but useful, applications of the RS-232-C interface (Transmission of data in and out of an M20). The first example describes the use of a small BASIC program to permit the user to set up the required transmission parameters (character length, baud rate, etc.) ready for transmission of data in or out of an M20. The second example describes the use of a small program to accept a string keyed in and to transmit it over the RS-232-C interface. The third example describes a small program to receive a string of character codes over the RS-232-C interface and display them on the M20 screen. The fourth example in addition uses a Status read to check that the data has been received without parity, frame or overrun transmission error.

Four example programs which could perform these functions are worked through.

SETTING UP TRANSMISSION PARAMETERS

This first example sets up the transmission parameters by defining an SCOMM (set communication port) command according to the desired transfer characteristics (lines 19Ø to 39Ø). The sample program is listed in Figure 4-1: it asks the user to make a selection from the transmission parameters offered or accepts default settings.

```
1Ø REM
2Ø REM program to set up transmission parameters
3Ø REM
4Ø DIM PN$ (3)
5Ø DATA "com:","com1:","com2:"
6Ø FOR I = 1 TO 3 : READ PN$(I):NEXT I
7Ø DIM R$(8)
8Ø DATA "5Ø","11Ø","3ØØ","6ØØ","12ØØ","24ØØ","48ØØ","96ØØ"
9Ø FOR I = 1 TO 8 : READ R$(I) : NEXT I
1ØØ DIM P$(3)
11Ø DATA "none", "odd", "even"
12Ø FOR I = 1 TO 3 : READ P$(I) : NEXT I
13Ø PRINT :INPUT "'rs' already executed? (yes-no) ",A$
14Ø IF A$<>"no" THEN 16Ø
15Ø EXEC "rs"
16Ø PRINT :INPUT "'ci' already resident? (yes-no) ",A$
```

Figure 4-1 Program to set up transmission parameters (cont.)

```
170 IF A$<>"no" THEN 190
180 EXEC "pl ci"
190 PRINT :INPUT "enter port number (0-1-2)", PN%
200 IF PN%<0 OR PN%>2 THEN PRINT "wrong selection" : GOTO 190
210 PRINT :INPUT "enter baud rate (50-110-300-600-1200-2400-4800-9600)
    ",BR$
220 FOR I=1 TO 8:IF R$(I)=GRI THEN 250
230    NEXT I
240    PRINT "wrong selection" : GOTO 210
250 PRINT:INPUT "enter parity (none-odd-even)", PA$
260 FOR I=1 TO 3:IF P$(I)=PA$ THEN 290
270    NEXT I
280    PRINT "wrong selection":GOTO 250
290 PRINT :INPUT "enter no. of stop bits (0=1 bit 1=1.5 bits 2=2 bits)"
    SB$
300 IF SB$<"0" OR SB$>"2" THEN PRINT "wrong selection" : GOTO 290
310 PRINT :INPUT "enter no. of data bits (5-6-7-8) ", DB$
320 IF DB$<"5" OR DB$>"8" THEN PRINT "wrong selection":GOTO 310
330 PRINT:INPUT "enter duplex (full-half) ", DU$
340 IF DU$<>"full" AND DU$<>"half" THEN PRINT "wrong selection": GOTO 330
350 PRINT :INPUT "enter handshake (on-off) ", HA$
360 IF HA$<>"on" AND HA$<>"off" THEN PRINT "wrong selection":GOTO 350
370 PRINT :INPUT "enter buffer size (>0) ", BS$
380 IF BS$<= "0" THEN PRINT "wrong selection" :GOTO 370
390 ES$ = "sc   "+PN$(PN%+1)+","+BR$+","+PA$+","+SB$+","+DB$+","+DU$+",
    "+HA$+","+BS$
400 EXEC ES$                                    'set communications port
410 E% = 0
420 CALL "ci" (PN%,"o",@E%)          'open port
430 IF E%=0 THEN GOTO 460
440    PRINT "port open error = ";E%
450    STOP
460 PRINT:INPUT "default transmission parameters ok? (yes-no)", A$
470 IF A$<>"no" THEN 650
480 HS% = 0
490 A$ = "transmit enable":GOSUB 700
500 HS% = HS% + 1%
510 A$ = "data terminal ready" : GOSUB 700
520 HS% = HS% + 2*I%
530 A$ = "receive enable":GOSUB 700
540 HS% = HS% + 4*I%
550 A$ = "send break character":GOSUB 700
560 HS% = HS% + 8*I%
570 A$ = "error reset" : GOSUB 700
580 HS% = HS% + 16*I%
590 A$ = "request to send" : GOSUB 700
600 HS% = HS% + 32*I%
610 CALL "ci" (PN%,"sw",@E%,,@HS%)          'set hardware status
620 IF E%=0 THEN GOTO 650
630    PRINT "status write error = "; E%
640    STOP
650 PRINT : PRINT "end of setup program"
660 END
700 PRINT
710 PRINT A$;" (1=yes,0=no)";
720 INPUT I%
730 IF I%<>0 AND I%<>1 THEN PRINT "wrong selection": GOTO 700
740 RETURN
```

Figure 4-1 Program to set up transmission parameters

RS-232-C SAMPLE PROGRAMS

PROGRAM SET-UP

In accordance with normal CI and BASIC programming practice:
all variables are defined as integer before usage; three arrays are dimensioned to contain the set of port device names (line 4Ø), baud rates and parity check values (line 1ØØ).

MAKING CI RESIDENT

Line 13Ø asks the user if the RS-232-C driver is already resident and if not, makes it resident by use of the EXEC statement in line 15Ø.

Line 16Ø, asks the user if CI is already resident and if not, makes the command resident by use of the EXEC statement in line 18Ø.

SELECTING TRANSMISSION PARAMETERS

Line 19Ø-37Ø allow the user to set up each of the SCOMM parameters port by port. In each case he is asked to choose one of the valid values available and advised of a wrong selection otherwise.

Then the desired port is opened (line 42Ø) and any discovered port open error indicated (line 44Ø). See also CI Open command.

SELECTING HARDWARE STATUS

The user is asked to make a selection from all the available possible defaults in the hardware status byte (see also Table 2-2). When the selection is complete the status byte is written on the desired port (line 61Ø).

Then the user is advised that the set-up program is complete (line 65Ø). Should such a set-up not be possible, the corresponding error message and value is displayed (line 63Ø). See also CI Status Read command.

TRANSMIT DATA FROM M20

The second example deals with the actual transmission of data from the M20.

The sample program listed in Figure 4-2 specifically transmits a string of characters entered at the keyboard when prompted by the system and is considered complete when a carriage return is entered. To interrupt program execution, the user must enter CTRL C.

```
10 REM
20 REM     program to transmit data
30 REM
40 PN% = 0
50 E% = 0
60 LINE INPUT "enter string",A$
70 CALL "ci" (PN%, "w",@E%, A$, 13)
80 IF E%<>0  THEN PRINT "write error ="; E% :STOP
90 GOTO 60
```

Figure 4-2 Program to transmit data from M20

DATA TRANSFER

Line 60 asks the user to enter a string of characters.

Each character of the string is then sent to the transmission line by the CI command (line 70).

The end character is not transmitted.

ERROR CHECK

The returned error value from the CI Write command is used to display a suitable message.

RECEIVE DATA IN M20

The third and fourth examples both deal with the actual receipt of data in an M20.

The sample programs both specifically receive a string of ASCII codes on the interface when prompted by the system and considers the string complete when a carriage return code is detected.

WITHOUT STATUS CHECK

Figure 4-3 lists this program.

```
1Ø REM
2Ø REM  program to receive data (without status check)
3Ø REM
4Ø PN% = Ø
5Ø E% = Ø
6Ø BC% = Ø
7Ø C$ = SPACE$(1)
8Ø PRINT : PRINT "ready to receive"
9Ø CALL "ci" (PN%, "sr",@E%,,,@BC%)        'check one character
1ØØ IF E%<>Ø  THEN PRINT "status read error = ";E% :STOP
11Ø IF BC%=Ø THEN GOTO 9Ø
12Ø CALL "ci" (PN%, "r",@E%,@C$, 1)        'receive one character
13Ø IF E%<>Ø  THEN PRINT "read error = ";E% : STOP
14Ø IF ASC(C$)<>13  THEN GOTO 17Ø
15Ø IF A$ = "END"  THEN GOTO 19Ø
16Ø PRINT A$ : A$ = "" : GOTO 8Ø
17Ø A$ = A$ + C$
18Ø GOTO 9Ø
19Ø CALL "ci" (PN%, "c",@E%)                'close port
2ØØ IF E%<>Ø  THEN PRINT "port close error = ";E% :STOP
21Ø PRINT : PRINT "end of receive program"
22Ø END
```

Figure 4-3 Program to receive in M20 (without status check)

Data Transfer

When set-up is complete, line 8Ø ensures that by a message on the screen the user is aware that the M20 is ready to receive. The CI command of line 12Ø receives the ASCII codes one by one over the port (returned variable C$).

Lines 14Ø to 16Ø build these codes into a string of ASCII characters displayed on the video screen once the carriage return terminator has been detected.

Receipt Check

Lines 9Ø to 11Ø make use of the buffer count (returned value BC) in a CI Status Read command to detect when a character has been received on the serial RS-232-C port.

Any Read error is indicated with a suitable message (line 13Ø).

End of Data

When the "END" string has been received (line 15Ø), input is returned to the keyboard (line 19Ø) and the port is closed with an indication of any related error (line 2ØØ) and the final message of line 21Ø.

WITH STATUS CHECK

Figure 4-4 lists this program which is similar to that of Figure 4-3 except for the status check (lines 18Ø to 26Ø) and the use of the composite status mask variable SM.

The description of the part common to the two programs is not repeated here. (See description of without status check version).

```
10 REM
20 REM  program to receive data (with status check)
30 REM
40 PN% = 0
50 E% = 0
60 HS% = 0 : DS% = 0
70 BC% = 0
80 C$ = SPACE$(1)
90 PRINT : PRINT "ready to receive"
100 CALL "ci" (PN%, "sr",@E%,@HS%,@DS%,@BC%)          'get status & buffer
110 IF E%=0 THEN GOTO 130                             'count
120 PRINT "status read error = ";E% : STOP
130 IF BC%=0  THEN GOTO 100
140 CALL "ci"  (PN%, "r",@E%,@C$, 1)                  'get one character
150 IF E%=0  THEN GOTO 270
160 PRINT "read error = ",E%
170 IF E%<>4  THEN STOP
180 SM% = HS% AND &H8
190 IF SM%=&H8 THEN PRINT "parity error"
200 SM% = HS% AND &H10
210 IF SM%=&H10  THEN PRINT "overrun error"
220 SM% = HS% AND &H20
230 IF SM%=&H20 THEN PRINT "framing error"
240 SM% = DS% AND &H100
250 IF SM%=&H100 THEN PRINT "buffer overflow error"
260 STOP
270 IF ASC(C$)<>13  THEN GOTO 300
280 IF A$="END"  THEN GOTO 320
290 PRINT A$ : A$ = "" : GOTO 90
300 A$ = A$ + C$
310 GOTO 100
320 CALL "ci" (PN%, "c",@E%)                          'close port
330 IF E%<>0  THEN PRINT "port close error = ";E% :STOP
340 PRINT :PRINT "end of receive program"
350 END
```

Figure 4-4 Program to receive in M20 (with status check)

Status Check

Lines 180 to 260 make use of the hardware status byte and driver status word. The returned values HS and DS read on the port (line 100) are used to detect parity, overrun and framing errors. The nature of the error detected is displayed on the screen (lines 190, 210, 230 and 250).

PART II - IEEE 488 PARALLEL INTERFACE

5. IEEE 488 RELATED CONCEPTS

ABOUT THIS CHAPTER

This chapter describes the hardware and software concepts behind IEEE 488 to aid understanding of the programming statements in chapter 6. The final section shows the relationship between these concepts.

CONTENTS

IEEE 488 RELATED CONCEPTS

THE INTERFACE HARDWARE

Inside the M20, the interface hardware consists of an extension board and a 24-wire cable, linking this to the connector socket on the back panel of the machine. The hardware conforms to the IEEE 488 Standard (see Appendix A), and uses the Port Addresses listed in appendix D. A half size representation of the board is shown in figure 5-1.

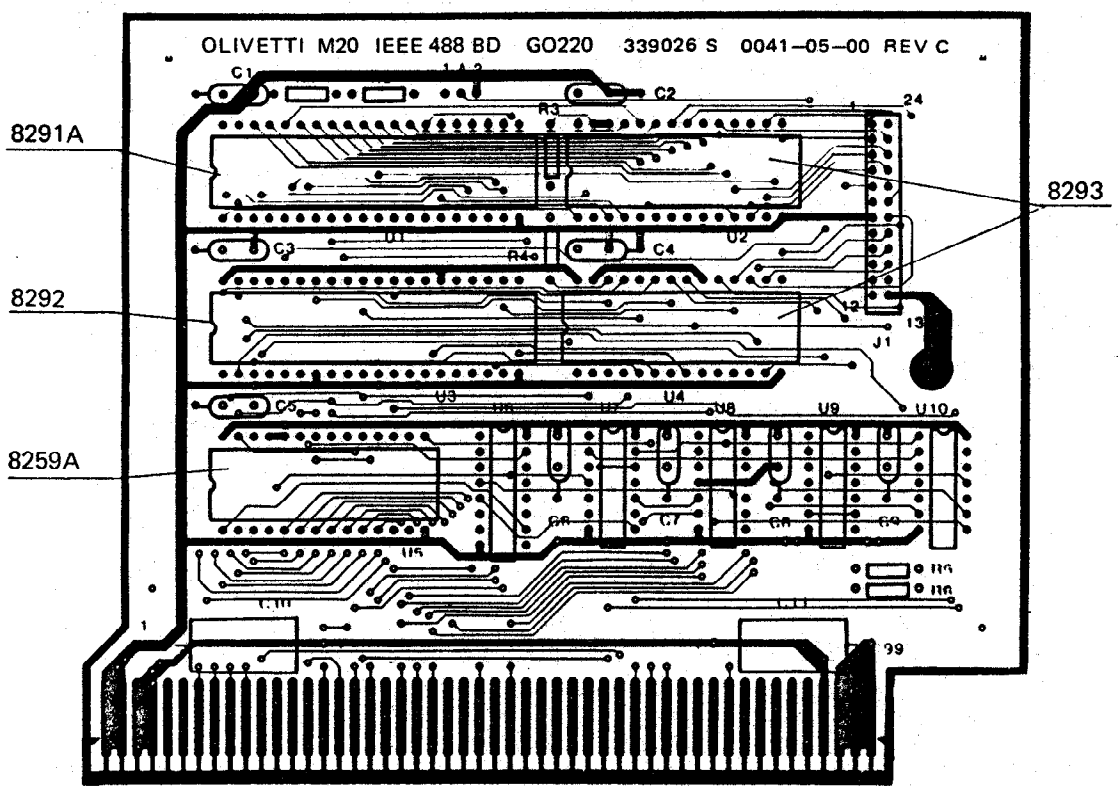


Figure 5-1 The IEEE 488 Expansion Board

The main IEEE 488 integrated circuit (IC or chip) is the 8291A GPIB (General Purpose Interface Bus), which performs all of the functions except for that of the controller, which function is performed by the 8292 GPIB Controller.

THE GENERAL PURPOSE INTERFACE BUS

Together, the 8291A and the 8292 form the complete interface which handles communication between the M20 and the IEEE 488 bus via the 24 wire

connector. They handle the handshake protocol, talker/listener addressing procedures, data transfers, service requests, and serial polling. The M20 itself is not interrupted unless a data byte has arrived, or is to be sent out.

The two 8293 GPIB ICs provide the electrical interface for the talker/listener and talker/listener/control configurations. They form the interface between the 8291A and 8292 and the IEEE 488 bus itself. Their main function is to convert incoming and out-going signals to a form the eventual receiver will understand - many signals need to be inverted.

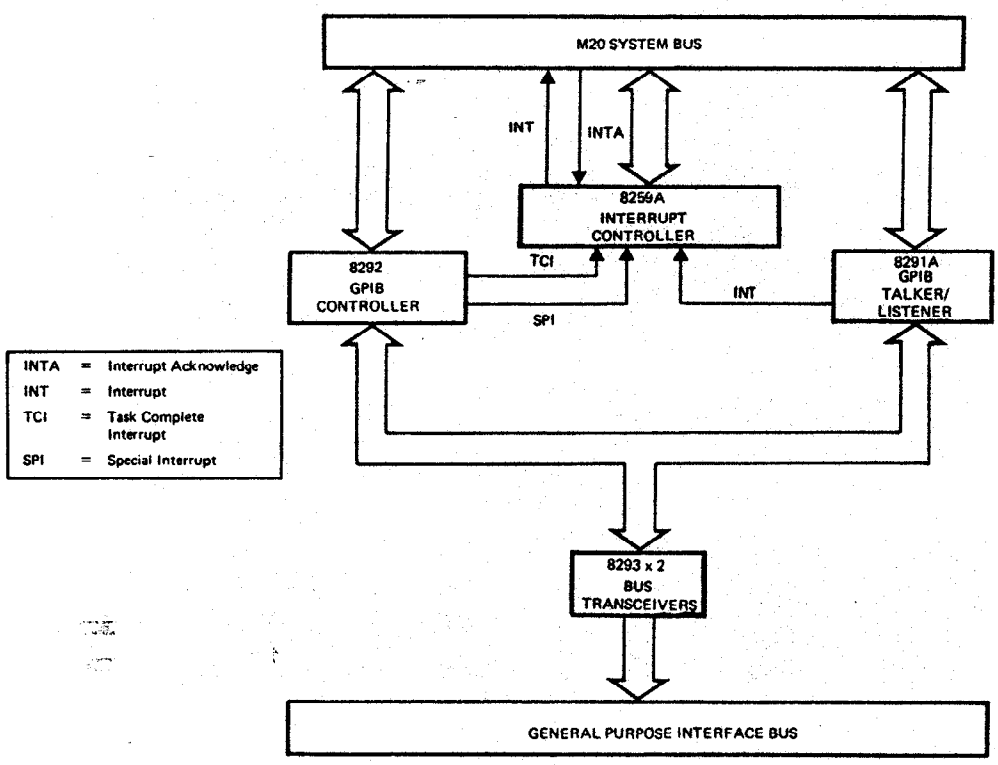


Figure 5-2 IEEE 488 Block Diagram

The last IC on the board, the 8259A, is the interrupt controller, which interrupts the M20 when the 8291A and 8292 request service (SRQ), and decides on an order of priority for these messages. The ICs are all clearly marked in Figure 5-1.

As Figure 5-2 shows, the 8259A and the 8293s do not have quite the same function. A simple analogy is that the 8293s act as a Toll gate, through which each bit must pass. The 8259A on the other hand, acts like traffic

police, merely keeping the information flowing, but not intruding unless it is absolutely necessary.

The location of the connector socket on the back panel is displayed in Figure 5-3. The IEEE 488 connector conforms to the specifications indicated in the IEEE 488 Standard (see Appendix A).

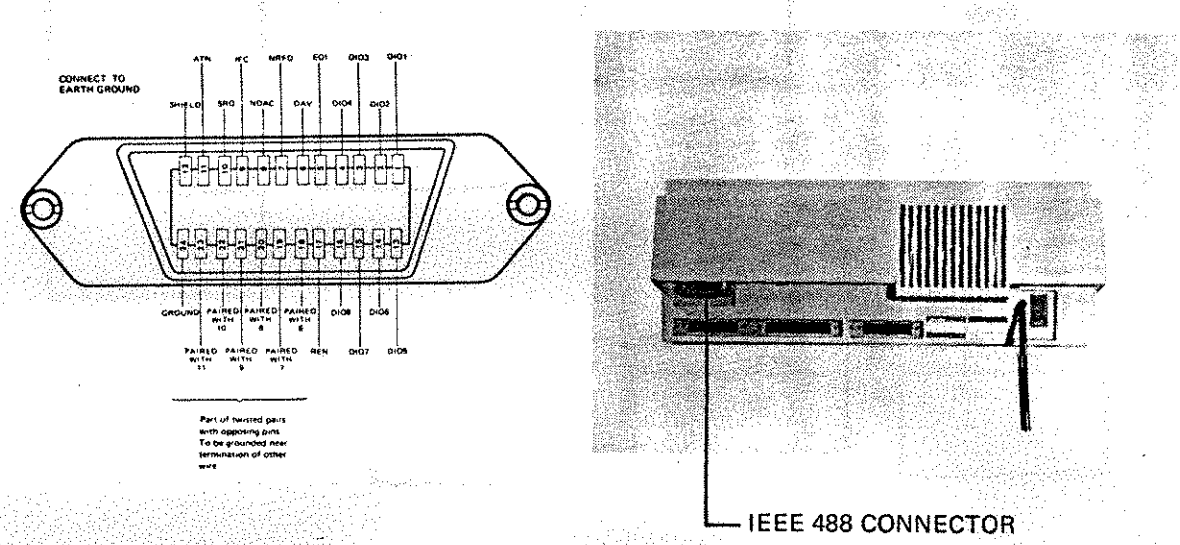


Figure 5-3 The IEEE 488 Connector and its Location on the M20

MOUNTING A CONNECTOR

Olivetti supplies IEEE 488 connection cables in 4 lengths (1.Ø, 2.Ø, 3.Ø and 4.Øm), all having a suitable connector on each end. Similar connection cables are available from IEEE 488 device manufactures. It is important to note that there are two types of mounting fasteners, metric and English. The M20 is supplied with metric screw fasteners, which are readily identified by their black colour. English threaded (6-32UNK) fasteners are silver. ON NO ACCOUNT attempt to mate black to silver or silver to black, as this can damage the hardware itself.

CONFIGURATIONS

Peripherals can be connected in a star or a linear combination network. In a linear network, each cable end connects either to a device only, or to a device an another cable.

In a star network, each cable end connects either to a central device or to another device (see Figure 5-4). This network has a limitation: no more than 4 cables should be fixed to a device. This limitation can be overcome by combining linear and star networks.

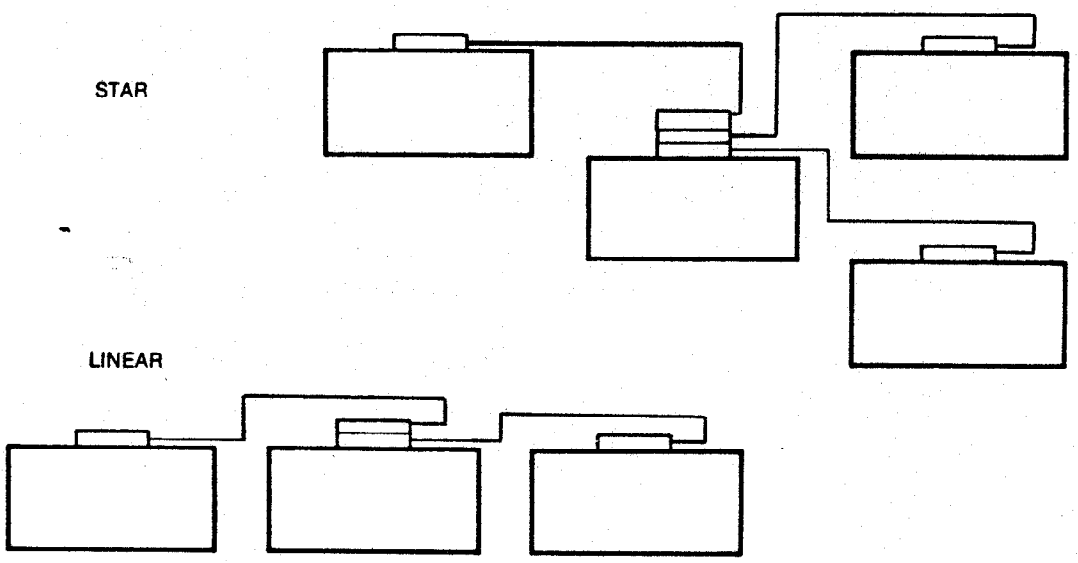


Figure 5-4 Linear and Star Networks

Any combination of these can be used, provided that no more than 15 devices are linked together, and provided that certain mechanical restrictions laid out below are adhered to.

MECHANICAL RESTRICTIONS

In order to ensure the accurate transmission of data, there is a 2Ø metre limit to the length of cabling allowed. Further, there should be no more than two metres per device. Thus, if there are 4 devices, there should be no more than 8 metres of cable, but if there are 12 units, they must be configured using only 2Ø metres of cable between them.

INTERFACE SOFTWARE

The following paragraph describes the interface at a user level only. Once these concepts are understood, further details can be obtained from the IEEE 488 Standard (see Appendix A).

IEEE 488 RELATED CONCEPTS

The IEEE 488 Standard defines a total instrumentation interface package which not only defines to very strict limits the hardware, as described in the section on hardware above, but also the software, so that the problems of instrumentation integration are minimised.

This means that instruments which are IEEE 488 compatible all have the same, standard, connectors and all can respond to the complete set (or a subset) of IEEE commands. This section describes how these commands are passed to and fro.

CONCEPTS

During communications between devices on the IEEE 488 Bus each device may perform one or more of the following roles defined by the Standard:

- System Controller

- Controller-in-Charge

- Talker

- Listener.

Only one System Controller can exist in a given Interface System and it must remain active as long as any operation is in progress on the System. (When the operation is finished the System Controller can yield the control to another device that has the appropriate capabilities). The System Controller is the only device that can interrupt any operation (by issuing the command Interface Clear-IFC) to start another activity right away. It can define itself to be the Controller-in-Charge or it can pass the control to another device with Controller capability.

Only one Controller-in-Charge can be active at a time on the IEEE 488 Bus. It controls actions in other devices on the Bus. Only one Talker can be active at a time on the IEEE 488 Bus. But the Controller-in-Charge can stop the current Talker and start another. In any case the active Talker places data onto the Bus for any current Listener(s).

One (or more) Listener(s) can be active as selected by the Controller-in-Charge. Each active Listener accepts data from the IEEE 488 Bus from the active Talker.

Data transfers from the active Talker to any Listeners are byte-serial bit-parallel. IEEE 488 Bus lines DIO1-8 carry data codes in either of two forms: when the Controller-in-Charge wants attention (ATN = true), addresses and commands are listened to by all devices, and lines DIO1 - 7

carry ASCII (ISO) bits 1 - 7 while line DIO8 is available for a parity bit; when the Controller-in-Charge no longer wants attention (ATN = false), only the device currently addressed to talk will talk, and only the device(s) currently addressed to listen will listen. Then all eight DIO lines can carry any code, of eight or fewer bits, that is understood by both the talker and the listener(s).

The Interface System must have at least one Talker device (for example, a digital voltmeter) and one Listener device (for example, a printer). The System can also have one (or more) Controller device(s) and one (or more) other Talker(s) and one (or more) Listener(s).

The IEEE 488 Interface defines the methods by which a "Controller" (here the M20) can control a "Talker" (usually a measuring instrument such as a voltmeter) and a "Listener", or "Listeners" (usually recording instruments, such as a printer, tape punch etc.). In actual fact, talker-only or listener-only devices are rare. Most have at least a limited capacity to do both. Nonetheless, a tape-recorder which can also talk would, for ordinary purposes, be referred to as a listener.

The IEEE 488 Interface bus is essentially a 24 wire device interconnector. These are divided into control and timing (8 wires), and data transmission (8 wires). In addition, there are 8 ground wires. This accounts for all 24 pins on the IEEE 488 connector.

The M20 Implementation

In the M20 implementation of IEEE 488 the M20 is the system controller. Only the M20 is able to be active controller; it cannot pass control to another device.

INTERFACE FUNCTIONS

The interface functions defined by the IEEE 488 Standard, and implemented on the M20 are:

- SH1 (Source Handshake): This means the M20 can be one of two or more devices that exchange data in an interlocked sequence. There must be an acceptor handshake function active in at least one other device.

- AH1 (Acceptor Handshake): This means the M20 can be one of two or more devices that exchange data in an interlocked sequence. There must be only one source handshake function active in another device.

- T6 (Talker): This means the M20 can send data to other devices.

- L4 (Listener): This means the M20 can receive data from another device.

- C1, C2, C3, C4, C27 (Controller): This means the M20 can send universal commands, addresses, and addressed commands to devices on the IEEE 488 Bus. The M20 can also conduct serial polls of other devices.

INTERFACE MESSAGES

The IEEE 488 Standard defines the operation of the functions above in terms of "states". Each function has several clearly defined states, but it can only exist in one state at a time. For example, the Listener state can be in one of the: Listener IDle State (LIDS); Listener ADdressed State (LADS); or Listener ACtive State (LACS). Thus at any one moment, each supported function, in each attached device, must be in ONE, AND ONLY ONE, permitted state. The most frequently discussed state is "active", as a device must be active to operate. An active listener is thus one which is in LACS state. Correspondingly, an active talker is TACS. A device may contain both talker and listener functions, without being active in either, if it is in the relevant idle state.

The type of communication permissible for a given device at any given time is determined by the states of its functions. For example, a talker cannot transmit data while it is in the Talker IDle State (TIDS), or can only respond to a serial poll if it is in the Serial Poll Active State (SPAS).

The rules governing when active functions can change from one state to another are defined in the IEEE 488 Standard, and are beyond the scope of this document, but in general, the elementary conditions causing a state transition may include:

- other current states (in other functions)

- the occurence of specified events within (or after) defined time intervals

- local messages - conditions occurring within the device, according to device-dependent logic

- interface messages - information received over the bus from the controller.

More details on these, and all the other states can be found in the IEEE 488 Standard.

The Interface message above is a remote message. A second type of Interface message is device-dependent messages. Interface messages control the interfaces in other devices, device-dependent messages include the data bytes transmitted between the devices. Interface messages thus function as commands.

Some commands are defined states of a single interface line, for example IFC true; these are called uniline messages. Other messages are defined combinations of states of both control and data lines, for example "untalk"; these are multiline messages. Only one multiline message (message byte) may be sent at any one time, but several uniline messages may be sent concurrently.

The BASIC statements discussed in Chapter 6 allow the user to issue any byte in Command Mode, but only a small number of bytes are valid multiline interface messages (see Appendix C). Appendix C shows messages involved in parallel polling and "take control" also which are not supported on the M20.

THE CONTROL LINES

The eight control lines are all assigned to connector pins, as follows:

PIN	MNEMONIC	LINE NAME
5	EOI	End Or Identify
6	DAV	Data Valid
7	NRFD	Not Ready For Data
8	NDAC	Not Data Accepted
9	IFC	InterFace Clear
10	SRQ	Service ReQuest
11	ATN	ATteNtion
17	REN	Remote ENable

These 8 lines can be further divided into 2 categories, Handshake (DAV, NRFD, NDAC), and Management (Control).

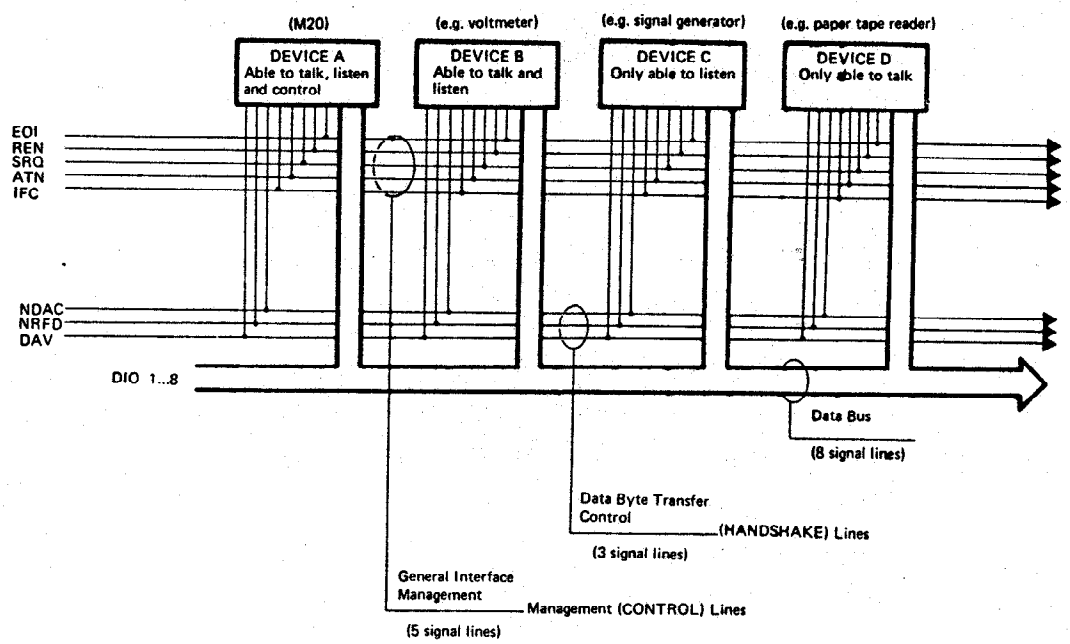


Figure 5-5 IEEE 488 Signal Lines

Figure 5-5 shows diagrammatically the IEEE 488 signal lines, and correlates some of the ideas already discussed with some of those that will be discussed now.

DAV, NRFD AND NDAC: THE THREE WIRE HANDSHAKE

Data transmission from the talker to the listener(s) occurs asynchronously so that devices of different speeds can operate from the same IEEE 488 Bus. Thus data is always transmitted at the speed of the slowest device on the Bus. To understand how, consider a competitive examination given to qualify candidates for admittance to a college. An administrator is assigned (addressed) and the candidates are invited (addressed). To ensure impartiality, certain procedures are followed, including displays of each question:

1. No characters of a question are made ready for display until all candidates are ready for a new question.

2. Only after all characters of a question have been made ready is a question displayed.

3. All candidates must accept the question (understand and agree with it).

4. As soon as all candidates have accepted the question, the display is no longer needed and is ended.

5. As each candidate accepts a question he/she starts work on the answer but each finishes at his/her own speed.

6. Steps 1 through 5 repeat until all questions have been answered.

Applying the above steps to an IEEE Std. 488 System, they can be restated with the Talker in place of the administrator, the Listeners in place of the candidates, and a byte of data transmitted instead of a question displayed:

1. No bits of a byte are made ready for transmittal until all Listeners are ready for a new byte. (Signal NRFD-Not Ready For Data-goes false).

2. Only after all bits of a byte have stabilised on the IEEE 488 Bus is the byte transmitted. (Signal DAV-DAta Valid-goes true).

3. All Listeners must accept the byte. (Signal NDAC - Not Data ACcepted - goes false).

4. As soon as all Listeners have accepted the byte it is no longer needed and is cancelled. (Signal DAV goes false).

5. As each Listener accepts the byte it starts to use the byte but finishes at its own speed. (Signal NRFD goes true).

6. Steps 1 through 5 repeat until all bytes have been used.

"Handshaking" thus uses three IEEE 488 Bus lines: DAV, NRFD, and NDAC.

ATN, IFC, REN, SRQ AND EOI: GENERAL INTERFACE MANAGEMENT

The data lines carry command instructions as well as data. To inform the peripheral that the incoming information is an interface message, the ATN line is set true - Command Mode. To comply with the Standard, every device must:

- monitor the ATN line at all times, so that it is aware how to treat the data

- respond to ATN going true within 2ØØ ns

- at the end of the instruction(s), ATN must be set false - Data Mode, so that the listener(s) can read the subsequent information as data.

ATN Line

The rest of this subsection describes the various commands that can be sent while ATN is set true. If it were not true, the byte(s) would be treated as data.

When ATN (ATteNtion) is set true, the listeners are to interpret the incoming byte(s) as commands, i.e. interface messages.

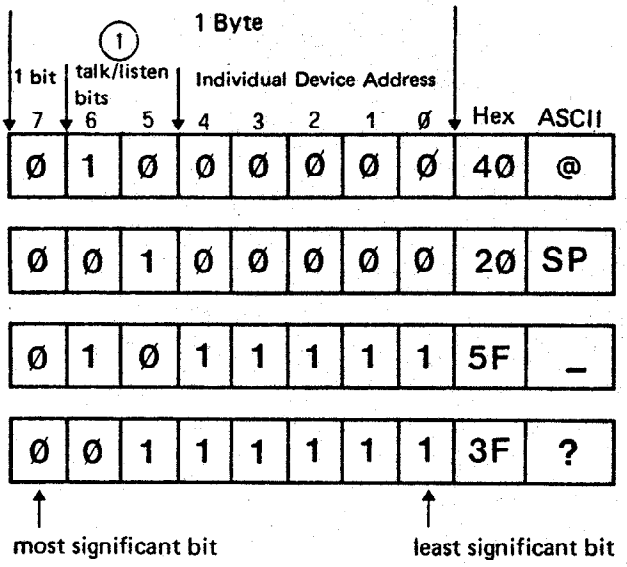
Addressing a peripheral: The Talker has 8 data lines with which to talk to the Listeners. The 8 data lines operate in binary format - either sending a logical Ø or logical 1. Each line corresponds to a "bit". They do this simultaneously so the Listener receives 8 bits at once. These 8 bits (or a byte) of data can be followed serially by further bytes. Thus the interface is bit parallel, byte serial.

In the simplest case there are only two attached devices, but up to 15 may be connected so it is important that each has a unique address. This is done using the 5 least significant bits of the (first) byte. Of the 32 possible values for the device address that this provides, only 31 (Ø-3Ø) are available. Decimal 31 (11111) is used by the system to show that the instruction is a universal "untalk" or "unlisten". If it is Universal Unlisten, 31 (hex 1F) is added to hex 2Ø (listen) making 3F. For Universal Untalk, 1F is added to 4Ø, making 5F (see Universal Commands below). Most devices come with a default value set. This can be altered using the Dip-switches provided, or some instruments allow front panel, programmable entry.

ADDRESS BINARY	DECIMAL	DEVICE
ØØØØØ	Ø	System controller
Ø1ØØØ	8	Oscillator
1ØØØØ	16	Tape punch
11ØØØ	24	Voltmeter 1
11Ø1Ø	26	Voltmeter 2
11111	31	Illegal - used for universal untalk or unlisten

These are just a sample of the addresses. Appendix C shows them all. The most significant bit of the byte is never used.

Bit 6 is hex 4Ø, bit 5 hex 2Ø. These are used to indicate Device Talk and Device Listen respectively.



(1) Bit 6 = 1, Bit 5 = Ø TALKER
Bit 6 = Ø, Bit 5 = 1 LISTENER

Figure 5-6 Bit Configuration Convention

To talk/listen a specific device, their respective number must be added to the device number, when using WBYTE or RBYTE. So, for example, to Talk (1ØØØØØØ) voltmeter 1, the byte 1Ø11ØØØ would be sent:

```
1ØØØØØØ +
  11ØØØ =

1Ø11ØØØ
```

Similarly, to Listen (Ø1ØØØØØ) the tape punch, Ø11ØØØØ would be sent. These addresses, with the talk or listen instruction added, are sometimes referred to as "my talk" or "my listen" address. If an active talker reads a talk address which is not its own, it automatically drops to its idle state. This is not true of the listen address, because more than one listener can be present on the system at one time. Appendix C shows the ASCII code and the binary value for all possible address numbers.

When using PRINT@ , INPUT@ , or LINE INPUT@ the relevant bit (hex 4Ø for talk, hex 2Ø for listen) is added automatically to the device number. The user is thus only concerned with the talk or listen command addition if WBYTE or RBYTE is being used.

Universal Untalk and Universal Unlisten, which are discussed below, are the talk and listen addresses, with the device address set at 11111 - which is why it is an illegal device address. Thus Universal Untalk becomes 1Ø11111 (5F), and Universal Unlisten becomes Ø111111 (3F).

Programming the interface is carried out using the nine BASIC statements discussed in Chapter 6. Two of these, RBYTE and WBYTE, can be used to transmit any of the nine supported IEEE 488 commands, or to address devices as talkers or listeners.

The IEEE commands can be divided into two categories, Universal commands which address all the attached devices; and addressed commands, which only talk to specified devices. The mnemonics of these commands are listed in appendix C.

Universal Commands: These six commands are illustrated in the table below (see also Appendix C):

COMMAND		CODE		
MNEMONIC	NAME	HEX	DECIMAL	ASCII
UNT	UNTalk	5F	95	-
UNL	UNListen	3F	63	?
DCL	Device CLear	14	2Ø	DC4

COMMAND MNEMONIC	NAME	CODE HEX	DECIMAL	ASCII
LLO	Local LockOut	11	17	DC1
SPE	Serial Poll Enable	18	24	CAN
SPD	Serial Poll Disable	19	25	EM

Table 5-1

- UNT (UNTalk): unaddresses the current talker. Since addressing one talker to talk automatically unaddresses all the other talkers, it is not necessary to use untalk when switching from one talker to another.

- UNL (UNListen): unaddresses all current listeners - it is not possible to unaddress one listener at a time. This command should be used each time a listener, or group of listeners, is to be addressed, to ensure that only the required listeners are actually listening (so the transfer is not held up by slow listeners who do not need the information anyway).

- DCL (Device CLear): causes all peripherals to be cleared. The definition of this state varies from one device to another, but individual manuals should make the situation clear for each device. The peripheral devices should respond to the command whether they are addressed or not.

- LLO (Local LockOut): disables the return-to-local or reset on a specific device. This means that local control cannot be regained by using these. Thus the settings made remotely through the interface cannot be tempered with. Like DCL, this is a universal command, so devices recognising it should enter a lockout state whether they are addressed or not. To return to normal use, REN must be set false, which places all devices back under local control.

- SPE/SPD (Serial Poll): on the M20, Serial Poll Enable and Disable are performed by the same BASIC statement, POLL. When a device receives a POLL, it responds with a single eight-bit byte which shows the device status. Only devices which are capable of talking (many devices functioning mainly as listeners can talk, e.g. printers can warn users that they are out of paper) can output this byte, so they are the only ones which can be polled. When the controller receives the status byte,

the serial poll disable byte is sent out to the device. The device remains in the talker state.

Addressed Commands: These three commands are illustrated in the table below (see also Appendix C):

COMMAND MNEMONIC	COMMAND NAME	CODE HEX	CODE DECIMAL	CODE ASCII
GET	Group Execute Trigger	Ø8	8	BS
SDC	Selected Device Clear	Ø4	4	EOT
GTL	Go To Local	Ø1	1	SOH

Table 5-2

- GET (Group Execute Trigger): provides a way to trigger several listeners simultaneously. All devices that are currently addressed to listen are triggered to start a previously programmed action.

- SDC (Selected Device Clear): the device which had been previously addressed to listen is now to go to its clear state. This is exactly the same as DCL, except that only one, named, peripheral is involved.

- GTL (Go To Local): returns control to Local lockout, i.e. control is returned to the front panel of the instrument in question.

IFC, REN, SRQ and EOI Lines

The other 4 lines which are in the General Interface Management organise the transmission. This is necessary to ensure that the right information is going to the right device(s) and that they are able to accept it (the printer has not run out of paper for example).

These 4 functions are:

- IFC (InterFace Clear): initialises, or reinitialises several functions to their idle states (talkers and listeners idled and unaddressed, and talkers serial poll mode exited, if previously active).

- REN (Remote ENable): allows a device to respond to the instructions issued by the controller or another talker. Whilst other talkers can issue the instructions, only the controller can issue the REN, which establishes remote control.

- SRQ (Service ReQuest): tells the controller that it needs to communicate, as one of the peripherals has a problem. SRQ does not indicate what the problem is, nor who has the problem, this is up to the controller to determine, by polling.

- EOI (End Or Identify): with ATN false, signals that this is the last byte of data in a multibyte sequence. On the M20, the control of EOI is automatically done by the PRINT@ and WBYTE statements (see Chapter 6).

The first two, IFC and REN, are used only by the controller. Devices must respond to the command within 1ØØ ms, but only devices which are capable of remote and local operation need monitor REN.

On the M20, SRQ acts as an interrupt, but only when the user has enabled the interrupt. When SRQ is received, the BASIC interpreter interrupts the current series of events, to find out what the problem was. In this context, data-ready-for-transmission would count as a problem. The controller performs this task by conducting a serial poll.

Serial Poll

A serial poll enables the controller to ascertain if a device, or group of devices, requires service. In return, the controller may receive a byte defining the status of each individual device on the interface. This return byte is called the Status byte. Since only talkers can return this byte, only talkers (or listeners with the ability to talk) can be serially polled.

To conduct the poll, the controller addresses each device sequentially (by sending an SPE (Serial Poll Enable) if IFC (InterFace Clear) is false, and then addressing the device to talk) and evaluates each of the returned status bytes in turn.

One problem with polling in this way is that two devices might send an SRQ at the same time, and only one of them will be discovered (the first one). Therefore it is advisable (but not essential) that the installation is programmed to check each and every device when and SRQ is requested.

PROGRAMMING THE INTERFACE

Programming the interface is done in the BASIC language. This has been upgraded with 9 statements specifically oriented towards IEEE. These statements are fully defined and detailed in the next chapter. They require both the presence of an IEEE 488 expansion board and the PLOADing of the PCOS command IEEE 488, the driver. The absence of either of these generates a user visible error (see Appendix D). This section is merely an introduction, to try to show how these statements relate to all the other lists of mnemonics which have been presented in this chapter, and Appendix C.

AN INTERFACE OVERVIEW

Figure 5-7 below attempts to show this relationship. The two main boxes represent, on the left the M20, and on the right a peripheral device. In between the two are arrows running in both directions. These represent the IEEE 488 interface.

The first set of mnemonics discussed in this chapter were the interface functions; the abilities required of the connected devices. It was pointed out there that not all functions were implemented on the M20 IEEE package. Only those implemented are listed. In order to keep the diagram as simple as possible, AH and SH are omitted, and since this implementation does not support peripheral devices being assigned active controller status, this function is not shown as being present in the "device". With these exceptions, the remaining functions are shown as they relate to each side of the diagram.

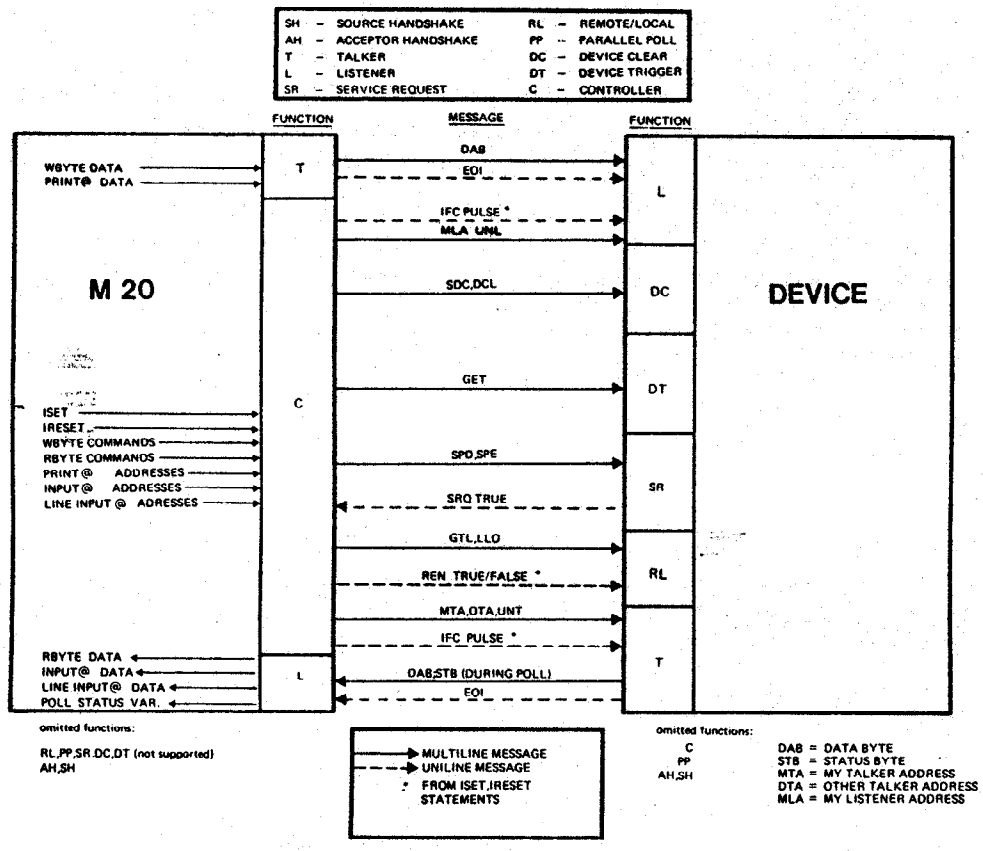


Figure 5-7 Functional Diagram of the Interface

Next come the control lines, These are listed under another of their "features", uniline messages. These are messages which are passed back and forth over the interface, in the direction shown. Since the handshake is not being indicated, DAV, NRFD and NDAC are not present. Nor is ATN, as it too is a special case; to show the effect it has on the multiline (addressed and universal) commands would over-complicate the diagram. This leaves four uniline commands, which are all shown.

The multiline messages come next. For the sake of completeness, the diagram contains multiline messages treated, in this document, not as "commands", but as special cases: data and status bytes (sent with ATN false) and talker and listener addresses. The abbreviations, MTA (My Talker Address) and MLA (My Listen Address) make reference to the particular device and OTA makes reference to other devices.

Lastly the BASIC statements themselves. These are used to allow the M20, as a programmable controller, to interact with the interface, so they are only shown on that half of the diagram. In some cases, the keyword appears twice, once followed by "data", and once followed by either "addresses" or "commands". Data is all the information passed over the interface after the semicolon of any particular statement. Addresses and commands are both written before the semicolon.

IEEE 488 AND BASIC

As discussed in the previous section, there are 8 data lines (DI01÷DI08). These are the ones which transfer the commands and data. The 9 IEEE 488 BASIC statements carry out the necessary instructions to perform these duties, thus making programming the interface considerably easier.

The control lines also affect the attached devices. For example, in order to address a particular instrument, the control line ATN (ATteNtion) must be set true. This is done automatically, by the BASIC interpreter, when a device address is issued by the (user) program.

6. INTERFACE STATEMENTS

ABOUT THIS CHAPTER

This chapter describes all the statements available to program the IEEE 488 Interface bus in the BASIC programming language, provided the PCOS command IE has been PLOADed. A general knowledge of BASIC is assumed throughout the chapter.

CONTENTS

INTERFACE STATEMENTS

INTRODUCTION

This chapter deals with those BASIC statements which are specifically related to IEEE 488. Of necessity, the examples contain other BASIC statements, of which a working knowledge is assumed. All statements used, but not described here, are fully illustrated in the M20 BASIC Language Reference Guide.

Before any of these statements can be utilised, the PCOS command IEEE 488 (IE) must be PLOADed, to load the group of programs which execute the BASIC statements.

The 9 BASIC IEEE extension statements can be classified into 4 groups:

1. Those which control single, specific, bus lines - ISET, IRESET

2. Those which enable, disable, or respond to service requests - ON SRQ GOSUB, POLL.

3. Those which write data to the bus - WBYTE, PRINT@.

4. Those which read data from the bus - RBYTE, INPUT@, LINE INPUT@.

The inclusion of IEEE facilities within the interpretative BASIC language aids the development and debugging of software programs.

The M20 contains a 'time out' feature. If one of the participating devices of the handshake routine (see below) is unable to complete its part of the hardshake within approximately 15 seconds, the M20 will abort the attempted BASIC statement. Such a procedure prevents the M20 from waiting indefinitely for an incompleted Input/Output operation (if this happened, the only resolution would be to reboot the system). Since ISET, IRESET and ON SRQ GOSUB do not use the handshake, this feature is not present in them.

The user is visually warned in some cases, eg. Talker = Listener Address. A full list of the error codes used is provided in Appendix D.

THE REN/IFC STATEMENTS

As discussed in the previous chapter, REN true allows a device to respond to instructions from the controller, or some other talker. Thus REN must be set true before using an instruction command (WBYTE or RBYTE). It must be set false again in order to restore exclusive local control to the devices.

ISET (PROGRAM/IMMEDIATE)

The ISET REN statement sets the Remote ENable to true.

The ISET IFC statement generates an Interface Clear pulse.

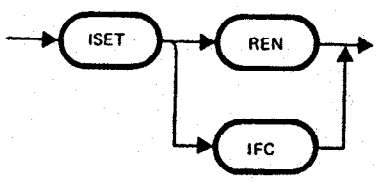


Figure 6-1 ISET Statement

Where

SYNTAX ELEMENT	MEANING
ISET REN	remote enabled is asserted true
ISET IFC	the interface clear pulse is generated

Remarks

REN must be set true in order that the devices on the interface can listen.

Example

For test purposes, a series of IFC pulses can be generated using a BASIC program loop.

```
1Ø FOR K=1 TO 1Ø
2Ø ISET IFC
3Ø NEXT K
```

IRESET (PROGRAM/IMMEDIATE)

Resets REN (Remote ENable) to false.

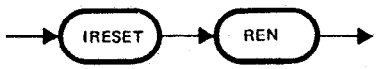


Figure 6-2 IRESET Statement

THE SERVICE REQUEST STATEMENTS

The ON SRQ GOSUB and POLL statements provide the user with a simple means of detecting, and responding to, service requests from both talkers and listeners. ON SRQ GOSUB Ø disables the SRQ (Service ReQuest) interrupt - the request will be ignored. ON SRQ GOSUB line number enables the SRQ interrupt.

ON SRQ GOSUB (PROGRAM)

Enables or disables the SRQ interrupt.

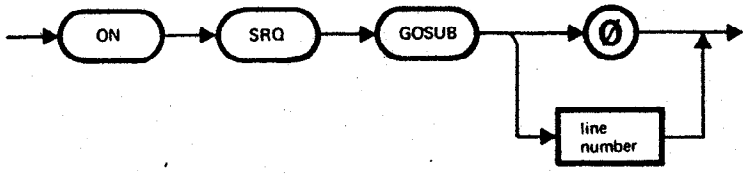


Figure 6-3 ON SRQ GOSUB Statement

Where

SYNTAX ELEMENT	MEANING
Ø	disables the service request interrupt
line number	enables the SRQ interrupt. On occurrence of an SRQ, control will be transferred to the subroutine starting with the line number specified. This statement would ordinarily initiate a BASIC interrupt service routine, containing a Serial Poll and ending with a RETURN statement.

Example

```
 1Ø DEFINT N                                  A series of "4Ø"s will be displayed
 2Ø N2=Ø                                      on the screen. When a service re-
 3Ø ON SRQ GOSUB 7Ø                           quest occurs, the message "service
 4Ø PRINT "4Ø"                                routine entered" will be displayed.
 5Ø IF N2=1 THEN STOP                         A serial poll will then be executed
 6Ø GOTO 4Ø                                   (see below).
 7Ø PRINT "Service routine entered"
 8Ø POLL 7,N
 9Ø PRINT "device status = ",N
1ØØ N2=1
11Ø RETURN
```

POLL (PROGRAM/IMMEDIATE)

Determines whether a device (connected to the IEEE 488 bus) has requested service. Only one-instrument is checked at a time.

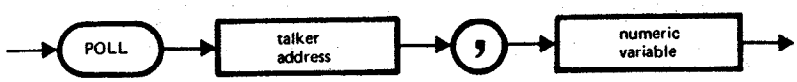


Figure 6-4 POLL Statement

Where

SYNTAX ELEMENT	MEANING
talker address	a hexadecimal number corresponding to the 5-bit binary code for the individual device's identify code.
numeric variable	the name of an integer variable which is to store the status byte returned from the polled device.

Characteristics

The POLL statement starts the following sequence:

If attention (ATN) is set false, it is asserted true.

The following values (hexadecimal) are written to the bus: 3F (UNListen), 18 (Serial Poll Enable), and the talker address specified (with hex 40 automatically added).

ATN is reset (false).

The M20 then inputs (reads) one byte from the bus (the status byte), and saves it in the numeric variable. The requesting device sets bit 6 ON to identify itself.

ATN is asserted true, and hex 19 (Serial Poll Disable - SPD) is written to the bus.

ATN is reset (false), thus returning the bus to Data Mode.

Example

The following example is very simple in order that the byte transmission in the proceeding diagram can be followed. A more typical example of the use of this statement is shown in Chapter 7.

DISPLAY	COMMENTS
1Ø DEFINT N 2Ø POLL 7,N 3Ø PRINT "device status = ",N	N is defined as the integer variable which will accept the status byte (line 1Ø). Line 2Ø polls device 7, reads one byte from the bus, and stores it in the variable N. Line 3Ø prints out the status byte.

Using the example above, the following bytes would be sent across the bus:

BYTE TRANSMITTED	BYTE RECEIVED	ATN		COMMENTS
		T		if ATN is false, it is set to true
3F	-	T	UNL	unlisten
18	-	T	SPE	serial poll enable
47	-	T		talker address (hex 7) plus hex 4Ø
-	43	F		example input from the polled device
19	-	T	SPD	serial poll disable
device status = 67				will be displayed on the screen. 67 is the decimal equivalent of the hex 43.

Table 6-1

Example

This second example shows the use of POLL statements within an interrupt service routine.

DISPLAY	COMMENTS
1ØØ ON SRQ GOSUB 2ØØ . . . 2ØØ POLL 5,N% 21Ø IF N% AND MASK=64 THEN GOSUB 3ØØ 22Ø POLL 9, N% 23Ø IF N% AND MASK=64 THEN GOSUB 4ØØ 24Ø POLL 14, N% 25Ø IF N% AND MASK=64 THEN GOSUB 5ØØ 26Ø RETURN . . .	On occurrence of an SRQ, control is transferred to the subroutine starting at 2ØØ. The 3 attached devices (5, 9 and 14) are addressed one at a time. The requesting device sets bit position 6 to identify itself (x1xxxxxx). Each of the devices which are thus detected then has a separate routine for handling its particular servicing problem. The AND MASK at 21Ø, 23Ø and 25Ø is set at Ø except for bit 6 (=64). Thus any value with bit 6 set will be caught in the logical gate.

THE WRITE STATEMENTS

The WBYTE and PRINT@ statements enable the user to send one or more command and/or data bytes to listeners.

WBYTE writes one or more commands (each command may also be a device address) followed by numeric data.

PRINT@ writes a listener address followed by numeric and/or string data (numeric data is transmitted as a string of characters, where each digit is represented by the associated ASCII character).

Before sending data bytes, WBYTE and PRINT@ sent 5F (UNT). This stops the current talker without enabling another.

An optional "@" at the end of these two statements sends END (EOI true, ATN false) with the last byte of data. If "@" is omitted, CR with END terminates the data.

WBYTE (PROGRAM/IMMEDIATE)

Writes commands to the bus with ATN true, then writes numeric values with ATN false.

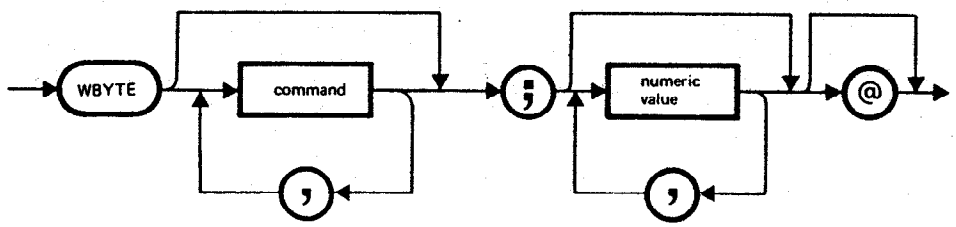


Figure 6-5 WBYTE Statement

Where

SYNTAX ELEMENT	MEANING
command	any numeric integer constant or variable which is the command to be transmitted (with ATN true). The command may also be a talker or listener address.
numeric value	any numeric integer constant or variable (Ø÷255) which is to be transmitted as data (with ATN false).
@	if " @ " is present at the end of the statement, the END message (EOI = true, ATN = false) is sent with the last byte of data. If not, CR (hex ØD) with END terminates the data.

Note: WBYTE is used by the controller to transmit data to other devices on the bus, which must therefore have the ability to listen (see Remark below).

Example

DISPLAY	COMMENTS
Ø5 DEFINT N 1Ø N1=85 2Ø N2=17Ø 3Ø WBYTE 3,5,7;N1,N2	after writing the command byte Ø3, Ø5 and Ø7, WBYTE writes the contents of the variables N1 and N2, i.e. 85 (hex 55) and 17Ø (hex AA)

The next diagram shows the bytes that would actually pass across the interface.

BYTE TRANSMITTED	BYTE RECEIVED	ATN	COMMENTS
Ø3	-	T	Ø3, Ø5 and Ø7 are command bytes transmitted across the bus
Ø5	-	T	
Ø7	-	T	
5F	-	T	UNT(UNTalker)
55	-	F	the hex code for 85, the first byte of data
AA	-	F	the hex code for 17Ø, the second byte of data
ØD	-	F	carriage return. This is accompanied by EOI true.

Table 6-2

Remark

To listen/talk to a specific device the listener or talker flag (hex 2Ø or 4Ø) must be added by the user to the device address.

PRINT@ (PROGRAM/IMMEDIATE)

Writes the listener address (with hex 2Ø automatically added) then data to the bus. Data items may be numeric or string.

If numeric, they are automatically ccnverted into decimal form (if hexadecimal or octal) and then transmitted as sequences of characters

where each digit is represented by the associated ASCII character. Moreover each sequence of characters corresponding to a numeric value is preceded and followed by a space (hex 2Ø).

Commas used to separate data within the statement are also transmitted to the bus.

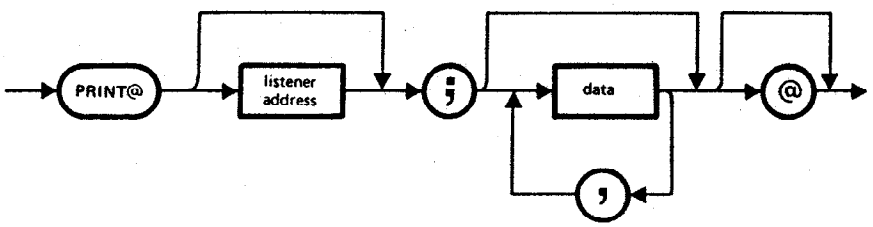


Figure 6-6 PRINT@ Statement

Where

SYNTAX ELEMENT	MEANING
listener address	the hex code for the number assigned to the listener. Hex 2Ø is added to this number before transmission.
data	a numeric or string constant or variable. (A string expression may also be used.)
@	END is sent true with the last byte of data. Without it, CR with END terminates the data.

Example

DISPLAY	COMMENTS
LIST 1Ø PRINT@ 13;12,24,36	the data 12, 24 and 36 are transmitted across the bus to device number 13. This latter is a listener address.

This data would be transmitted as the following bytes, across the interface:

BYTE TRANSMITTED	BYTE RECEIVED	ATN	COMMENTS
2D	-	T	the listener address, decimal 13, is hex ØD. ØD plus hex 2Ø is 2D.
5F	-	T	UNT (UnTalker)
2Ø	-	F	hex 2Ø is ASCII code for space.
31	-	F	31 1
32	-	F	32 2
2Ø	-	F	2Ø space
2C	-	F	2C ,
2Ø	-	F	2Ø space
32	-	F	32 2
34	-	F	34 4
2Ø	-	F	2Ø space
2C	-	F	2C ,
2Ø	-	F	2Ø space
33	-	F	33 3
36	-	F	36 6
2Ø	-	F	2Ø space
ØD	-	F	ØD carriage return accompanied by EOI true.

Table 6-3

Remarks

Hex 2Ø is added automatically to the listener address to tell the peripheral to listen. This does not have to be done by the user.

THE READ STATEMENTS

Three input statements are available:

- the RBYTE statement writes commands and reads values (Ø ÷ 255) into numeric variables
- the INPUT statement outputs addresses and inputs numeric and/or string data into numeric and/or string variables
- the LINE INPUT statement outputs addresses and inputs a line of data into a string variable.

RBYTE (PROGRAM/IMMEDIATE)

Writes command bytes to the bus (as WBYTE), then reads values from the bus and places them in the specified numeric integer variables.

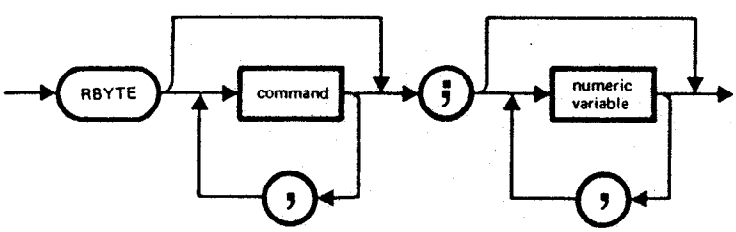


Figure 6-7 RBYTE Statement

Where

SYNTAX ELEMENT	MEANING
command	any numeric integer constant or variable which is to be transmitted to the bus in Command Mode (thus with ATN true). The command may also be a talker or listener address.
numeric variable	the name of a numeric integer variable into which a value (Ø ÷ 255) read from the bus will be stored (with ATN false). At the end of transmission END is received (EOI true, ATN false).

Here is a simple example using the RBYTE statement:

```
DISPLAY                                    COMMENTS

Ø5 DEFINT N                        Commands 14 (hex ØE), 5 and 7 are
1Ø N3=14                           transmitted, then the data return-
2Ø RBYTE N3,5,7;N1,N2               ed by the device are stored in the
3Ø PRINT "values received are",N1,N2  variables N1 and N2.
                                   Line 3Ø prints out the data
                                   returned.
```

The following table shows the bytes transmitted and received across the bus:

BYTE TRANSMITTED	BYTE RECEIVED	ATN	COMMENTS
ØE	-	T	hex ØE (dec 14)
Ø5	-	T	
Ø7	-	T	
-	55	F	example input

BYTE TRANSMITTED	BYTE RECEIVED	ATN	COMMENTS
	AA	F	example input (as this is the last data byte read EOI is set true).
values received are 85	17Ø		will then be displayed on the screen.

Table 6-4

Remark

To listen/talk to a specific device the value hex 2Ø or 4Ø (the listener and talker flags) must be added by the user to the device address.

INPUT@ (PROGRAM/IMMEDIATE)

Outputs addresses (with ATN true) and then inputs data into specified variables (with ATN false). END must be sent true with the last byte. The driver ignores a single CR, LF at the end of each line.

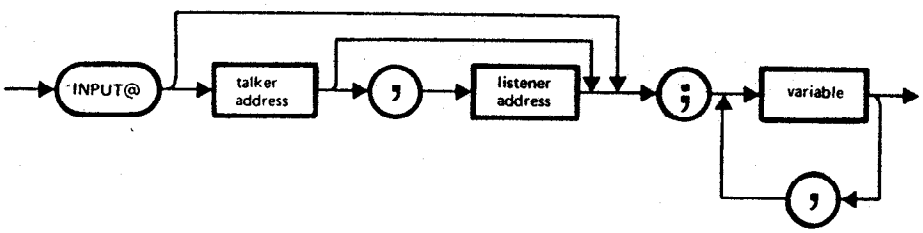


Figure 6-8 INPUT@ Statement

Where

SYNTAX ELEMENT	MEANING
talker address	the hex code for the talker address. The talker flag, hex 4Ø, is automatically added to this before transmission.
listener address	the hex code for the listener address. The listener flag, hex 2Ø, is automatically added to this before transmission.
variable	the name of either a string or a numeric variable into which a value read from the bus will be stored. Data must be separated by commas and be of the same type as the variable. END must be sent true with the last byte.

Example

DISPLAY	COMMENTS
1Ø INPUT@ 3,6;N1,N2,N3 2Ø PRINT "received = ";N1,N2,N3	talker number 3 is addressed to talk, listener number 6 is addressed to listen. The controller (M20) will assign the data transmitted by the talker to the variables N1, N2, and N3. The listener with address number 6 will also receive the information. Finally, the data will be displayed on the screen.

If the three values sent to the M20 are 75, 26 and 14 respectively, then the following bytes will be sent across the interface:

BYTE TRANSMITTED	RECEIVED	ATN	COMMENTS
43	-	T	device address hex 3, plus hex 4Ø to indicate its talker status, is hex 43.
26	-	T	device address hex 6, plus hex 2Ø to indicate listener status, is hex 26
-	2Ø	F	hex 2Ø is ASCII code for space
-	37	F	37 7
-	35	F	35 5
-	2Ø	F	2Ø space
-	2C	F	2C ,
-	2Ø	F	2Ø space
-	32	F	32 2
-	36	F	36 6
-	2Ø	F	2Ø space
-	2C	F	2C ,
-	2Ø	F	2Ø space
-	31	F	31 1
-	34	F	34 4
			The last byte is sent with EOI true, to indicate the end of the data stream.
received = 75, 26, 14			will then be displayed on the screen.

Table 6-5

Remark

The values hex 2Ø or hex 4Ø (the listener and talker flags) are added automatically, and do not need to be added by the user.

LINE INPUT@ (PROGRAM/IMMEDIATE)

Outputs addresses with ATN true and then inputs a line of data and assigns it to a string variable (with ATN false). END must be sent true with the last byte of data. The driver ignores a single CR, LF at the end of each line.

When 255 characters have been received, no other character can be input (as the maximum length of a string is 255 characters, see also the LINE INPUT statement).

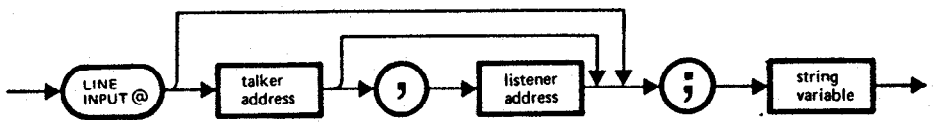


Figure 6-9 LINE INPUT@ Statement

Where

SYNTAX ELEMENT	MEANING
talker address	the hex code for the talker address. Hex 4Ø is automatically added to this before transmission.
listener address	the hex code for the listener address. Hex 2Ø is automatically added to this before transmission.
string variable	the variable name to which the string input is to be assigned.

Example

DISPLAY	COMMENTS
1Ø LINE INPUT@ 3,6;N$ 2Ø PRINT "received = ";N$	Statement 1Ø outputs the address 3 as a talker, and address 6 a listener. It then receives one line of data input and assigns it to the string variable N$. The listener 6 also receives the output from the talker. Statement 2Ø displays the value of the N$ variable.

If the data input in response to the program above was "IEEE TEST", the following bytes would be sent across the interface:

BYTE TRANSMITTED	BYTE RECEIVED	ATN	COMMENTS
43	-	T	device address 3, plus the talker address 4Ø is 43.
26	-	T	device address 6, plus the listener flag 2Ø, is 26.
-	49	F	hex 49 is ASCII code for I
-	45	F	45 E
-	45	F	45 E
-	45	F	45 E
-	2Ø	F	2Ø space
-	54	F	54 T
-	45	F	45 E
-	53	F	53 S
-	54	F	54 T this last being sent with EOI true, to indicate the end of the message.
received = IEEE TEST			will then be displayed on the screen.

Table 6-6

Remarks

The values hex 2Ø and hex 4Ø (the listener and talker address flags) are set automatically and do not need to be added by the user.

7. IEEE 488 SAMPLE PROGRAMS

ABOUT THIS CHAPTER

This chapter illustrates, with the use of example programs, two typical uses for this interface. These are followed by an example showing how these particular devices might be used together, and how IEEE 488 can be used to run such a system.

CONTENTS

IEEE 488 SAMPLE PROGRAMS

USING THE INTERFACE

IEEE 488 is used for many different purposes, for example:

CREATING TEST/CONTROL EQUIPMENT

In this case, the IEEE 488 is used to connect together otherwise independent pieces of (test) equipment. Limitations on the maximum length of cabling allowed necessitate that these devices are located close to one another, usually on the same test-bench. If distance between devices is a problem, it can be got round in several ways, including, for example, linking the test equipment to the controller via an RS-232-C line.

By providing such a connection, the separate devices are linked together, each able to communicate with the other, and so can be thought of as one unit. In addition, one of these devices can be a processor, (e.g. the M20) which can run these peripherals instead of an engineer having to operate each in turn. The final stage in this process is to program the processor, so that little or no human intervention is required.

To take an example, suppose that a particular circuit-board needs to be tested. The same test may need to be carried out hundreds, or even thousands, of times. This repetition may be carried out thousands of times on the same board, in a QA or experimental environment; or once, on thousands of boards, in a production environment.

With a set up as described above, instead of an engineer connecting up each device manually, adjusting the settings and then taking the readings, a technician can do the job. With a suitable program on the machine, all that would be required of such a person is to read the results from the screen of the processor.

AUTOMATION

A common area for the use of IEEE is in industrial automation. By having detectors and the devices which change the environment (thermometers and air-conditioners, tachometers and generator power supply, to take two simple examples) as closely linked as IEEE enables them to be, the machines can run without manual intervention. Service ReQuest (SRQ) should be suitably programmed to warn of conditions outside the control of the automatic system, so that manual control may be re-established.

Such a system not only alleviates the necessity for constant vigilance, and the human errors that inevitably result, but even more importantly, allows a far greater degree of accuracy. Conditions controlled by such a system can be tested and adjusted thousands of times a second if required.

IEEE 488 AND COMPUTERS

Because of the three-wire handshake, the interface is able to transfer data at very high speeds and with a great deal of accuracy - because the listener governs the speed, it can always keep up. For this reason, IEEE is very common as a connector between main-frame machines and peripheral equipment. The interface can thus cope with the differences in speed that a disc drive and a printer, for example, require their data to be transmitted.

SOME PRACTICAL EXAMPLES

The rest of this chapter describes two very simple, but useful, applications of the IEEE interface. The first example describes the use of a small program to write data to a plotter, and have it draw such information. The second describes the use of a voltmeter in connection with the M20. These examples are followed by a third, more complex, example which combines these two, with a program that plots out the data read from the voltmeter.

The PCOS command IEEE 488 must be PLOADed before any of these programs can be run.

The examples are described and three sample programs which could perform each of these applications are worked through.

USING A PLOTTER

The following example requires the plotter to draw a circle and to label it "OLIVETTI M20". The program described uses a specific plotter, and some of the data sent to it as instructions may therefore be different for another machine. What these instructions are, and how they are interpreted, is always made clear by use of comments within the program itself.

IEEE 488 SAMPLE PROGRAMS

This example shows how data can be transmitted to a peripheral device, using the interface. However, it is essential that the programmer is familiar with the peripheral itself, as well as with the IEEE programming statements, if the device is to be able to respond to commands and detect the difference between commands and genuine data.

For example, line 7Ø of the following program sends the data "POLIVETTI M20" to the plotter. In the case of this plotter, the first character, P, is read as the command "Print" which indicates that all the following characters should be treated as data and printed as required. If this first character had been S for example, then the following characters would have been included in the command, telling the plotter the size of printing required. This situation is illustrated in line 5Ø.

Figure 7-1 shows an example program to carry out the requirements above.

```
Ø5 REM PROGRAM SENDING DATA TO A PLOTTER
1Ø ISET REN
2Ø PA=6 'plotter address
3Ø XY$="1ØØØ,5Ø"
4Ø PRINT@ PA;"M"+XY$
5Ø PRINT@ PA;"S15"
6Ø PRINT@ PA;"QØ"
7Ø PRINT@ PA;"POLIVETTI M2Ø"
8Ø R=1ØØØ
9Ø XY$=STR$(18ØØ+R*COS(Ø))+","+STR$(13ØØ+R*SIN(Ø))
1ØØ PRINT@ PA;"M"+XY$ 'move to first point on circle w/pen up.
11Ø FOR X=Ø TO 6.3 STEP .1
12Ø XY$=STR$(18ØØ+INT(R*COS(X)))+","+STR$(13ØØ+INT(R*SIN(X)))
13Ø PRINT@ PA;"D"+XY$
14Ø NEXT
15Ø PRINT@ PA;"H" 'home w/pen up.
```

Figure 7-1 Example Program Sending Data to a Plotter

The Plotter Example

ISET REN in line 1Ø sets all devices on the interface to Remote ENable.

Line 2Ø sets the plotter address (in this case 6) to the variable PA.

Line 3Ø sets the variable XY$ equal to "1ØØØ,5Ø". These are the co-ordinates of a point, that will be sent to the plotter as data.

Line 4Ø asks the plotter to receive the data M, 1ØØØ, comma and 5Ø. This specific plotter interprets this as "Move the pen to the absolute co-ordinate "1ØØØ, 5Ø" with the pen up". The next data to be sent to the plotter is "S15", which means that any text is to be printed at 15 times the basic size. "QØ" then specifies the orientation to print in and finally the string "POLIVETTI M20" is sent, which tells the plotter to print OLIVETTI M20 (in orientation Ø and character size 15).

Lines 8Ø to 14Ø look more complicated, but in fact the complication is only the mathematics required to draw the circle.

Line 9Ø shows that XY$ is to become a string containing: the value 18ØØ + R * COS(Ø), (where R has been set in line 8Ø to 1ØØØ), a comma, then another string value, this time 13ØØ + R * SIN (Ø). Thus XY$ becomes equal to X,Y as it did in line 3Ø, but now the co-ordinates specify the first point of the circle.

Line 12Ø is exactly the same, except that the trigonometric functions have Ø replaced by X, where X is set by the loop at lines 11Ø and 14Ø.

Line 11Ø shows that the X parameter will change 63 times, so the circle will be made up of 64 straight line connections, joining these points.

Line 13Ø sends the data "D" and XY$. This means draw a straight line from where the pen is now to the co-ordinates specified in XY$. The final instruction, "H", tells the plotter to return the pen to the Home position, with the pen up (not all plotters return with the pen up, a separate command is then required to prevent the pen from drawing across to the Home position).

Although these instructions may vary from machine to machine, the method of transmission of this data remains the same.

USING A VOLTMETER

This example requires the voltmeter to produce a voltage reading each time the SRQ is triggered. For demonstration purposes this could be done manually, using an external trigger, so that readings can be performed whenever necessary. These readings are displayed on the VDU. Figure 7-2 gives an example of a program that would carry out the requirements above. As in the previous example, some of the instructions issued may be specific to the peripheral used.

```
 Ø5 REM PROGRAM TRIGGERING AND READING VOLTMETER OUTPUT
 1Ø CLEAR
 2Ø DEFINT A-Z
 3Ø DVM = 1
 4Ø ISET REN
 5Ø WBYTE 33,4; 'Send SDC
 6Ø PRINT @ DVM; "F1R7M3T2D1AØ"
 7Ø ON SRQ GOSUB 1ØØØ
 8Ø PRINT "Waiting for srq"
 9Ø 'P=POS(1) 'Returns row of text cursor.
1ØØ CURSOR (1,2) : I=I+1 : PRINT I : GOTO 1ØØ
11Ø END
1ØØØ 'SRQ ROUTINE
1Ø1Ø PASS = PASS + 1
1Ø2Ø POLL DVM,STATUS
1Ø3Ø CURSOR (1,1Ø)
1Ø4Ø PRINT "STATUS=";STATUS,"PASS=";PASS
1Ø5Ø LINE INPUT@ DVM ; VOLTS$
1Ø6Ø PRINT "READING IS",VOLTS$
1Ø7Ø PRINT @ DVM;
1Ø8Ø ON SRQ GOSUB 1ØØØ : RETURN
```

Figure 7-2 Example Program Triggering and Reading Voltmeter Output

The Voltmeter Example

Line 1Ø is the BASIC statement which closes all previously open files. Line 2Ø then declares that all variables in the program starting with an alpha-character will be integer variables. Line 3Ø sets the first of these variables, DVM, to 1. This is a mnemonic for the Digital Voltmeter, and its address, 1. ISET REN sets all the device on the interface to Remote ENable (although for the purposes of this program the DVM is the only device on it).

The WBYTE statement at line 5Ø then sends Selected Device Clear to the DVM. The device address is 1, plus listen (32) adds to decimal 33, the first command. Following the comma, the second command, 4, is the code for SDC. (See Appendix C).

In order to send string information, the PRINT statement is used next. As in the previous example, the data is recognised and interpreted by the DVM. The string "F1R7M3T2D1AØ" is interpreted by this specific voltmeter as:

- F1 DC Volts

- R7 Auto range

- M3 Maths off

- T2 Trigger external

- D1 Data ready RQS (sets SRQ on valid data)

- AØ Auto calibration off.

Line 7Ø then sets up the GOSUB statement to jump to the routine which will take the voltmeter reading each time there is a service request.

Line 1ØØ provides a waiting loop, which counts from 1 onwards until the SRQ is triggered. By repositioning the cursor each time, each number overwrites its predecessor. The end of the program proper follows, then the service request subroutine starts at line 1ØØØ.

After the title, the first line merely counts the number of times the subroutine has been called. Line 1Ø2Ø then POLLs the DVM for its STATUS. The voltmeter outputs this status information which is printed, with the pass number, at line 1Ø4Ø. Line 1Ø3Ø has meanwhile repositioned the cursor so that it does not overwrite the counter.

Line 1Ø5Ø addresses the DVM to output its current reading to the controller (M2O). This data is displayed on the VDU by the PRINT@statement at line 1Ø6Ø. Line 1Ø7Ø is a dummy PRINT@ statement which causes the DVM to become a listener again, which it needs to be to listen for the next SRQ trigger.

Line 1Ø8Ø resets the GOSUB statement then, in the same line, returns control to the main body of the program. By having these two statements on the same line, the interrupt routine is completed before another can be acknowledged.

USING A VOLTMETER AND A PLOTTER

This example uses the DVM to take readings of the mains voltage and plot each of the readings graphically, using the plotter. The example is more complicated because of the interaction between the two instruments.

```
  Ø REM PROGRAM USING TWO PERIPHERALS
  1 TIME = 1ØØØ
  5 ON ERROR GOTO 9ØØØ
  7 CALL "pl ie"
 1Ø XMIN = -3Ø
 2Ø XMAX = 14Ø
 3Ø INPUT "Enter your A.C. line voltage: ";ACV
 4Ø YMAX = .2 * ACV
 5Ø YMIN = -YMAX
 6Ø MAXY = YMAX / 2
 7Ø MINY = -MAXY
 8Ø NUMTIC = MAXY - MINY
 9Ø TICINC = INT(13ØØ/NUMTIC)
1ØØ PLOTTER = 6
1Ø5 'Functions FNX$ & FNY$ return plotter units in string form that is
    suitable for sending to plotter.
11Ø DEF FNX$(X) = STR$(INT((X-XMIN)*36ØØ/(XMAX-XMIN)))
12Ø DEF FNY$(Y) = STR$(INT((Y-YMIN)*26ØØ/(YMAX-YMIN)))
13Ø C$ = "Move" : X = Ø : Y = MINY : GOSUB 1ØØØ
14Ø GOSUB 2ØØØ
17Ø C$ = "Move" : X = Ø : Y = Ø : GOSUB 1ØØØ
175 XØ% = Ø : X1% = 1ØØ
18Ø XØ$ = FNX$(XØ%)
182 X1$ = FNX$(X1%)
184 TICINC = INT((X1%-XØ%)/1Ø)
186 PRINT PLOTTER;"X1," +STR$(TICINC) + ",1Ø"
188 GOSUB 21ØØ
2ØØ DVM = 1
21Ø ISET REN
22Ø WBYTE 63,95;
23Ø PRINT@ DVM;"F2R7T2T3"
24Ø WBYTE 33,8;
25Ø INPUT@ DVM;ACV1
255 WBYTE 63; 'UNL
26Ø Y = ACV - ACV1
27Ø C$ = "Move": X = Ø : GOSUB 1ØØØ
28Ø FOR X = 1 TO 1ØØ
285 C$ = "D"
29Ø WBYTE 33,8; 'GET
3ØØ INPUT@ DVM;ACV1
31Ø Y = ACV - ACV1
32Ø GOSUB 1ØØØ
322 C$ = "Move" : GOSUB 1ØØØ
325 FOR K = 1 TO TIME : NEXT K
33Ø NEXT X
34Ø PRINT@ PLOTTER;"H"
35Ø END
1ØØØ 'Sub plot
1Ø1Ø CODE$ = LEFT$(C$,1) 'Get rid of all but the first character.
1Ø2Ø IF CODE$ = "H" THEN PLOT$ = CODE$ : GOTO 1Ø7Ø 'Home needs no
     parameters
```

Figure 7-3 Example Program Using Two Peripherals (cont.)

```
1Ø25 'X$ & Y$ are the plotter units in string form while X% & Y% are the
     same values in numeric form.
1Ø3Ø X$ = FNX$(X)
1Ø4Ø Y$ = FNY$(Y)
1Ø5Ø PLOT$ = CODE$ + X$ +"," + Y$ 'Build string to send to plotter.
1Ø7Ø PRINT@ PLOTTER ; PLOT$ 'Send string to plotter.
1Ø75 WBYTE 63; 'UNL
1Ø8Ø RETURN
2ØØØ 'Sub Y-axis
2ØØ5 PRINT@ PLOTTER;"XØ" + "," + STR$(TICINC) + "," + STR$(NUMTIC)
2Ø1Ø C$ = "Move" : X =-4
2Ø2Ø PRINT@ 6,"S2" 'Small character size
2Ø3Ø FOR Y = MAXY TO MINY STEP-1
2Ø4Ø GOSUB 1ØØØ
2Ø5Ø PRINT@ PLOTTER; "P" + STR$(Y)
2Ø6Ø NEXT Y
2Ø65 WBYTE 63; 'UNL
2Ø7Ø RETURN
21ØØ 'Sub Xaxis
21Ø5 XT = 1ØØ
211Ø C$ = "Move" : Y =-1
212Ø FOR X = 98 TO 8 STEP-1Ø
213Ø GOSUB 1ØØØ
214Ø PRINT@ PLOTTER;"P" + STR$(XT)
2145 XT = XT-1Ø
215Ø NEXT X
216Ø RETURN
9ØØØ 'Error recovery
9Ø1Ø RESUME NEXT
```

Figure 7-3 Example Program Using Two Peripherals

Line 1 sets the time interval between the readings. If this is to be a long period (more than 1Ø seconds) the program should be amended to lift the pen after each point is plotted, so that the pen does not leak onto the paper. This is easily done using a dummy Move statement, causing the pen to move to the co-ordinates it is already at (so it does not move) but the pen is lifted until the next Draw statement. Inserting line 322 as C$ = "Move" : GOSUB 1ØØØ would perform this operation.

Line 5 prevents the program being halted by an error - the error routine causes the program to go on to the next plot. Line 7 calls up and loads the IEEE 488 package, using the PCOS command IE.

Lines 1Ø and 2Ø define the upper and lower limits of the X axis, and 3Ø allows the user to input the line voltage which will be used. Lines 4Ø and 5Ø define the limits of the Y axis as plus or minus 2Ø% of the line voltage input, while 6Ø and 7Ø define the limits that the axis will be plotted to. Thus the scale will cover the middle half of the paper, but

the actual graph can be plotted to the edges (given a 2Ø% fluctuation in voltage).

Lines 8Ø and 9Ø set the number of increments and their interval for the Y axis (13ØØ is the number of units used internally by the plotter). As before, a mnemonic (PLOTTER) is assigned the device address (6) at line 1ØØ.

The next 3 lines, as 1Ø5 describes, return the plotter units in string form, which is how they will be sent to the plotter.

Line 13Ø sets the pen up to plot the Y axis, starting at X=Ø as the lowest value of Y. To transmit this data to the plotter, the subroutine at line 1ØØØ is used.

Subroutine 1ØØØ: Sending Co-ordinates to the Plotter

Line 1Ø1Ø strips CODE$ of all but the first character. Lines 1Ø3Ø and 1Ø4Ø assign the values of X and Y to plotter units, then 1Ø5Ø strings the 3 values into PLOT$. The PRINT@ statement then sends this string to the plotter.

If the letter left in CODE$ is "H", the PLOT$ is assigned this in line 1Ø2Ø, and lines 1Ø3Ø, 1Ø4Ø and 1Ø5Ø are jumped over, as Home does not need the extra parameters X and Y.

Before returning to the main body of the program, the Universal UNListen is sent by line 1Ø75, so that the system is ready for further instructions.

Drawing the Y Axis

Returning to the main body of the program, another subroutine is called (line 14Ø). This subroutine, starting at line 2ØØØ actually draws the Y axis, and the increment numbers beside it. Line 2ØØ5 tells the plotter to plot the Y axis (XØ command for that specific plotter) with NUMTIC number of segments, drawn at TICINC intervals.

Line 2Ø1Ø then sets C$ to Move again (the XØ command used in 2ØØ5 leaves the pen down) and X is set to -4, so that the scale is drawn to the left of the Y axis.

The PRINT@ statement at line 2Ø2Ø sends the plotter (device 6) the data S2, which sets the character size (as in the first example). For

efficiency, since the Y axis was drawn from bottom to top, the scale is printed from top to bottom, so that the pen does not have to travel the length of the axis to start. Thus line 2Ø3Ø starts a loop, decrementing Y by 1 each time; beginning with the value MAXY and ending at MINY.

The string variable PLOT$ is built by subroutine 1ØØØ again. X is constant and already set at -4 (line 2Ø1Ø) and the loop varies Y to move the pen vertically down the page.

Line 2Ø5Ø, another PRINT statement, addresses the plotter with the data "P" and STR$(Y). This instructs the plotter to print the value Y.

Subroutine 1ØØØ then resets the pen to the next position where 2Ø5Ø prints the value, and so on. When this is completed, line 2Ø65 sends a Universal UNListen (thus including the voltmeter), then at line 2Ø7Ø control returns to the main program body.

Line 17Ø resets C$ to Move and X and Y to Ø. Subroutine 1ØØØ then carries this out, i.e. moves the pen to Ø,Ø.

Drawing the X Axis

Line 175 assigns XØ% and X1% variables.

Lines 18Ø and 182 set string variables XØ$ and X1$ to the minimum and maximum of X, in plotter units, so these can be sent to the plotter. TICINC is reset, this time to be the integer part of the value (X1%-XØ%)/1Ø. Thus there are to be 10 intervals along the X axis.

Line 186 is similar to 2ØØ5, but this time the X axis is to be drawn, with 1Ø ticks, at intervals of TICINC. Line 188 transfers control to subroutine 21ØØ which adds the scale to the X axis.

Line 21Ø5 sets the start point for the X co-ordinate, the 211Ø sets C$ to Move once more and Y to -1 (so that the legend will be below the axis). The loop is set up in line 212Ø, starting at 98 so that the numbers are directly under the tick marks.

As before, subroutine 1ØØØ then moves the pen to the co-ordinate required, with Y constant at -1 and X decreasing by 1Ø each time. Line 214Ø, as 2Ø5Ø, then instructs the plotter to print the new value for XT, which is altered each time round by line 2145. Line 215Ø completes the loop, and when this is finished 216Ø returns control to the main program once more.

Setting up the DVM

The plotter is now ready, and all that remains before the 'experiment' can run is to set up the voltmeter.

As on previous occasions, a mnemonic is used for the device address. Line 2ØØ sets DVM to 1. Line 21Ø Remote enables the system (so all devices can listen) then 22Ø performs a Universal UNListen (63) and UNTalk (95), so that the plotter will not listen to the proceeding instructions.

Line 23Ø sends the string "F2R7T2T3" to the DVM. This is interpreted as:

- F2 AC Volts
- R7 Auto range
- T2-T3 A special trigger mode to enable the DVM to respond to a GET signal.

In order that the first point on the graph is a 'real' point, not where the pen happens to be (normally Ø), the first GET is triggered outside the main experiment loop, at line 24Ø. This addresses the DVM as a listener (1+32) and then sends the GET command.

The GET (Group Execute Trigger) will be sent each time line 29Ø is reached.

Plotting the Graph

Line 25Ø accepts the input from the DVM and stores it as variable ACV1. Line 26Ø then computes the difference between the "mean" line voltage input at line 3Ø (ACV) and the measured voltage (ACV1) and assigns this value to the Y co-ordinate. Line 27Ø sets C$ to Move, X to Ø and diverts control to subroutine 1ØØØ which positions the pen at this point.

Line 28Ø starts the loop of the experiment. 1ØØ readings are to be taken and the loop variable is used to increment the X co-ordinate. Line 285 sets C$ to Draw, so the pen will map movement from now on.

Lines 29Ø, 3ØØ and 31Ø correspond exactly to 24Ø, 25Ø and 26Ø, triggering and accepting the data from the DVM. As usual, this information is used by subroutine 1ØØØ to reposition the pen (but this time the pen is drawing).

Line 325 runs the time interval set at line 1, then the process repeats until all 1ØØ points are plotted (1Ø1 including the first).

Finally, the plotter is sent the data "H" by the PRINT@ statement, which returns the pen to the Home position. This could equally have been done by C$=HOME : GOSUB 1ØØØ as the subroutine is prepared for such action in line 1Ø2Ø. Line 35Ø then terminates the program.

Figure 7-4 shows an example of the output from this program. How constant the graph is will depend on the time interval chosen, and the variability of the mains voltage in the area.

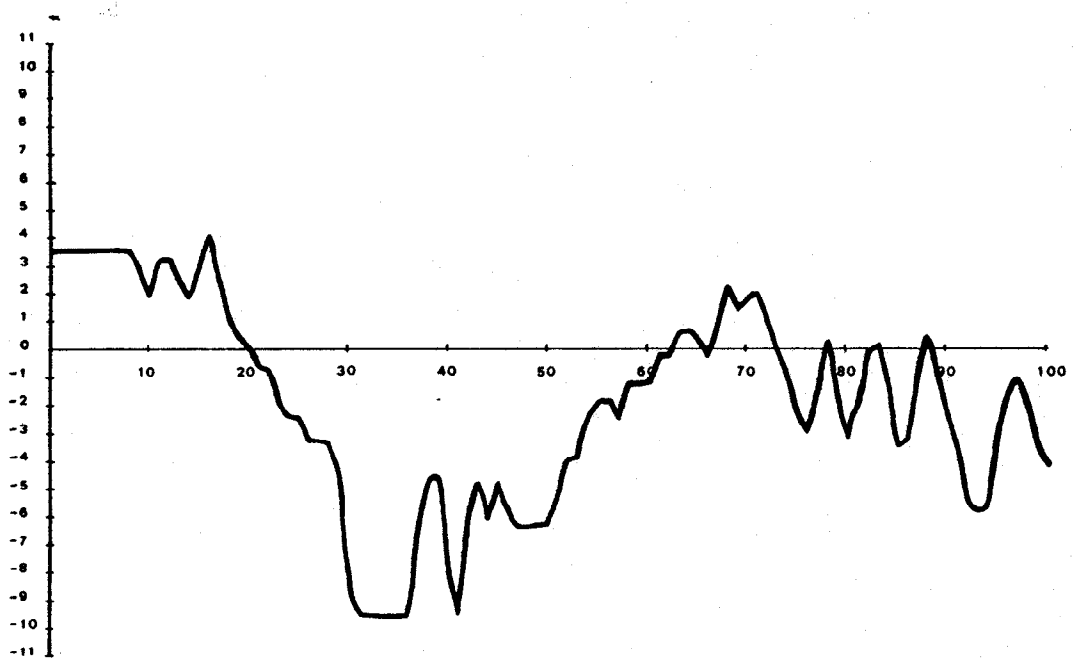


Figure 7-4 Plotting the Graph (Sample Output)

A. STANDARDS PUBLICATIONS

EIA STANDARD RS-232-C

Interface between Data Terminal Equipment and Data Communication Equipment employing serial binary data interchange, dated: August 1969

Published by: ELECTRONIC INDUSTRIES ASSOCIATION
Engineering Department
2ØØ1 Eye Street, NW
WASHINGTON, DC 2ØØØ6

CCITT V.24

This standard is entitled:

Recommendation V.24

LIST OF DEFINITIONS FOR INTERCHANGE CIRCUITS BETWEEN DATA-TERMINAL EQUIPMENT AND DATA CIRCUIT-TERMINATING EQUIPMENT

and forms part of Volume VIII.1 (i.e. Part 1 of Volume VIII) of:

CCITT SIXTH PLENARY ASSEMBLY
Geneva, 27 September - 8 October 1976

VOLUME VIII.1

DATA TRANSMISSION OVER THE TELEPHONE NETWORK, dated: 1977

Published by: INTERNATIONAL TELECOMMUNICATIONS UNION
GENEVA, Switzerland

ANSI/IEEE Std 488-1978

IEEE standard digital interface for programmable instrumentation

Published by: The Institute of Electrical and Electronic Engineers, Inc.
345 East 47th Street,
New York,
New York 1ØØ17

B. RS-232-C SIGNALS AND SETTINGS

ABOUT THIS APPENDIX

This appendix describes the RS-232-C interface in terms of its signals, definitions and functions as applied to a particular peripheral, or printer. As an aid to ensure correct functioning of the M20 RS-232-C interface, the jumper settings corresponding to port Ø (asynchronous usage) as well as ports 1 and 2 (asynchronous or Current Loop usage) are shown.

CONTENTS

RS-232-C SIGNALS AND SETTINGS

INTERFACE SIGNALS

The following table shows the signal descriptions by V.24 and RS-232-C name and circuit identifier; direction relative to source; M20 name and pin on connector.

INTERFACE SIGNALS

CCITT V.24		EIA RS-232-C		SIGNAL DIRECTION RELATIVE TO SOURCE*	SIGNAL NAME	PIN ON CONNECTOR
CIRCUIT	SIGNAL DESCRIPTION	CIRCUIT	SIGNAL DESCRIPTION			
1Ø1	Protective ground	AA	Protective ground	--	GROUND	1
1Ø2	Signal ground	AB	Signal ground	--	EARTH	7
1Ø3	Transmitted data	BA	Transmitted data	from	T1Ø3A	2
1Ø4	Received data	BB	Received data	to	R1Ø4A	3
1Ø5	Request to send	CA	Request to send	from	T1Ø5A	4
1Ø7	Data set ready	CC	Data set ready	to	R1Ø7A	6
1Ø8/2	Data terminal ready	CD	Data terminal ready	from	T1Ø8A	2Ø
1Ø9	Data channel received	CF	Received line signal detector	to	R1Ø9A	8
**	Reverse channel	**	Reverse channel	from	RECHØ	11
118	Transmitted Backwards	SBA	Secondary transmitted data	from	T118A	14

INTERFACE SIGNALS

CCITT V.24		EIA RS-232-C		SIGNAL DIRECTION RELATIVE TO SOURCE*	SIGNAL NAME	PIN ON CON-NECTOR
CIRCUIT	SIGNAL DE-SCRIPTION	CIRCUIT	SIGNAL DE-SCRIPTION			
12Ø	Transmit-ted channel line signal	SCA	Secondary request to send	from	T12ØA	19

NOTES: * M20 on MODEM cable and PERIPHERAL on PERIPHERAL cable

** Circuits not present on both standards.

ELECTRICAL LEVELS AND LOGIC REPRESENTATION

VOLTAGE LEVELS ON LINE

	NEGATIVE (V=-3 MIN.)	POSITIVE (V=+3 MIN.)
Binary status	1	Ø
Signal	MARK	SPACE
Function	OFF (disabled)	ON (enabled)

The voltage levels are measured at the line connector pins.

The max voltage level for MARK must be -25V, while for SPACE it must be +25V.

The time which elapses for signal excursion between the voltage levels +3V and -3V must not exceed 3% of the bit nominal time.

INTERFACE SIGNAL DESCRIPTION

The following table shows how these signals are used with a printer.

CCITT CIRCUIT IDENTIF.	PIN	SOURCE	NAME	DESCRIPTION
1Ø	1	--	Protective ground	To be connected to the machine cabinet
1Ø2	7	--	Signal ground	Logic ground. It must be connectable to circuit 1Ø1 through a removable jumper
1Ø3	2	Printer	Transmitted data	This circuit can be used to signal the printer status to the driving host system. Normally it is in MARK condition (-V) and goes to SPACE condition (+V) when the printer is "Busy" or in "anomaly" status (e.g.: paper-out, out of service). This circuit can be used as an alternative to circuit 118 (Secondary transmitted data) or to Reverse Channel circuit
1Ø4	3	Host system	Received data	During the intercharacter time intervals, and when circuit 1Ø9 is OFF (-V) this signal is maintained at MARK level (-V)
1Ø5	4	Printer	Request to send	It is maintained at ON level (+V) when circuit 1Ø3 is used. Otherwise it is maintained at OFF level (-V)
1Ø7	6	Host system	Data set ready	The ON level (+V), indicates that the host system is ready to transfer data

CCITT CIRCUIT IDENTIF.	PIN	SOURCE	NAME	DESCRIPTION
1Ø8/2 minal	2Ø	Printer	Data ter-	The ON level (+V), indicates that the printer is ready to receive data. It goes OFF (-V) when "anomaly" or "local" conditions occur
1Ø9	8	Host system	Received line signal detector	The ON level (+V), indicates that data are present on circuit 1Ø4 (if activated)
118	14	Printer	Secondary transmitted data	It is used alternatively to circuit 1Ø3 or Reverse channel circuit, to signal the "Busy" or "anomaly" status to the host
12Ø	19	Printer	Secondary request to send	Permanently maintained at ON level (+V)
--	11	Printer	Reverse channel	It is used alternatively to circuits 1Ø3 and 118 transmit the "Busy" or "anomaly" status to the host

JUMPER SETTINGS

MOTHERBOARD

For the built-in RS-232-C Interface, the user should check that the following jumpers are in position for port Ø:

Y1	to	Y24
Y3	to	Y22
Y5	to	Y20
Y7	to	Y18
Y9	to	Y16
Y11	to	Y14
P1	to	P
N2	to	P2

MINIBOARD

For the extension RS-232-C Interface, the user should check that the following jumpers are in position:

ALWAYS

1. B to B

   I to Z

AND

2. FOR PORT 1:

   1 to G
   2 to R
   2 to S

   AND FOR PORT 2:

   A to A
   2 to W
   2 to Y

EITHER

FOR ASYNCHRONOUS USE

3. FOR PORT 1

   1 to M

   AND FOR PORT 2

   1 to T
   1 to U

OR

FOR 2Cma CURRENT LOOP USAGE

4. FOR PORT 1

   3 to M

AND

TRANSMIT				OR	RECEIVE			
ACTIVE			PASSIVE		ACTIVE			PASSIVE
J	to	K	J to L		A	to	P	A to C
L	to	M	M to N		D	to	c	D to E
N	to	P			E	to	F	

FOR PORT 2

					2	to	T	
					2	to	U	

AND

TRANSMIT				OR	RECEIVE			
ACTIVE			PASSIVE		ACTIVE			PASSIVE
7	to	8	7 to 9		1	to	2	1 to 3
9	to	10	10 to 11		3	to	4	4 to 5
11	to	12			5	to	6	

C. IEEE 488 CHARACTER CODES

(SENT AND RECEIVED WITH ATN = 1)

Bits b_6 b_5 b_4 / b_3 b_2 b_1 b_0 ②		0 0 0	① MSG	0 0 1	MSG	0 1 0	MSG	0 1 1	MSG	1 0 0	MSG	1 0 1	MSG
1 1 1 1		0		1		2		3		4		5	
0 0 0 0	0	NUL		DLE		SP		0		@		P	
0 0 0 1	1	SOH	GTL	DC1	LLO	!		1		A		Q	
0 0 1 0	2	STX		DC2		"		2		B		R	
0 0 1 1	3	ETX		DC3		#		3		C		S	
0 1 0 0	4	EOT	SDC	DC4	DCL	$		4		D		T	
0 1 0 1	5	ENQ	PPC③	NAK	PPU③	%		5		E		U	
0 1 1 0	6	ACK		SYN		&		6		F		V	
0 1 1 1	7	BEL		ETB				7		G		W	
1 0 0 0	8	BS	GET	CAN	SPE	(		8		H		X	
1 0 0 1	9	HT	TCT③	EM	SPD	)		9		I		Y	
1 0 1 0	10	LF		SUB		*		:		J		Z	
1 0 1 1	11	VT		ESC		+		;		K		[	
1 1 0 0	12	FF		FS		,		<		L		\	
1 1 0 1	13	CR		GS		–		=		M		]	
1 1 1 0	14	SO		RS		.		>		N		^	
1 1 1 1	15	SI		US		/		?	UNL	O		—	UNT

MLA ASSIGNED TO DEVICE

MLA ASSIGNED TO DEVICE

MTA ASSIGNED TO DEVICE

MTA ASSIGNED TO DEVICE

ADDRESSED COMMAND GROUP (ACG)

UNIVERSAL COMMAND GROUP (UCG)

LISTEN ADDRESS GROUP (LAG)

TALK ADDRESS GROUP (TAG)

PRIMARY COMMAND GROUP (PCG)

NOTES:
① MSG = INTERFACE MESSAGE
② b_ϕ = DI01 . . . b_6 = DI07
③ NOT AVAILABLE ON THE M20

Table C-1

D. IEEE 488 BASIC ERROR CODES

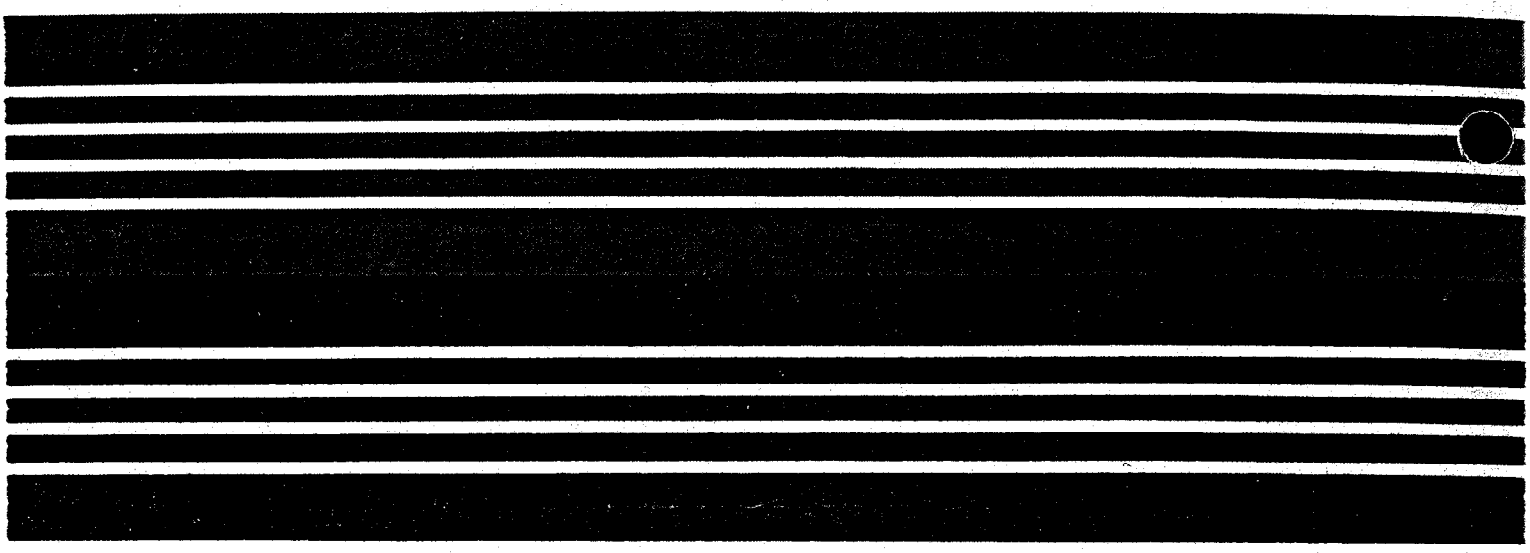
ERROR CODE	DESCRIPTION AS OUTPUT	COMMENTS
31	IEEE invalid Talker/ Listener address	use of illegal talker listener address
32	IEEE: Talker = Listener address	an attempt has been made to talk to a talker, or listen to a listener
33	IEEE: Unprintable error	an error message is not printable i.e. corresponds to an error with an undefined error code
34	IEEE: Board not present	an attempt has been made to use IEEE on a machine which does not have the optional IEEE board

NOTICE

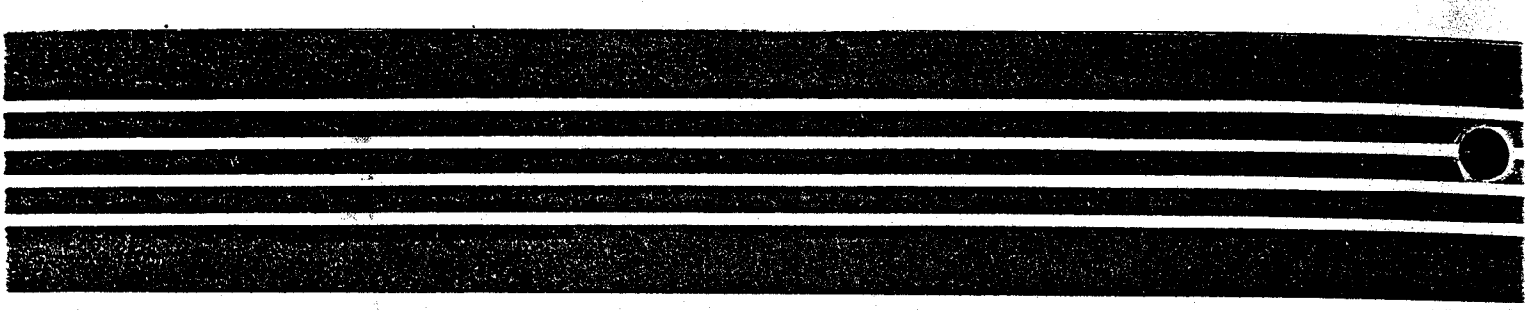
Ing. C. Olivetti & C. S.p.A. reserves the right to make improvements in the product described in this manual at any time and without notice.

This material was prepared for the benefit of Olivetti customers. It is recommended that the package be test run before actual use.

Anything in the standard form of the Olivetti Sales Contract to the contrary not withstanding, all software being licensed to Customer is licensed "as is". THERE ARE NO WARRANTIES EXPRESS OR IMPLIED INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTY OF FITNESS FOR PURPOSE AND OLIVETTI SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL OR INCIDENTAL DAMAGES IN CONNECTION WITH SUCH SOFTWARE.



GU Code 3982300 N (0)
Printed in Italy



GU Code 3982300 N (0)
Printed in Italy