850

THE ATARI® 400/800™
COMPUTER SYSTEM

INTERFACE
MODULE

TECHNICAL MANUAL

ATARI

ATARI®

ATARI® 850™
INTERFACE MODULE
TECHNICAL MANUAL

ATARI®

A Warner Communications Company

Important Information: Like any electrical appliance, this ATARI Home Computer equipment uses and produces radio frequency energy. If it's not installed and used properly according to the instructions in this guide, the equipment may cause interference with your radio and television reception.

It has been type tested and found to comply with the limits for a Class B computing device in accordance with the specifications in Subpart J of Part 15 of the FCC rules. These rules are designed to provide reasonable protection against such interference when the equipment is used in a residential setting. However, there is no guarantee that interference will not occur in a particular home or residence.

If you believe this equipment is causing interference with your television reception, try turning the equipment off and on. If the interference problem stops when the equipment is turned off, then the equipment is probably causing the interference. With the equipment turned on, you may be able to correct the problem by trying one or more of the following measures:

- Reorient the radio or television antenna.
- Reposition the equipment in relation to the radio or television set.
- Move the equipment away from the radio or television.
- Plug the equipment into a different wall outlet so the equipment and the radio or television are on different branch circuits.

If necessary, consult your ATARI Computer retailer or an experienced radio-television technician for additional suggestions.

CONTENTS

APPENDICES

INDEX

ILLUSTRATIONS

TABLES

WHAT IS RS-232-C?

RS-232-C is a technical standard of the Electronic Industries Association (EIA). Published in August of 1969, it is titled "Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange." The standard specifies electrical signal characteristics and names and defines the functions of the signal and control lines that make up a standard interface called RS-232-C.

Figure 1-1 shows, diagrammatically, the kind of hook-up that RS-232-C was designed to standardize. A **data terminal** is at each end of the communication link. The data terminal either generates or receives data (or does both). It could be a keyboard/screen "terminal" in the normal sense of the word; it could be a computer; etc. The idea is that the data terminal is at the end of the communication link – hence it is called "terminal." However, the data terminal need not really be at the end – you can think of "data terminal" as just the name of one of the two ends of a RS-232-C connection.

At the other end of a RS-232-C connection is the **data set**. In the example of Figure 1-1, each data set takes data from the data terminal it is connected to and sends/receives the data over the communications link. The most familiar example of a data set is the modem (such as the ATARI 830 Acoustic Modem), which takes data from a terminal and converts it for sending and receiving over a telephone line.

The ATARI Home Computer with the interface module should be thought of as a unit comprising a RS-232-C data terminal.

A full-duplex connection is one where data can be sent and received by both ends of the RS-232-C connection simultaneously. In a half-duplex connection, data cannot be sent by both ends at the same time. Therefore, one terminal must be able to tell the other terminal, "I'm through now; it's your turn." The second terminal then sends a signal saying, "OK, here I come." This exchange is called "handshaking"; it is simply the sending and receiving of required signals to prepare each end of the connection for the sending or receiving of data. Handshaking can be used in full-duplex operation to tell the sending terminal to stop sending until the receiving terminal can catch up.

The data-set/data-terminal distinction should be kept in mind because the RS-232-C interface is **directional**. That is, each line in a RS-232-C interface has a direction — one device drives the line (sends information) and the other receives the information. Each line in an RS-232-C interface is defined as being driven by either the data-set end or the data-terminal end.

DATA TERMINAL (TERMINAL)	DATA SET (MODEM)	DATA SET (MODEM)	DATA TERMINAL (COMPUTER)
RS-232-C CONNECTION	DATA COMMUNICATION LINK (TELEPHONE)	RS-232-C CONNECTION	
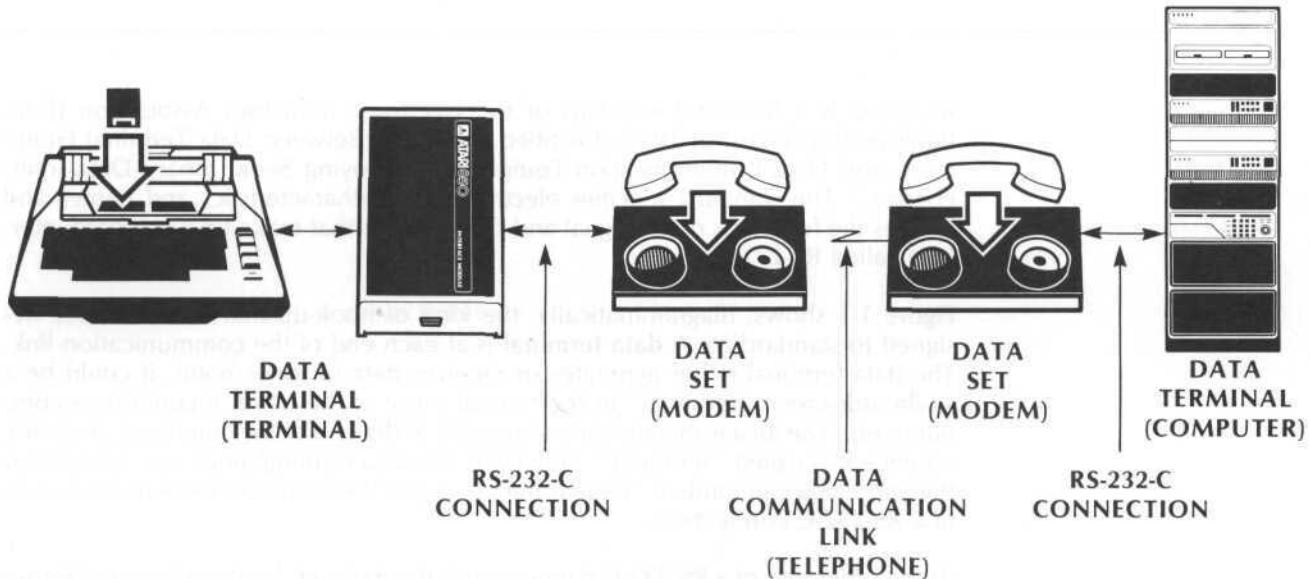
Figure 1-1 Communications Hook-Up Showing Role of RS-232-C

The RS-232-C standard defines some 20 signaling lines, or "circuits" as the standard refers to them. Most of them are optional and rarely used. Even with many omissions and deviations from the standard, a link may still be referred to as RS-232-C. It is more common to refer to the link loosely as "RS-232" or "RS-232 compatible."

The most commonly used RS-232-C lines are listed in Table 1-1. The table shows the name of each line in the RS-232-C standard and commonly used mnemonics.

Table 1-1 The Most Common RS-232-C Circuits

LINE (CIRCUIT) NAME	DIRECTION	DESCRIPTION	ABBREVIATION
BA	terminal→set	Transmitted data	XMT
BB	terminal←set	Received data	RCV
CA	terminal→set	Request to send	RTS
CB	terminal←set	Clear to send	CTS
CC	terminal←set	Data set ready	DSR
AB	(none)	Signal ground	
CF	terminal←set	Signal (carrier) detect	CRX
CD	terminal→set	Data terminal ready	DTR

It is common practice to use common names or abbreviations for the RS-232-C signals, and not the two-letter names in the official standard.

Transmit **(XMT)** and receive **(RCV)**, for any given device, is usually *relative to that device*. That is, data goes out of a device on XMT and comes in on RCV. To connect two RS-232-C devices when given the common names of the signals, you should connect XMT to RCV (in one direction) and RCV to XMT (in the other direction). If one of the devices is wired as a data set and the other as a data terminal, then you should connect DTR to DTR, DSR to DSR, RTS to RTS, and so on. If, on the other hand, they are each wired as data terminals, you should be careful how things are connected.

The **Signal Ground** connection must always be made. RS-232-C requires that the ground potential of the two devices be equal. That is, their grounds must be connected together. Devices for which this requirement cannot be met cannot be connected via a RS-232-C interface.

Data terminal ready **(DTR)** is used by RS-232-C to allow the terminal to signal its readiness to send or receive data. This is a signal to automatic answering modems that they have permission to answer the ringing of the telephone line.

Data set ready **(DSR)** is used by the data set to signal its readiness to send or receive data. This indicates that communications are established.

Request to send **(RTS)** is used by the data terminal to tell the data set it wishes to send data. Some modems (Bell 202 for example) require this line to switch directions.

Clear to send **(CTS)** allows the data set to signal its readiness to pass data from the data terminal.

The carrier detect **(CRX)** line allows the data set to tell the data terminal that the communication link is established. This often differs little from DSR, except that DSR usually refers to "telephone off the hook" (answered), whereas CRX means something like "I hear the modem at the other end and we can talk now." When CRX goes OFF, data set ready OFF usually follows a few seconds later, indicating that the other end has "hung up."

In normal operation, DTR, DSR, and CRX are all ON. For full-duplex operation RTS and CTS are also both ON. However, it is often unnecessary to have all these lines ON — either one or the other devices on the RS-232-C connection does not have all the lines, or it is all right to ignore them (one of the properties of the RS-232-C standard is that not all of it needs to be implemented — it's perfectly all right to leave parts out). To operate the ATARI 830 Acoustic Modem, for instance, none of the control lines need to be used. In fact, the ATARI 830 Acoustic Modem ignores DTR and RTS, and it turns DSR, CTS, and CRX on and off together (with carrier).

Note that the communication link shown in Figure 1-1 is not defined by RS-232-C. In particular, this link seldom has more than the "equivalent" of XMT and RCV — that is, only data lines and no control. However, as often as not this link is a full-duplex link, so data can go both ways simultaneously. ASCII characters are the most common data sent, so the data sent each way can be either "control" data or "data" data.

With full-duplex operation, two devices can handshake with data in various ways. Common terminals usually do not have an internal connection between the keyboard and display (or they have a switch, usually called half/full duplex, to make or break this internal connection) so when talking with a computer in full-duplex mode (the most common mode), the computer at the other end "echoes" (sends back) each character to be displayed as it is typed. This allows you to see exactly what the computer at the other end receives. It also allows the computer at the other end to decide NOT to let you see what you have typed, as in "suppressing" the echo of a password.

In half-duplex operation, somewhere along the communications path data may pass in only one direction at a time. Not all parts of the communications path need be half duplex, but if any part is, then the whole system will probably have to send data only one way at a time. In half-duplex mode, the computer at the other end does not echo back what you type. In this case, in order to see what you type, the connection from keyboard to screen must be set locally; that is, set your terminal to "half duplex."

Note: The ATARI TeleLink™ I cartridge does not have the equivalent of a half/full switch. However, the ATARI 830 Acoustic Modem does have such a switch, and when it is placed in the half-duplex position, it echoes any data sent out over the phone back to the computer console.

A common handshake that requires full duplex is the XOFF/XON (transmit off/transmit on) handshake. The receiver of data can send XOFF to the sender to ask the sender to pause the data transmission and XON to resume. This allows the user of a screen terminal to stop the data so he can read the screen, and it allows a computer that is receiving data from another computer to effectively control the rate at which it can accept data.

ATARI BASIC AND THE ATARI 850
INTERFACE MODULE

This section will show you an example of how the interface module is programmed. Using ATARI BASIC, two ATARI Home Computer Systems will be programmed to let one user talk to another user through ATARI 830 Acoustic Modems.

Let's start simply — just a program to send a message line, then receive a line, and so on. The main part of such a program might be:

```
100 INPUT MESSAGE$
110 PRINT #5; MESSAGE$
120 INPUT #5, MESSAGE$
130 PRINT MESSAGE$
140 GOTO 100
```

Here, unit #5 is assumed to have been opened to the RS-232-C port attached to the modem. Thus, line 100 gets a line from your keyboard, line 110 sends it to the modem, line 120 gets a line from the modem, and 140 prints that on your television screen.

Here's the whole program:

```
10 DIM MESSAGE$(120)
20 OPEN #5, 13, 0, "R1:"
30 XIO 40, #5, 0, 0, "R1:"
100 INPUT MESSAGE$
110 PRINT #5; MESSAGE$
120 INPUT #5, MESSAGE$
130 PRINT MESSAGE$
140 GOTO 100
```

Line 10 allows space for the variable MESSAGE$ (used to both send and receive). We've assumed the modem is attached to port 1. Line 20 opens the RS-232-C port 1, allowing input and output, and enabling Concurrent Mode I/O. Concurrent Mode I/O is required for input (see Sections 4, 5, 6, 9, and 10, and Appendix B for more details about Concurrent Mode I/O). Line 30 turns on the Concurrent Mode I/O. Once the Concurrent Mode I/O is started, you may INPUT and/or PRINT at any time to the concurrent RS-232-C port; but no other I/O to any peripheral is allowed until Concurrent Mode I/O is stopped by closing the port. Input/output to the keyboard and screen is allowed while RS-232-C Concurrent Mode I/O is active, and that is what we're doing here, so there's no problem.

HOW TO USE
THIS PROGRAM

How is this program used? The first thing to know is that since it uses an RS-232-C port, the RS-232-C I/O handler for the RS-232-C ports must be loaded into the ATARI Home Computer System. This is done automatically for you by the computer and the interface module, but you have to turn on your equipment in the right order. Specifically, the interface module must be turned on before the computer console (or at the same instant), or the computer will not know the interface module is there. The RS-232-C handler can only be loaded when the computer console is turned on. See Section 3 for more details.

After turning your computer on correctly, you can type in the program. Or, the program can be loaded from tape or diskette. The person on the other end does all these things, too, but there's one small difference. One of you has to write his program to listen first, and the other to talk first. One way to do this is by adding the statement:

        40 GOTO 120

to one of the programs. This way, the program starts reading from the modem instead of the keyboard.

You're ready to go! Get on the phone to each other. One of you sets his modem to Answer mode, and the other sets his to Originate mode (it doesn't matter which). Both modems should be set to Full Duplex. Note that when the Answer modem is turned on, it starts to squeal. This is the same squeal you hear over the phone when you call up a timesharing service, and it is a signal to the Originate modem to start communications. To avoid the bother of listening to this, don't turn the modem on until you are ready to start talking (only the interface module needs to be turned on before the computer console — it's all right to turn on the modem later). Now put your phones in the modem cradles.

The person with the send-first program types a line (up to 120 characters), ending with a `RETURN`. Then the other can answer...and so on.

The listener will notice something odd right away: nothing appears on his television screen for a while, and then *POOF!* there's a message. Here's why: The sender's program gets to the INPUT statement, line 100. BASIC waits for input from the keyboard, and doesn't go on to PRINT to the modem (line 110) until the `RETURN` is typed by the sender. Meanwhile, the listener's program is also doing INPUT (line 130), and that INPUT won't complete until an **EOL** (End of Line) is received from the sender. (**EOL** is similar to `RETURN`, and is used inside the computer to mark the end of a line.) **EOL** is the last character sent by PRINT in line 110. So the sender's message doesn't start over the phone until he presses `RETURN`, and the listener doesn't see the message until it's completely received!

How is the program stopped? (Note that the program is an infinite loop.) Try the `BREAK` key. You'll find it doesn't work, because the `BREAK` key is turned off whenever Concurrent Mode I/O is active. In general, you should always try the `BREAK` key first whenever you want to stop a BASIC program, even if you don't expect it to work. But, if it doesn't work, don't be upset – just press the `SYSTEM RESET` key (**Note:** On early versions of the interface module, the `BREAK` key is not disabled during Concurrent Mode I/O.)

IMPROVING THE PROGRAM

You may find the limitation of one line at a time bothersome. Also, there is no way to interrupt and get the talker's attention when you're the listener. An improved version of the program should allow either of you to talk at any time, and send characters immediately as they're typed. This improved program is somewhat more complicated than the simple program you've seen, and some background is needed to understand it.

The basic idea is to use GET and PUT instead of INPUT and PRINT. GET and PUT work with single characters (represented by the character's numeric equivalent inside the computer), and characters are immediately available to BASIC without having to wait for the [RETURN] from the keyboard or **EOL** from the modem. (Note that when you use GET, the [RETURN] key produces the code for **EOL**, but this is not the same as "ending the input line" which is the meaning of [RETURN] when using INPUT.)

GET shares a problem with INPUT: BASIC waits until all the input is available. Admittedly GET is only looking for one character but until it arrives, BASIC waits. So suppose your copy of the program is executing the GET from the modem link to your friend, and he hasn't typed anything. This means you can't type, because your computer isn't reading your keyboard!

The trick is to avoid actually doing the GET until you know a character is there. Fortunately, there is a way to check for a character before GET, both for the keyboard and for the concurrent RS-232-C port. Thus the "flow" of our improved program will be like this:

100 IF *no character from keyboard yet* THEN 200

110 GET *character from keyboard*

120 PUT *that character to the modem*

200 *get* STATUS *of the modem buffer*

210 IF *character not available* THEN 100

220 GET *character from modem*

230 PUT *that character to television screen*

240 GOTO 100

The program continually alternates between checking for a character from the keyboard and one from the modem, and in each case only gets and sends along a character if one is ready.

(You may wonder how a character can become available even when BASIC has not yet tried to GET it. The Operating System of the computer, using techniques of "interrupts" and "buffering," accepts and saves the characters as they appear. Then, when BASIC does the GET, the saved character is handed over to BASIC from the Operating System. The keyboard has a one-character buffer, that is, only one character is saved this way. If you press another key before BASIC receives the first character, BASIC will find the second character you typed and the first is lost. The default buffer for each RS-232-C port holds 32 characters; that is, BASIC can fall 31 characters behind before anything is lost. You can set up a buffer larger than 32 if you need it — see Section 9 for more details.)

So much for the general idea — now for the details. First of all, GET does not work from the Operating System's screen editor; to use GET you must open the keyboard (K:). For the PUT commands to the television screen, you can use either the screen device (S:) or the editor device (E:). (You might want to experiment with E: and S: to see which suits your needs — they differ in their treatment of the editing keys such as DELETE BACK S , INSERT , and the cursor movement keys. See the *ATARI BASIC Reference Manual* for details.) Here, we'll use the editor.

To check whether a key has been typed, PEEK at the keyboard buffer. It is in memory at location 764 (decimal), and has the value 255 whenever a key has not yet been pressed. This location is reset to 255 when you GET from the keyboard.

Here's the improved program:

```
10 LET KB = 764: NOKEY = 255
20 LET MODEM = 747: NOCHAR = 0
30 OPEN #5, 4, 0, "K:"
40 OPEN #2, 13, 0, "R1:"
50 OPEN #3, 8, 0, "E:"
60 XIO 40, #2, 0, 0, "R1:"
100 IF PEEK(KB) = NOKEY THEN 200
110 GET #5, KEY: PUT #3, KEY
120 PUT #2, KEY
200 STATUS #2, XXX
210 IF PEEK(MODEM) = NOCHAR THEN 100
220 GET #2, CHAR
230 PUT #3, CHAR
240 GOTO 100
```

Lines 10 and 20 of this program set up some *symbolic constants*. The variables (symbols) are given values which will not be changed as the program runs (they're constant) – but see how much more readable these variables make lines 100 and 200. Lines 30 through 60 open the keyboard (K:), modem (attached to port R1:), and screen editor (E:), and start up the Concurrent I/O through R1. The rest of the program is just like the "flow" program shown before, except it's been turned into real BASIC. Note the extra PUT in 110 to place your keyboard typing onto your own television screen.

Lines 200 and 210 work together. First 200 gets the status of port R1: (which was opened through IOCB #2). Part of the status is the count of characters that have been received and placed in the receiving buffer, and that count is in location 747 (decimal) after the STATUS check. Line 210 checks 747 to see whether it's zero or not – if not, line 220 gets a character from the buffer.

This program is used just like the simpler one. Since either of you can talk at any time, you can both use the same program — it's not necessary to modify one of them to listen first. If you both type at the same time, though, your two messages will get mixed together. You'll have to learn to take turns typing, but allow for the other to interrupt if he wants. One character you might find useful is the bell (buzzer), which is typed as CTRL 2.

This program can be used to "talk" with a computer other than another ATARI Home Computer. If you get double characters when you are entering data into the other computer, then delete **PUT #3, KEY** in line 110.

INTERFACE MODULE CAPABILITIES

In effect, these two programs have changed your ATARI Home Computer into a teletypewriter, using the modem and interface module. With the latter program, you can access computer networks, such as THE SOURCE, AMERICA'S INFORMA-TION UTILITY* and COMPUSERVE**. The program is not intended to replace the TeleLink™ I cartridge, as it will print all the control characters to your television screen and has no provision for using an ATARI Printer.

Both programs can be helpful in learning how to use, and get more out of, the ATARI 850 Interface Module.

The interface module has many capabilities not mentioned in the above examples. The rest of this manual contains information about the interface module and how to take advantage of some of these capabilities. After you've read more about the interface module, look at the Section 8 examples of how to use it.

* THE SOURCE and AMERICA'S INFORMATION UTILITY are service marks of Source Telecomputing Corporation, a subsidiary of The Reader's Digest Association, Inc.

**COMPUSERVE INFORMATION SERVICE is a registered trademark of CompuServe, Inc., an H & R Block company.

HOW THE INTERFACE MODULE
INTERACTS WITH THE SYSTEM

TURN-ON OPERATION

The Operating System (OS) of the ATARI Home Computer does not contain the information necessary to operate the interface module SERIAL INTERFACE ports or the **ATARI 810™ Disk Drive.** This information comes from the peripheral itself.

The computer asks for the data when it is turned on. If the peripheral is turned on before or at the same time as the computer, it will answer the computer's request and send the necessary information. This turn-on and initialization procedure is called "automatic bootstrap," "autoboot," or just "boot." The term comes from the expression "pulling yourself up by your bootstraps," indicating that you start with nothing and reach your goal by your own efforts.

The bootstrap information contained in the interface module is called the RS-232-C "handler." The disk drive information is called the Disk Operating System (DOS).

WITHOUT A DISK DRIVE

When the power is turned on to the computer console, the computer issues a disk drive request. If there is no disk drive in the system (or if the disk drive is turned off), the interface module will respond to the disk drive request. The computer then loads the RS-232-C handler bootstrap program from the interface module, just as though it were reading the program from a diskette. The bootstrap program is then run, and it gets the RS-232-C handler from the interface module and relocates it into the computer's RAM. The memory occupied by the bootstrap program is then freed (but the handler remains).

WITH A DISK DRIVE

If the disk drive is set for Drive 1, it will respond to the disk drive request when the computer console is turned on. The interface module will not respond. A special start-up program is loaded from the diskette and this program then loads the handler from the interface module.

In the ATARI 810 Master Diskette, CX8104, this job is handled by a file called AUTORUN.SYS that is supplied with your DOS II Diskette. Read the instructions supplied for details on AUTORUN.SYS.

Caution: The RS-232-C handler shares RAM space with a portion of the DOS utilities. When DOS is called (by typing **DOS** and pressing RETURN from BASIC), DOS will overwrite the RS-232-C handler and destroy it. To protect against this, add MEM.SAV to your diskette (item N on the DOS Command Menu). Then, when you call DOS, the RS-232-C handler will be saved with your program.

Note: ATARI 810 Master Diskette, Model CX8101, does not contain the AUTORUN.SYS file and cannot be used with the interface module RS-232-C SERIAL INTERFACE ports.

As a general rule, turn on any peripheral that you intend to use with your ATARI Home Computer before turning on the computer. This allows those devices that need booting to do so.

There are exceptions to this rule, though. For example, since an ATARI Printer does not use the SERIAL INTERFACE I/O ports of the interface module, it can be turned on at any time, as can the ATARI 830 Acoustic Modem.

USING A PRINTER

In the case of the ATARI Printer, turning the computer on *before* turning on the interface module saves RAM, as the RS-232-C handler takes over 1 1/2 K-bytes of memory. The RS-232-C handler is not needed to use an ATARI Printer attached to the printer port of the interface module. To see how much memory can be saved, boot your system **with** the RS-232-C handler and type **?FRE (0).** This will give you the amount of free memory with the RS-232-C handler loaded into memory. Repeat the action **without** loading the RS-232-C handler. The difference is the amount of RAM saved by not using the handler when it is not needed.

PROGRAMMING THE SERIAL INTERFACE PORTS

As with any peripheral device attached to an **ATARI 400™** or **ATARI 800™ Home Computer**, the ATARI 850 Interface Module requires a program to tell it what to do. This program may be pre-written or you may want to use the SERIAL INTER-FACE ports from your own BASIC program. In the case of pre-written programs, such as the ATARI TeleLink I cartridge, read the instructions for using that particular program with the interface module.

Using software instructions to set the specific values for the parameters of the port is called "configuring the port." The configuration and use of the SERIAL INTER-FACE ports on the ATARI 850 Interface Module can be complex. Many details must be remembered and complicated procedures must be followed exactly. This section gives an overview of the effects of commands and their relation to each other.

INPUT/OUTPUT CONTROL BLOCK

IOCB is an acronym for Input/Output Control Block. It is that portion of the computer's Operating System (OS) that controls the input and output of data within the system.

An IOCB allows the computer to keep track of the I/O functions, both its own and the user's. Therefore, an IOCB acts as an interface between the user and the computer I/O system.

From ATARI BASIC, the user has seven IOCBs available to use. These are numbered 1 to 7. IOCB #7 is used by the OS for LPRINT and and IOCB #6 is used for GRAPHICS MODE functions. These IOCBs should not be used with the interface module if you have graphics or line printer commands in your BASIC program. To be on the safe side, specify IOCB #5.

ACCESSING AN RS-232-C DEVICE

STEP 1 - CONFIGURE THE SERIAL INTERFACE PORT

The first thing to be done to access an RS-232-C device is to configure the SERIAL INTERFACE port to which the device is connected by using instructions in your program. In configuring the port you may set the following:

- Baud rate — bits-per-second sent/received
- Number of bits-per-word sent/received
- Number of stop bits-per-word sent
- Whether the incoming control signals DSR, CTS, and CRX are monitored

- Whether input parity is checked
- Whether output parity is set
- Whether Line Feed is added after every Carriage Return sent
- Translation of the word being sent or received (three types of translation)

- How the outgoing control signals DTR and RTS are used

These are shown as three groups, corresponding to the three configuration commands; otherwise, the division into groups is arbitrary.

If you do not configure the port, the system sets default values of the port variables, as follows:

- 300 Baud
- 8 bits-per-word
- 1 stop bit-per-word transmitted
- Input parity is not checked
- Output parity (bit 7) is set to zero
- Line feed is not added after every Carriage Return sent
- Light-translation
- Outgoing control signals DTR and RTS are set off

If the default (preset option) is what you want, then the parameters of the port configuration do not have to be set.

Each of these groups of conditions can be changed with a configuration command and each port can be configured independently. Configuration of one port has no effect on the configuration of any of the other ports.

The CONFIGURE BAUD RATE command is used to set the Baud rate, number of bits per "word," and the number of stop bits to transmit. This command also establishes the monitoring of DTR, CTS, and CRX. The CONFIGURE TRANSLA-TION MODE command will set up the translation mode, input and output parity modes, and the automatic appending of LF (Line Feed) after CR (Carriage Return). The CONTROL command will let you turn the DSR and RTS control lines on or off.

STEP 2 - USING THE SERIAL INTERFACE PORT

Once the SERIAL INTERFACE port is configured, you can OPEN the port for I/O. There are two fundamentally different ways of doing I/O to a SERIAL INTERFACE port: Block Output Mode and Concurrent Mode. As its name implies, the Block Output Mode can only be used for output from the computer to your RS-232-C compatible device. The Block Output Mode is simpler to use, allows use of the DTR, CTS, and CRX monitoring, and carries none of the concurrency restrictions of the Concurrent Mode. Concurrent Mode I/O is required for input from the RS-232-C compatible device and is required for full-duplex (input and output at the same time) operation.

Block mode output is performed by simply doing normal BASIC PRINT or PUT statements to the appropriate SERIAL INTERFACE port (after opening it, of course).

Your output characters will be placed in a 32-byte buffer and transmitted to the interface module and then to your RS-232-C compatible device. This is done when:

- The buffer fills up
- You close the channel to the RS-232 port
- A CR (decimal 13) is placed in the buffer
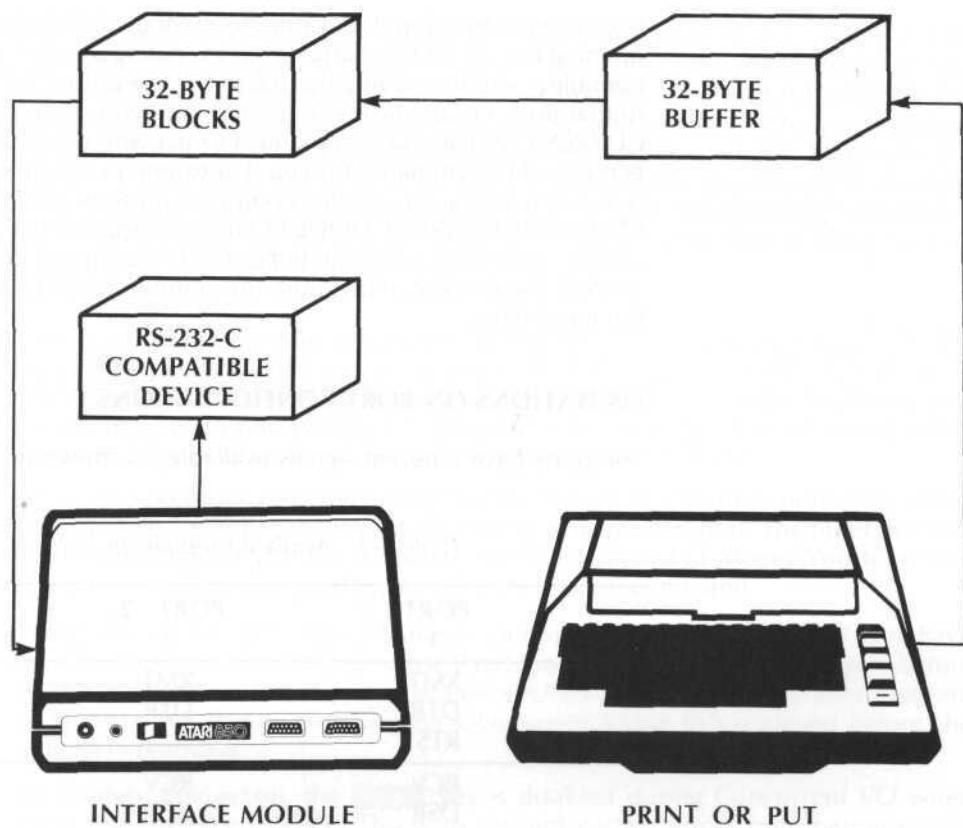
Figure 4-1. Block Output Mode I/O

On occasion, you may want to force the sending of the information in the buffer. For example, if you have specified the Append LF translate option, the LF will be sent at a different time, later than the CR. You may want to send the LF immediately if the external device is a terminal. As another example, if you are using the DTR, CTS, or CRX monitoring feature to avoid sending more characters to a device than it can handle, you can use the FORCE SHORT BLOCK operation to send your characters one (or a few) at a time. That way you can ensure that the device won't lose characters you send it because it became not ready in the middle of an output block.

The FORCE SHORT BLOCK operation is only valid if you are using Block Output Mode. If you are using Concurrent Mode, you cannot use this command.

If you issue a FORCE SHORT BLOCK command when the buffer is empty, no action will be taken. Doing this is not an error. Since you can alternate output to two SERIAL INTERFACE ports when using Block Output Mode, you can also alternate FORCE SHORT BLOCK commands from one port to another. The ports must be opened through different IOCBs, of course. The BASIC command for FORCE SHORT BLOCK is listed in Appendix B.

Concurrent Mode input, or Concurrent Mode (full duplex) I/O, is performed by first opening the file for Concurrent I/O, executing a START CONCURRENT MODE I/O operation, and then doing normal BASIC INPUT, GET, PRINT, and PUT operations to that port. In Concurrent Mode I/O, after you have performed the START CONCURRENT MODE I/O command, I/O is going on at the same time BASIC is executing other commands for you. For example, if your RS-232-C compatible device is sending characters to the computer through the interface module, after the START CONCURRENT MODE I/O those characters will be saved for your program as they arrive into a holding buffer by the computer. If you subsequently perform an INPUT statement to that port, the computer will really just look in that buffer for the input data.

LIMITATIONS ON PORT CONFIGURATIONS

The ports have different signals available, as shown in Table 4-1.

Table 4-1   Available Signals on Ports 1, 2, 3, and 4

PORT 1	PORT 2, 3	PORT 4
XMT	XMT	XMT
DTR	DTR	
RTS		
RCV	RCV	RCV
DSR	DSR	
CTS		
CRX		

For 8-bit operation, the following limitations are imposed:

- Input requires Concurrent Mode I/O.
- Output allows the option of Concurrent Mode I/O or Block Mode I/O.
- Full-duplex operation requires Concurrent Mode I/O.
- Full-duplex supports all Baud rates.

For 7-, 6-, and 5-bit operation, the following limitations are imposed:

- Full-duplex is not allowed.
- Half-duplex input can be in Concurrent Mode I/O only.
- Half-duplex output can be in Block Mode only.
- Input and output can be only 300 Baud or less.

Other limitations on port configurations are imposed by using a port in the Concurrent I/O mode. These limitations are described in this section under Restrictions.

If any default condition is to be changed, the port must be configured *before* it is used. Configuring a port is accomplished by one or more commands described in this section. There are three principal commands. Each command is concerned with several configuration variables. The parameters of the commands are coded to signify different values of the several variables. The details of the coding are presented in Appendix B.

A particular default condition is Block Output Mode in which, of course, you cannot input data. To input from a port you must put it into Concurrent Mode I/O with the START CONCURRENT I/O command. Thus, the START CONCURRENT I/O command is a configuration command. It is different from the other configuration commands in that the port to which it applies must be opened first. Moreover, it is much more complex than other configuration commands. You should think of the START CONCURRENT I/O mode command as being a configuration command in one aspect and as having more important effects on other aspects of using the configured port and, indeed, all the ports.

RESTRICTIONS

You must observe certain precautions when you use Concurrent Mode I/O. The only I/O operations that are permitted with this mode are GET, INPUT, PUT, PRINT, STATUS and CLOSE to the opened port, and I/O to the keyboard and screen (which do not involve any peripheral device). Input/output to any other peripheral is not allowed while Concurrent Mode I/O is active.

Using one port for Concurrent I/O prevents the use of any other port of the interface module, including the PARALLEL INTERFACE (printer) port. The other ports remain inaccessible until you terminate the Concurrent I/O Mode. You terminate Concurrent I/O by closing the port with the CLOSE command.

Using any of the SOUND commands during Concurrent Mode I/O can have disastrous effects, from changing the Baud rate to stopping I/O completely before your data is transferred. If you must use SOUND commands, write your program so that the IOCB you are using for Concurrent Mode I/O is closed before the SOUND command.

As a safety precaution, the `BREAK` key is disabled during Concurrent I/O when using the interface module. This is to prevent certain errors from harming your work sessions. You may use the `BREAK` key as you normally would to stop your program, but be prepared for nothing to happen if Concurrent I/O is active. If that should be the case, use `SYSTEM RESET`.

Note: `SYSTEM RESET` can mess up your disk files. Use `SYSTEM RESET` only after trying `BREAK`. The BREAK key was not disabled on early versions of the ATARI 850 Interface Module. These early versions can be upgraded at your local Authorized ATARI Computer Service Center. The ATARI TeleLink I cartridge will transmit a break signal when you type `SHIFT` `BREAK`. Refer to your *ATARI TeleLink I Operators Manual* for details.

During Concurrent I/O, incoming data may overflow the computer's buffer. In that case, data is lost. Methods for avoiding loss of data in this way are described in Section 10.

Port configuration cannot be done after having started Concurrent Mode I/O. For that reason, specify port parameters (with XIO 34, XIO 36 and XIO 38) before entering Concurrent Mode I/O.

Once set, configured parameters will not change until you change them with an appropriate command. Pressing `SYSTEM RESET` on the computer will *not* reset any parameter to its default value. Turning off the power on the interface module may reset some parameters but not others. This action may result in peculiar operation because information about some of these parameters is saved both in the computer and in the interface module. Turning off the power to the interface module during a session with the computer is not recommended.

Turning off the power to the computer also resets the interface module. When you turn the computer on again, and the interface module boots, all of the above parameters will have reverted to their preset default values.

SETTING THE BAUD, WORD SIZE, STOP BITS, AND READY CHECKING

Common convention and other standards have settled on a fairly universal serial data transmission convention. When data is not being sent, the data line will sit idle in the MARK state. A data character (sometimes called a transmission WORD) is signalled by one START BIT, represented by the SPACE state. It is followed by the data bits, each bit being represented by SPACE for 0 and MARK for 1. The word is terminated by 1 (sometimes 2) STOP BIT(s), represented by the MARK state. The next word can immediately follow with its start bit. If it does not, the line stays idle in the MARK state. Effectively, the stop bit lasts indefinitely.

The most common transmission word size is 8 bits. When sending ASCII, which is a 7-bit code, the 8th bit usually represents the parity, is just set to 1 or 0, or is used as a marker bit of some sort. ASCII is very rarely sent in 7-bit words. The interface module supports 7-bit words for these cases and can also be used for communication with 7-bit or 6-bit codes such as BCD (with or without parity). Five-bit words are also allowed so you can communicate with old Baudot code teletypes for radioteletype and similar uses.

The receiver can receive all the bits in the word because it knows when each will arrive. Each bit is the same duration as established by the Baud rate (bits-per-second rate).

The CONFIGURE BAUD RATE command allows you to set the Baud rate, "word" size, number of stop bits to transmit, and enable or disable checking of DSR, CTS, and CRX. The command may be issued through an open IOCB to the RS-232-C SERIAL INTERFACE port, or through an IOCB which isn't being used. If you have opened an IOCB to the port you are configuring, you must use that IOCB. You cannot configure any port if a Concurrent Mode I/O operation is active.

The CONFIGURE BAUD RATE command looks like this in BASIC:

XIO 36, #IOCB, Aux1, Aux2, "Rn:"

The 36 makes this a CONFIGURE BAUD RATE command.

The #IOCB is the number of the IOCB that BASIC should use to execute the command. The IOCB should either be open to the port you are configuring, or should not be open at all. No Concurrent Mode I/O should be active when you issue this command.

Aux1 is a number or expression that specifies the Baud rate, "word" size, and number of stop bits to send with each "word." For each of these, pick a number from Tables B-1, B-2, and B-3 in Appendix B, and then add the numbers together to form Aux1. You may add them yourself or you can let BASIC add them for you. For example: **XIO 36, #1, 10 + 0 + 128, 0, "R:"** and **XIO 36, 138, 0, "R:"** both specify the same thing.

Aux2 is a number of expression that specifies whether or not the interface module should check Data Set Ready (DSR), Clear to Send (CTS), and/or Carrier Detect (CRX) when a Block Mode output or START CONCURRENT MODE I/O operation is performed. If you ask to have the interface module check one or more of these, the interface module will return error status if the line(s) checks is not ON. The error status is returned from the Block Mode output attempt or from the START CON-CURRENT Mode I/O attempt. The condition of the line(s) being checked does not matter at the time you do the CONFIGURE BAUD RATE command to turn on the checking; the line(s) will be checked only when the Block Mode output or START CONCURRENT I/O is attempted. You may TRAP the error and program ATARI BASIC to take the action you desire. See Table B-4 in Appendix B for values of Aux2.

The last XIO parameter, **Rn:**, specifies which SERIAL INTERFACE port of the inter-face module you are configuring. For **n** put 1, 2, 3, or 4, just as you would in the OPEN command.

Note that the default (preset) values of Aux1 and Aux2 for all four ports are zero, corresponding to 300 Baud, 8-bit words, one stop bit transmitted, and no checking of DSR, CTS, or CRX.

You should know the following things about this command:

The configured parameters will stay as you set them until you either reset them or until you reboot the system (turn the power off and back on). The `SYSTEM RESET` key will *not* reset any of these parameters.

You may configure each RS-232-C SERIAL INTERFACE port independently.

If you specify 8-bit words, there are no restrictions on operation of the port. However, the following restrictions apply to 7-, 6-, and 5-bit words:

- Full-duplex is not allowed
- Half-duplex input can be in Concurrent Mode I/O only
- Half-duplex output can be in Block Mode only
- Input and output can be only 300 Baud or less

If you specify 7-, 6-, or 5-bit words, there is no restriction on the number of stop bits you may specify. Note that most applications of these word sizes will probably be to devices that require more than one stop bit — you should specify two. Each word sent or received will be converted from, or to, an 8-bit byte within the com-puter by ignoring the most significant bit(s). This will very likely interact with the translation operation and, in particular, there may be no way you can receive an EOL. If this is the case, you cannot use the BASIC INPUT statement to read the port, and you must retrieve characters one at a time using GET. More details will be found in Section 10.

If you specify that you want the interface module to check DSR, CTS, and/or CRX, it will check them whenever you try to start Concurrent mode I/O and whenever you try to send a block of data in block output mode. If any one of the lines you asked to be checked is not ready (OFF), then the Concurrent Mode I/O will not be started or the block of data will not be sent. The interface module will then return ERROR 139 (device not acknowledged) to BASIC, and you may TRAP the error and take corrective action. Following the TRAP, you may perform a STATUS request from the interface module which will provide bit 4 in Location 746. See Section 10 for details.

Note that CTS and CRX are not supported on ports 2, 3, and 4, and that DSR is not on port 4. The interface module behaves as if they are really there, however, and as if they are always ready (ON).

You may look at the states of DSR, CTS, and CRX any time that Concurrent Mode I/O is not active (you must have an IOCB open to the port) by issuing a STATUS request for the port. Thus, enabling this automatic checking of these lines is not the only option available to you, and you may prefer checking them directly with STATUS. See Section 10 for details.

SETTING THE TRANSLATION MODES AND PARITY HANDLING

The interface module handler can be configured to perform certain types of code conversions (translations) and do parity generating and checking for you. These two operations interact with each other. For this reason, they will be described together in this section. The various options you may select for each are specified by executing the same command — CONFIGURE TRANSLATION AND PARITY.

Three factors must be kept in mind when setting up code translations. Translation, of course, is one of them, since it results in (possibly) changing one code into another. Parity generation and checking also may result in changing one code into another. The third factor to remember is the word size you are transmitting/receiving. Inside the computer, all words are the same as bytes; that is, all words are 8 bits. If you are sending/receiving 7-, 6-, or 5-bit words, these shorter words have to come from 8-bit computer words by chopping out some bits, or expanded into 8-bit computer words by adding some bits. These operations are similar to changing one code into another.

Each of these three possible code changes takes place separately from the others, one at a time. For output, translation comes first, followed by parity generation, and finally truncation (shortening by leaving bits off). Of course, at each stage a change may not occur, depending on what selection of options you have configured and depending on which character (code) the computer is sending. For example, if you have configured 8-bit words, the truncation operation does nothing. For input, the order of code changing is expansion (from short words to 8-bit words), followed by parity checking, and finally translation.

At each of the three stages, a code change may occur. If a change *does* occur, then it is the *changed* code that will be operated on in the next stage. For example, (in a particular configuration of translation and parity options) if you output an ATASCII EOL (End-of-Line), it would first be translated to an ASCII Carriage Return (CR) and then parity would be generated for the CR. The parity step operates on the result of the translation step, in this case the CR.

Note: ASCII is an acronym for the American Standard Code for Information Interchange. In ASCII, each letter (both upper- and lowercase), numeric code, and control key has a number assigned to it. In order for the ATARI Home Computer to display its spectacular screen graphics, the ATARI Computer engineers devised a modified version of ASCII, which is called ATASCII. In ATASCII, the codes used for certain ASCII control characters have been assigned to ATARI Computer graphics characters. Appendix A contains a conversion chart.

There is one other translation option which is very specific; namely, the option to have an ASCII Line Feed (LF) sent after each transmitted CR. This code change occurs at the translation step. Consequently, the generated LF will go through the parity and truncation (small word) phases just like the CR.

TYPES OF CODE TRANSLATION

You have three options from which to choose: no translation at all, "Light" translation, or "Heavy" translation. Whichever option you choose will apply both to incoming and outgoing characters. The no-translation option is just what it says — no change is made to the characters, whether being received or sent. This statement applies only to the translation step, of course — you can still get changes from parity and small words. The no-translation option is useful if you are going to do your own special processing on the characters you are sending and receiving. This can be particularly useful in the small word situations, since many of the cases where small words are used do not (or cannot) involve ASCII. You may also want to use the no-translation option if the RS-232-C compatible device you are communicating with understands ATASCII. An example of this is communication with another ATARI Home Computer via modem.

No matter which translation option you choose, if you use a BASIC INPUT statement to read data, the data must have an ATASCII EOL character at the end of each line. This requirement applies AFTER all translation. Thus, if you select the no-translation option, your incoming data must either contain EOLs or you should use GET instead of INPUT. Remember that using short words and checking parity affect data coming in, so you may still need to use GET.

Heavy and Light translation are two ways to convert between ASCII and ATASCII. In either translation mode, the ATASCII EOL (9B in hexadecimal, 155 in decimal) is converted to and from the ASCII CR (OD in hex, 13 in decimal). In the case of output, EOL is changed to CR; if you also selected the Append LF option, EOL is changed to CR followed by LF, that is, the translation function produces two characters out for one in. On input, a CR will be translated to EOL. Both Heavy and Light translation modes assume ASCII in the outside world and they assume ATASCII in the computer. ASCII is treated as a 7-bit code; that is, the eighth (most significant) bit is always treated as if it is zero. On input, then, if you select Heavy or Light translation, the eighth bit of each word is cleared to zero. On output, the translation step will set this bit to zero.

Light translation performs the fewest changes between ASCII and ATASCII. The assumption is that you wish to work with ATASCII within the computer but treat it as if it were really ASCII. Note, for example, that the ATASCII graphics codes are the same numerically as some ASCII control codes (1 - 26). So for input, the character has its high bit stripped (set to zero), and that's all — except if the code is found to be a CR, it is changed to an EOL. For output, if the character being sent is EOL it is changed to CR; then, no matter what the character is, the high bit is set to zero. Light translation is the preset default mode.

Heavy translation is a more thorough translation mode. Here the assumption is that if there is no direct correspondence between the character in ASCII and ATASCII, then the code should not be translated. So for input, after the high bit is cleared to zero, if the character is CR it is changed to EOL; otherwise, the character is checked to see if it is the same in ATASCII as in ASCII. If it is not, it is translated to the Won't-Translate character. Specifically, if the code for the ASCII character is less than 32 decimal (i.e., the character is a control character) or greater than 124 decimal (7C hex) it will be translated to the Won't-Translate character. If the steps preceding Heavy translation leave the most significant bit (bit 7, corresponding to parity) set to 1, the character has a value greater than 124 and therefore, it won't translate. Thus, heavily translated ASCII corresponds to the printable characters from blank through vertical bar. The Won't-Translate character is specified by you in the CONFIGURE TRANSLATION command. If you do not specify it, the preset default value for it is zero (ATASCII graphic heart).

On output, Heavy translation converts EOL to CR, and will output any character whose ASCII meaning is the same as it is in ATASCII. That is, characters whose values range from 0 - 31 decimal (ASCII control values) or whose values are above 124 decimal (7C hex) will not be sent. Note that characters whose high bit is one will be translated to nothing; that is, characters that would show on the television screen as inverse video will not be sent in Heavy translation mode. Note also the difference between input and output in the Heavy translation mode: untranslatable characters in the input are converted to the Won't-Translate value, where untranslatable output simply is not sent out.

The (optional) sending of LF after CR is produced in the translation step. If you specify no translation, the option of adding LF to CR is not available. If you specify Light translation, LF will follow EOL (which of course becomes CR). Note that if you send the 13-decimal code (CR), LF will be added to it (when the Append LF feature is on). Each character in the CR/LF pair is independently sent through the parity and word shortening steps on its way out. The preset default setting of the Append LF feature is Off, that is, the default is to **not** append the LF. LF will be appended only if translation is enabled. If NO TRANSLATION is set, LF will not be automatically appended to anything.

PARITY

You may select input and output parity handling separately. Thus, you may choose to send, for example, even parity while you ignore the parity of what you are receiving. The parity is always the most significant bit of each 8-bit byte (bit number 7). Parity operation is not useful then, if you are working with 7-, 6-, or 5-bit words.

In the default parity condition, the parity bit of input or output is not altered. However, the parity bit of outgoing messages may have been changed during the translation step.

For output, you may select even parity, odd parity, set parity bit to 1, or no parity change.

For input, your choices are "don't touch," "check even," "check odd," and "don't check." Each of these last three options will clear the top bit to zero, whether or not a parity check is made. If an input parity error is found, the character will still be input as if it were all right; the parity error flag will be turned on in the status bytes. See Sections 10 and 13 for details.

SHORT WORD CONVERSION

The third operation which affects your code translation is the short word conversion (if you are using 8-bit words, this is a "no-effect" operation). Short words sent out are made from 8-bit computer characters by omitting the most significant bits. That is, a 7-bit word is bits 0 - 6 of the character, a 6-bit word is bits 0 - 5, and a 5-bit word is bits 0 - 4. Thus the parity, if generated, is lost. ASCII is a 7-bit code; you can send ASCII in 7-bit form without parity (this is not common practice, though — usually 8 bits are sent even if the 8th bit is not used for parity). With 6-bit and 5-bit codes, you will not be using ASCII, so you will have to concern yourself with the codes you want to be sending. With these word sizes, you should turn translation off so the translation performed by the interface module handler will not affect the codes you are using.

On input, small words are converted to 8-bit computer characters by adding high-order bits. These added bits are always set to 1. Thus, if you are receiving 7-bit ASCII, the parity and translation steps will be getting ASCII with the 8th bit set high. If you are receiving 6- or 5-bit codes, there is no way you can receive the 13 decimal (0D hex) code (ASCII CR) — after all, you cannot receive ASCII in 6 or 5 bits anyway. This means that in BASIC you will have to use the GET statement, not INPUT. Of course, you will be doing your own code conversion, so you should turn off the conversions of the interface module handler.

The CONFIGURE TRANSLATION MODE command is specified in BASIC this way:

XIO 38, #IOCB, Aux1, Aux2, "Rn:"

38 specifies the CONFIGURE TRANSLATION MODE command.

The #IOCB specifies the IOCB number (from 1 to 7) you wish to use to configure the translation mode. You may open a new IOCB if you have no channel open to the port you are configuring; otherwise you must use the IOCB you have opened to that port. You cannot issue the CONFIGURE TRANSLATION MODE command if any Concurrent Mode I/O is active.

Aux1 specifies the translation mode, the input parity mode, the output parity mode, and the Append LF option. You specify these options by adding numbers taken from Tables B-5, B-6, B-7 and B-8 in Appendix B. You may add the numbers yourself and put the sum in your program for Aux1, or you may let BASIC add them for you (e.g., you can say either 2+8+32 or 42 to mean even parity in, even parity out, and no translation). Do not add in more than one value from each table.

Aux2 is the numeric representation of the Won't-Translate character for Heavy translation. Remember that the BASIC function ASC will give you the numeric representation of a character. For example, 65 and ASC("A") mean the same number. The number you specify should be from 0 through 255.

"Rn:" specifies the port you are configuring. For **n**, you put 1, 2, 3, or 4. **R:** means you are configuring port 1.

The default configuration is Aux1 = 0 and Aux2 = 0. If you execute the CONFIGURE TRANSLATION MODE command for one of the RS-232-C SERIAL INTERFACE ports, that configuration will remain in effect until you do another CONFIGURE TRANSLATION MODE for that port. SYSTEM RESET will not change the translation mode for any port. Of course, you can configure each port a different way.

CONTROLLING THE
OUTGOING LINES

DTR, RTS,
AND XMT

There are up to three outgoing RS-232-C signals on each of the RS-232-C SERIAL IN-TERFACE ports of the ATARI 850 Interface Module: Data Terminal Ready (DTR), Request to Send (RTS), and Data Transmit (XMT). Each of these lines can be turned on or off with the CONTROL command.

Port 1 supports all three outputs. Ports 2 and 3 have DTR and XMT. Port 4 has only XMT. You may use this command the same way with any port — it is not an error to try to control a line that does not exist. Your attempt will simply have no effect.

You may control any or all of these lines on a single RS-232-C SERIAL INTERFACE port with the CONTROL command (controlling lines on other ports requires one CONTROL command for each port). The CONTROL command may be issued to a port which is not OPEN through an IOCB by specifying any unopen IOCB number in the CONTROL command. If the port has been opened through an IOCB, you must use that IOCB in the CONTROL command. You may not issue a CONTROL command if any Concurrent Mode I/O is active.

Controlling XMT line has very limited use and few users will be concerned with it. If you change XMT you are likely to interfere with the normal transmission of data. In the serial communication world the only practical use of control of the XMT line is to send a BREAK signal. The BREAK is simply a period of holding the XMT line out of its normal resting state. Specifically, the normal resting state is called MARK, which corresponds to the binary 1 state. A BREAK is a period of the state called SPACE, which corresponds to binary 0. (Actually, since MARK and SPACE are the only legal states of any RS-232-C SERIAL INTERFACE signal, all data consists of alternating MARKS and SPACES.) What distinguishes BREAK from other uses of SPACE is that a BREAK is a SPACE which is a lot longer in duration than the time that a transmitted word would be. This is true because any transmitted word ALWAYS has one or more MARK bits in it — in particular, each word ends with one or more stop bits represented by MARK. Thus to send a BREAK, first issue a CONTROL command to set the XMT line to SPACE (0), then a little while later issue a control to set it back to MARK (1).

The uses of the other lines will depend on your application. For some guidelines, see Section 10.

CONTROL
COMMAND

The preset default state of the DTR and RTS lines is OFF. The preset default state of the XMT line is MARK. Once you change any of them with the CONTROL command, the new setting will remain until you either turn the computer off or issue another CONTROL command to change things. The SYSTEM RESET key has no effect on these lines.

The form of the CONTROL command in BASIC is:

XIO 34, #IOCB, Aux1, Aux2, ''Rn:''

34 specifies the CONTROL command.

The #IOCB specifies the IOCB number (1-7) you wish to use for the command. If no IOCB is open to the RS-232-C SERIAL INTERFACE port, specify an unused IOCB. If not doing concurrent I/O and the port is open through an IOCB, use that IOCB.

Aux1 is the sum of three numbers chosen from Tables B-9, B-10, and B-11 in Appendix B to control DTR, RTS, and XMT. Choose only one number from each table. You may add the numbers together yourself and put the resulting sum in your program for Aux1, or you may put an expression for the sum and let BASIC do the arithmetic for you.

Aux2 is not used by this command; the best value to specify is zero.

''Rn:'' specifies the RS-232-C SERIAL INTERFACE port you are acting on. For **n** you put 1, 2, 3, or 4. If you omit **n**, the interface module handler will assume you mean port 1.

BASIC I/O COMMANDS

The commands OPEN and CLOSE, and the I/O commands GET, INPUT, PUT and PRINT, LIST, SAVE described here should be familiar from using the BASIC language.

OPENING A PORT

You must open an IOCB (using the BASIC OPEN command) to a RS-232-C SERIAL INTERFACE port before you can read from it, write to it, start Concurrent Mode I/O or read its status. You may configure a port without having opened it.

The OPEN command in BASIC is:

OPEN #IOCB, Aux1, Aux2, "Rn:"

IOCB is the number of the IOCB that other BASIC commands for the opened port must use. Any IOCB number (1 through 7) may be used. Do not use an IOCB if another file is already open through it.

Aux1 specifies the direction of the port:

- 5 signifies that you are going to use the port for input only (Concurrent Mode I/O)
- 8 signifies that you are going to use the port for output only (Block Mode I/O)
- 9 signifies that you are going to use the port for output only (Concurrent Mode I/O)
- 13 signifies that you are going to use the port for input or output (Concurrent Mode I/O, full duplex)

Aux2 is not used in this command; make Aux2 zero.

Rn: is the RS-232-C SERIAL INTERFACE port being opened. **n** can be 1, 2, 3, or 4. **R:** is interpreted as **R1:**. For a given port no more than one IOCB may be open at one time.

CLOSING A PORT

Having opened and used a port, you may disconnect the IOCB by closing the port with the BASIC command CLOSE, as follows:

CLOSE #IOCB

IOCB is the IOCB number previously opened.

CLOSE is also used to terminate Concurrent Mode I/O. In this case the IOCB number is the one through which the Concurrent Mode I/O is active. CLOSE is the only way to terminate Concurrent Mode I/O from a program.

Note: Always close Concurrent Mode I/O before closing anything else.

To restart Concurrent Mode I/O to the port, you must first reopen an IOCB to it with the OPEN command.

When you CLOSE the IOCB, all data in the input buffer is lost, and all data in the output buffer is sent.

Closing a file does not change the configuration of the IOCB. You may change any configuration parameters after closing the port.

Note: Failure to terminate Concurrent Mode I/O properly before attempting I/O to other peripherals (or even other SERIAL INTERFACE ports) will probably result in program failure.

Pressing `SYSTEM RESET` on the computer closes all open IOCBs and reestablishes most of the I/O system's registers and pointers. This method of closing files results in the loss of data being held in input and output buffers. The interface module may be "interrupted" by the `SYSTEM RESET` and so transmit only part of the character being sent at the time `SYSTEM RESET` was pressed. Another possible effect of `SYSTEM RESET` is a short burst of random data to an active Concurrent Mode I/O RS-232-C SERIAL INTERFACE port.

The exclusion of peripheral I/O to anything other than the active Concurrent Mode I/O port applies to the CLOSE command. If you have any other peripheral device or RS-232-C SERIAL INTERFACE port open, you cannot close it while one open port is in the Concurrent Mode I/O. Close the Concurrent Mode I/O port first.

If files are not specifically closed, BASIC will close them when it interprets END or comes to the end of the program. All files are closed in the descending order of the IOCB number you have assigned.

ATARI BASIC has reserved IOCB #7 for LPRINT and IOCB #6 for the Graphics Modes. These two IOCBs are user programmable. However, problems will occur if you have opened IOCB #6 and then use the LPRINT command, or have IOCB #7 open and change Graphics Modes. For this reason, it is suggested that IOCB #5 be used when configuring a serial port and IOCB's numbered less than #5 be used for your other files. Then, when your program ends, the serial port will automatically be closed *before* any other program files.

Note: Always make sure that an active Concurrent Mode I/O IOCB is closed before any other closes occur.

ATARI BASIC I/O STATEMENTS

This section contains information on how these statements are used with the SERIAL INTERFACE ports. Before reading this section you should read and understand the material in Sections 4, 5, and 6, Appendix B, and the preceding material on opening and closing ports.

GET, INPUT, PUT, AND PRINT

The BASIC input statements are GET and INPUT. The BASIC output statements are PUT and PRINT. Refer to the *ATARI BASIC Reference Manual* for details about these statements. In this context, PRINT and INPUT must always include the proper IOCB number. The formats are given here as a reminder.

Formats: GET #IOCB,var
          INPUT #IOCB{;}{avar[,{avar}...]
                     {,}{svar}[,{svar}...]
          PUT #IOCB,aexp
          PRINT #IOCB{;}exp[,exp...]
                  {,} [;exp...]}

avar — Arithmetic variable
exp — Any expression, whether a string expression or an arithmetic expression
svar — A string variable
var — Any string variable, string or arithmetic

(See the *ATARI BASIC Reference Manual* for details.)

INPUT and PRINT are line oriented. They process a ''line'' of characters at a time. A line ends with an ATASCII EOL character. The translation mode you set up (or the one preset for you) can be used to translate the EOL character to an ASCII CR on output, and CR to EOL on input. *An EOL is required for INPUT* — a BASIC INPUT statement will not finish until an EOL is read in. If your input does not have EOL, or if your translate mode will not produce it on input, you should not use INPUT, but use GET instead.

Remember that if you place a comma or semicolon at the end of a PRINT command, EOL is not produced when the PRINT command is executed.

When you use a BASIC input statement, the input data must be in the proper form for BASIC. For example, if you read into numeric variables, the input must consist of digits with optional sign, decimal point, and exponent. Multiple input numbers must be separated by commas or EOLs. For more details see the *ATARI BASIC Reference Manual.*

GET and PUT are character-oriented. You can input or output only one character at a time. This is much slower than INPUT and PRINT, but it gives you more control over what you send and receive. You may alternate between the different types of BASIC input statements, and between the output statements, to the same port if necessary.

Concurrent Mode input, or Concurrent Mode (full-duplex) I/O, is performed by first opening the file for concurrent I/O, executing a START CONCURRENT MODE I/O operation, and then doing normal BASIC INPUT, GET, PRINT and PUT operations to that port. In CONCURRENT MODE I/O, after you have performed the START CONCURRENT MODE I/O command, I/O is going on at the same time BASIC is executing other commands for you. For example, if your RS-232-C compatible device is sending characters to the computer through the interface module, after the START CONCURRENT MODE I/O, those characters will be saved for your program as they arrive into a holding buffer by the computer. If you subsequently perform an INPUT statement to that port, the computer will just look in that buffer for the input data.

Output may be done either in Block Mode or in Concurrent Mode. When you do output in Block Mode, do *not* start Concurrent Mode I/O before doing the output. For this reason, full-duplex operation is not allowed with Block Mode output.

Block Mode output sends your data out to the interface module in 32-character blocks (whenever 32 characters have been collected by the handler from your PUT or PRINT statements). The computer waits while the interface module sends the block over the RS-232-C SERIAL INTERFACE port. Between blocks the computer's I/O port is not being used as it is when Concurrent Mode I/O is active, so if you use Block Mode there are no restrictions on using other I/O devices at the "same time."

Block Mode I/O is the only mode in which you can transmit 5-, 6- or 7-bit words (the word size option of the SET BAUD RATE command will not work with Concurrent Mode I/O output). 8-bit words may be transmitted in either Block or Concurrent Mode I/O.

Note: On rare occasions, the Operating System may resend a block to the interface module. This may result in part or all of the block being sent twice to the RS-232-C peripheral. To avoid this problem, use Concurrent Mode I/O ouput.

Concurrent Mode I/O output does not work in blocks. Instead, whenever your program tries to output any characters to the interface module, they are first moved into a 32-byte buffer. As long as there are characters in the buffer which have not been sent they will be sent as fast as possible. This "draining" of the buffer takes place concurrently with the execution of other BASIC statements in your program. The buffer will fill up when you try to output characters faster than they can be transmitted at the Baud rate you are using. Your program will be held up until space becomes available in the buffer. If you do not want your program to be held up, use the STATUS command to find out how much buffer space has been used and let your program use that information to decide whether or not to execute an output statement.

Concurrent Mode I/O input may be used at the same time you are using Concurrent Mode I/O output (full-duplex operation). Note that Block Mode output is **not** allowed if you do this. Also note that 5-, 6-, or 7- bit words can only be input in half-duplex mode. If you select anything other than 8-bit words you cannot output and input them at the same time. The 5-, 6-, and 7-bit words can be input at speeds up to 300 Baud.

Concurrent Mode I/O input data is placed in the input buffer as it is received from the RS-232-C SERIAL INTERFACE port. Your program must get the data out of the buffer with GET or INPUT before the buffer fills or data will be lost from the buffer. If the buffer fills, the data that has been in the buffer the longest will be replaced by the newer data. What your program will see is that characters are missing. The STATUS REQUEST command will tell you if data has been lost this way. STATUS REQUEST can be used to find out how many characters are in the input buffer, so you can program the machine to decide when to do an INPUT. STATUS can also be used to determine some kinds of errors in data reception and parity. This is fully described in Section 10, The STATUS Command.

LIST, SAVE, LOAD, AND ENTER

The BASIC statements LIST, SAVE, LOAD and ENTER perform "compound I/O" operations. Their operations can be thought of as consisting of combinations of other I/O operations. Each statement will input or output all or part of your program. This section defines how these statements work with the RS-232-C SERIAL INTERFACE ports.

Each of these statements can be thought of as consisting of first an OPEN, then one or more input or output operations, then a CLOSE. These operations do **not** include any configuration of RS-232-C SERIAL INTERFACE ports, and they do not include any START CONCURRENT MODE I/O action. Thus you cannot use the two input statements (LOAD and ENTER) with the RS-232-C SERIAL INTERFACE ports. Instead, enter your program as data to a program you write in ATARI BASIC and put the program on cassette or diskette. Then you can ENTER the program to ATARI BASIC from the cassette or diskette. The RECEIVE program on page 52 is an example of the type of program needed.

Since the configuration commands may be executed without opening an IOCB to a RS-232-C SERIAL INTERFACE port, you can configure the Baud rate, translation modes, and so forth, before you execute a LIST or SAVE to the port. (Saving a program to the RS-232-C SERIAL INTERFACE port will send the program in BASIC'S internal 8-bit tokenized format — this will probably be useful only if you are sending the program to another ATARI Home Computer System.) Since LIST has no implicit interface module status checking, the program will simply be sent out at the maximum rate allowed by the Baud rate you have selected. The receiving device must therefore be able to receive the data at that rate.

STARTING CONCURRENT MODE I/O

Use the command START CONCURRENT I/O (XIO 40) to start Concurrent I/O Mode. This mode may be used for output and must be used for input or full duplex. The port must be open before you can start Concurrent Mode I/O. Once Concurrent Mode I/O is in effect, no other I/O operations that use the computer I/O connector can be performed. Input/output operation to another serial port, for example, cannot be performed; I/O to the keyboard editor, the screen editor, and the controller jacks can still be performed.

Operations that are allowed while Concurrent Mode I/O is active are input and output operations to the active port (GET, INPUT, PUT, PRINT), STATUS commands to that port, and CLOSE commands to that port.

There are two different forms of the START CONCURRENT MODE I/O command. The main difference between them is that one specifies the use of a small input buffer built into the interface module handler (in the computer), and the other allows you to give your own buffer to the handler so it can be any size you wish. In Assembly Language these two options are really just different forms of the same command.

The form of the START CONCURRENT MODE I/O command which allows you to specify your own I/O buffer has two disadvantages. The command is complicated to specify in this form, and the BASIC array you use as the buffer may be moved by the BASIC interpreter. Once created, BASIC arrays are *not* moved while a program is being run, but arrays *are* moved whenever you add or delete a BASIC statement, even in the immediate mode.

The handler for the interface module is told of the location of the buffer only when you start Concurrent Mode I/O. If BASIC is allowed to move the array, data will be inserted in unpredictable locations. Ongoing concurrent input could wind up in other arrays or variables or even in your BASIC program, possibly destroying it. Therefore, if using user-program supplied buffers, it is imperative to close the Concurrent I/O when the program stops. See "Closing a Port," Page 31.

None of these problems occur if you use the buffer which is built into the interface module handler, since that buffer does not move. On the other hand, that buffer is quite small (32 bytes) and this may not be adequate for all programs.

With any size input buffer you need to GET or INPUT the data from the buffer before the buffer fills up with data that you have not yet read. Of course, if in the long-range average you read the data out of the buffer more slowly than it is arriving, you will eventually lose data anyway. If this is the case, you will either have to put up with losing it (which is not all that bad in some cases — see program example READING A DIGITIZER, Section 11), or you will have to figure out a way to slow down the device that is sending the data to you (such as setting a lower Baud rate). Even if your program processes the data fast enough in the long run, a small buffer puts demands on your program to get data quickly and often.

The BASIC interpreter can be quite slow relative to incoming data, if you want to do some processing on each and every character that comes in. In that case, even 300 Baud would be fast for BASIC. On the other hand, the system is more than fast enough to read in a line of data (terminated by CR) at 9600 Baud (960 cps) — as long as there is enough time between lines for your program to do its processing. It pays to read a whole line of input at a time (use INPUT wherever possible instead of GET), and it's really helpful if the inputting device will pause for you after each line. Even if the inputting device will not pause, inputting a whole line at a time may buy you the processing time you need. The best thing to do is try it.

Note: In order to perform line-oriented input using the BASIC INPUT statement, the input must either have an ATASCII EOL at the end of each input line, or an ASCII CR must terminate each line. In the latter case, you must configure the translation mode of the interface module port to convert the CR into EOL. This is discussed more fully in Section 6.

A large input buffer will be needed if you can read the data from the buffer only in large, occasional bursts. For example, if you do not know how long it will take to process a line of input because some lines require a lot of work, you will want to allow lines to "back up" in the input buffer. This will work fine as long as you do not get too many of these "slow" lines at once. You will probably have to determine the needed size of your input buffer by trial.

The number of characters that can come in every second depends on the Baud rate — the higher the Baud rate the faster characters can arrive. Thirty characters may arrive each second at 300 Baud; 480 may arrive in the same time at 4800 Baud. Of course, if the sending device does not run at the maximum possible speed — if there are "gaps" between characters anywhere — then the speed of the characters will not be important. Thus the Baud rate controls the *maximum* data transfer rate, but the actual or *effective* data transfer rate may be smaller.

What it amounts to is that your program in BASIC must INPUT data from the input buffer faster than the interface module puts them there from your RS-232-C compatible device; that is, your BASIC program must read the data faster than your device's effective data transmission rate (on average). You can control that rate by setting the Baud rate, and possibly there are other ways to control the transfer rate (that depends on the device itself). Be prepared to experiment to find the best mode of operation.

In BASIC, the START CONCURRENT MODE I/O operation which uses the built-in input buffer looks like this:

XIO 40, #IOCB, 0, 0, "Rn:"

Specify the appropriate open IOCB, and specify 1, 2, 3, or 4 for **n** in **Rn:**. The computer assumes port 1 for **R:**. You *must* specify zero for both Aux1 and Aux2, since this is the way the RS-232-C handler is told to use its own input buffer.

If you opened the port for output only, then only concurrent output is enabled. If the port is open for input only, then only concurrent input is started. If the port was opened for both, then Concurrent Mode input and output are started (full duplex). See Section 4 for details on how these various modes operate.

In BASIC, the START CONCURRENT MODE I/O operation in which you supply the input buffer for the handler is specified by a series of POKE statements followed by calling the Central I/O (CIO) through a USR function. The POKE statements specify the type of operation and specify the buffer address and length. Poke these values into the IOCB corresponding to the IOCB you have opened for the RS-232-C SERIAL INTERFACE port. Here is an example program:

```
10 DIM BUF$(500), RSTART$(7)
20 LET RSTART$ = "hhh*LVd":REM * and d are inverse video
30 LET FILE = 2
40 OPEN #FILE, 13, 0, "R4:"
50 LET IOCB = 16*FILE
60 LET BUF = ADR(BUF$)
65 LET BUFLEN = 499
70 LET RSTART = ADR(RSTART$)
80 POKE 832+IOCB+2, 40
90 POKE 832+IOCB+4, BUF-(INT(BUF/256)*256)
100 POKE 832+IOCB+5, INT(BUF/256)
110 POKE 832+IOCB+8, BUFLEN-(INT(BUFLEN/256)*256)
120 POKE 832+IOCB+9, INT(BUFLEN/256)
125 POKE 832+IOCB+10, 13
130 DUMMY = USR(RSTART,IOCB)
140 STARTSTATUS = PEEK(832+IOCB+3)
```

In this program, a full-duplex file is opened through IOCB #2 to RS-232-C SERIAL INTERFACE port number 4 (the 13 in line 40 specifies full duplex). Lines 50 through 70 set up some values that are used by the START CONCURRENT MODE I/O operation. The buffer is set up in lines 80 through 130. Line 140 gets the status value returned by the I/O call. Each POKE statement puts some needed value into the IOCB. The address to poke is specified as the sum of the following: the first address of the IOCBs (832), a value specifying which IOCB, and an "Offset" into the IOCB for the particular value you are poking. The value specifying the IOCB is 16 times the IOCB number through which you have opened the RS-232-C SERIAL INTERFACE port (in this case we set the variable IOCB to 32 in line 50, since the IOCB is #2).

The values poked into the IOCB are: 40 into Offset 2; the buffer location (address) into locations 4 and 5; the buffer length (minus one) into Offsets 8 and 9; and 13 into Offset 10. Pay special attention to the fact that the buffer address and the buffer length are both 2-byte values, requiring two pokes to put them into the IOCB. Those complex looking expressions in lines 90 through 120 are simply splitting the address and length into their low-part and high-part so each part can be poked individually.

Line 130 calls the I/O system through a USR function. This USR function has two arguments: the address of the function, and the IOCB specifier (the same as was used in specifying the poke locations). The address of this USR function was found in line 70, so you see that the function is the character array called RSTART$. The function itself is the odd-looking sequence of characters in line 20. Be sure to type this character sequence carefully before you call this USR function – any mistakes and your program will probably produce an unrecoverable failure.

Assembler note: This USR function is the following in Assembly Language: PLA; PLA; PLA; TAX; JMP $E456. The first four instructions get the IOCB number into the X-register, but leave the return address on the stack, so the I/O system is "called" by jumping to it at address E456.

Line 140 gets the I/O status after the USR I/O call. You do not need to get the status if you do not want to. To get status, PEEK at Offset 3 in the IOCB. The status will be 1 if all went well. Otherwise, the status is the same as the error number that BASIC prints after an I/O call fails. (Note that the variable DUMMY in the program above does not get any meaningful value.)

By using USR to call the I/O system you are bypassing any checking by BASIC of the success of the I/O. The only result of any error will be that Offset 3 in the IOCB will not have the value 1. Therefore, no BASIC error will occur and no TRAP you set will occur.

Once this START CONCURRENT MODE I/O operation has been performed, the concurrent I/O is active. The operation may be either in-only or out-only, or it may be full-duplex (as specified in the OPEN). If you are running full duplex or out-only, the output buffer is built into the interface module handler. The input and output buffers are accessed through normal input and output statements in BASIC. See Section 8 for details.

Remember, BASIC may move arrays around if your program stops. If Concurrent Mode input continues after your program stops, this may result in overwriting something outside your buffer array.

There is a 256-byte area at address 1536 (decimal) which you may use as an input buffer or anything else. Be sure that area is only being used for the one thing you wish. Most ATARI software does not use this area. However, if you plan to use this area as a buffer, read the instructions that come with your ATARI software. Address 1536 splits nicely into low and high parts (so does 256), so you could replace lines 90 through 120 of the program as shown on page 39:

```
                              :
                              :
                              :
90 POKE 832 + IOCB + 4, 0
100 POKE 832 + IOCB + 5, 6
110 POKE 832 + IOCB + 8, 0
120 POKE 832 + IOCB + 9, 1
                              :
                              :
                              :
```

If you use this area, you do not need to worry about it when your program stops since BASIC will not move it.

BASIC commands that will automatically close all of the IOCBs are: RUN, END, BYE, DOS, NEW, and ENTER. Allowing your program to "run off the end" also closes the IOCBs automatically.

If BASIC is allowed to close your IOCBs, it will close all open IOCBs according to IOCB number in descending order. However, since a CLOSE operation to a file usually involves serial port I/O, it is not allowed during Concurrent I/O (unless it is to the Concurrent I/O port itself). Therefore, since IOCB #7 is reserved for LPRINT and IOCB #6 is reserved for graphics, we strongly suggest that you use IOCB #5 for your Concurrent I/O and IOCBs numbered less than 5 for your program files. In that way Concurrent I/O will be closed before your files.

There are cases where your program may stop but the IOCBs will remain open. These are I/O error, program error, pressing the `BREAK` key when enabled, and the BASIC command STOP. The BASIC command CONT will not close any open IOCBs.

If you have followed the above suggestion using IOCB #5 and wish to stop Concurrent Mode I/O and close your files, you can enter the BASIC command END from the immediate mode. If you have not followed the suggestion, you must close each file individually with the BASIC command CLOSE (CLOSE #4, CLOSE #3, etc.). In any case, the Concurrent I/O *must* be closed first.

THE STATUS COMMAND

USES OF STATUS COMMAND

The STATUS command is useful for determining many facts about an RS-232-C SERIAL INTERFACE port and the state of the interface module. You can check for certain specific error conditions to find out why certain errors have occurred, to check parity, and so on. The STATUS command allows you to determine the amount of data in the input and output buffers while Concurrent Mode I/O is in effect. STATUS also allows you to check the state of the RS-232-C control lines DSR, CTS, CRX (and the state of RCV at the time you issue the STATUS command).

The STATUS command may be issued only through an IOCB opened to a RS-232-C SERIAL INTERFACE port. You may issue the command whether or not Concurrent Mode I/O is in effect. If this mode is in effect to a port, you cannot obtain status information (via the STATUS command) from any other port.

The information returned by a STATUS command is different according to whether or not Concurrent Mode I/O is in effect. When Concurrent Mode I/O is in effect, the STATUS command allows you to see how full your input and output buffers are, but you cannot check on the state of the control lines DTR, CTS, CRX and RCV. When Concurrent Mode I/O is not in effect, you get no information about buffers, but the state of the control lines can be checked. There are other minor differences in the effect of the STATUS command in the two cases.

In BASIC, the STATUS REQUEST command is implemented as a "compound" command — that is, you must code multiple BASIC statements to get the status. The first is the STATUS command. This is followed by uses of the PEEK function to retrieve status which is placed in a small status area by the STATUS command.

The STATUS command looks like this in BASIC:

STATUS #IOCB, avar

Here, #IOCB specifies the IOCB (1-7) through which you have opened the RS-232-C SERIAL INTERFACE port. You may issue this statement to the port before or after Concurrent Mode I/O is started.

Avar is a variable which will get the status *of the STATUS statement itself*. That is, avar will be set to the input/output system's one-byte status that is returned when BASIC calls the I/O system. Since the I/O system call here is STATUS, the value returned is the I/O system's determination about how the STATUS command went. This number is the same kind of number returned to BASIC by the I/O system after **any** I/O call, but in the other BASIC I/O statements, BASIC looks at the number itself to see if the I/O was completed without error. The STATUS command simply puts the number in the avar. This status number can be interpreted just like one of the ERROR codes — for example, you will get an ERROR 130 if you neglected to OPEN the IOCB, since an unopen IOCB does not specify any peripheral device and ERROR 130 means "Nonexistent Device Specified." The status number will be 1 if the STATUS call was completed without error. The status number will be some error number greater than 127 if there was some problem with the STATUS call.

If the STATUS call is successful, up to four bytes of information are stored in locations 746, 747, 748, and 749 (decimal). Location 746 always contains error status bits relating to the status history of the RS-232-C SERIAL INTERFACE port. The other three locations will contain buffer use information if Concurrent Mode I/O is active. If Concurrent Mode I/O is not active, 747 contains status bits relating to DSR, CTS, CRX, and RCV on the RS-232-C SERIAL INTERFACE port, and locations 748 and 749 hold nothing.

Table 10-1 shows the definition of the error bits in location 747. The table gives each bit a decimal value which shows how that bit, if "on" or 1 (as opposed to "off" or 0), adds to the total value of the byte when interpreted as a decimal number. The meaning of each of these error bits is discussed later in this section, but first here is a BASIC example showing how you can check one of the bits:

```
               :

               :

               :

160 STATUS -1, IGNORED

170 LET ERRORBITS = PEEK(746)/128

180 IF INT(ERRORBITS) < > INT(ERRORBITS+0.5) THEN PRINT
"OVERRUN!"

               :

               :

               :
```

In statement 160 the STATUS call is made to a dummy variable IGNORED. We do not use this variable, because we assume the STATUS call will work all right. The STATUS call must be made in order to put a value into location 746.

Statement 170 peeks at location 746. This value is then divided by twice the decimal number of the error bit being checked (this information is taken from Table 10-1). This value becomes the variable ERRORBITS. If the bit being check is 0, then adding 0.5 will not increase the integer part of the number. If the bit is 1, the integer part of the value changes when 0.5 is added.

In the above example, we are checking for the BYTE OVERRUN error. From the table, we find this to be 64. PEEK(746) is divided by twice 64 (128).

Statement 180 makes the comparison. If there is an error, OVERRUN! will be printed.

Table 10-1   Decimal Representation of the Error Bits in Location 746

Decimal Equivalent Error	Error
128	Received data framing error
64	Received data byte overrun error
32	Received data parity error
16	Received data buffer overflow error
8	Illegal option combination attempted
4	External device not fully ready flag
2	Error on block data transfer out
1	Error on command to interface module

Following are the descriptions of these error status bits in location 746 after STATUS command.

ERROR STATUS BITS

RECEIVED DATA FRAMING ERROR (bit 7, decimal value 128)

This error bit indicates that a framing error was encountered in the data coming from the external RS-232-C compatible device: the 10th bit of some character was not a STOP bit (9th, 8th or 7th in the cases of 7-, 6-, or 5-bit received words). This error can be caused either by garbled data (for instance, noise on the phone lines) or by improper configuration to receive the data (for example, wrong Baud rate).

This condition is monitored in one of two places: in the ATARI Home Computer, or in the ATARI 850 Interface Module. The computer watches for this error in the case of 8-bit data. The interface module catches this error if you are receiving 7-, 6-, or 5-bit data. In both cases, the error status is set at the time the erroneous character is received (not the time you read it out of the holding buffer).

In the 8-bit data case, where the computer monitors the error, you may find out about the error any time after it occurs by issuing STATUS while the Concurrent Mode input is active. The error bit will be cleared when you issue the STATUS command or when you CLOSE the Concurrent Mode IOCB. In the 7-, 6-, and 5-bit cases, the error is monitored by the interface module and cannot be interrogated while the Concurrent Mode input operation is active. In this case, you must close and reopen the Concurrent Mode IOCB and then issue STATUS to determine if the error occurred. The error bit in the interface module is cleared by STATUS when Concurrent Mode I/O is not active. It is also cleared by most of the configuring and control XIO's (but not all), and it may be cleared by CLOSE when Concurrent Mode I/O is not active.

In general, the error bits read from location 746 after a STATUS request apply only to the most recent I/O operation to the RS-232-C SERIAL INTERFACE port; that is, they are cleared as the I/O operation is started and then set if the error occurs. You can see that the previous error is an exception to this rule.

RECEIVED DATA BYTE OVERRUN ERROR (bit 6, decimal value 64)

This error bit is maintained by the computer and indicates that the computer got too busy to read all the data as it was arriving (due to overly heavy interrupt loading, or perhaps interrupts being masked off totally). This error is flagged when the first character of data following the error is read from the port and placed in the holding buffer. The error should not occur at all under normal conditions.

RECEIVED DATA PARITY ERROR (bit 5, decimal value 32)

This error bit is maintained by the computer and indicates that a received character had the wrong parity. The bit will not be set if no parity checking has been enabled. This error occurs during the translation from the external (received) form of the character to the internal (INPUT, GET) form, which takes place as the data is read out of the holding buffer. The error flag bit is cleared by the STATUS command.

RECEIVED DATA BUFFER OVERFLOW ERROR (bit 4, decimal value 16)

This error flag indicates that more data has arrived than can be held in the input buffer — data has not been read from the buffer (INPUT, GET) soon enough. This error is maintained by the computer, and it occurs when the overflowing character arrives from the RS-232-C compatible device. The new character replaces the oldest one in the buffer. This error bit is cleared by the STATUS command.

ILLEGAL OPTION COMBINATION ATTEMPTED (bit 3, decimal value 8)

This error flag is kept in the interface module and may be read by STATUS only if Concurrent Mode I/O is not active. It is set by an attempt to start Concurrent Mode input with short words (7-, 6-, or 5-bit) with the port open for both input and output or output only (short words are allowed in only) or too high a Baud rate (short words are allowed for input at a maximum rate of 300 Baud). This error may be checked immediately after the interface module produces a NAK (Error 139, which may be trapped) for the refused command. The bit is cleared by the STATUS request. Error bit zero (command error, decimal value 1) will always be set when this bit is set.

EXTERNAL DEVICE NOT FULLY READY (bit 2, decimal value 4)

This bit is kept in the interface module and may be read by STATUS only when Concurrent Mode I/O is not active. It is set whenever a START CONCURRENT MODE I/O or block output command is refused by the interface module because one or more of the external status lines being monitored is not ON. Any of the external status lines not being monitored (as set by the SET BAUD RATE command) is ignored; if none is being monitored this bit will not be set and the I/O operation will proceed normally. Read this flag bit with a STATUS request immediately after the interface module refuses the operation with Error 139, which can be trapped. This flag is cleared by the STATUS command.

DATA BLOCK ERROR (bit 1, decimal value 2)

This error bit is maintained in the interface module and may be read by STATUS immediately after a command is refused by Error 139, which can be trapped. In a block output, the data block was unsuccessfully received from the computer by the interface module. This error should not occur in normal operation; it indicates problems in communication between the computer and interface module.

COMMAND ERROR TO INTERFACE MODULE (bit 0, decimal value 1)

This error bit is maintained in the interface module and may be read by STATUS immediately after a command is refused by an Error 139 from the interface module. This bit indicates that the interface module did not recognize a command sent to it from the computer, or that the interface module is not configured properly to perform the command (see ILLEGAL OPTION COMBINATION ERROR).

BUFFER CHECKING

During active Concurrent Mode I/O, the STATUS command will return the number of characters in the input buffer in locations 747 and 748, and the number of characters in the output buffer in location 749. To find the number of characters in the input buffer in ATARI BASIC:

    LET BUFFERUSE = PEEK(747) + 256 * PEEK(748)

If you only want to find out whether or not there are characters in the input buffer, you do not need to multiply by 256:

    IF PEEK(747) + PEEK(748) = 0 THEN input buffer empty...

or:

    IF PEEK(747) + PEEK(748) < > 0 THEN input buffer not empty...

If you are using the built-in buffer, or if your supplied buffer has fewer than 256 bytes, then location 748 will always be zero and you need to look only at location 747. The output buffer holds only 32 characters; location 749 will never exceed 32.

When Concurrent Mode I/O is not active, location 747 will contain information about the readiness lines (DSR, CTS, and CRX) and the data receive line (RCV) of the specified port after a STATUS request. (Locations 748 and 749 will not contain anything useful after a STATUS request when there is no active Concurrent I/O.) Location 747 will contain the sum of four numbers, shown in Table 10-2. The current and past status of DSR, CTS, and CRX as well as the current status of RCV are included. The past status of DSR, CTS, and CRX applies back to the time the interface module was booted, or to the most recent STATUS command to the specified port which was made while Concurrent Mode I/O was not active (i.e., the last time that DSR, CTS, and CRX were supplied to a STATUS request). No other operations affect the past status of these lines, which is supplied by STATUS. In particular, whether or not you enable readiness checking before I/O (in the SET BAUD RATE command) will have no effect on the information supplied by STATUS.

Ports 2 and 3 will always show CTS and CRX as being ON. Port 4 will show CTS, CRX, and DSR as being ON.

This is a quick way to check whether or not a port is ready:

    STATUS #n, XXX IF PEEK(747) < 128 THEN *not ready* ...

or to check if it has stayed ready since the last check:

    IF PEEK(747) > = 192 THEN *always ready* ...

In other words, the DSR status bits are the most significant bits in the sense byte, and you can check them this way without having to worry about the states of the other bits in the byte.

Table 10-2   Sense Values Added Into Location 747

DATA SET READY (DSR)
192   Ready now (ON); on since previous STATUS
128   Ready now (ON); not always on since last STATUS
 64   Not ready now (OFF); not always off since last STATUS
  0   Not ready now (OFF); always off since last STATUS

CLEAR TO SEND (CTS)
 48   Clear now (ON); on since previous STATUS
 32   Clear now (ON); not always on since last STATUS
 16   Not clear to send now (OFF); not always off since last STATUS
  0   Not clear to send now (OFF); always off since last STATUS

CARRIER DETECT (CRX)
 12   Carrier now (ON); on since previous STATUS
  8   Carrier now (ON); not always on since last STATUS
  4   No carrier now (OFF); not always off since last STATUS
  0   No carrier now (OFF); always off since last STATUS

DATA RECEIVE (RCV)*
  1   MARK (1) now
  0   SPACE (0) now

*No information is supplied about the past status of RCV.

SAMPLE PROGRAMS

TRANSFERRING BASIC SOURCE PROGRAMS

This section describes a pair of programs that can be used to transfer information from one ATARI 800 Home Computer to another over the telephone. These two programs demonstrate an example of the technique called "handshaking." Handshaking, which was described in Section 1, is an overextended term in the computer world. What is meant here is that the receiving program will respond to the sender with an "I've got it!" message when it has successfully received each line of information from the sender.

The trick here is that the sending program must not miss the "I've got it!" message. Likewise, the receiving program must not only have got the line when it says "I've got it," but the receiver must be ready to receive the next line immediately because, theoretically, the sender might send the next line immediately. These programs show how this is done.

Both programs operate on one line (up to 255 characters) at a time. Each program starts by dimensioning its line array, and each asks its user for the filename to be sent/received. Each program then opens its modem port (R1:) and disk file (assuming the send/receive files are disk files).

The RECEIVE program must be started first, in order to be ready for the sender's first line. The SEND program will send the first line with no prior signal from the RECEIVE program.

In line 540, the SEND program gets a line from the disk file. The program then prints the line on the television screen (so you can watch the data being sent). Then, in lines 570-590, the line is sent over the phone. Note that port R1: is opened full duplex: SEND assumes when RECEIVE gets the line, that there might be an immediate reply. (Of course, this can't happen but it's best to write the program as though it could.) In line 600, SEND waits for the reply (a line that is empty except for an EOL is used as a reply).

The RECEIVE routine, meanwhile, has set itself up to get a line from the modem (lines 280-290, 530). When line 530 completes (the line of data has been received), RECEIVE closes the modem port (R1:) in order to save the data on the diskette (lines 540, 580) and echoes the data on the television screen (line 590). Then RECEIVE opens the modem again and sends the reply (lines 610-630). Note that port R1: is opened full duplex: RECEIVE assumes that it might start getting the next line immediately after it has sent its reply. Note also that it is not necessary for RECEIVE to INPUT the data immediately, but it is necessary that RECEIVE have started the Concurrent Mode data receive (line 620).

When SEND gets the reply, it knows it can safely close the modem port (R1:) to get another line of data from its diskette (lines 600-610). It then goes back to get another line of data (lines 530-540) and the whole cycle repeats. Note how the SEND program checks for the end of the disk file and how it sends a specially encoded line (EOF EOF EOF) to the RECEIVE program to signal this. Also note that both programs explicitly close their files.

To use these programs, assume that you and your friend are talking on the phone and have prepared your computers (you have loaded your SEND program and your friend has the RECEIVE program). You each RUN your programs, and each program gets a filename from each of you — type the name but don't yet press RETURN . Now one of you sets this modem to **A** Answer and the other sets his to **O** Originate. Looking at your watches, you decide that your friend will press his RETURN key as soon as the READY light comes up on his modem and you will press your RETURN key 10 seconds later. In other words, the RECEIVE program must be ready to receive before the SEND program sends the first line. Now you each put your phone handsets in the modem muffs and you proceed to send a program to your friend.

Since these programs work on LINES of data, you cannot send tokenized BASIC. You should send BASIC source, that is, send a file you saved on the diskette with the LIST (not SAVE) command. Your friend should ENTER the file he receives (not LOAD). You may modify these programs to send and receive the information one character at a time (using GET and PUT instead of PRINT and INPUT), doing the handshake every 40 characters or so. You'll have to pay particular attention to the question of sending the end-of-file information if you try this modification; however, such a modification should allow you to send any kind of data, not just lines of text.

The RECEIVE program will probably need modification if you intend to put the received information on cassette. The cassette handler requires that the first 128-byte record be written within about 30 seconds after you OPEN the cassette for output. A little experimentation should get you going.

Lines 210 and 220 will have to be modified if these programs are to be used for communications with computers other than an ATARI Home Computer.

SEND PROGRAM

```
110 DIM OUTLINE$(255)
200 REM
201 REM ==========
202 REM
210 LET TRANSLATE=32:REM Full ATASCII
220 XIO 38,#5,TRANSLATE,0,"R1:"
230 REM
240 PRINT "Send file's full name";
250 INPUT OUTLINE$
260 OPEN #1,4,0,OUTLINE$
500 REM
501 REM ==========
502 REM
510 FOR ETERNITY=0 TO 0 STEP 0
520 REM
530 TRAP 900:REM Trap END OF FILE
540 INPUT #1;OUTLINE$:REM Get line
550 PRINT OUTLINE$:REM Echo onscreen
560 REM
570 OPEN #5,13,0,"R1:"
580 XIO 40,#5,0,0,"R1:":REM Start I/O
590 PRINT #5;OUTLINE$:REM Send line
600 INPUT #5;OUTLINE$:REM Get reply
610 CLOSE #5:REM Stop I/O
620 REM
630 NEXT ETERNITY
900 REM
901 REM ==========
902 REM
910 OPEN #5,8,0,"R1:":REM Send EOF
920 PRINT #5;"EOF EOF EOF"
930 CLOSE #5:CLOSE #1:REM All done
999 END
```

RECEIVE PROGRAM

```
110 DIM INLINE$(255)
200 REM
201 REM = = = = = = = = = =
202 REM
210 LET TRANSLATE=32:REM Full ATASCII
220 XIO 38,#5,TRANSLATE,0,"R1:"
230 REM
240 PRINT "Receive file's full name";
250 INPUT INLINE$
260 OPEN #2,8,0,INLINE$
270 REM
280 OPEN #5,13,0,"R1:"
290 XIO 40,#5,0,0,"R1:":REM Start I/O
500 REM
501 REM = = = = = = = = = =
502 REM
510 FOR ETERNITY=0 TO 0 STEP 0
520 REM
530 INPUT #5;INLINE$:REM Get line
540 CLOSE #5:REM Stop I/O
550 REM
560 IF INLINE$="EOF EOF EOF" THEN 900
570 REM
580 PRINT #2;INLINE$:REM Save line
590 PRINT INLINE$:REM Echo onscreen
600 REM
610 OPEN #5,13,0,"R1:"
620 XIO 40,#5,0,0,"R1:":REM Start I/O
630 PRINT #5:REM Send reply
640 REM
650 NEXT ETERNITY
900 REM
901 REM = = = = = = = = = =
902 REM
910 CLOSE #2:REM EOF received
999 END
```

BAUDOT TERMINAL EMULATOR

Here is a sample program showing the use of odd character transmission sizes and non-ATASCII (also non-ASCII) character codes. This program turns your ATARI Home Computer into a **Baudot** teletype emulator.

Warning: The ATARI 850 Interface Module was not designed for connection to old teletype equipment. Such equipment used 60 milliamp current loops rather than the more modern 20 milliamps. High voltages could be present in such old equipment. These voltages could be dangerous to you and could damage your interface module. This program is intended to allow you to communicate, via a modem, over a telephone or radio link with someone owning a Baudot teletype.

The Baudot code is an old 5-bit serial code which is actually two codes in one. Half of the characters in Baudot are in the LETTERS SHIFT category and half are in the NUMBERS SHIFT category. The latter category includes digits 0 - 9 and some special characters. This program takes care of sending and receiving the shifting control characters.

This program is actually much simpler than it looks. In lines 110 - 210, the program's *symbolic constants* and starting values are set up. The *symbolic constants* are values which are not changed in the program, but for readability they are represented symbolically (as variables). Constants include: logical constants (YES and NO); PEEK and POKE addresses (SWITCH, KB); character constants (RETURN, FEED, UPSHIFT, DOWNSHIFT); BASIC line number constants for GOSUBs and GOTOs (RECEIVE, SEND, and TESTSWITCH); and useful numbers (NOPUSH, NOKEY). Setting INSHIFT to zero establishes LETTERS SHIFT for received data; setting ALPHA to YES establishes LETTERS SHIFT for sent data; and setting TALK to NO establishes LISTEN mode.

Lines 300 - 390 fill in the ASCII-Baudot translation tables from the data values in lines 2000 - 2460. Remarks are interspersed in the data to show what character is being translated. Notice that all the characters are represented within this program as numbers — the number is the "internal" character code for the corresponding letter (this is true for both ATASCII and Baudot, but, of course, the numbers representing a particular letter are different for each).

In order to make the code conversion easy, the translation mode is set to 32 — no translation. The Baud rate is set to 45.5 Baud (60 wpm). This is the most common speed for old Baudot equipment.

Lines 500 - 650 are the receive routine. The computer informs you that you are entering Listen Mode, then opens the RS-232-C SERIAL INTERFACE port R3: for input and starts the Concurrent Mode input (510 - 540). The receive loop (560-650) first does a GOSUB TESTSWITCH to check for switching to Send Mode (TESTSWITCH is discussed later). The STATUS and IF PEEK... statements (580 -585) see if there are any characters received. If there are, a character is input in line 590 and translated to ASCII in lines 600 - 630, and printed to the television screen in line 640. ATASCII table values less than zero mean untranslatable characters; 0 means the LETTERS SHIFT character is received; 1 means NUMBERS SHIFT.

Lines 700 - 950 are the send routine. Talk mode is announced, and port R3: is opened for output. The first send loop (750 - 950) action is a GOSUB TESTSWITCH. Line 770 checks for the typing of a keyboard key. In lines 780 - 800 the key's value is retrieved and its high bit is stripped (it is forced to be less than 128 — this has the effect of disregarding inverse video and allows the conversion to table to require only 128 elements). The key is translated in line 810; if it translated to zero, that means it has no Baudot equivalent and line 820 restarts the loop. Otherwise, it is echoed to the television screen (830); it then undergoes further translation in lines 840 - 890, where a LETTERS or NUMBERS shift character is added if needed. Line 900 sends the character itself, and if it was RETURN, lines 920 - 930 add a LINEFEED and LETTERS SHIFT.

The TESTSWITCH routine (lines 1000 - 1060) checks whether one of the yellow buttons is pushed (START, SELECT or OPTION). If not pushed, TESTSWITCH just returns. Otherwise, the subroutine waits for the button to be released, restores BASIC's GOSUB/FOR-NEXT stack, flips from SEND to RECEIVE mode (or vice-versa) and does a GOTO to the proper routine.

In operation, the 32-character internal buffer fills with characters to be sent. When the buffer is full, the interface module sends the characters as a block. While the characters are being sent, the keyboard will accept one character (which you won't see on the screen), so you should type the next character you want to send and wait for it to appear on the television screen. Note that this program, as written, sends the block immediately when you type RETURN. You may want to experiment with variations, such as sending each character as it is typed from the keyboard (using the FORCE SHORT BLOCK) or reading a line at a time (this allows you to use backspace to correct your typing, but the person at the other end of the connection won't see anything except when you type RETURN).

```
110 DIM ATASCII(64),BAUDOT(128)
120 REM
121 REM Set up constants...
122 REM ——————————-
130 LET YES=1:NO=0
140 LET SWITCH=53279:NOPUSH=7
150 LET KB=764:NOKEY=255
160 LET RETURN=8:FEED=2
170 LET UPSHIFT=27:DOWNSHIFT=31
180 LET RECEIVE=500:SEND=700
190 LET TESTSWITCH=1000
200 REM
201 REM Starting values...
202 REM ————————
210 LET INSHIFT=0:ALPHA=YES:TALK=NO
300 REM
301 REM Fill Baudot to ATASCII table...
302 REM ————————————-
310 FOR I=1 TO 64
```

```
320 READ IN
330 LET ATASCII(I) = IN
340 NEXT I
350 REM
351 REM Fill ATASCII to Baudot table...
352 REM ————————————————
360 FOR I = 1 TO 128
370 READ IN
380 LET BAUDOT(I) = IN
390 NEXT I
400 REM
401 REM Set up I/O...
402 REM ——————-
410 LET BAUD = 128 + 48 + 1:TRANSLATE = 32
420 XIO 36,#5,BAUD,0,"R3:"
430 XIO 38,#5, TRANSLATE,0,"R:3"
440 REM
450 OPEN #1,4,0,"K:"
500 REM
501 REM Receive routine...
502 REM —————————
510 PRINT:PRINT "Listen..."
520 REM
530 OPEN #5,5,0,"R3:":REM Input
540 XIO 40,#5,0,0,"R3:":REM Start
550 REM
551 REM Receive loop...
552 REM ————————-
560 FOR INLOOP = 0 TO 0   STEP 0
570 GOSUB TESTSWITCH
580 STATUS #5,PORT4
585 IF PEEK(747) = 0 THEN NEXT INLOOP
590 GET #5,IN
600 LET IN = ATASCII(IN-224 + INSHIFT + 1)
610 IF IN < 0 THEN NEXT INLOOP
620 IF IN = 0 THEN INSHIFT = 0:NEXT INLOOP
630 IF IN = 1 THEN INSHIFT = 32:NEXT INLOOP
640 PRINT CHR$(IN);
650 NEXT INLOOP
```

```
700 REM
701 REM Send routine...
702 REM ————————-
710 PRINT:PRINT "Talk..."
720 REM
730 OPEN #5,8,0,"R3:":REM Output
740 REM
741 REM Send loop...
742 REM ——————
750 FOR OUTLOOP=0 TO 0   STEP 0
760 GOSUB TESTSWITCH
770 IF PEEK(KB)=NOKEY THEN NEXT OUTLOOP
780 GET #1,KEY
790 LET OUT=KEY
800 IF OUT>127 THEN LET OUT =OUT -128
810 LET OUT=BAUDOT(OUT+1)
820 IF OUT=0 THEN NEXT OUTLOOP
830 PRINT CHR$(KEY);
840 IF ALPHA THEN 880
850 IF OUT<0 THEN 900
860 LET ALPHA=YES:PUT #5,DOWNSHIFT
870 GO TO 900
880 IF OUT>0 THEN 900
890 LET ALPHA=NO:PUT #5,UPSHIFT
900 PUT #5,ABS(OUT)
910 IF OUT< >RETURN THEN NEXT OUTLOOP
920 PUT #5,FEED:PUT #5,DOWNSHIFT
930 XIO 32,#5,0,0,"R3:"
940 LET ALPHA=YES
950 NEXT OUTLOOP
1000 REM
1001 REM Listen/Talk switch test...
1002 REM ——————————————
1010 IF PEEK(SWITCH)=NOPUSH THEN RETURN
1020 IF PEEK(SWITCH)< >NOPUSH THEN 1020
1030 POP:POP:REM Pop GOSUB & FOR-loop
1040 CLOSE #5
1050 IF TALK THEN TALK=NO:GO TO RECEIVE
1060 LET TALK=YES:GO TO SEND
2000 REM
```

```
2001 REM Baudot to ATASCII table...
2002 REM ——————————————
2010 REM NUL,E,LINEFEED,A,SPACE,S,I,U
2020 DATA -1,69,-1,65,32,83,73,85
2030 REM RETURN,D,R,J,N,F,C,K
2040 DATA 155,68,82,74,78,70,67,75
2050 REM T,Z,L,W,H,Y,P,Q
2060 DATA 84,90,76,87,72,89,80,81
2070 REM O,B,G,Numbers, M,X,V,Letters
2080 DATA 79,66,71,1,77,88,86,0
2090 REM NULL,3,LF,-,SPACE,BELL,8,7
2100 DATA -1,51,-1,45,32,253,56,55
2110 REM RETURN,$,4,',COMMA,!,:,(
2120 DATA 155,36,52,39,44,33,58,40
2130 REM 5,'',),2,#,6,0,1
2140 DATA 53,34,41,50,35,54,48,49
2150 REM 9,?,+,Numbers,.,/,;,Letters
2160 DATA 57,63,43,1,46,47,59,0
2200 REM
2201 REM ATASCII to Baudot table...
2202 REM ——————————————
2210 REM Graphics characters incl. CR
2220 DATA 0,0,0,0,0,0,0,0
2230 DATA 0,0,0,0,0,0,0,0
2240 DATA 0,0,0,0,0,0,0,0
2250 DATA 0,0,0,8,0,0,0,0
2260 REM SPACE,!,'',#,$,%,&,'
2270 DATA 4,-45,-17,-20,-9,0,-26,-11
2280 REM (,),*,+,COMMA,-,.,/
2290 DATA -15,-18,0,-26,-12,-3,-28,-29
2300 REM 0,1,2,3,4,5,6,7
2310 DATA -22,-23,-19,-1,-10,-16,-21,-7
2320 REM 8,9,:,;,<,=,>,?
2330 DATA -6,-24,-14,-30,0,0,0,-25
2340 REM @,A,B,C,D,E,F,G
2350 DATA 0,3,25,14,9,1,45,26
2360 REM H,I,J,K,L,M,N,O
2370 DATA 20,6,11,15,18,28,12,24
2380 REM P,Q,R,S,T,U,V,W
```

```
2390 DATA 22,23,10,5,16,7,30,19
2400 REM X,Y,Z,Graphics characters
2410 DATA 29,21,17,0,0,5,0,0
2420 REM A - Z again
2430 DATA 0,3,25,14,9,1,45,26
2440 DATA 20,6,11,15,18,28,12,24
2450 DATA 22,23,10,5,16,7,30,19
2460 DATA 29,21,17,0,0,5,0,0
9999 END
```

PROGRAMMING A PRINTER

Here are two examples of programming printers connected serially through RS-232-C SERIAL INTERFACE ports. It is assumed that there are fundamental differences between the two — the characteristics of each printer control how that printer must be programmed. These two sample programs (or program fragments) are not intended to show general techniques, but are examples of how certain specific needs can be met.

The printer being programmed here is able to buffer and hold characters ahead of its printing (or it is so fast that it is always ready to accept characters to print). When it does not want you to send more data, it sets a READY line OFF; that line is connected here to the DSR pin on the RS-232-C SERIAL INTERFACE port. However, the printer sets its READY line OFF early — it is still able to collect up to 32 characters after it says it's full. In other words, since the RS-232-C SERIAL INTERFACE ports send data out in blocks of up to 32 characters, it is only necessary to monitor the DSR line once per block.

The automatic monitoring of DSR once per block is set up in line 150. In line 160, we tell the interface module to add LF to each CR (this printer wants the LF).

When a block is about to be sent, the interface module checks DSR (per our request). If it is OFF, the resulting NAK error is trapped (line 360), and in the TRAP routine (900 etc.) the program checks that the TRAP was really caused by the DSR being OFF. If this was the cause, the PRINT is simply retried — eventually it will succeed because the printer will become ready again.

```
                    :
                    :
                    :

140 OPEN #5,8,0,"R2:"
150 XIO 36,#5,0,4,"R2:":REM Monitor DSR
160 XIO 38,#5,64,0,"R2:":REM Add LF to CR
                    :
                    :
                    :

360 TRAP 900
370 PRINT #5;.... REM PRINT something to R2:
                    :
                    :
```

:

```
900 STATUS #5,PORT2:REM Get R2: status
910 LET READY = PEEK(746)/8:REM Check readiness error
920 IF INT(READY) < > INT(READY + 0.5) THEN 360:REM If so, retry
930 REM If here then some error other than port-not-ready
```

:

:

:

The printer being programmed in the example below also has a READY line to signal that it is not ready to accept data. However, when it is not ready, it cannot accept any data. Therefore, the data must be sent to the printer one character at a time, checking DSR before each character. Since the PRINT statement cannot be made to send data one character at a time, we assume that the file to be printed was first written to a diskette or cassette. Here is a program to read that file off the diskette or cassette and print it on this printer.

The operation of this program should be fairly obvious. Once again, we assume the printer wants both CR and LF at the end of a line (lines 230 - 240). The file is read from diskette (or tape) one character at a time. Then if the printer on port R2: is ready (540 - 550), the character is PUT (560). The output is then forced (FORCE SHORT BLOCK) in line 570.

```
110 DIM FILE$(16)
200 REM
201 REM = = = = = = = = = =
202 REM
210 LET BAUD = 13:REM 4800 Baud
220 XIO 36,#5,BAUD,0,"R2:"
230 LET TRANSLATE = 64:REM Add LF to CR
240 XIO 38,#5,TRANSLATE,0,"R2:"
250 REM
260 PRINT "List file's full name";
270 INPUT FILE$
280 OPEN #1,4,0,FILE$
290 REM
300 OPEN #5,8,0,"R2:"
500 REM
501 REM = = = = = = = = = =
502 REM
510 FOR ETERNITY = 0 TO 0 STEP 0
520 TRAP 900:REM Trap end of file
530 GET #1,CHARACTER
```

```
540 STATUS #5,XXX:REM Check ready
550 IF PEEK(747)<128 THEN 540
560 PUT #5,CHARACTER
570 XIO 32,#5,0,0,"R2:"
580 NEXT ETERNITY
900 REM
901 REM ==========
902 REM
910 CLOSE #5:CLOSE #1
920 END
```

READING A DIGITIZER

This is an example of reading data from a digitizing pad. A digitizing pad is a device that is capable of sensing the position of a handheld object (a special pen or whatever) and reporting its location to the computer.

The digitizing pad used in this example is capable of sending its information to the computer at speeds up to 4800 Baud, so that Baud rate is used here. Each sampled pen position is 14 characters long: a digit indicating whether or not the button on the pen is being pushed, the x-coordinate (6 characters), the y-coordinate (6 characters), a CR and a LF. Since the LF follows the CR, the interface module will read it as the first character on the following input line.

If we assume that the digitizer sends the pen coordinates as fast as it can, then BASIC will not be able to keep up at 4800 Baud. A lower Baud rate might allow BASIC to get every sample, but at 300 Baud, for example, it would take about half a second for each sample to come in (15 characters at 30 cps). Thus we want the data to come in at the highest possible rate. It really doesn't matter if we miss samples, because the pen is usually in pretty much the same place sample after sample.

Therefore, it is all right if the digitizer sends samples as fast as it can and the program just grabs them now and then when it can. However, take into account the way the interface module behaves when data arrives too fast: when the computer's holding buffer fills up, the newest data replaces the oldest. An INPUT statement reads the oldest data — which is messed up by being replaced by the newer data!

This is actually very trivial to solve. Look at line 100. A sample is INPUT twice. The first INPUT gets the messed-up sample which has been written over by new data. Then the second INPUT gets a sample from the buffer which is unharmed. (This works because the sample contains enough characters to allow an INPUT to get significantly ahead of the arriving character stream, and because the sample contains fewer characters than the holding buffer.)

Lines 110 - 130 extract the coordinates from the sample. It was not possible to use an INPUT statement with these number variables because the sample does not have commas between the sample numbers. The details of what the program does with the samples is not shown (in order to keep the example to the important points).

```
10 DIM IN$(16)
20 XIO 36,#5,13,0,"R2:"
30 OPEN #5,5,0,"R2:"
40 XIO 40,#5,0,0,"R2:"
            :
            :
            :

100 INPUT #5,IN$:INPUT #1,IN$
110 LET BUTTON=VAL(IN$(2,2))
120 LET X=VAL(IN$(3,8))
130 LET Y=VAL(IN$(9,14))
            :
            :
            :
590 GO TO 100:REM Get next point
            :
            :
            :
```

INTERFACE MODULE
ELECTRICAL SPECIFICATIONS

RS-232-C STANDARD

RS-232-C is the standard adopted by the Electronic Industries Association (EIA) to ensure the uniformity of transmission of data between data communications equipment and data processing terminals. This standard is followed by most equipment manufacturers.

The RS-232-C standard defines a range of values of electrical parameters for a communication link. The ATARI 850 Interface Module is the device used in ATARI Home Computer to adapt to the values of these parameters. The interface module organizes the bit stream of communication according to software-coded intructions.

When we refer to a communication port as a RS-232-C SERIAL INTERFACE port, we mean that signals to or from that port conform to the RS-232-C standards. We also use the adjective "RS-232-C-compatible" when the communication conforms to essential aspects of the RS-232-C standard. Perhaps the most important aspect of the standard is the specification of voltage levels corresponding to mark and space. Accordingly, many other publications may use the term "RS-232-C compatible" to mean "using the voltage levels in the RS-232-C standard."

RS-232-C SPECIFICATIONS

RS-232-C compatibility has come to cover many devices that are not "data sets" or "data terminals," particularly in the personal computer world. This usually means the device conforms to the electrical RS-232-C specification, which is shown in Table 12-1. Sometimes such devices (which include printers, plotters, digitizing pads, and many other devices) also have lines that are called DSR, DTR, RTS - and so on. However, their use is often different from the use covered by the RS-232-C standard and usually the use is specific to the device. One such use is to signal readiness to accept data from your computer (as opposed to sending XOFF/XON over a data line). Unfortunately, there is no standard of how many characters the device will accept after the line goes OFF, nor a good way to determine where to start up again when the device becomes ready (if characters have been lost). You will have to familiarize yourself with your device's characteristics and then program your ATARI Home Computer and the interface module accordingly.

Table 12-1   RS-232-C Electrical Specifications

TYPE OF SIGNAL	FIRST STATE −24 volts to −3 volts	SECOND STATE +3 volts to +24 volts
Binary signal	1	0
Signal condition	MARK	SPACE
Control function	OFF	ON

It is common practice when using the 25-pin D-connector most used with RS-232-C to connect XMT to pin 2, RCV to 3, RTS to 4, CTS to 5, DSR to 6, common signal ground to 7, CRX to 8, and DTR to 20. However, these conventions may not be followed; you may also run into cases where the other pins in the connector have either entirely unrelated functions (such as other types of communication standards on the same connector) or possibly related functions (such as setting the Baud rate by connecting two pins). *Carefully read the instructions of any device you intend to connect to the ATARI 850 Interface Module!* You may have to make your own cable to connect the device to the interface module.

The RS-232-C standard does not specify how data should be transmitted on XMT and RCV. In fact, RS-232-C explicitly avoids this issue. Fortunately, common convention and other standards have settled on a fairly universal serial data transmission convention. When data is not being sent, the data line sits idle in the MARK state. A data character (sometimes called a transmission WORD) is signalled by one START BIT, represented by the SPACE state. It is followed by the data bits (most commonly 8 of them), each bit being represented by SPACE for 0 and MARK for 1. The word is terminated by 1 (sometimes 2) STOP BIT(s), represented by the MARK state. The next word can immediately follow with its start bit. If it does not, the line stays idle in the MARK state (effectively, the stop bit lasts indefinitely).

The data bits are sent least-significant first. The bit numbered 0 is sent first, 1 next, and so on. The receiver does not know when a character will be coming, so it constantly monitors the stopped MARK state looking for the transition to a start bit. The receiver can then receive the rest of the bits in the word because it knows when each will arrive — each bit has the same duration as established by the Baud rate of the communication. Of course, both the transmitter and receiver must use the same Baud rate.

There are only a small number of common Baud rates, and the ATARI 850 Interface Module supports all of the most common ones. The most common transmission word size is 8 bits; when sending ASCII, which is a 7-bit code, the 8th bit usually represents the parity, is just set to 1 or 0, or is used as a marker bit of some sort. ASCII is very occasionally sent in 7-bit words. The interface module supports 7-bit words for these cases, and can also be used for communication with 7-bit or 6-bit codes such as BCD (with or without parity). Five-bit words are also allowed so you can communicate with old Baudot code teletypes for radioteletype and similar uses.

ELECTRICAL SPECIFICATIONS OF THE SERIAL PORTS

Refer to the interface module schematic diagram in Appendix C while reading this section.

There are basically two types of circuits for the serial port lines: a receiving circuit, and a transmitting circuit. One of these circuits connects each RS-232-C signal line to a pin of one of the two computer I/O chips in the interface module.

The sending circuit consists of an operational amplifier (op-amp) followed by a 10-ohm protective resistor. The op-amp is driven "to the rail," and produces approximately + 9 volts for SPACE, and − 5.5 volts for MARK (guaranteed at least + or − 5 volts), when driving a 3000-ohm load (3000 ohm is the worst-case load allowed by the RS-232-C standard; any lower resistance may result in improper operation). The driver circuit will withstand short circuits to ground, and will withstand connection to voltages within their driving range. Shorting a driver to a voltage outside the range −5.5 to + 9 volts may result in damage to the interface module.

The receiving circuit consists of a diode and transistor whose function is to convert the minus/plus RS-232-C voltages to the voltages used by the I/O chips. A 4700-ohm input resistor protects the outside device from having to deliver too much current. Notice that the DSR inputs have 1800-ohm resistors attached to ground which ensure that DSR will seem OFF if nothing is attached to DSR. A long unterminated wire attached to DSR can cause DSR to go ON and OFF if there is activity in other leads in the same cable. This is called the "antenna effect," because the unterminated wire acts as an antenna and "receives" the signals in the adjacent wires.

Port 4 may be set up for 20-mA current loop operation (see Figure C-4, page 90). In current-loop operation, pins 4 and 7 (RTS + 10v, and RCV) are tied together (pin numbers are of the 9-pin connector of the interface module). When the attached teletype keyboard-sending contacts are closed, pin 9 pulls RCV negative (MARK). This is the idle state of the teletype. Whenever the switch opens during transmission of a character from the teletype, RCV is pulled positive (SPACE). Notice that if the teletype is turned off, this switch may be open and the interface module will receive a BREAK signal.

For current loop output, the teletype's printer solenoid is tied between pins 1 and 3 (+ 10v DTR and XMT). XMT is normally negative (MARK); thus the solenoid is activated in the MARK state. XMT goes to nearly + 10v for SPACE so very little current passes through the solenoid and it disengages. Be careful when connecting a current loop device that it does not apply excessive voltages to the interface module. Also note that if the send and receive loops are connected together within the teletype the send or receive loop may not work correctly (the signal may be shorted out). If this happens, try swapping the send or receive wire pairs.

PRINTER PORT SPECIFICATIONS

Refer to the interface module schematic in Appendix C and the printer port timing diagrams (Figure 12-1).

All signals on the printer port are TTL level (0 to + 5 volts). The output lines are buffered by transistors to supply the necessary drive for the printer electronics. Input lines are buffered to protect the I/O chips.

The open-collector output circuit can sink 5 mA. That is, the circuit is capable of pulling 1000-ohm pull-up resistors in the printer to TTL zero. The output circuit expects some such pull-up in the printer; if pull-ups are not present, the output lines will be pulled to + 5 volts only by the internal 10,000-ohm pull-up resistors and the lines may slew too slowly to TTL one.

The interface module detects the presence of the printer via the FAULT line. If this line is low, the interface module will not respond to printer requests from the computer. This line is low if the ATARI 825 Printer is turned off (or disconnected). This feature allows you to connect more than one ATARI printer to the computer, and switch between them by turning only one of them on at a time. If you attach your own printer to the printer port, FAULT must be high (TTL one) for the interface module to operate the printer. If there is no appropriate signal from your printer to which FAULT may be attached, you may connect FAULT (pin 12) to the + 5 volt pull-up at pin 9. Be sure you do not connect FAULT to a busy-type line which will alternate on and off; FAULT should stay on. Do not attach FAULT to a voltage above + 5 volts.
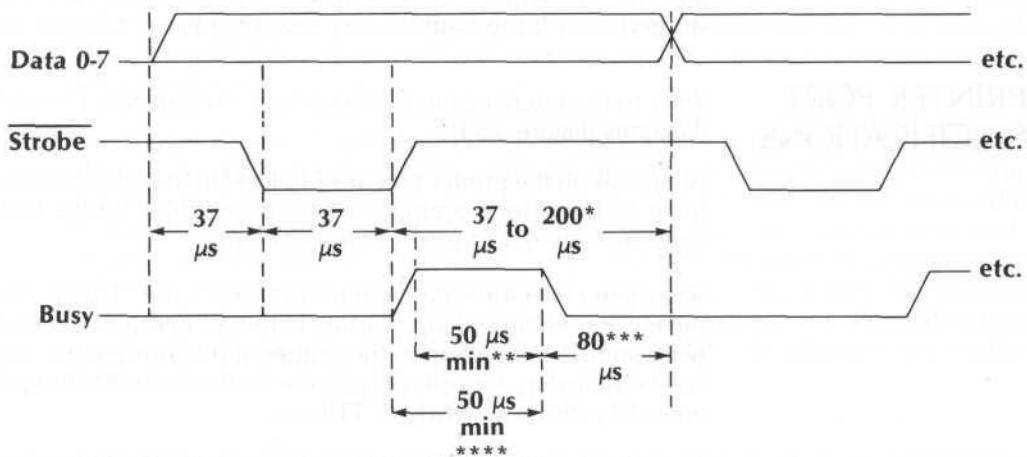
The eight data lines are positive-logic. The data lines normally rest at zero (ASCII NULL). A data byte is sent to the printer (when it is not BUSY) by placing the data on the eight data lines and pulsing the data STROBE. The STROBE is normally high, and goes low during the strobe pulse.

After sending each data byte to the printer, the interface module waits for a BUSY signal. The ATARI 825 80-Column Printer sends a positive-logic BUSY signal as it processes each byte of data. The BUSY is quite short for most data bytes since the printer merely saves each character in its own memory, but BUSY is quite long when the printer prints. The interface module does not care how long the printer is BUSY — the only requirement is that the printer respond to all 40 characters (that is, go not BUSY after the last character) within 30 seconds. Immediately after BUSY goes low again, the interface module sends the next character to the printer. When all the characters have been accepted by the printer, the interface module signals the computer that the print operation is finished.

Note: Early versions of the ATARI 850 Interface Module will wait four seconds for the printer to respond. If you suspect that you may have an early version, contact your nearest Authorized ATARI Computer Service Center for details on upgrading.

Some printers using the Centronics-type interface do not signal BUSY for each character received, but only go BUSY during printing. For this reason, the interface module only waits 200 microseconds for BUSY after sending a data byte. If BUSY does not go on within this time, the interface module sends the next character, assuming the printer has completed its processing of the preceding character.

Figure 12-1   Timing of Printer Ports

*One byte sent every 280 microseconds without BUSY
**Pulse must be > 50 microseconds, no maximum. However, 40 characters must be accepted by the printer in 30 seconds.
***Approximate
****BUSY may follow either leading or trailing edge of $\overline{\text{STROBE}}$. However, it must remain at least 50 microseconds after trailing edge of $\overline{\text{STROBE}}$.

PRINCIPLES OF OPERATION

The ATARI 850 Interface Module is an "intelligent" device. It contains a microprocessor, built-in program in ROM, and extensive I/O capability. The I/O forms the PARALLEL INTERFACE (printer) and SERIAL INTERFACE (RS-232-C) ports, and is also used for communication between the interface module and the ATARI 400 or ATARI 800 Home Computer.

SOFTWARE OPERATION

Once booted, the RS-232-C SERIAL INTERFACE port handler is linked in as the **R:** device. This handler contains code to reestablish itself whenever a `SYSTEM RESET` occurs.

The RS-232-C handler is called by the Central Input/Output subsystem (CIO) to execute each type of I/O operation for the **R:** device. (The CIO is that part of the OS that handles input/output.) The exception to this is output calls from BASIC which bypass CIO by calling the RS-232-C handler directly. Some of the commands are executed entirely by the handler (set-up), but most are passed on to the interface module. Some commands cause set-up in both the handler and in the interface module.

The CONFIGURE BAUD RATE command is a set-up command which is executed by both the handler and the interface module. Both the handler and the interface module keep separate tables for each of the four RS-232-C SERIAL INTERFACE ports. This command allows you to set the Baud rate, "word" size, number of stop bits to transmit, and enable or disable the checking of DSR, CTS, and CRX. See Sections 4, 5, and 9 for further definitions and details.

The CONFIGURE TRANSLATION MODE command is executed by the handler. This command sets values that control the translation and parity handling during I/O. See Sections 4, 6, and 9 for further definitions and details.

The CONTROL command is executed by the interface module. Outgoing control lines for the indicated port are set ON (or MARK), set OFF (or SPACE), or left alone, as specified by the control parameter. Each line is left alone until another CONTROL command is executed. Note that if the XMT line is set to SPACE, it will return to SPACE following any subsequent data transmission until another CONTROL command sets it to MARK. See Sections 4, 7, and 9 for more information.

The OPEN command is executed entirely by the handler. It establishes control information for the port being opened. The CLOSE command is executed mostly by the handler; OPEN flags are cleared; any data in output buffers is sent; Concurrent Mode I/O is shut down. Any data in an input buffer is lost at CLOSE time. See Sections 4 and 8.

Block Mode output takes data from BASIC PRINT or PUT statements, puts each character through translation, and puts each character into the 32-byte output buffer. The buffer is transmitted when it fills, or when 13 (decimal) is stored into the buffer (automatic short block on CR). Data from the buffer is sent to the interface module as 8-bit bytes. If 7-, 6-, or 5-bit words are configured, the interface module strips the necessary number of high-order bits from each byte before transmitting it to the port. If monitoring of any external status line has been configured for the port, the readiness is checked by the interface module whenever a block is sent to it. If not ready, the interface module returns a NAK. The ATARI Home Computer waits while the interface module transmits a block.

The FORCE SHORT BLOCK command causes the handler to transmit the block of data before 32 bytes have been collected. If there is no data in the buffer, the FORCE SHORT BLOCK command has no effect. See Section 4 and Appendix B.

When START CONCURRENT MODE I/O is performed, a number of things occur. The handler marks the Concurrent Mode I/O as active (if there are no errors while starting Concurrent Mode I/O). The handler sets up its own serial input/serial output interrupt handlers as necessary (depending on I/O direction) to field data going in and out. The handler establishes the initial (empty) state of the input and output buffers. Then the handler informs the interface module that Concurrent Mode I/O should be started.

During Concurrent Mode I/O, each character being received from the interface module is taken in by the handler's interrupt driver and placed in the input buffer. Characters to be sent to the interface module are translated and put in the output buffer. As the serial hardware in the computer finishes sending each character, the output interrupt driver immediately sends another character from the buffer (unless it is empty). If the input buffer overflows, an error is flagged; output buffer overflow stops putting data into the buffer until data is sent to free buffer space. See Sections 4, 9, and 10 for more information.

Input and output statements (GET, PUT, PRINT, INPUT) executed to a channel through which Concurrent I/O is active do not directly cause any I/O to the RS-232-C SERIAL INTERFACE port. Rather, input statements simply retrieve data that is in the input buffer (translation occurs at this time), and output statements put data into the output buffer. If an input statement wants more data but the input buffer is empty, BASIC will wait until the data arrives. If an output statement attempts to put data into a full output buffer, BASIC will wait until space becomes available. This is a result of the interrupt-driven sending of data from the output buffer, which starts as soon as data is put into the buffer. The data is moved into and out of each buffer circularly—that is, the buffer is automatically reused. The maximum amount of data a circular buffer can hold at once is one byte less than its size.

The interface module handles Concurrent I/O in one of two ways. The most common mode is used when 8-bit words are being transmitted, no matter what the rate or I/O direction. In this mode, the interface module "connects" (through the interface module's microprocessor) the transmit (XMT) and receive (RCV) lines of the selected port to the I/O connector going to the computer. The data is not interpreted by the interface module in this mode; all serialization of the data is performed by the serial I/O hardware in the computer console. Note that the "connection" between the RS-232-C SERIAL INTERFACE port and the computer's peripheral I/O port is handled by software. Each line coming into the interface module (one from the computer, one from the RS-232-C SERIAL INTERFACE port) is sampled (checked) over and over, and its value is then passed on to the "connected" outgoing line. The sampling rate is 34.6 kHz; the lines are sampled every 28.9 microseconds.

The other Concurrent Mode I/O is established in the interface module for low speed (300 Baud or less) 7-, 6-, or 5-bit input (half-duplex). In this mode, the interface module receives a 7-, 6-, or 5-bit character from the port and then transmits a corresponding 8-bit character to the computer. This is done because the computer's hardware is not capable of receiving anything but 8-bit serial words. The interface module receives the data by sampling it at a rate of 16 samples per bit. As each character is sent from the interface module to the computer, extra high-order 1-bits are added to get 8-bit words. The interface module sets an internal error flag if a framing error occurs in the incoming data. This flag may be queried with STATUS after the Concurrent I/O is stopped.

The interface module leaves Concurrent Mode when it is instructed by the handler when the Concurrent I/O IOCB is closed, either with a BASIC command or when the program "runs off the end."

The interface module is constantly keeping track of all incoming RS-232-C readiness lines, for the purpose of being able to report their state to the STATUS command. This does not apply to the RCV lines or any lines on the printer port. The readiness lines are checked periodically through sampling. The sampling rate depends on the activities the interface module is asked to perform. In order not to be missed, a pulse on a readiness line should be at least a few dozen milliseconds in duration.

The STATUS command is performed either by the RS-232-C handler alone (when Concurrent I/O is active) or by both the handler and the interface module. In the former case, the handler supplies the user with information about its current operation. In the latter case, the handler combines some of its own information with status and sense information supplied by the interface module. See Section 10 for more information.

PRINTER SOFTWARE OPERATION

The interface module responds to commands to an ATARI printer whenever it senses a printer attached to the parallel port (see Parallel Printer Port Specifications, Section 12, for signal requirements between the interface module and a printer).

The ATARI Home Computer Operating System contains a printer handler program which will address one printer, called **P:**. Four commands are allowed by the **P:** handler: OPEN, CLOSE, output (represented by PUT, PRINT, and LIST in BASIC), and STATUS.

To use the printer, one must OPEN an IOCB to the printer. CLOSE releases the IOCB when it is no longer needed.

ATARI printers and the interface module operate in Block Output Mode (as described in Section 4 of this manual). The printer handler builds a 40-byte buffer, and when the buffer fills, the 40 bytes are sent to the printer. When a printer is attached to the interface module, the interface module accepts the 40 characters and sends them, one at a time, over the PARALLEL INTERFACE port to the attached printer. The printer must acknowledge all 40 characters within 30 seconds (four seconds on early models). If it cannot, the computer will return an error (usually 139). See the note at the end of this section.

There is one exception to the above description: When the printer handler is asked to print an ATASCII End-of-Line character, it fills any unused part of the 40-character buffer with blanks (following the EOL) and sends it immediately. For this reason, the interface module ignores any characters in the buffer that follow an EOL.

The interface module translates EOL into ASCII CR (Carriage Return, 13 decimal). No other translations are made. In particular, bit 7 (high bit) of each byte is not changed, and LF is not added following CR. However, multiple EOL's in a row, without intervening characters, are sent to the printer as alternating CR's and blanks.

A special note about LPRINT in BASIC: LPRINT is equivalent to OPEN, PRINT and CLOSE all in one. Execution of an LPRINT statement with a comma or semicolon at the end will send to the interface module a 40-character buffer which is padded with blanks but does **not** have an EOL character. The interface module will send all 40 characters to the printer (including the blanks), but the printer will probably not respond because most printers wait for CR before activating a print cycle. Therefore, LPRINT should **not** be used when you want a comma or semicolon on the end of your statement. Use PRINT in those cases. LPRINT uses IOCB #7 exclusively.

The STATUS request for device **P:** is answered by the interface module if there is a printer attached and it is turned on. The status returned in location 746 (decimal) is 128 if the previous operation to the printer was successful; 129 if the previous command to the interface module printer port was bad; 130 if the previous 40-byte data frame had an error (this should not happen); and 132 if the previous command timed out—that is, the printer stayed BUSY more than 30 seconds.

NOTICE

Use of multiple printer control codes that involve carriage motion (with the exception of end-of-line), may cause an ERROR 139 (Device NAK). Carriage motion includes backspace, forward and reverse linefeeds, and partial linefeeds.

The ATARI 850 Interface Module sends data to the printer in 40-character blocks. If there is more than one carriage motion in each block, the printer may not recover in time to receive the next 40-character block.

If you should have this problem, check your program. Try to arrange your printer control codes in such a way that there is no more than one carriage motion in each 40-character block. This can be done by preceding each carriage motion with 40 "null" characters. Null characters can be generated with control comma (CTRL ,) or with the BASIC command CHR$(0).

APPENDIX A
CODE TABLES

Table A-1. Decimal, Hexadecimal, ASCII, ATASCII Code

DECIMAL CODE	HEXADECIMAL CODE	ATASCII CHARACTER	ASCII CHARACTER	MEANING
00	00		NUL	Null
01	01		SOH	Start of heading
02	02		STX	Start of text
03	03		ETX	End of text
04	04		EOT	End of transmission
05	05		ENQ	Enquiry
06	06		ACK	Acknowledge
07	07		BEL	Bell
08	08		BS	Backspace
09	09		HT	Horizontal tabulation
10	0A		LF	Line feed
11	0B		VT	Vertical tabulation
12	0C		FF	Form feed
13	0D		CR	Carriage return
14	0E		SO	Shift out
15	0F		SI	Shift in
16	10		DLE	Data line escape
17	11		DC1	Device control 1 (XON)
18	12		DC2	Device control 2
19	13		DC3	Device control 3 (XOFF)
20	14		DC4	Device control 4
21	15		NAK	Negative acknowledge
22	16		SYN	Synchronous idle
23	17		ETB	End of transmission block
24	18		CAN	Cancel
25	19		EM	End of medium
26	1A		SUB	Substitute

DECIMAL CODE	HEXADECIMAL CODE	ATASCII CHARACTER	ASCII CHAR	MEANING
27	1B		ESC	Escape
28	1C		FS	File separator
29	1D		GS	Group separator
30	1E		RS	Record separator
31	1F		US	Unit separator
32	20		SP	Space
33	21		!	Exclamation point
34	22		"	Quotation mark
35	23		#	Number sign
36	24		$	Dollar sign
37	25		%	Percent sign
38	26		&	Ampersand
39	27		'	Apostrophe
40	28		(	Opening parenthesis
41	29		)	Closing parenthesis
42	2A		*	Asterisk
43	2B		+	Plus
44	2C		,	Comma
45	2D		-	Hyphen (minus)
46	2E		.	Period (decimal point)
47	2F		/	Right Slant
48	30		0	Zero
49	31		1	One
50	32		2	Two
51	33		3	Three
52	34		4	Four
53	35		5	Five
54	36		6	Six
55	37		7	Seven
56	38		8	Eight
57	39		9	Nine
58	3A		:	Colon
59	3B		;	Semicolon
60	3C		<	Less than

DECIMAL CODE	HEXADECIMAL CODE	ATASCII CHARACTER	ASCII CHAR	MEANING
61	3D	=	=	Equals
62	3E	>	>	Greater than
63	3F	?	?	Question mark
64	40	@	@	Commercial at
65	41	A	A	Uppercase A
66	42	B	B	Uppercase B
67	43	C	C	Uppercase C
68	44	D	D	Uppercase D
69	45	E	E	Uppercase E
70	46	F	F	Uppercase F
71	47	G	G	Uppercase G
72	48	H	H	Uppercase H
73	49	I	I	Uppercase I
74	4A	J	J	Uppercase J
75	4B	K	K	Uppercase K
76	4C	L	L	Uppercase L
77	4D	M	M	Uppercase M
78	4E	N	N	Uppercase N
79	4F	O	O	Uppercase O
80	50	P	P	Uppercase P
81	51	Q	Q	Uppercase Q
82	52	R	R	Uppercase R
83	53	S	S	Uppercase S
84	54	T	T	Uppercase T
85	55	U	U	Uppercase U
86	56	V	V	Uppercase V
87	57	W	W	Uppercase W
88	58	X	X	Uppercase X
89	59	Y	Y	Uppercase Y
90	5A	Z	Z	Uppercase Z
91	5B	[	[	Opening bracket
92	5C	\	\	Left slant
93	5D	]	]	Closing bracket
94	5E	^	^	Circumflex

DECIMAL CODE	HEXADECIMAL CODE	ATASCII CHARACTER	ASCII CHAR	MEANING
95	5F		—	Underscore
96	60		`	Grave accent
97	61		a	Lowercase a
98	62		b	Lowercase b
99	63		c	Lowercase c
100	64		d	Lowercase d
101	65		e	Lowercase e
102	66		f	Lowercase f
103	67		g	Lowercase g
104	68		h	Lowercase h
105	69		i	Lowercase i
106	6A		j	Lowercase j
107	6B		k	Lowercase k
108	6C		l	Lowercase l
109	6D		m	Lowercase m
110	6E		n	Lowercase n
111	6F		o	Lowercase o
112	70		p	Lowercase p
113	71		q	Lowercase q
114	72		r	Lowercase r
115	73		s	Lowercase s
116	74		t	Lowercase t
117	75		u	Lowercase u
118	76		v	Lowercase v
119	77		w	Lowercase w
120	78		x	Lowercase x
121	79		y	Lowercase y
122	7A		z	Lowercase z
123	7B		{	Opening brace
124	7C				Vertical line
125	7D		}	Closing brace
126	7E		~	Tilde
127	7F		DEL	Delete

DECIMAL CODE	HEXADECIMAL CODE	ATASCII CHARACTER
128	80	♥
129	81	├
130	82	│
131	83	┘
132	84	┤
133	85	┐
134	86	╱
135	87	╲
136	88	◢
137	89	▪
138	8A	◥
139	8B	▫
140	8C	▪
141	8D	▬
142	8E	▁
143	8F	▪
144	90	♣
145	91	┌
146	92	─
147	93	┼
148	94	●
149	95	▬
150	96	│
151	97	┬
152	98	┴
153	99	█
154	9A	└
155	9B	⏎
156	9C	↑

*For any ATASCII value above 127, the video display of that value will be in inverse video (blue on white instead of white on blue) as compared to that same value minus 128.

Example: ATASCII 128 displays a blue heart on a white background, while ATASCII 0 (128 minus 128) displays a white heart against a blue background. For character transmission, however, the interface module will automatically subtract 128 from any ATASCII value exceeding 128. Therefore, any inverse video character will be transmitted as a normal character.

Additional changes may occur to certain ATASCII control characters during character transmission (see Section 6, "Setting the Translation Modes and Parity Handling").

DECIMAL CODE	HEXADECIMAL CODE	ATASCII CHARACTER
157	9D	↓
158	9E	←
159	9F	→
160	A0	□
161	A1	!
162	A2	''
163	A3	#
164	A4	$
165	A5	%
166	A6	&
167	A7	'
168	A8	(
169	A9	)
170	AA	*
171	AB	+
172	AC	,
173	AD	—
174	AE	.
175	AF	/
176	B0	0
177	B1	1
178	B2	2
179	B3	3
180	B4	4
181	B5	5
182	B6	6
183	B7	7
184	B8	8
185	B9	9
186	BA	:
187	BB	;
188	BC	<
189	BD	=
190	BE	>

DECIMAL CODE	HEXADECIMAL CODE	ATASCII CHARACTER
191	BF	?
192	C0	"
193	C1	A
194	C2	B
195	C3	C
196	C4	D
197	C5	E
198	C6	F
199	C7	G
200	C8	H
201	C9	I
202	CA	J
203	CB	K
204	CC	L
205	CD	M
206	CE	N
207	CF	O
208	D0	P
209	D1	Q
210	D2	R
211	D3	S
212	D4	T
213	D5	U
214	D6	V
215	D7	W
216	D8	X
217	D9	Y
218	DA	Z
219	DB	[
220	DC	\
221	DD	]
222	DE	^
223	DF	_
224	E0	◆

DECIMAL CODE	HEXADECIMAL CODE	ATASCII CHARACTER
225	E1	a
226	E2	b
227	E3	c
228	E4	d
229	E5	e
230	E6	f
231	E7	g
232	E8	h
233	E9	i
234	EA	j
235	EB	k
236	EC	l
237	ED	m
238	EE	n
239	EF	o
240	F0	p
241	F1	q
242	F2	r
243	F3	s
244	F4	t
245	F5	u
246	F6	v
247	F7	w
248	F8	x
249	F9	y
250	FA	z
251	FB	♠
252	FC	|
253	FD	↰
254	FE	◀
255	FF	▶

BAUDOT CODE

Below is a table of the most common Baudot codes. All Baudot codes are identical for letters, numbers, and control characters, but some may differ in punctuation. The DECIMAL VALUE column gives the 5-bit Baudot serial binary code converted to decimal. When transmitted, a start bit (space) precedes the character, the character itself is sent low bit first, and 1.5 or 2 stop bits (mark) follow. Mark is sent for 1, space for 0.

The HEX/DEC columns show the value of the Baudot character when interpreted as an 8-bit word with the three high-order bits set to 1. These are the codes which represent the Baudot characters with the interface module's no-translation mode (translation mode 32).

LETTERS	FIGURES	DECIMAL VALUE	HEX/DEC
A	-(dash)	3	E3/227
B	?	25	F9/249
C	:	14	EE/238
D	$	9	E9/233
E	3	1	E1/225
F	!	13	ED/237
G	+ or &	26	FA/250
H	# or STOP	20	F4/244
I	8	6	E6/230
J	' (apost.)	11	EB/235
K	(	15	EF/239
L	)	18	F2/242
M	. (period)	28	FC/252
N	, (comma)	12	EC/236
O	9	24	F8/248
P	0 (zero)	22	F6/246
Q	1 (one)	23	F7/247
R	4	10	EA/234
S	BELL	5	E5/229
T	5	16	F0/240
U	7	7	E7/231
V	;	30	FE/254
W	2	19	F3/243
X	/	29	FD/253
Y	6	21	F5/245
Z	''	17	F1/241
NULL	NULL	0	E0/224
RETURN	RETURN	8	E8/232
LINEFEED	LINEFEED	2	E2/226
SPACE	SPACE	4	E4/228
LETTERS	LETTERS	31	FF/255
FIGURES	FIGURES	27	FB/251

XIO COMMANDS AND TABLES

IOCB is an acronym for Input/Output Control Block. It is that portion of the computer's Operating System that controls the input and output of data within the system.

IOCB is the number of the IOCB that BASIC commands for this port must use. From the ATARI BASIC language the user has seven IOCBs to use, numbered 1 through 7.

The Operating System uses IOCB #7 for LPRINT and IOCB #6 for Graphics Modes functions. These IOCBs may be used, but care must be taken not to use LPRINT or Graphics Mode functions while the appropriate IOCB is open.

Note: It is strongly suggested that to save yourself problems, you assign IOCBs with numbers other than 6 or 7. You should also use IOCB #5 for Concurrent I/O.

Rn: is the SERIAL INTERFACE port being opened; **n** can be 1, 2, 3, or 4. **R:** is interpreted as **R1:** For a given port, no more than one IOCB may be open at one time.

FORCING EARLY TRANSMISSION OF OUTPUT BLOCKS

The BASIC form of the FORCE SHORT BLOCK command is:

Format: **XIO 32,** #IOCB, Aux1, Aux2, "Rn:"
Example: **XIO 32,** #5, 0, 0, "R1:"

32 specifies the FORCE SHORT BLOCK command.

Aux1 and Aux2 are ignored in this command. In general, set these parameters at zero.

SETTING THE BAUD, WORD SIZE, STOP BITS, AND READY CHECKING

Format: **XIO 36,** #IOCB, Aux1, Aux2, "Rn:"
Example: **XIO 36,** #5, 138, 6, "R:"

This command sets the Baud rate, word size, and number of stop bits in transmitted messages. It also controls the checking of incoming control signals. 36 specifies this command.

Aux1 is coded to specify three variables — Baud, word size and the number of stop bits. The coding is given in Tables B-1, B-2, and B-3. Add one number from each table.

Aux2 is coded to specify which of the incoming control signals (if any) should be checked. These signals are DSR (Data Set Ready), CTS (Clear to Send) and CRX (Carrier Detect). The coding is given in Table C-4. (See Section 5 for a detailed explanation of this checking.)

Table B-1 Baud Rate Specifiers To Add to Aux1

ADD	BAUD RATE (bits per second)
0	300
1	45.5*
2	50*
3	56.875*
4	75**
5	110
6	134.5***
7	150
8	300
9	600
10	1200
11	1800
12	2400
13	4800
14	9600
15	9600

*These Baud rates are useful for communications with Baudot teletypes, for RTTY (radioteletype) applications. They are more commonly referred to as 60, 67, and 75 words per minute.

**This Baud rate is sometimes used for ASCII communications and may also be used for 5-bit Baudot RTTY. The latter is commonly referred to as 100 wpm.

***This Baud rate is used by IBM systems.

Table B-2 Word Size Specifiers To Add to Aux1

ADD	WORD SIZE (bits)
0	8
16	7
32	6
48	5

Table B-3 Specifier for Two Stop Bits To Add to Aux1

ADD	STOP BITS SENT WITH EACH WORD
0	1
128	2

Table B-4 Aux2 Specification To Check DSR, CTS, CRX

ADD	TO MONITOR
0	None
1	CRX
2	CTS
3	CTS, CRX
4	DSR
5	DSR, CRX
6	DSR, CTS
7	DSR, CTS, CRX

SETTING TRANSLATION MODES AND PARITY

Format: **XIO 38,** #IOCB, Aux1, Aux2, "Rn:"
Example: **XIO 38,** #5, 64, 33, "R2:"

This command controls the translation of internal codes and external codes. For example, ASCII to ATASCII. 38 specifies this I/O command.

Aux1 is coded to specify the translation mode, input parity mode, output parity mode and whether a Line Feed is added after Carriage Return. The coding is given in Tables B-5, B-6, B-7 and B-8. Add one number from each table.

Aux2 is the number equivalent to the "won't translate" character in one translation mode. (See Section 6 for detailed explanation.)

Table B-5 Translation Mode Options Added to Aux1

Add	To Get
0	Light ATASCII/ASCII translation
16	Heavy ATASCII/ASCII translation
32	No translation

Table B-6 Input Parity Mode Options Added to Aux1

Add	To get
0	Ignore and do not change parity bit
4	Check for odd parity, clear parity bit
8	Check for even parity, clear parity bit
12	Do not check parity but clear parity bit

Table B-7   Output Parity Mode Options Added to Aux1

Add	To get
0	Do not change parity bit
1	Set output parity odd
2	Set output parity even
3	Set parity bit to 1

Table B-8   Append Line Feed Options Added to Aux1

Add	To get
0	Do not append LF
64	Append LF after CR (translated from EOL)

CONTROLLING THE OUTGOING LINES DTR, RTS, AND XMT

Format: **XIO 34,** #IOCB, Aux1, Aux2, "Rn:"
Example: **XIO 34,** #5, 160, 0, "R1:"

This controls the use of XMT and the outgoing control lines DTR, RTS.

34 specifies this I/O command.

Aux1 is coded to specify control of DTR, RTS, and XMT. The coding is given in Tables B-9, B-10, and B-11. Add one number from each table.

Aux2 is not used by this command. It should be set to zero.

Table B-9   Control Values for DTR Added to Aux1

Add	To get
0	No change from current DTR setting
128	Turn DTR OFF
192	Turn DTR ON

Table B-10   Control Values for RTS Added To Aux1

Add	To get
0	No change from current RTS setting
32	Turn RTS OFF
48	Turn RTS ON

Add	To get
0	No change from current XMT setting
2	Set XMT to SPACE (0)
3	Set XMT to MARK (1)

Starting Concurrent I/O Mode

Format: **XIO 40,** #IOCB, 0, 0, ''Rn:''
Example: **XIO 40,** #5, 0, 0, ''Rn:''

This command is used to start Concurrent I/O Mode. 40 specifies this I/O command.

With this command — the input buffer is in the handler in the computer. The buffer holds 32 bytes. For some purposes a longer buffer is more convenient. Section 11 shows how to specify any size of buffer. The BASIC coding is more complex.

Note: Failure to terminate Concurrent Mode I/O properly before attempting I/O to other peripherals (or even other SERIAL INTERFACE ports) will probably result in program failure. The only way to recover from such failure is to turn the computer console off and then on again, which results in the loss of the information stored in RAM.

APPENDIX C

PORT DIAGRAMS AND INTERFACE MODULE SCHEMATIC

Pin 3 Send Data (Out)

Pin 4 Receive Data (In)

Pin 5 Signal Ground

Pin 2 Carrier Detect (CRX, In)

Pin 1 Data Terminal Ready (DTR, Ready Out)

Pin 8 Clear to Send (CTS, In)

Pin 6 Data Set Ready (DSR, Ready In)

Pin 7 Request to Send (RTS, Out)
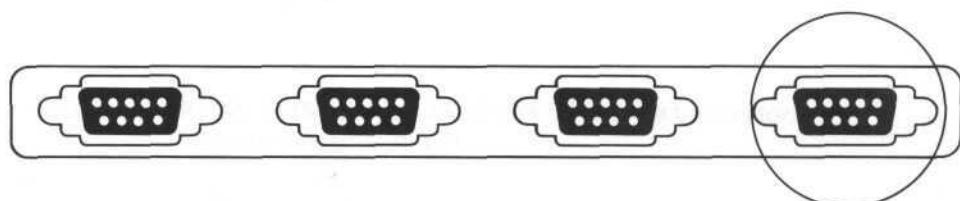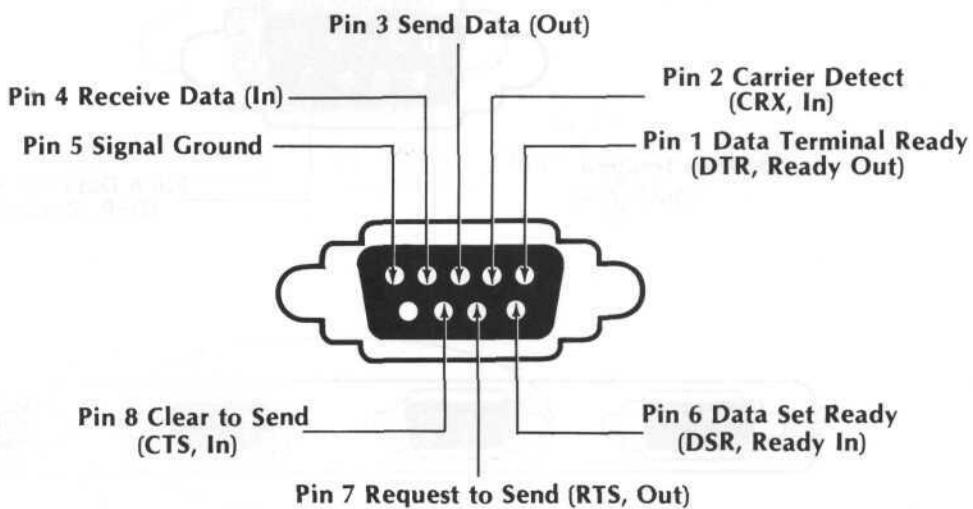
Figure C-1   Pin Functions of SERIAL INTERFACE Port 1 (9-pin female connector)
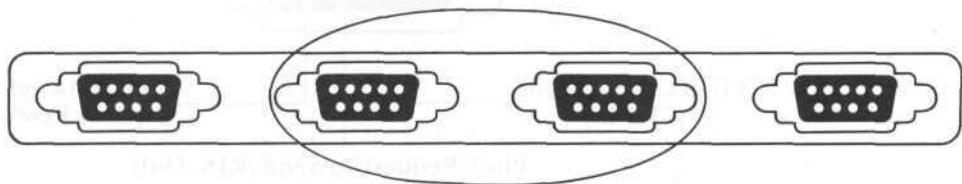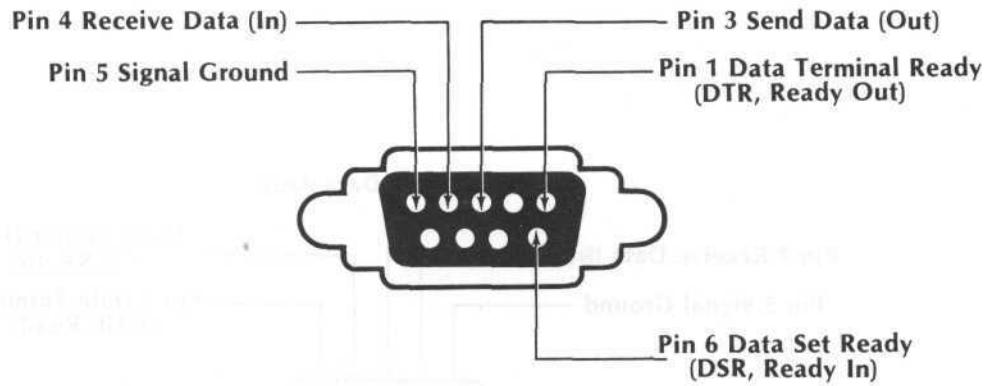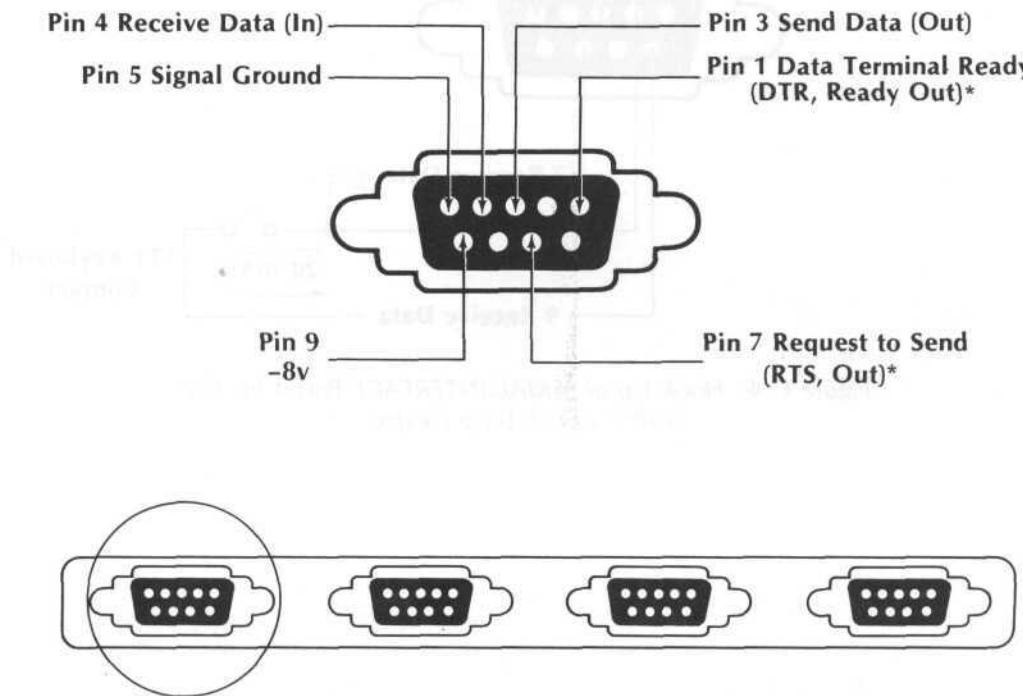
Pin 4 Receive Data (In) ———————————————— Pin 3 Send Data (Out)

Pin 5 Signal Ground ———————————————— Pin 1 Data Terminal Ready
(DTR, Ready Out)

Pin 6 Data Set Ready
(DSR, Ready In)

Figure C-2   Pin Functions of SERIAL INTERFACE Ports 2 and 3
(9-pin female connector)

Pin 4 Receive Data (In) ———— Pin 3 Send Data (Out)

Pin 5 Signal Ground ———— Pin 1 Data Terminal Ready
(DTR, Ready Out)*

Pin 9 ———— Pin 7 Request to Send
-8v (RTS, Out)*

*These pins are not computer-controlled and are always ON (+10v).

Figure C-3 Pin Functions of SERIAL INTERFACE Port 4 (9-pin female connector)
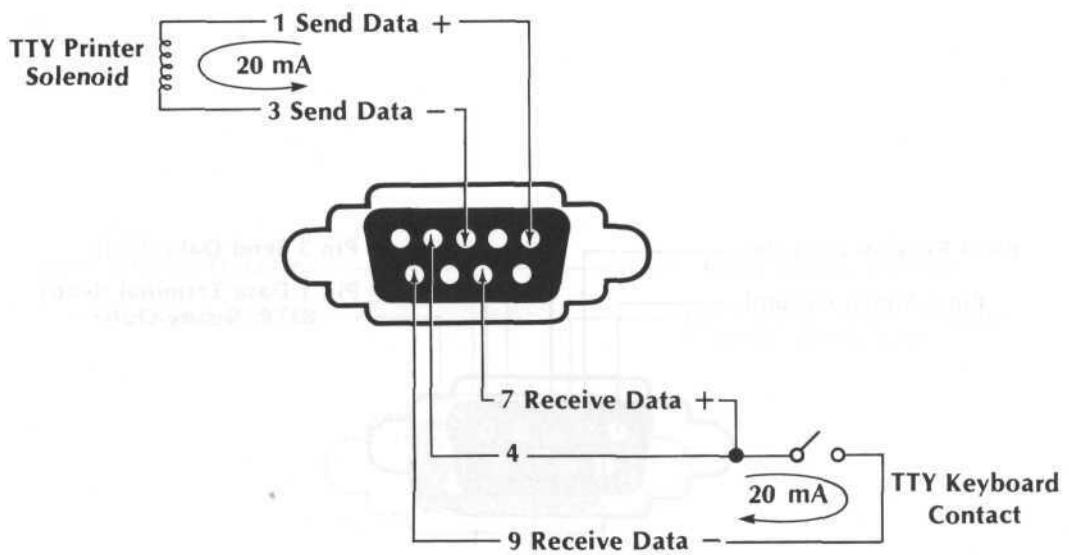
TTY Printer
Solenoid

1 Send Data +

20 mA

3 Send Data −

7 Receive Data +

4

9 Receive Data −

20 mA

TTY Keyboard
Contact

Figure C-4   Hook Up of SERIAL INTERFACE Port 4 for Use
With a 20-mA Loop Device

Pin 5 Data Bit 3 ——————————————— Pin 4 Data Bit 2

Pin 6 Data Bit 4 ——————————————— Pin 3 Data Bit 1

Pin 7 Data Bit 5 ——————————————— Pin 2 Data Bit 0

Pin 8 Data Bit 6 ——————————————— Pin 1 $\overline{\text{Data Strobe}}$

Pin 9 Data Pins
Pull-Up (+5v)

Pin 15 Data Bit 7 ——————————

Pin 13 Busy ——————————— Pin 11 Signal Ground

Pin 12 $\overline{\text{Fault}}$
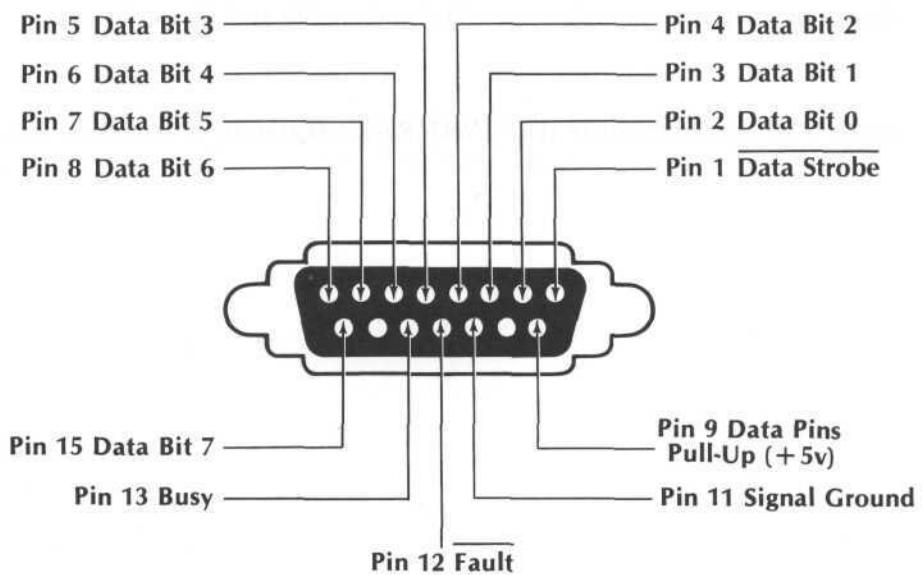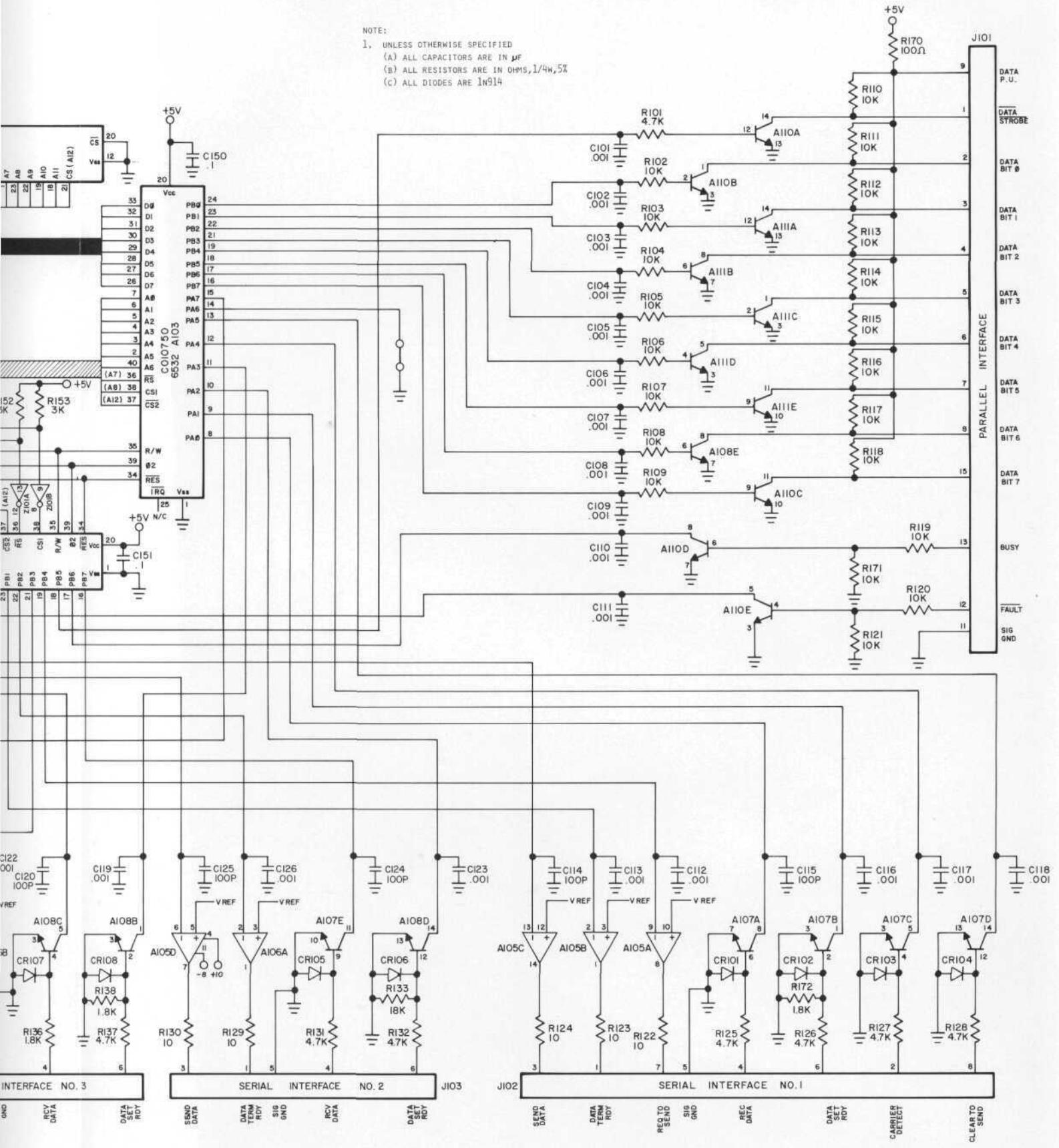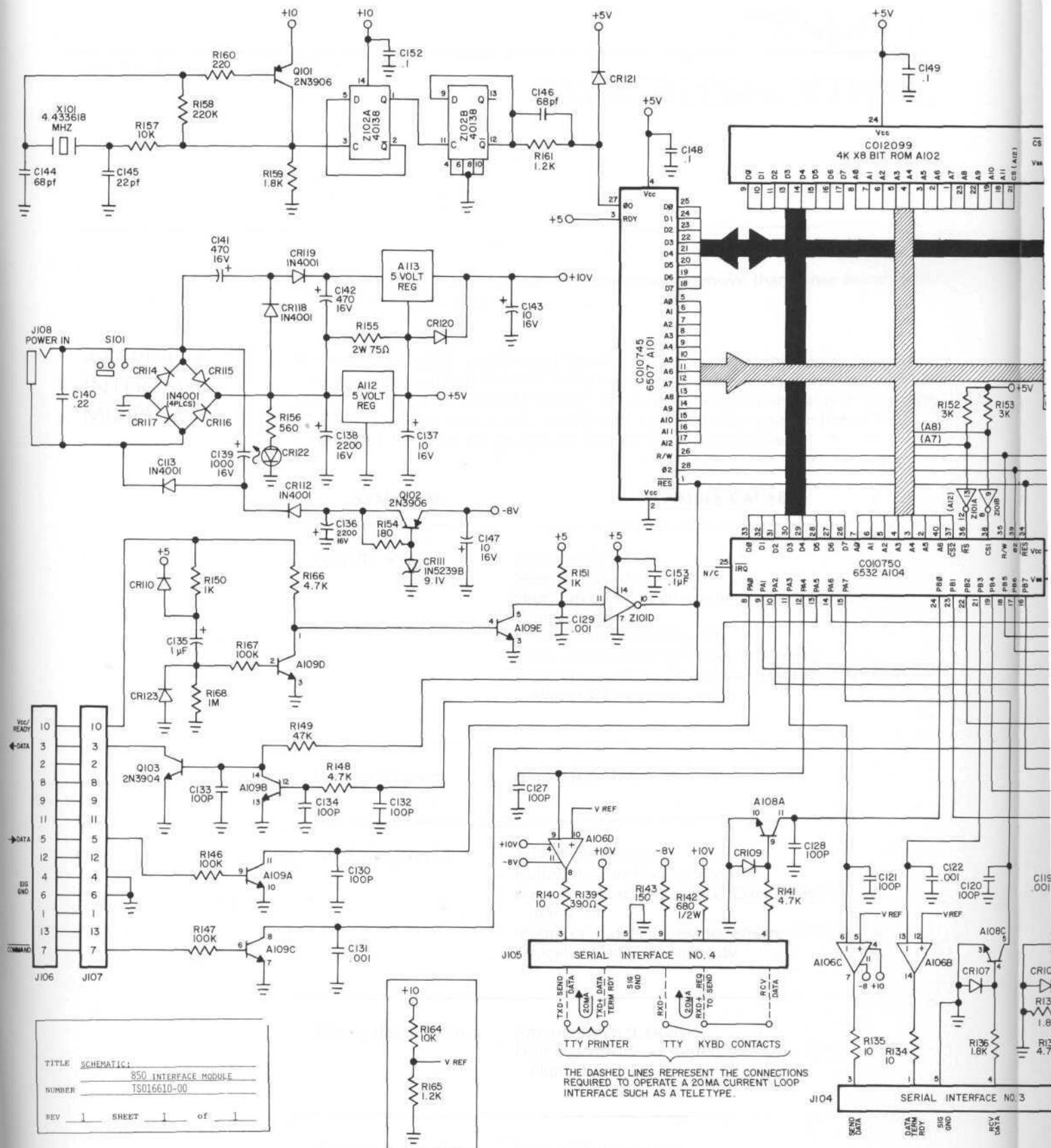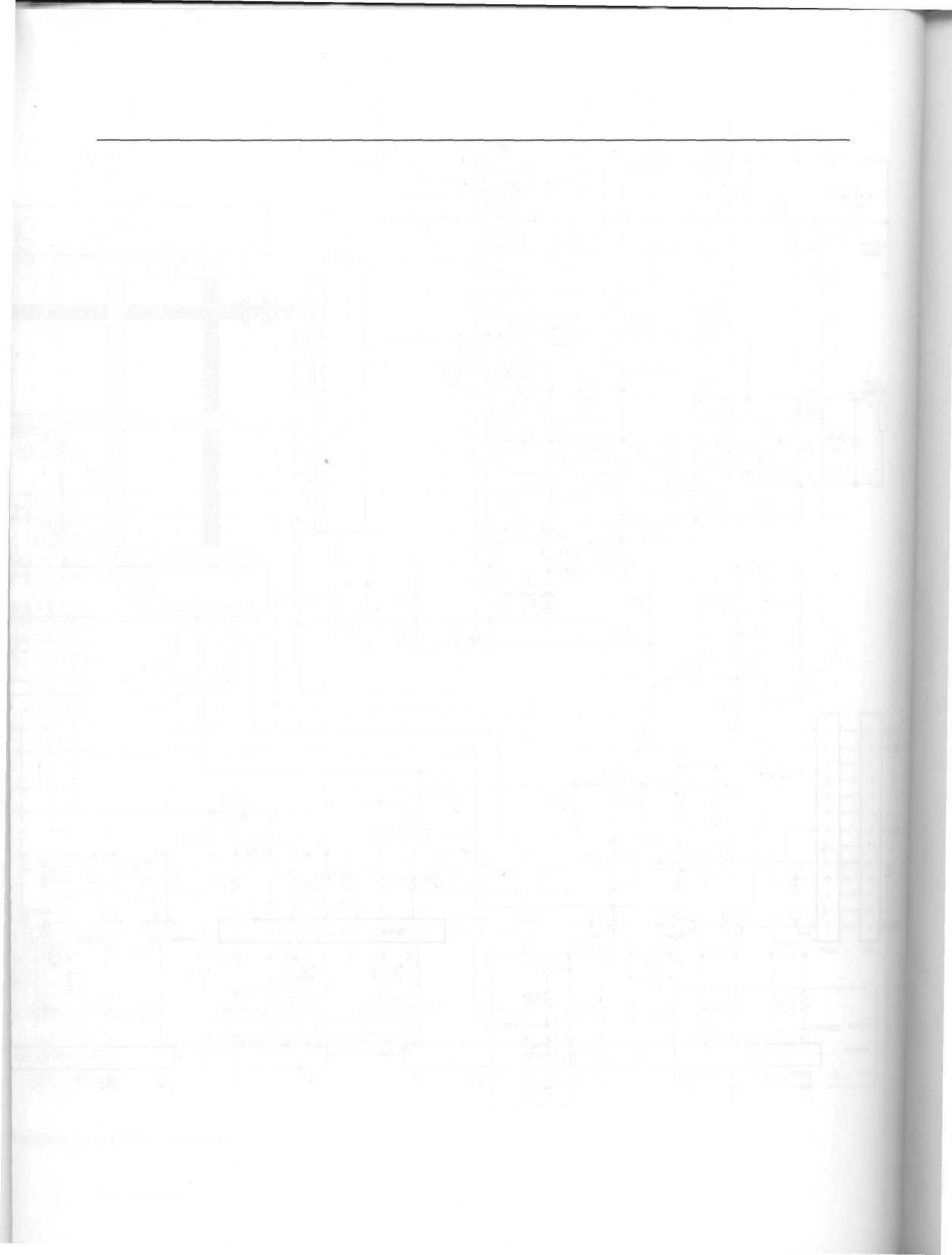
Figure C-5   Pin Functions of the Printer Port (15-pin female connector)

NOTE:
1. UNLESS OTHERWISE SPECIFIED
   (A) ALL CAPACITORS ARE IN μF
   (B) ALL RESISTORS ARE IN OHMS, 1/4W, 5%
   (C) ALL DIODES ARE 1N914

+5V
R170 100Ω
J101

R110 10K
R111 10K
R112 10K
R113 10K
R114 10K
R115 10K
R116 10K
R117 10K
R118 10K
R119 10K
R171 10K
R120 10K
R121 10K

R101 4.7K
C101 .001   R102 10K   A110A   A110B
C102 .001   R103 10K   A111A
C103 .001   R104 10K   A111B
C104 .001   R105 10K   A111C
C105 .001   R106 10K   A111D
C106 .001   R107 10K   A111E
C107 .001   R108 10K   A108E
C108 .001   R109 10K   A110C
C109 .001
C110 .001   A110D
C111 .001   A110E

+5V
C150 .1

20 Vcc
C010750
6532 A103

D0 33
D1 32
D2 31
D3 30
D4 29
D5 28
D6 27
D7 26
A0 7
A1 6
A2 5
A3 4
A4 3
A5 2
A6 40
RS 36 (A7)
CS1 38 (A8)
CS2 37 (A12)
R/W 35
Ø2 39
RES 34
IRQ
Vss 25

PB0 24
PB1 23
PB2 22
PB3 21
PB4 19
PB5 18
PB6 17
PB7 16
PA7 15
PA6 14
PA5 13
PA4 12
PA3 11
PA2 10
PA1 9
PA0 8

CS 20
Vss 12
CS (A12)

A7 23
A8 22
A9 21
A10 20
A11 19
CS (A12)

R152
R153 3K
+5V

+5V N/C
C151 .1
20 Vcc

J101 PARALLEL INTERFACE
9 DATA P.U.
1 DATA STROBE
2 DATA BIT 0
3 DATA BIT 1
4 DATA BIT 2
5 DATA BIT 3
6 DATA BIT 4
7 DATA BIT 5
8 DATA BIT 6
15 DATA BIT 7
13 BUSY
12 FAULT
11 SIG GND

C122 .001   C120 100P
C119 .001
C125 100P   C126 .001
C124 100P   C123 .001
C114 100P   C113 .001   C112 .001
C115 100P   C116 .001   C117 .001   C118 .001

V REF

A108C   A108B   A105D   A106A   A107E   A108D
A105C   A105B   A105A   A107A   A107B   A107C   A107D

CR107   CR108   CR105   CR106   CR101   CR102   CR103   CR104

R138 1.8K
R136 1.8K   R137 4.7K
R130 10   R129 10
R131 4.7K   R133 18K   R132 4.7K
R124 10   R123 10   R122 10
R125 4.7K   R172 1.8K   R126 4.7K   R127 4.7K   R128 4.7K

INTERFACE NO. 3

SERIAL INTERFACE NO. 2   J103

J102   SERIAL INTERFACE NO. 1

GND RCV DATA DATA SET RDY
SEND DATA DATA TERM RDY SIG GND RCV DATA DATA SET RDY
SEND DATA DATA TERM RDY REQ TO SEND SIG GND REC DATA DATA SET RDY CARRIER DETECT CLEAR TO SEND

Interface Module Schematic Diagram

Figure C-6 ATARI 850 Interface Module

The troubleshooting hints listed here are more extensive than those found in the *ATARI 850 Owner's Manual.*

ATARI 850 INTERFACE MODULE

If you have trouble communicating with a device, check the cable connections first. If you are using the ATARI BASIC Computer Language, try using GET and PUT instead of INPUT and PRINT; sending characters one at a time can give you useful clues to problems. Reread the instructions for the device you are trying to use. If you have a terminal, try plugging it into simulate your device.

SYMPTOM	POSSIBLE CAUSES
No data sent	Loose cable Selection of wrong port Short circuit or open connection Wrong Baud rate Wrong translation or parity Control lines incorrectly set (not ready) Incorrect control line sense (setting control ON when OFF required, etc.)
Wrong data sent	Any of the above
No data received	Any of the above Failure to start Concurrent I/O Data arrived during period Concurrent I/O not active Incorrect handshaking to ensure that the data is received only when ready
Wrong data received	Any of above reasons Framing error — started reading during arriving character
Characters missing from received data	Any of above reasons Buffer overrun — failure to retrieve data from buffer faster than it arrives

System stops when INPUT is tried	No data — see above No EOF: data contains no EOF or translation mode is incorrect (CR not becoming EOF)
System behaves sluggishly when program is no longer running	Concurrent input still active — be sure to terminate (close IOCB)
System dies	Failure to terminate Concurrent I/O before doing other serial I/O Editing BASIC program or executing BASIC statements in immediate mode while Concurrent I/O active — remember to close IOCB

ATARI 830 ACOUSTIC MODEM

If you have problems, the most likely reason is the phone line. Noise on the line or a weak phone line signal can often result in lost or invalid data. Try to redial the call to ensure that the connection is noise-free and there is no interference.

The ATARI 830 Acoustic Modem has a test mode to verify that the modem is working properly. The test mode switches the transmitter frequencies to match the receiver. All data into the modem will be looped back to the computer console for verification. The test requires an isolated acoustic path between the speaker and receiver of the telephone handset.

To test the originate mode, use the ATARI TeleLink I cartridge or the special Test Program shown below. Set the O/A switch for Originate and the F/H switch to the center position (TEST). If there is no tone, the unit is defective. Dial a single digit on the telephone to get a quiet line, then immediately place the handset into the acoustic muffs. Wait for the READY light and then type a message on the keyboard. The TEST function will display the message. Check the television screen for errors.

After the originate test, quickly push the O/A switch to **A** with the telephone still connected to the modem. Wait for the READY light and repeat the test.

Note: A quiet line is required to prevent dial tone interference. By dialing one digit, you may only get a quiet line for 30 seconds. You may have to repeat the process. A longer quiet time can be obtained by calling a friend. The mouthpiece of the friend's phone must be covered or removed to prevent room noise interference.

If communication still cannot be established and the modem checks out in the TEST mode, see tables below for other possible causes for failure.

MODEM TEST PROGRAM

Here is a simple program for testing your ATARI 830 Acoustic Modem. This program simply reads a key from the computer console keyboard, sends it to the modem, then reads it back from the modem and displays on the television screen.

This test may be used to check a port of the interface module by looping a wire between the XMT and RCV lines of the SERIAL INTERFACE port.

Caution: Using a wire too large in diameter can damage the SERIAL INTERFACE port connector.

To terminate the program, press the SYSTEM RESET key.

```
10 OPEN #1,4,0,"K:"
20 OPEN #2,13,0,"R1:"
30 XIO 38,#2,32,0,"R1:"
40 XIO 40,#2,0,0,"R1:"
50 FOR ETERNITY=0 TO 0 STEP 0
60 GET #1, KEY:REM from keyboard
70 PUT #2,KEY:REM to modem
80 GET #2,KEY:REM back from modem
90 PRINT CHR$(KEY);:REM to television
100 NEXT ETERNITY
```

SYMPTOM	POSSIBLE CAUSES
Ready Light Off:	Is modem power ON?
	Is handset in proper position? Label Indicates direction of cord.
	Are mode switches set properly?
	• When communicating with a time share computer, the modem should be set to ORIG mode. Modem at remote computer end will be in answer mode.
	• When commmunicating with another terminal, mode selection is determined by prior agreement between users. Remember one modem must be in answer mode, the other in originate mode.

Double Character Display:	Is Modem in half-duplex mode?
	1. If remote computer echoes all characters the modem should be in **full**-duplex mode.
	2. If communication system is half-duplex (no echo), the modem should be in **half**-duplex.

Garbled Display:	Is telephone handset fully seated in the rubber muffs?
	Is Baud rate correct? Both local and remote terminals must send data at the same Baud rate (300 Baud or less).
	Is received signal too weak or noisy? Pick up handset and listen for a clean tone (if remote modem is in answer mode). If additional tones, dialing pulses, static noise or voices are present, data may be garbled. Re-dial call.

ERROR CONDITIONS, CAUSES, AND CORRECTIONS

RS-232-C SERIAL INTERFACE PORT

This section contains descriptions of the errors you might encounter while using the interface module. Many of these errors also occur with other ATARI peripherals; they are listed here so you can see what they mean when using the interface module.

There are a number of **new** errors that you can get from the interface module but which no other peripherals will produce. These new error codes are listed in **boldface type.**

"ERROR" 1   Success. This is the status which successful completion of an I/O operation produces. BASIC does not report this to you except by continuing in normal fashion.

ERROR 128   Break abort. This means you pressed the BREAK key while I/O was proceeding.

ERROR 129   IOCB already OPEN. Your choice of IOCB number (#n) was that of an IOCB that was already OPEN. This can happen if you restart a program in a manner other than RUN (RUN closes files). Be careful not to put your OPEN statement inside a programmed loop. The second time OPEN is encountered it will produce ERROR 129.

ERROR 130   Nonexistent device. You specified something other than R:, R1:, R2:, R3:, or R4: . Perhaps you were trying to access a file on disk whose name starts with "R" and forgot the D: . THIS ERROR WILL OCCUR IF YOU ATTEMPT TO USE AN RS-232-C SERIAL INTER-FACE PORT AND THE RS-232-C HANDLER HAS NOT BEEN BOOTED WHEN THE SYSTEM WAS TURNED ON. In that case, you should save your program and start a new session, allowing the RS-232-C SERIAL INTERFACE handler to boot.

ERROR 131   Write only. You tried to read (GET, INPUT) from a port you opened as write only.

ERROR 132   Invalid command. You specified something incorrectly in an XIO command to the interface module.

ERROR 133   IOCB not OPEN. You neglected to OPEN the IOCB to the I/O device you are trying to access.

ERROR 135   Read only. You tried to write (PUT, PRINT) to a port you opened for read access only.

ERROR 138   Device timeout. The interface module did not respond to a command. Check the cables. Make sure the interface module is turned on.

ERROR 139   NAK. The interface module refused to perform some command. You may issue a STATUS request to find out what was wrong. Most common causes are: attempts to perform 5-, 6-, or 7-bit input at too high a Baud rate; automatic readiness checking was enabled and the connected device was not ready.

ERROR 150   Port already OPEN. You attempted to OPEN a RS-232-C SERIAL IN-TERFACE port but it was already OPEN through another IOCB. You can access a RS-232-C SERIAL INTERFACE port through only one IOCB at a time.

ERROR 151   Concurrent Mode I/O not enabled. You attempted to start Concurrent Mode I/O (XIO 40) but the port was not opened with an odd number specified for Aux1 (Aux1 bit 0 not set).

ERROR 152   Illegal User-supplied buffer. In the START CONCURRENT MODE I/O command with the user-supplied buffer, the buffer address and/or the buffer length were inconsistent.

ERROR 153   Active Concurrent Mode I/O error. You attempted to perform I/O to a RS-232-C SERIAL INTERFACE port while Concurrent Mode I/O was active to some other RS-232-C SERIAL INTERFACE port. Only input, output, CLOSE and STATUS commands to the active Concurrent Mode port are allowed while Concurrent Mode I/O is active. This error message is not always produced — attempting to do disallowed I/O while Concurrent Mode I/O is active may result in the computer "hanging up."

ERROR 154   Concurrent Mode I/O not active. Concurrent Mode I/O must be activated in order to perform input (GET, INPUT).

CAUTION

If a program is using Concurrent Mode input, always make sure the Concurrent Mode operation is stopped before your program stops. If files are not specifically closed, BASIC will close them when it interprets END or comes to the end of the program. All files are closed in the descending order of the IOCB number you have assigned.

Failure to terminate Concurrent Mode I/O properly before attempting I/O to other peripherals (or even other RS-232-C SERIAL INTERFACE ports) will probably result in program failure. The only way to recover from such failure is to turn the computer console off and then on again, which results in the loss of the information stored in RAM.

Pressing ▮SYSTEM RESET▮ on the computer console closes all open IOCBs and reestablishes most of the I/O system's registers and pointers. This method of closing files results in the loss of data being held in input and output buffers. The interface module may be "interrupted" by the ▮SYSTEM RESET▮ and so transmit only part of the character being sent at the time ▮SYSTEM RESET▮ was pressed. Another possible effect of ▮SYSTEM RESET▮ is a short burst of random data to an active Concurrent Mode I/O RS-232-C SERIAL INTERFACE port.

The Operating System has reserved IOCB #7 for LPRINT and IOCB #6 for the Graphics Modes. These two IOCBs are user programmable. However, problems will occur if you have opened IOCB #7 and then use the LPRINT command, or have IOCB #6 open and change Graphics Modes. For this reason, it is suggested that IOCB #5 be used when configuring a RS-232-C SERIAL INTERFACE port and IOCBs less than #5 for your program. Then, when your program ends, the RS-232-C SERIAL INTERFACE port will automatically be closed *before* your program files.

Using any of the SOUND commands during Concurrent Mode I/O can have disastrous effects, from changing the Baud rate to stopping I/O completely before your data is transferred. If you must use SOUND commands, write your program so that the IOCB you are using for Concurrent Mode I/O is closed before the SOUND command.

PARALLEL INTERFACE (PRINTER) PORT

This section describes error conditions that could occur when using the PARALLEL INTERFACE (printer) port. There are no new error codes associated with the interface module's printer port. However, the meaning of some of the errors is slightly different between the interface module and other ATARI Printers.

If an error occurs that is not listed here, consult the *ATARI BASIC Reference Manual*. Errors are listed here only if they have some new meaning when reported by the interface module.

ERROR 138   Timeout. The interface module did not respond to a request from the computer. Check the cables. Make sure the interface module and attached printer are turned on. The interface module will **not** respond to printer control commands from the computer if the FAULT wire to the printer is low (caused by loose cable or printer off).

ERROR 139   NAK. The interface module refused an illegal printer command. Make sure that Aux1 and Aux2 are specified as zero (0) in your OPEN command for the printer. This error also occurs when the printer appears active (FAULT line is high) but the printer fails to respond to characters sent to it within 30 seconds. Check the switches on the printer. Is it online? If the printer is not performing within 30 seconds, change your PRINT statements to break down transmissions into smaller chunks.

Use of multiple printer control codes that involve carriage motion (with the exception of end-of-line), can cause an ERROR 139 (Device NAK). Carriage motion includes backspace, forward and reverse linefeeds, and partial linefeeds.

The ATARI 850 Interface Module sends data to the printer in 40-character blocks. If there is more than one carriage motion in each block, the printer may not recover in time to receive the next 40-character block.

If you should have this problem, check your program. Try to arrange your printer control codes in such a way that there is no more than one carriage motion in each 40-character block. This can be done by preceding each carriage motion with forty "null" characters. Null characters can be generated with control comma (CTRL ,) or with the BASIC command CHR$(0).

Note: Attempts to operate more than one printer at a time will result in unpre-dictable operation. While one printer may ''win'' most of the time, errors are always possible, and exactly which error occurs is a matter of chance. If you have more than one ATARI Printer attached to your computer, turn on only one at a time.

PRODUCT SPECIFICATIONS

ATARI 850 Interface Module

- SIZE
  9 5/8 x 6 5/8 x 2 inches

- TEMPERATURE
  Operating environment: 50 to 110 degrees Fahrenheit. (10 to 43 degrees Centigrade)

  Storage: 40 to 160 degrees Fahrenheit. (40 to 71 degrees Centigrade)

- ELECTRICAL REQUIREMENTS
  Uses 117 VAC (17 watts) with power adapter
  10-VAC/1.5-A supplied by UL listed transformer with 10-foot cord

- HUMIDITY
  Operating environment: 20 percent to 80 percent relative humidity (no condensation).

  Storage: 5 percent to 95 percent (no condensation).

- CONTROLS
  ON/OFF switch with red POWER ON indicator lamp.

- DATA INTERFACE
  Four serial interface ports for use with the ATARI 830 Acoustic Modem and other EIA RS-232-C compatible peripherals. All have Send and Receive data signals and signal ground. Port 1 has five additional control signals. 20-mA current loop is connectible on Port 4 for teletype.

  Centronics-type 8-bit parallel output interface port for use with the ATARI 825 Printer.

ATARI 830 Modem

- SIZE
  10.2 x 4.7 x 2.3 inches

- WEIGHT
  1.5 lb.

- TEMPERATURE
  Operating environment: 32 to 122 degrees Fahrenheit. (0 to 50 degrees Centigrade)

  Storage: -40 to 140 degrees Fahrenheit. (-40 to 60 degrees Centigrade)

- ELECTRICAL REQUIREMENTS
  Uses 117 VAC (4 watts) with power adapter
  24 VAC/150mA supplied by UL-listed wall-mount transformer with 6-foot cord.

- HUMIDITY
  Operating environment: 10 percent to 90 percent relative humidity (no condensation).

  Storage: 5 percent to 95 percent (no condensation).

- TRANSMITTER FREQUENCIES
  Originate:
Mark:	1270 Hz
Space:	1070 Hz

  Answer:
Mark:	2225 Hz
Space:	2025 Hz

- RECEIVE FREQUENCIES
  Originate:
Mark:	2225 Hz
Space:	2025 Hz

  Answer:
Mark:	1270 Hz
Space:	1070 Hz

- TRANSMIT/RECEIVE RATE
  300 Baud (Max.)

- RECEIVE SENSITIVITY
  -45 dBm

- CONTROLS
  FULL/TEST/HALF

FULL:	Sets Full-Duplex operation.
TEST:	Sets up audio self-test mode.
HALF:	Sets Half-Duplex operation.

- ANS/OFF/ORIG

ANS:	Sets answer mode.
OFF:	Turns unit power off.
ORIG:	Sets originate mode.

- INDICATORS
  - POWER: Red indicates ON.
  - READY: Ready to communicate when ON.

- DATA INTERFACE
  The modem provides an RS-232-C interface via a standard 25-pin female D-connector. The table below lists the signals used by the modem.

  OUTPUTS: Mark(OFF):   -8V
                        Space(ON): +10V

  INPUTS: Mark(OFF):   -3 to -25V
                        Space(ON): +3 to +25V

Table F-1 Pin Connections

PIN NUMBER	SIGNAL MNEMONICS	FUNCTION	SIGNAL DIRECTION
2	XMT	Transmit Data	Input to modem
3	RCV	Receive Data	Output to interface module
5	CTS	Clear to Send (On with Carrier Detect)	Output to interface module
6	DSR	Data Set Ready (ON with Carrier Detect)	Output to interface module
7		Signal Ground	Common
8	CRX	Carrier Detect	Output to interface module

INDEX

H

Half Duplex 3, 6, 18, 22, 69
Handshake, handshaking 3, 6

I

IOCB 10, 17, 19-21, 23, 28-33, 38-43, 45,
    69, 70, 81, 85
IOCB (defined) 15, 81

L

Line Feed (see LF)
LF (Line Feed) 15-17, 25-28, 54, 58-60, 70,
    71

M

Mark 21, 29, 63-65, 67, 79

N

NAK (Not Acknowledge) 46, 58, 68

O

Operating System (see OS)
OS (Operating System) 9, 10, 13, 34, 67

P

PARALLEL INTERFACE 67, 69
Parity 15, 16, 21, 25, 27

R

RCV (Received Data) 4, 5, 18, 43, 44, 47, 48,
    64, 65, 69
Received Data (see RCV)
Request to Send (see RTS)
RS-232-C 7, 9
RS-232-C defined 3-6, 63
RS-232-C Port 7-9, 13, 15, 16, 22, 26
RS-232-C Handler 8, 13, 37, 38, 40, 67, 69
RTS (Request to Send) 4, 5, 16, 18, 29, 30, 63,
    64, 84

S

Serial Interface Port (See RS-232-C Port) 13,
    15-17, 22, 29-32, 34, 35, 39, 43-45, 53, 58,
    63, 67, 68, 81, 87-92
Set Baud Rate (see Configure Baud Rate)
Set Translation Mode (see Configure Translation
    Mode)
Signal Detect (see CRX)
Signal Ground 4, 5
THE SOURCE 11
Space 21, 29, 63-65, 79
Start Bits 21, 79
Start Concurrent Mode 18, 19, 33, 35, 37-41,
    46, 68, 85
Stop Bits 15, 16, 22, 45, 64, 79
Status Request 34, 70
Symbolic Constants 10

T

TeleLink 6, 11, 15, 19
Terminal 3
Translation 15, 16, 79
Transmit Off (see XOFF)
Transmit On (see XON)
Transmitted Data (see XMT)

X

XIO Commands 45, 81-85
XIO 32 81
XIO 34 19, 30, 84
XIO 36 19, 21, 81
XIO 38 19, 28, 83
XIO 40 10, 37, 38, 85
XMT (Transmitted Data) 4, 5, 18, 29, 30, 64,
    65, 67, 68, 84, 85
XOFF (Transmit Off) 6, 63
XON (Transmit On) 6, 63

ATARI®

ATARI®
A Warner Communications Company

C017652 REV. B