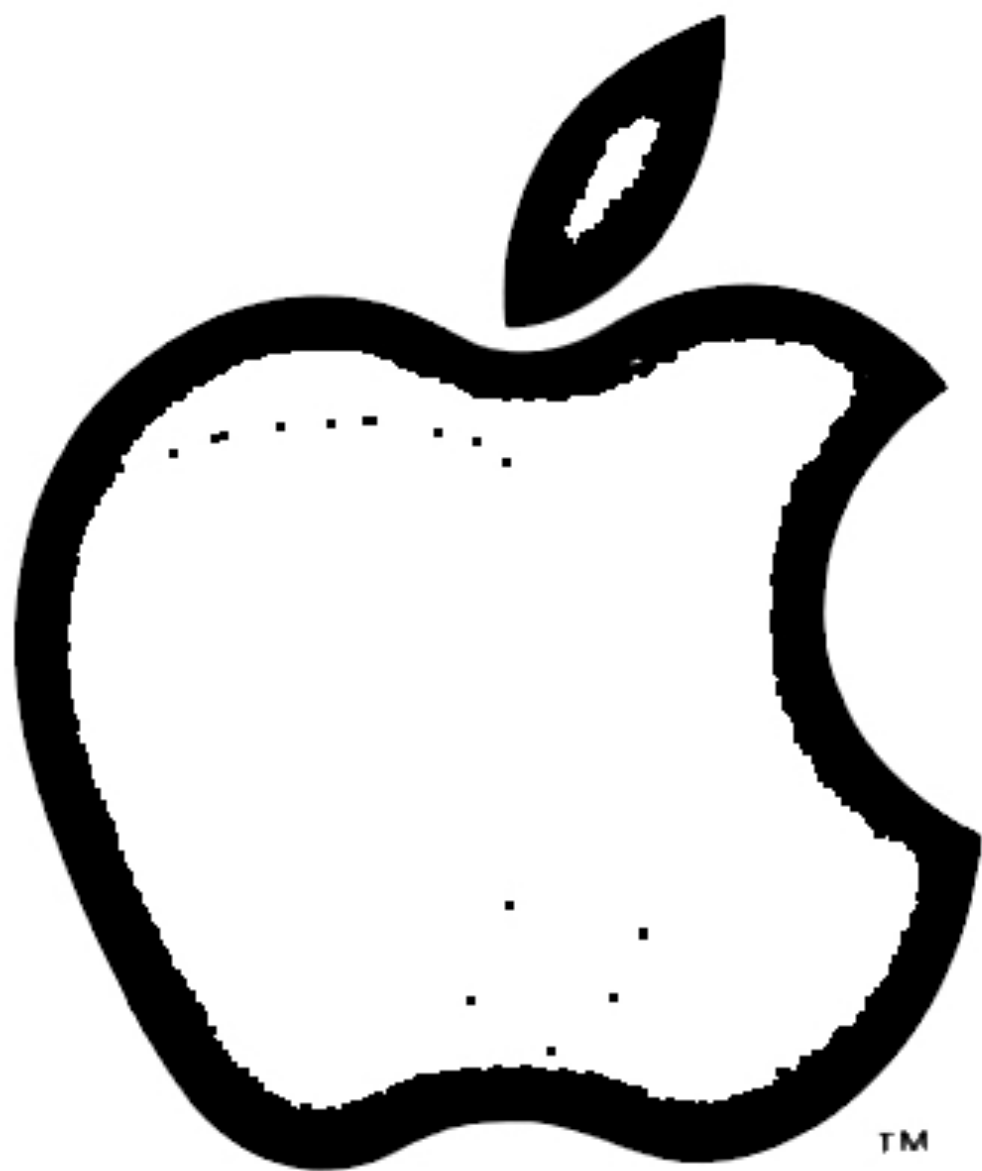


SERIAL INTERFACE CARD

INSTALLATION AND OPERATING MANUAL



Apple Intelligent Interfaces

© Apple Computer Inc.

WP-A2L0008/030-0012 Printed in U.S.A. 3/79-10K RP

APPLE II
SERIAL INTERFACE CARD (A2L0008)
INSTALLATION AND OPERATING MANUAL

**PLEASE READ THIS MANUAL BEFORE ATTEMPTING TO INSTALL
THE SERIAL INTERFACE CARD INTO THE APPLE II.**

TABLE OF CONTENTS

SECTION	TITLE	PAGE
	INTRODUCTION	
I	INSTALLATION	3
	Installing the Serial Interface	
	Compatibility with External Devices	
	RS232 Connector Usage	
	Current-Loop Operation with a Teletype	
II	OPERATION	9
	Using the Serial Interface	
	Preliminary Discussion of IN# and PR#	
	Sending Output and Receiving Input	
III	DEFAULT PARAMETERS AND THE DIP SWITCH	15
	Initialization	
	Operating Parameters	
	Setting the DIP Switch Defaults	
	Permanent Defaults	
IV	ACCESS TO OPERATING PARAMETERS	19
	Description of Serial Interface Operation	
	Transmission Data Sequence	
	Lower-Case Characters	
	Changing Interface Parameters through Software Commands	
V	DIRECT USE OF THE INTERFACE	27
	Transmitting a Character without Using PR#1	
	Batch Moves	
VI	APPENDIX: SERIAL INTERFACE TIMING	31
	Table of Baud Rate Quantum Numbers	
	Circuit Diagram	
	Assembly Listing	

APPLE II SERIAL INTERFACE CARD

INTRODUCTION

These are the fundamental abilities of the APPLE Serial Interface, using the nearly universal RS232 standard:

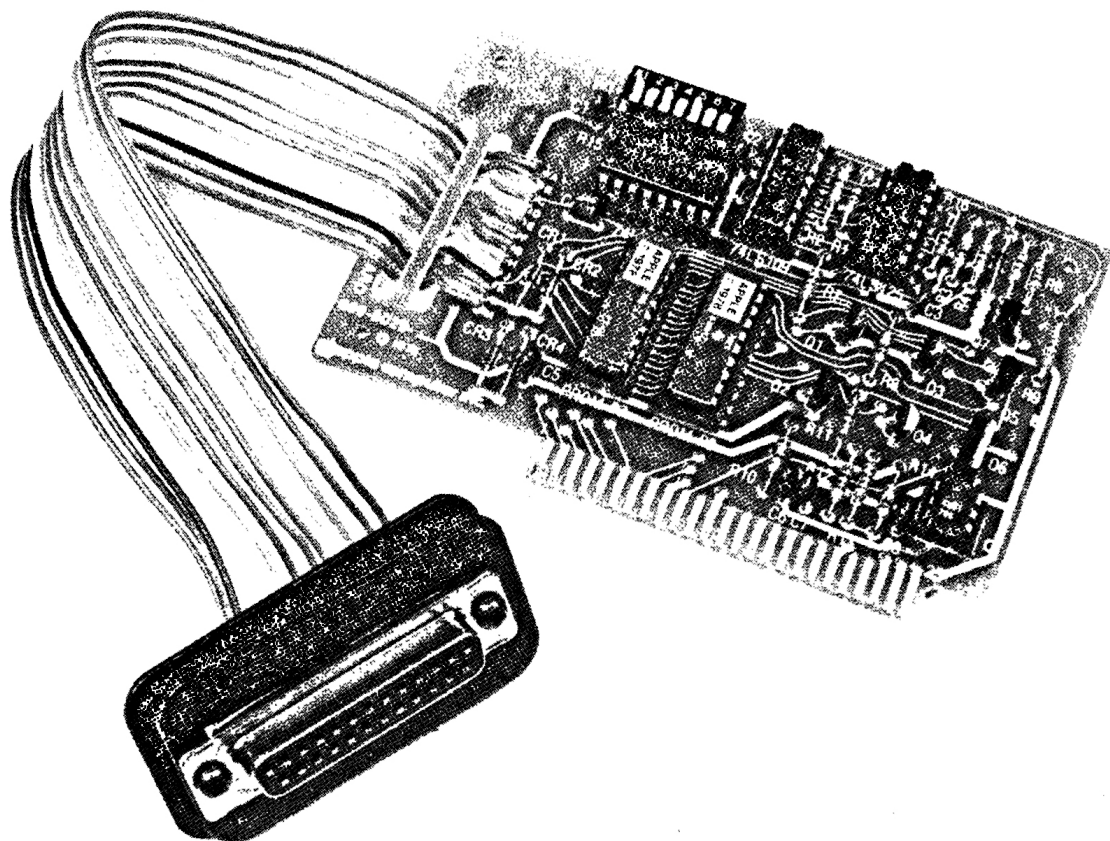
1. Output from the APPLE II can be sent to a serial printer or other external serial device, to the APPLE's TV screen, or to both. The Serial Interface can supply the necessary line-feeds with carriage-returns, etc.
2. Input for the APPLE II can be taken either from an external device or from the APPLE's keyboard, or from both simultaneously.
3. The APPLE II can handle half-duplex communications at rates from 75 to 19,200 baud, in both directions, with a printer, another APPLE, a terminal, modem or other RS232 external device.
4. The Serial Interface can also be connected for current-loop operation with a Teletype.

While this document is intended primarily for APPLE users who are familiar with the RS232 interface, many of the terms and concepts will be explained.

I INSTALLATION

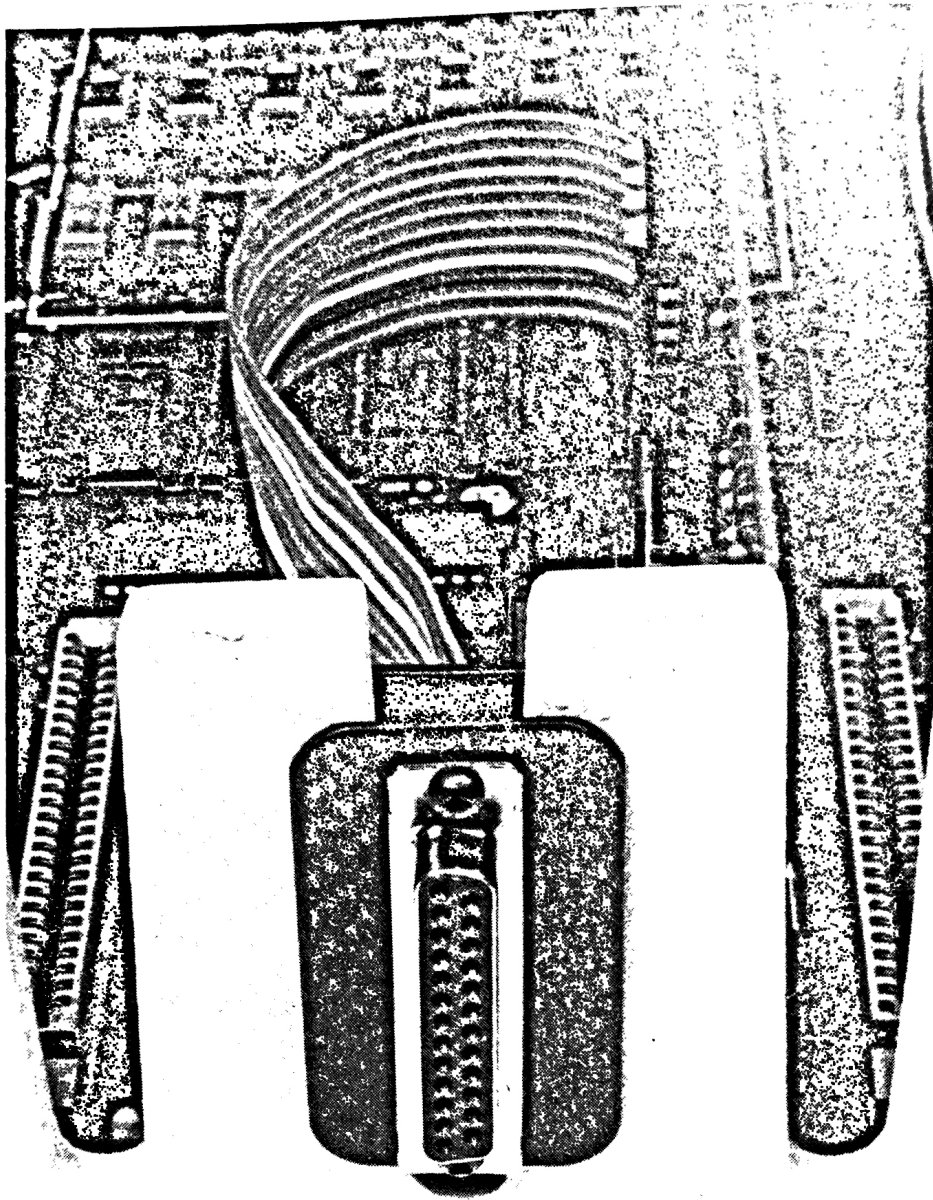
HOW TO INSTALL THE SERIAL INTERFACE

The Serial Interface consists of three parts: the Interface printed-circuit card itself, a female DB-25 connector, and a flat ribbon cable between them.



To install the Serial Interface, you will simply plug the Interface card into a socket inside the APPLE II, and then tighten a clamp to hold the DB-25 connector in place, as follows:

1. **Turn off the power switch** on the back of the APPLE II. This is important to prevent damage to the computer.
2. Remove the cover from the APPLE II. This is done by pulling up on the cover at the rear edge (the edge farthest from the keyboard) until the two corner fasteners pop apart. Do not continue to lift the rear edge, but slide the cover backward until it comes free.
3. Inside the APPLE II, across the rear of the circuit board, there is a row of eight long, narrow sockets called "**slots.**" The leftmost one (looking at the computer from the keyboard end) is slot #0, and the rightmost one is slot #7. Insert the "fingers" portion of the Serial Interface card into slot #1, the *second* socket from the left. The "fingers" portion will enter the socket with some friction and will then seat firmly. The Interface card may be placed in any slot except slot #0, the leftmost. However, APPLE's stan-



- standard location for printer interfaces is slot #1 (the second from the left). This manual and most APPLE software for the Serial Interface are written assuming you have installed the Serial Interface card in slot #1.
- Slip the DB-25 connector and its two metal plates as far down as possible into one of the three long vertical openings in the back of the APPLE II case. One plate goes on the inside of the case; the other plate goes on the outside of the case with the connector's flange on the outside of this plate. Any of the three large vertical openings may be used, but it is customary to use the middle one. Notice that the connector is not symmetrical. When seen from the back of the APPLE II, the longer side of the connector should be on the left (although it will work in either position).
 - Tighten the screws on the DB-25 connector just until the connector assembly can no longer be moved in the opening. Excessive tightening will cause the metal plates to bend.
 - Replace the cover of the APPLE II, remembering to start by sliding the front edge of the cover into position. Press down on the two rear corners until they pop into place.
 - The Serial Interface is installed, and the APPLE II may now be turned on.

COMPATIBILITY WITH EXTERNAL DEVICES

For communications between computers and computer-related equipment, the most widespread and universal standard is the RS232 standard. The RS232 standard specifies the electrical parameters, the form of the signals, and even the type of connector to be used in an interface. The APPLE Serial Interface complies with this standard.

The RS232 standard allows for a number of different communication speeds. These speeds are measured in terms of a unit called the "baud." Each multiple of 10 baud is equal to about 1 character sent or received per second; 300 baud is roughly equal to 30 characters per second. The Serial Interface can operate at any of 256 different speeds, from 75 baud to 19,200 baud.

Computers and their related devices do not actually send the keyboard characters themselves, of course. Each character is encoded in the form of electrical signals, and it is these electrical signals which are sent and received.

The APPLE Serial Interface can communicate with any device that specifies RS232 operation between 75 and 19,200 baud. Many devices can operate at a number of speeds. Very often a set of switches or a rotary dial selects the baud rate. These external devices should be set to a particular baud rate before being connected to the APPLE. The highest baud rate available is usually preferred. The Serial Interface should be set for the same baud rate, using the first 3 levers of the Serial Interface's DIP switch (this is explained in the section, SERIAL INTERFACE OPERATING PARAMETERS). All common baud rates are listed in the section, SERIAL INTERFACE TIMING.

While such operation does not conform to the RS232 standard, the APPLE Serial Interface can also be operated in the current-loop mode necessary to communicate with a serial teleprinter such as the Teletype Model 33ASR.

RS232 CONNECTOR USAGE

The standard DB-25 connector, which is supplied with the Serial Interface, has 25 pins. Six of these are connected internally to the APPLE Serial Interface, but for most applications only three of them need be used. If you don't have a ready-made cable that can go from the Interface's DB-25 connector to the external device, then you will have to wire an interconnecting cable. A cable is just a number of electrically distinct wires that physically run alongside each other. When you wire the cable, you will have to refer to the DB-25 connector's pin numbers. These numbers are molded into the connector, although sometimes they are almost vanishingly small.

The following list describes the functions of the active pins on the Serial Interface connector. The other pins may be left unconnected.

PINS 4 & 5 These pins have been wired together (jumpered) at the Serial Interface card. No connection need be made to these pins.

PINS 6, 8 & 20 These pins have been wired together (jumpered) at the Serial Interface card. No connection need be made to these pins.

PIN 7 This is called "signal ground." It should be wired to pin 7 at the other end of the cable. If there is no connector at the other end of the cable, then the Serial Interface's pin 7 should be wired to a signal ground connection on the external device. (If this is insufficient information, any additional data would have to be supplied by the manufacturer or designer of the external device.)

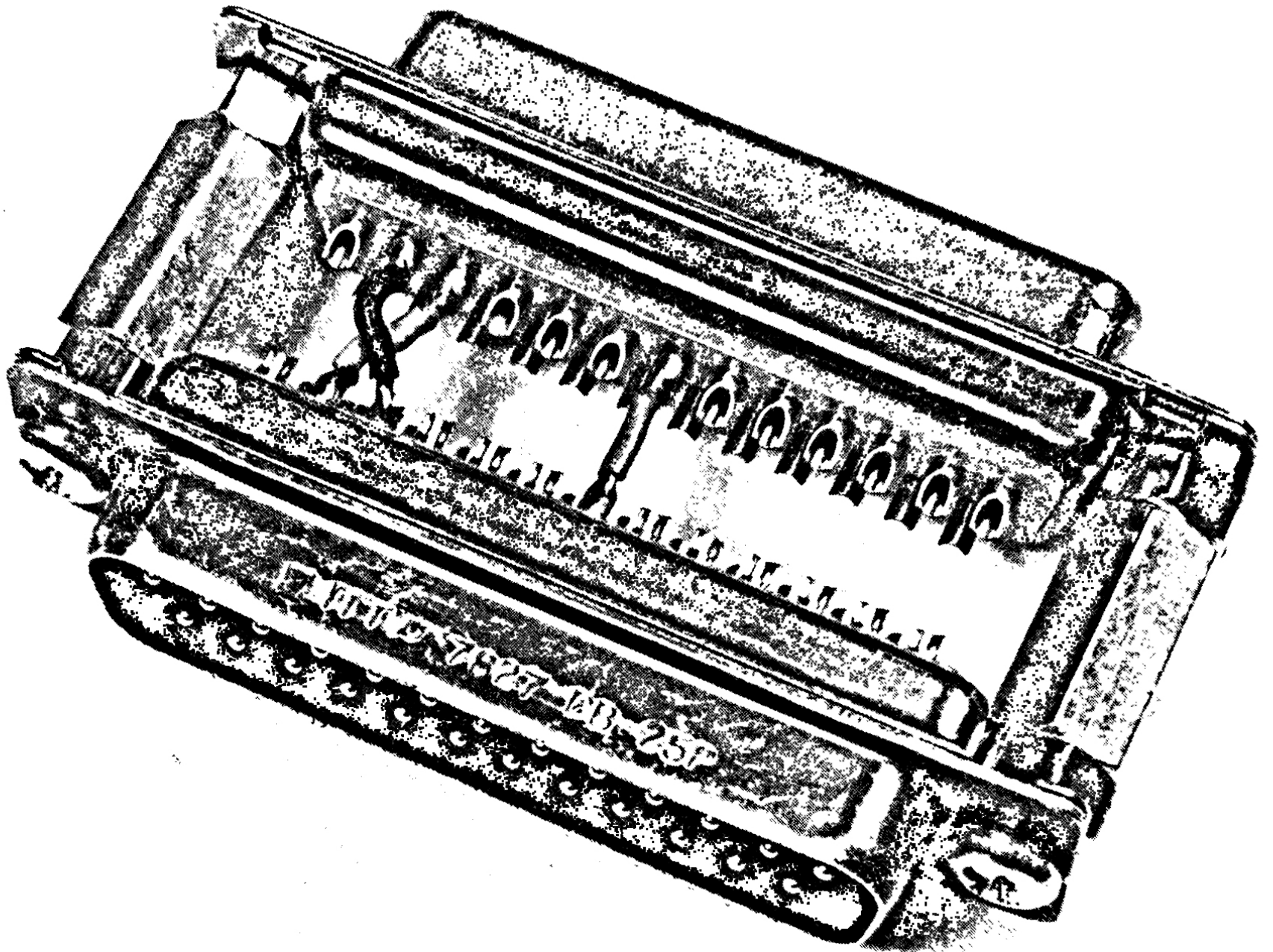
PIN 2 The characters from the external device arrive at the computer via this pin.

PIN 3 The characters leaving the computer, on their way to the external device, exit via this pin.

Pins 2 and 3 have been left for last since, if the external device end of the cable is another 25 pin connector, there are two ways that they might be wired. No damage is caused by wiring these pins the wrong way, but characters will not be sent out or received.

A. If the external device is a terminal or printer with an RS232 interface itself, then pin 2 on the APPLE's end of the cable should be wired to pin 2 at the external device's end of the cable. Similarly, pin 3 on the APPLE's end of the cable should be wired to pin 3 at the external device's end of the cable. Most of these devices, like the APPLE Serial Interface, also have a **female** DB-25 connector. Therefore your cable will (most likely) need to have a **male** DB-25 connector at *each* end.

B. If the external device is a modem, or another computer with a standard serial interface, then *its* interface will send characters out via pin 3 and receive characters via pin 2 just as the APPLE Serial Interface does. Therefore, you must wire pin 2 at the APPLE's end of the cable to pin 3 at the modem end of the cable; and wire pin 3 at the APPLE's end of the cable to



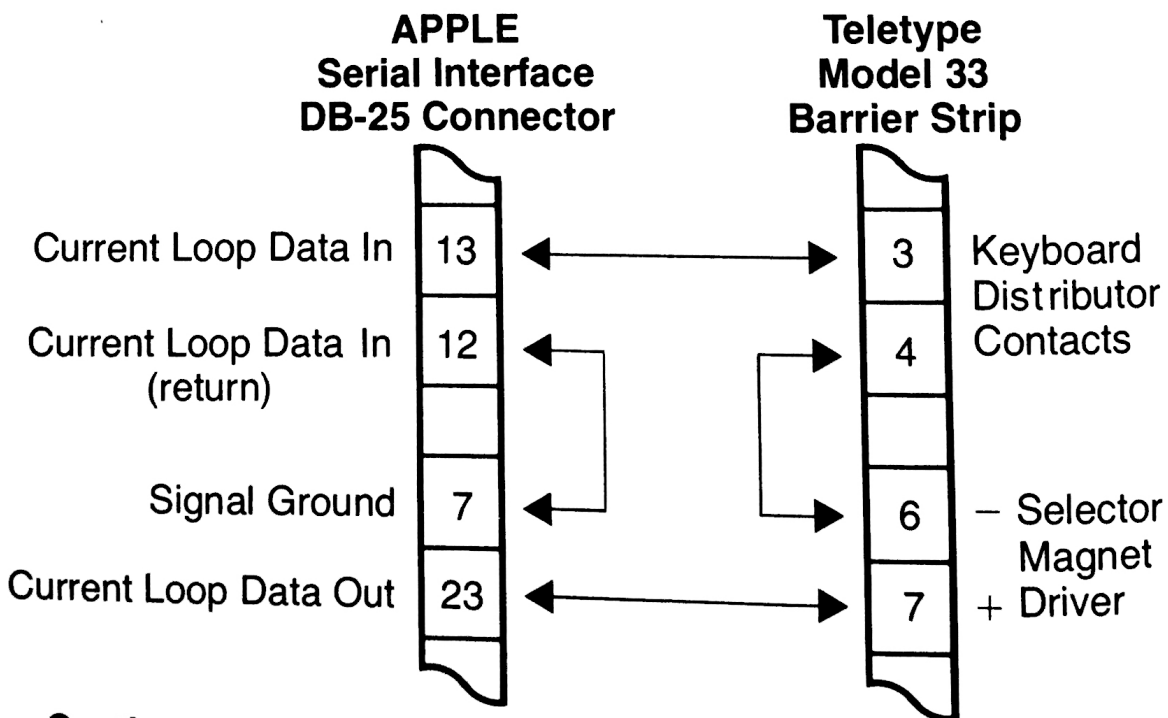
CURRENT-LOOP OPERATION WITH A TELETYPE

If you wish to use the APPLE Serial Interface to communicate with a current-loop teleprinter such as the Teletype Model 33ASR, you will be interested in three other Interface pins.

PIN 13 In current-loop operation, the characters from the Teletype arrive at the APPLE via this pin. This pin should be wired to terminal 3 on the Teletype Model 33's barrier strip.

PIN 12 The return path of the input current loop (the loop for characters arriving from the Teletype) should be connected to this pin. (In fact, the Serial Interface does not care which way current flows through this input loop. We have arbitrarily chosen pin 13 as the input and pin 12 as the return, but the roles of these two pins can be interchanged.) We suggest using signal ground (at pin 7) for the return path, in which case you should connect pin 12 to pin 7.

PIN 23 In current-loop operation, the characters leaving the APPLE, on their way to the Teletype, exit via this pin. Wire this pin to terminal 7 on the Teletype Model 33's barrier strip. The return path for this output current loop is also the signal ground at pin 7. For half-duplex operation, connect terminals 4 and 6 on the Teletype Model 33's barrier strip.



Caution! Pins 1 and 2 on the Model 33 barrier strip are connected to 120 Volts AC.

II OPERATION

USING THE APPLE SERIAL INTERFACE

The Serial Interface allows the APPLE II to communicate with other electronic devices which are *external* to the computer. These devices may be—to give a few examples—terminals, printers, or other computers. The Serial Interface can be controlled through BASIC programs or through assembly-language programs. It can also be controlled directly, by typing a few characters on the APPLE's keyboard.

In the following discussion, it will be assumed that you are familiar with the APPLE II BASIC Programming Manual, and that your APPLE II is operating in BASIC, with the Serial Interface installed in slot #1.

Here is a list of the most common tasks the Serial Interface is called upon to do, and the commands that accomplish them.

1. Send subsequent output to the Serial Interface.

PR#1

2. Cancel the effect of PR#1, sending output only to APPLE's TV screen.



PR#Ø

3. Accept subsequent input from the Serial Interface, as well as from the APPLE's keyboard.

IN#1

4. Force the APPLE to convert all lower-case characters to upper-case, as they arrive from the external device.



(type the  key, then type )

5. Allow the APPLE to accept lower-case characters that arrive from the external device. If displayed on APPLE's TV screen, the lower-case characters will appear as upper-case characters in inverse video.



These tasks are more fully explained, and some fine points considered, in the next few pages.

To understand exactly how the Serial Interface operates, it is useful to think of the APPLE II as divided into three parts:

1. The APPLE's keyboard, which **generates** characters (when you type on it).
2. The APPLE's TV screen, which can **absorb** characters (and make them visible).
3. The APPLE's processor, or "brain," which can **control** the flow of characters, and act upon them.

You can also think of any external device as being able to **generate** characters or **absorb** characters, or both. The external device may or may not have a "brain," but this is not important in understanding the operation of the Serial Interface.

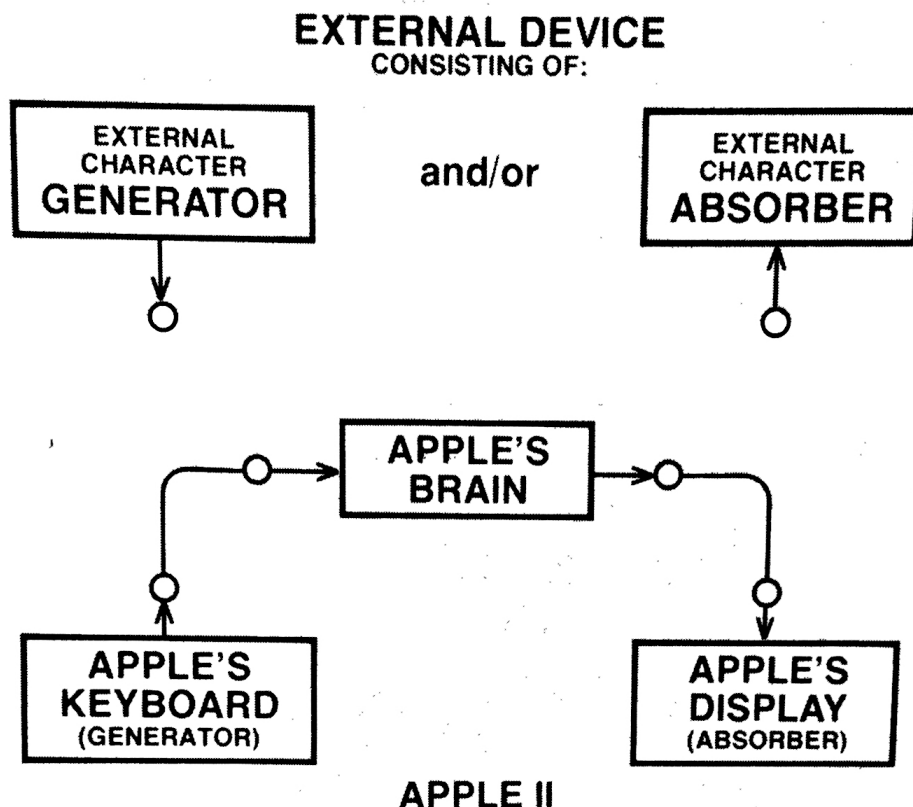


FIGURE 1

Figure 1 shows the three parts of the APPLE II, and the only parts of any external device that affect the Serial Interface. It also shows the normal interconnection between these parts. When the APPLE II is first turned on, it will ignore the external world, listening only to its own keyboard and displaying characters only on its own TV screen.

PRELIMINARY DISCUSSION OF THE IN# AND PR# COMMANDS

There are eight sockets, called "slots," on the back of the main circuit board inside the APPLE II. The leftmost one (as viewed from the keyboard end of the computer) is slot #0, and the rightmost one is slot #7. (See the section, HOW TO INSTALL THE SERIAL INTERFACE.) APPLE BASIC has two commands for selecting among these slots for input and output. In effect, when you first invoke BASIC, the commands

IN#0

and

PR#0

are automatically executed. The first of these commands, **IN#0**, tells the APPLE to

Take **IN**put from the APPLE keyboard.

And the second command, **PR#0**, instructs the APPLE to

Send **PR**inting to the APPLE's TV screen. This is the "normal," or APPLE-alone condition shown in Figure 1. Now, however, if the command (or program statement)

IN#1

is executed, the APPLE will henceforth take its input from whatever is plugged into slot #1. Similarly, if the command

PR#1

is executed, all output will be sent to whatever is plugged into slot #1. If there is nothing plugged into the specified slot, then the system may hang, or your program may be erased, or other strange behavior may result. Notice that slot #0 is special, and refers to the APPLE itself.

SENDING YOUR OUTPUT TO AN EXTERNAL DEVICE AND RECEIVING INPUT FROM AN EXTERNAL DEVICE

In the following examples, the commands

PR#1

and

IN#1

will be typed on the APPLE's keyboard. If you have put your Serial Interface into slot #1 (the second one from the left, as described in the section, HOW TO INSTALL THE SERIAL INTERFACE) the commands will work exactly as shown. If you use some other slot, you will have to substitute the number of that slot. **Slot #0 may not be used for the Serial Interface.**

Attach an appropriate cable (see the section, RS232 CONNECTOR USAGE) from the DB-25 connector of the Serial Interface to the external device with which you wish to communicate. Reset your APPLE II by pressing the

CTRL

CTRL

RESET key, and invoke BASIC typing a **B** (**B** means depressing the **B** key while simultaneously holding down the key marked **CTRL**). If you are not familiar with this procedure, see your APPLE II BASIC Programming Manual. When the prompt character appears, type the command

PR#1

(and press the **RETURN** key, of course) . From now on, any characters you type will be sent out through the Serial Interface to the external device. The characters will appear on the APPLE's TV screen only if levers 5 and 6 of the Serial Interface's **DIP** switch (located directly on the Interface card itself, near its upper edge) are **ON** when **PR#1** is typed. (This is more fully explained in the section, SERIAL INTERFACE OPERATING PARAMETERS.) Characters coming in from the external device will be ignored. The operation of the system after you type **PR#1** is shown in Figure 2.

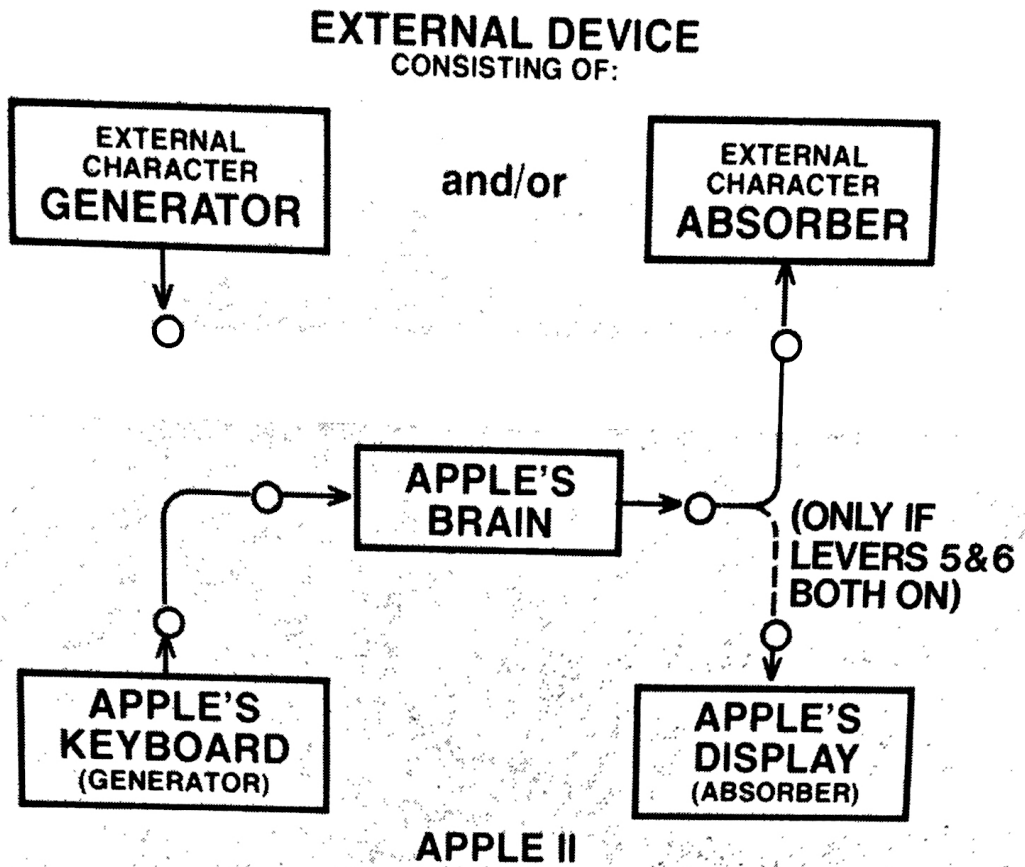


FIGURE 2

Operation of the system after executing PR#1

The APPLE's "brain" is still connected, and the command **PR#0** will restore normal (Figure 1) operation, in which the APPLE's output characters are not sent through the Serial Interface. Normal operation can also be

restored by pressing the **RESET** key and then typing a **CTRL C**, but this option is not available if the Interface is being controlled by a program.

To let the external source of characters control the APPLE II, use the command **IN#1**

After this command, the APPLE will accept input from the external device connected to the Serial Interface, as well as from its own keyboard. Figure 3 shows this condition. If there is no external device connected to the Serial

Interface, the system will "hang" after this command. Use **RESET CTRL C** to recover.

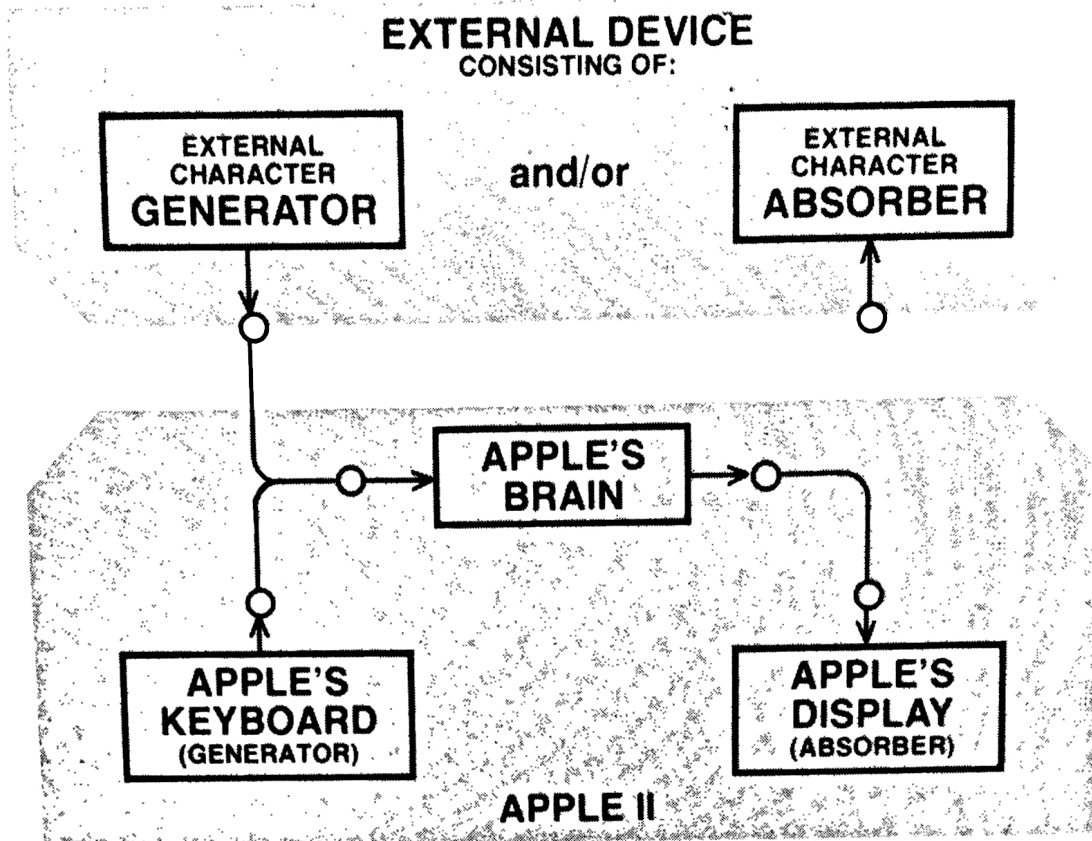


FIGURE 3

Operation of the system after executing IN#1

Normal operation is restored if the command **IN#0** is typed on the APPLE's keyboard. Normal operation is also restored if the external device sends the command **IN#0**

CTRL

Pressing the **RESET** key and typing a **C** on the APPLE's keyboard will also restore normal operation; but this cannot be done from a program, and will not be mentioned again.

Typing **both** commands, **PR#1** and **IN#1** will give the external device full control of the APPLE II. In this "remote mode" (shown in Figure 4), a friend could use your APPLE from across the country—or across the room.

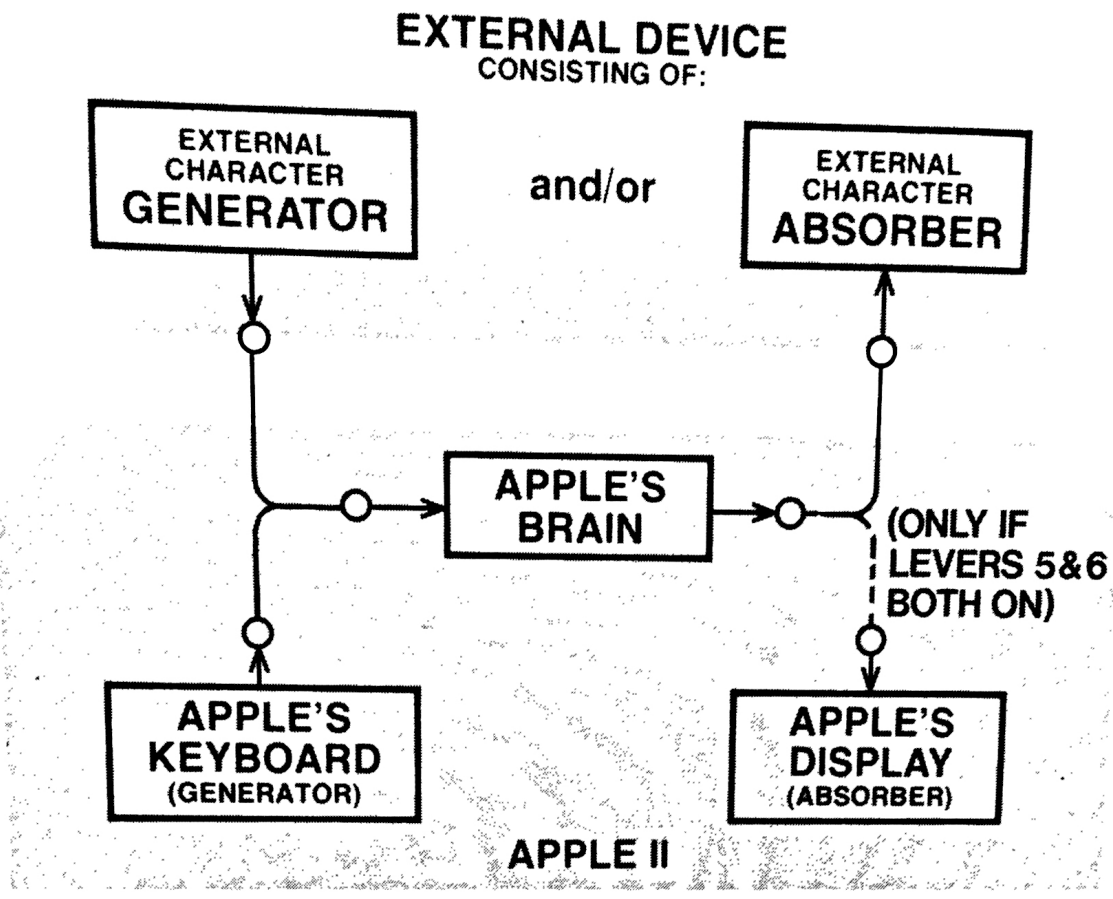




FIGURE 4

Remote Mode: operation of the system after executing PR#1 and IN#1.

III DEFAULT PARAMETERS AND THE DIP SWITCH

INITIALIZING THE SERIAL INTERFACE

Before the Serial Interface can be used, it must be **initialized**. Initializing the Interface sets all of the Interface operating parameters to their **default** values. Assuming slot #1, the Interface is initialized each time either of the following BASIC commands is typed:

PR#1  or IN# 1 

and each time any of the following Monitor commands are typed:

 
   or    or C100G 

When used within a *program*, a command (such as PR#1) that transfers APPLE's output to the Serial Interface does *not* initialize the Interface until the first character is actually sent out (with a PRINT statement, for instance). Similarly, if during a *program* a command (such as IN#1) tells the APPLE to get its input from the Serial Interface, the Interface is *not* initialized until the APPLE actually looks for its first input character (in an INPUT statement, for instance).

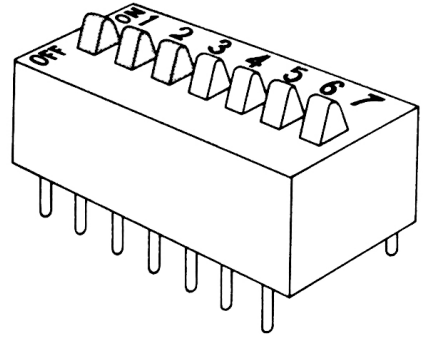
SERIAL INTERFACE OPERATING PARAMETERS

The Serial Interface has ten user-definable operating parameters. Each time the Serial Interface is initialized, the ten operating parameters are given their **default** values. Five of the default values are determined by the 7 levers of the Serial Interface's **DIP** switch (located on the Interface's printed-circuit card, near the upper edge). The **DIP** switch levers set the default values for these five operating parameters: **Baud Rate** (levers 1, 2 and 3), **Carriage Return Delay** (lever 4), **Line Width plus APPLE Video** (levers 5 and 6), and **Line Feed** (lever 7). Changing the settings of the **DIP** switch levers *after* initialization has no effect until the *next* initialization.

SETTING THE DIP SWITCH DEFAULTS

1) Baud Rate

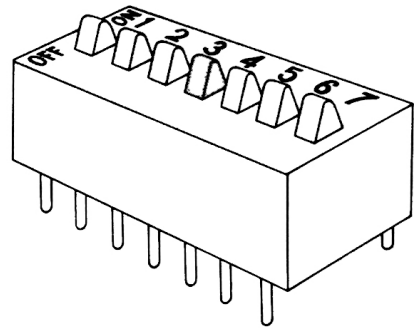
DIP Switch Levers				Default Baud Rate	
1	2	3	=		
On	On	On	=	110	baud
Off	On	On	=	134.5	baud
On	Off	On	=	300	baud
Off	Off	On	=	1200	baud
On	On	Off	=	2400	baud
Off	On	Off	=	4800	baud
On	Off	Off	=	9600	baud
Off	Off	Off	=	19200	baud



On initialization, the settings of **DIP** switch levers 1, 2 and 3 determine the rate at which bits may be transmitted to the external device. 300 baud is 300 bits per second. Under default conditions, each character is transmitted using 11 bits (1 start bit, 8 data bits, and 2 stop bits).

2) Carriage Return Delay

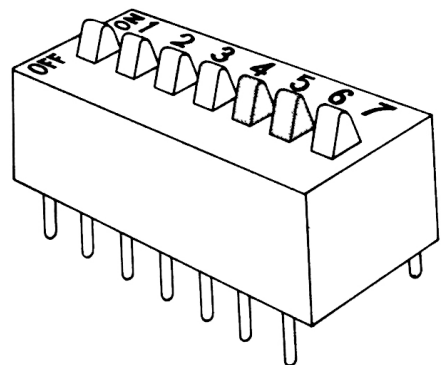
DIP Switch Lever		Default Car. Return Delay
4		
On	=	Disabled
Off	=	Enabled



If **DIP** switch lever 4 is Off (Delay Enabled), the Serial Interface will wait briefly (approximately 1/4 second) after transmitting a carriage return, to allow the printer to complete this movement. If you are transmitting to an external TV screen, this delay is probably unnecessary, and lever 4 may be turned On (Delay Disabled).

3) Line Width plus APPLE Video

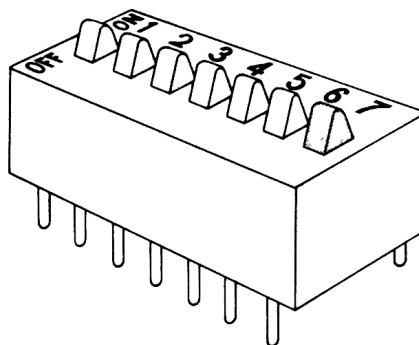
DIP Switch Levers			Default Line Width	Default APPLE Video
5	6			
On	On	=	40 Char/Line	Enabled
Off	On	=	72 Char/Line	Disabled
On	Off	=	80 Char/Line	Disabled
Off	Off	=	132 Char/Line	Disabled



After a carriage return, the settings of **DIP** switch levers 5 and 6 determine the maximum number of characters transmitted before the Serial Interface will force another carriage return to be sent out. Characters will be displayed on the APPLE's TV screen only if the default line width is set to 40 characters per line (levers 5 and 6 On). After initialization, the line width can be changed from 40 characters per line, but the display on the APPLE's TV screen will not correspond to the display on the external device, as transmitted carriage returns are not accompanied on the APPLE's TV screen by line feeds.

4) Line Feed

DIP Switch Lever	Default Line Feed
7	
On	= Disabled
Off	= Enabled



If **DIP** switch lever seven is Off (Line Feed Enabled), the Serial Interface will transmit a line feed after each carriage return it transmits. If the external device automatically supplies its own line feed after each carriage return received, you can set lever seven to On (Line Feed Disabled) to avoid double-spacing.

PERMANENT DEFAULTS

During each initialization, the five remaining operating parameters are set to their **permanent** default values:

1. **Parity** defaults to its disabled condition (no parity bit).
2. **Checksum** defaults to its disabled condition (no checksum character).
3. **Lower-Case** defaults to its disabled condition (converts all incoming lower-case characters to upper-case).
4. **Number of Data Bits** defaults to 9 (8 data bits plus one start bit).
5. **Number of Stop Bits** defaults to 2.

These parameters will be explained in the following section; they can only be changed by software commands *after* initialization.

DESCRIPTION OF SERIAL INTERFACE OPERATION

For most applications, the default operating parameters (both those that are fixed and those that can be set with the **DIP** switch) will be just what you need: your parameters will be set each time the Interface is initialized. In that case, the following section will be interesting but not necessary. However, the Serial Interface is designed to be very flexible, so that its operating parameters can be easily modified for use in a wide variety of special applications. This section gives a rather detailed description of the Serial Interface's operation. The following section shows you how to make any of the many possible modifications to that operation, should they be necessary.

Each character that is sent out through the Serial Interface is transmitted as a series of bits, in the following sequence:

1. One "Start Bit," a "low" voltage which tells the external device that a character is going to be transmitted.
2. From two to eight "Data Bits" (default is eight bits; can be changed by the user), a sequence of "high" and "low" voltages that represent the actual character code being transmitted. The default is eight bits because the APPLE normally handles data in eight-bit groups. If your external device sends and receives data in groups of fewer than eight bits, you must set the Serial Interface to send and receive these smaller groups.
3. *If enabled*, one "Parity Bit" (default is no parity bit; even or odd parity can be enabled by the user). This is a transmission-accuracy checking bit which the external device looks at for errors and then discards.

The parity bit is found as follows: all the 1-bits in the actual character code are added together, and the result's evenness or oddness is compared with the type of parity-checking selected. For instance, the binary ASCII code for the letter S is 1010011; the sum of the 1-bits is 4, an *even* number. If *Even* parity has been enabled, the comparison is *true*, and a 0-bit is sent at the end of the character. If *Odd* parity has been enabled, the comparison is *false*, and a 1-bit is sent. If parity has not been enabled, no extra bit is sent. Check your device's operation manual to see if it sends and receives parity bits.

4. From one to 127 "Stop Bits" (default is two bits; can be changed by the user), a "high" voltage which tells the external device that a character has been completely transmitted. Each external device requires a particular number of stop bits after every character; see the device's operation manual.

This same sequence will be used by the external device when it transmits to the APPLE. Timing is very important to correct transmission and reception. The Serial Interface sends out and receives bits at fixed intervals of time set by the "Baud Rate" (default is set by **DIP** switch levers 1, 2 and 3; can be

changed by the user). The Serial Interface and the external device *must* be set to the same baud rate and parity option, in order to interpret the sequence of “high” and “low” voltages correctly. The same is true when the external device is transmitting characters to the APPLE Serial Interface.

When the Serial Interface has sent out the number of characters set by the “Line Width” (default is set by **DIP** levers 5 and 6; can be changed by the user), it transmits a “Carriage Return” to the external device. After sending a carriage return, it may wait during a fixed ¼-second “Carriage Return Delay” (if enabled: default set by DIP lever 4; can be changed by the user) before sending the next character, to allow the printer to complete this movement. Then the Interface may send a “Line Feed” (if enabled: default set by **DIP** lever 7; can be changed by the user), so that subsequent characters will appear on the following line.

Finally, each time it completes sending 256 characters in a “Batch Move,” the Serial Interface may send a “Checksum” character (default is no checksum character; can be changed by the user). This is a transmission-accuracy checking character which the external device looks at for errors and then discards. The checksum character is found by XORing the previous 256 characters, as follows: the second character is XORed with the first, the third with that result, the fourth with *that* result, and so on. Check your device’s operation manual to see if it sends and receives checksum characters. During Batch Moves, the Serial Interface and the external device must be set to the same checksum option.

LOWER-CASE CHARACTERS

While the APPLE generates and displays characters only in upper case (capital letters), your external device may generate both upper-case and lower-case characters. When lower-case characters are received by the APPLE through the Serial Interface, they can be treated in two different ways:

1. Convert all incoming lower-case characters to upper-case characters (incoming upper-case characters are not affected). Since this is the usual APPLE mode, any incoming characters displayed on the APPLE’s TV screen will look fine. This is the permanent default condition; but it can be changed by the user, after initialization.
2. Accept all incoming lower-case characters as lower-case (again, incoming upper-case characters are not affected). APPLE’s TV screen *display* of characters is designed for upper-case only, so any display of incoming lower-case characters will look strange. If the characters are being displayed as they arrive from the external device, APPLE will show the lower-case characters as upper-case characters in *inverse video* (black letters on a white background). Once stored in APPLE’s memory, lower-

case characters will be displayed (when LISTed, for instance) as a strange assortment of upper-case characters in normal (white on black) video. There is one exception to this: if the stored lower-case characters are being displayed as they are sent out through the Serial Interface, they will again appear as upper-case characters in inverse video. Note that these peculiar *displays* do not reflect the lower-case characters themselves: in this mode they are *stored* correctly in APPLE's memory, and may be printed correctly on any appropriate external device. Note also that this mode does not add any capability to *generate* lower-case characters from the APPLE keyboard.

CHANGING SERIAL INTERFACE PARAMETERS THROUGH SOFTWARE COMMANDS

Ten of the Serial Interface parameters can be changed from their initialized (default) values, through the use of commands in machine language or BASIC. Once an Interface parameter is set by a software command, that parameter remains unchanged until the Serial Interface is reinitialized or the parameter is reset by another software command. For more discussion of the various parameters functions, see the previous section.

In the following descriptions, the letter "s" refers to the number of the printed-circuit board slot inside the APPLE, in which the Serial Interface card is installed (see the section, HOW TO INSTALL THE SERIAL INTERFACE).

1. BAUD RATE (assembly-listing variable: BRATE)

Memory location 1144+s (\$478+s, in Hexadecimal) contains APPLE's baud "quantum" number, which specifies how many "quanta" the APPLE is to wait between sending out bits through the Serial Interface. One quantum equals 53 APPLE II cycles (51.94 microseconds) per transmitted bit. The default value is set with levers 1, 2 and 3 of the Interface card's DIP switch (see the section, SETTING THE DIP SWITCH DEFAULTS). From BASIC, to change the baud rate from the default value to B use the command

POKE 1144+s, r

where r is the integer, from 0 through 255, that is closest to

$1 / (.00005194 * B)$

For further information, see the section SERIAL CARD TIMING.

2. STOP BITS (assembly-listing variable: STBITS)

Memory location 1272+s (\$4F8+s, in hexadecimal) contains the Number of Stop Bits (Note: the one parity bit is *included* in this number, if

parity is enabled). The default value is 2 stop bits (and no parity bit). To change the number of Stop Bits from BASIC, use the command

POKE 1272+s, r

where r is an integer, from 1 through 127. To determine the correct number of stop bits for your external device, see the external device's operation manual.

Note: you must add the one parity bit to the Number of Stop Bits, if parity is enabled.

3. PARITY/CHECKSUM OPTIONS (assembly-listing variable: STATUS)

Memory location 1400+s (\$578+s, in hexadecimal) contains a number, the lower three bits of which determine two parity options (enable/disable and even/odd) and one checksum option (enable/disable). If the remote device with which your Serial Interface is communicating requires a parity bit to be sent or received with each character, you can tell your Serial Interface to do this task. You can also specify which type of parity check (even parity or odd parity) is to be sent and received. If your remote device requires that a checksum be sent after every 256 characters in a Batch move, you can tell the Serial Interface to send one. To decide whether your external device requires either a parity bit or a checksum character (or both), consult the device's operation manual. The three Parity/Checksum options are changeable from BASIC by using the command

POKE 1400+s, r

where r is an integer from 0 through 7. The actual value that r should be assigned is determined as follows:

- | | | |
|--------|---------------------|---------------------------------|
| Bit 0: | 1 = odd parity | (This is the least significant, |
| | 0 = even parity | or rightmost, bit.) |
| Bit 1: | 1 = no parity | (initial default value) |
| | 0 = parity enabled | |
| Bit 2: | 1 = no checksum | (initial default value) |
| | 0 = checksum enable | |

First determine whether or not a parity bit need be sent (Bit 1). If yes, then decide whether the parity should be odd or even (Bit 0). Also, determine whether or not a checksum character need be sent during Batch moves (Bit 2). For example, let's assume that an even parity bit must be sent, with no checksum. Bit 0 gets a value of 0, Bit 1 gets a value of 0, and Bit 2 gets a value 1. This binary number 100 is converted to its decimal equivalent of 4 and POKE'ed (assuming slot #1):

POKE 1401, 4

4. INPUT/OUTPUT BUFFER (assembly-listing variable: BYTE)

Memory location 1656+s (\$678+s, in hexadecimal) is the input buffer for the individual character that has just been received through the Serial Interface from the external device. Assuming the Interface is in slot #1, the BASIC command

PRINT PEEK (1657)

will print on the APPLE's TV screen the ASCII value of the character just received.

5. LINE WIDTH (assembly-listing variable: PWDTH)

Memory location 1784+s (\$6F8+s, in hexadecimal) contains the "Printer Width," or number of characters per line. After transmitting this number of characters, the Serial Interface will then transmit its Carriage-Return sequence. To change the number of characters per line, from BASIC, use the command

POKE 1784+s, r

where r is an integer, from 0 through 255, specifying the number of characters per output line. To determine the maximum line width for your external device, consult the device's operation manual.

Note: if r is set to zero, the Serial Interface will not force any carriage returns to be transmitted. The output characters will be transmitted in a continuous stream.

6. DATA BITS (assembly-listing variable: NBITS)

Memory location 1912+s (\$778+s, in hexadecimal) contains the number of Data Bits, plus one for the start bit. In the APPLE, data is handled in groups containing eight bits. If you are communicating with an external device which also handles data in eight-bit groups, the default Number of Data Bits is perfect (8 data bits plus 1 start bit). However, if your external device handles data in groups of fewer than eight bits, you must set the Serial Interface to send and receive these smaller data groups.

When receiving data groups of fewer than eight bits, the Serial Interface will supply 1's to fill the remaining high-order bits of each eight-bit group in APPLE's memory. Similarly, when the Serial Interface is transmitting data groups of fewer than eight bits, the unused high-order bits in each of the APPLE's eight-bit data groups must be set to 1's.

To change the Number of Data Bits from BASIC, use

POKE 1912+s, r

where r is an integer, from 3 (2 data bits plus one start bit) through 9 (8 data bits plus one start bit).

Note: to calculate *r*, you must add one start bit to the number of data bits. If *r* is set to less than the default value of 9 (8 data bits plus one start bit), you must set the unused high-order data bits to *ones* before transmitting the data. Received data will also have unused high-order data bits set to *ones*.

Example: Binary Coded Decimal is a code for sending numbers in four-bit data groups. The BCD code for the number 7 is 0111. If the Number of Data Bits, *r*, is set to 5 (4 data bits plus 1 start bit), BCD for the number 7 must be stored in the APPLE's eight-bit byte as 11110111 before the data group 0111 can be transmitted. Similarly, if the data group 0111 is received by the Serial Interface, it will be stored in the APPLE's eight-bit byte as 11110111.

7. OPERATION MODES (assembly-listing variable: FLAGS)

Memory location $2040+s$ ($\$7F8+s$, in hexadecimal) contains a number, four of whose bits determine four separate modes of operation. To alter the operation modes from BASIC, use the command

POKE $2040+s$, *r*

where the value of *r* is determined by use of the following table:

r's Binary Bit	(Decimal Equiv., If Bit=1)	Operation Set By Bit Value	Default
Bit#0	(1)	1 = Line feed after carriage return 0 = No line feed	(Set by Lever 7)
Bit#5	(32)	1 = Lower-case input enable 0 = Convert lower-case to Upper-case	(Permanent Default)
Bit#6	(64)	1 = No delay after carriage return 0 = Carriage return delay enable	(Set by Lever 7)
Bit#7	(128)	1 = No display on APPLE's TV 0 = APPLE's display enabled	(Set by Levers 5&6)*





*APPLE's TV display is only enabled during initialization if DIP switch levers 5 and 6 are both On.

For example, let us assume that you wish to have line feeds, upper-case only, carriage return delay, and no APPLE display. This would require that bits 0 and 7 have a value of 1, and bits 5 and 6 have a value of 0. Add up the decimal equivalents of all of the bits that were assigned the value 1 (the decimal equivalents are the numbers in parentheses, next to

the Bit #'s). The decimal equivalents for bits 0 and 7 are 1 and 128 respectively; therefore the total of the decimal equivalents is 129. This value is assigned to r, and POKE'd (assuming slot #1):

POKE 2041, 129





If you wish to change only Bit #5 (lower-case input enable/convert), you can do so with the following commands:

  (Press and release the  key, and then type )

This changes r's Bit #5 to a zero, the default value. After this command, all lower-case characters arriving through the Serial Interface from an external device will be converted to upper-case characters. Incoming upper-case characters are not affected. This is the APPLE's usual mode, so any APPLE display will look fine.

This changes r's Bit#5 to a one. After this command, lower-case characters arriving through the Serial Interface from an external device will be stored as lower-case characters in APPLE's memory. Upper-case characters are not affected. Since the APPLE was designed for upper-case characters only (BASIC will accept lower-case characters only in quoted strings), any APPLE *display* of these lower-case characters will look strange on the TV screen. See the previous section for details. However, the characters are *stored* correctly, and may be printed correctly on any appropriate external device.

Note: the commands   and   are Serial Interface *input* commands. They will have no effect unless the Interface has been initialized for input (by IN#1, for instance).

8. TAB

The TAB and comma functions in Integer BASIC (HTAB in APPLESOFT) will sometimes work in conjunction with the Serial Interface, but have several restrictions (fewest for comma-tabbing). A TAB of less than 18, if it would end directly on a character already printed, may be simply tabbed from that character's position. No TAB can cause printing to occur to the *left* of the last printed character on the current line. An attempt to do so usually causes printing to occur in the first available position to the *right* of the last printed character. Both Integer BASIC's TAB and APPLESOFT's HTAB send out a carriage return for every 40 positions in the tab instruction, and then tab the remaining positions. For tabbing to any position (including those beyond position 40), you can use the BASIC command

POKE 36, r

where r is an integer, from 0 through 255, equal to the number of print

positions to be tabbed. This command suffers most of TAB's restrictions, except for the 40-position limit. In APPLESOFT, the TAB function (used inside a PRINT statement) can also cause tabbing of more than 40 positions.

V DIRECT USE OF THE INTERFACE

TRANSMITTING A CHARACTER WITHOUT USING PR#1

Occasionally, it is useful to send a character out through the Interface *without* using a PR# command to change the “output vector” (the system pointer that tells your APPLE where to send its output, normally to its TV screen). To use the Serial Interface *directly*, follow these two steps:

1. Into APPLE’s accumulator, put the ASCII code of the character to be sent.
2. CALL –16384 + (256 * s)

where s is the Interface’s slot number (the equivalent hexadecimal location to CALL is \$Cs00). There are various ways to get a number into the APPLE’s accumulator, but one way is to write a very small machine-language subroutine to do it, and then CALL that subroutine from your BASIC program. To begin, press the **RESET** key to enter the Monitor (prompt character: *), and then (assuming slot #1) type

```
300: A9 11 4C 00 C1 RETURN
```

Check your work by typing
300L

Ignoring most of the resulting display, the first two lines should look like this:

```
0300— A9 11 LDA #11  
0302— 4C 00 C1 JMP $C100
```

The first instruction (at hexadecimal location \$300) tells the APPLE to **LoaD** the **Accumulator** with the number in the next location (hexadecimal location \$301). For now, that number is \$11. The second instruction is equivalent to CALL –16128 in BASIC: it tells the APPLE to **JuMP** to hexadecimal location \$C100, which starts the Serial Interface character output routine. To use this subroutine in your BASIC program, you must first put into hexadecimal location \$301 (that’s location 769, in decimal) the ASCII code for the character you want the Serial Interface to send out. Then you will CALL the subroutine at hexadecimal location \$300 (768, in decimal). Here is a short program that uses the above machine-language routine to send out one character at a time:

```
10 INPUT “LETTER?”, L$  
20 POKE 769, ASC(L$)  
30 CALL 768  
40 GOTO 10
```

BATCH MOVES

At times it is useful to send or receive large amounts of information very quickly. This can be accomplished through use of a *Batch Move*. The Batch subroutines are “utility” routines. They are intended to be used for special

applications such as Data Collection, Mass Storage and Retrieval Systems, and sending program sequences to control external devices. To understand how to use the Batch routines on the simplest level, refer to the examples below. For an example of using the Batch Moves from BASIC, see the next section, BATCH MOVES FROM BASIC.

Note: before using the Batch routines, you must have initialized the Serial Interface (by typing PR#1, for instance) and you must have set the desired parity and checksum parameters. The Batch Move commands deal directly with the Serial Interface (not through the input and output "vectors" set by IN# and PR#); therefore these commands are the same, regardless of which slot contains the Interface card.

1. Batch Output

When in the Monitor (prompt character: *), type

3F8: 4C 41 C9 **RETURN**

This prepares the APPLE to jump to hexadecimal location \$C941 in the

CTRL

Serial Interface's Read-Only Memory when a **Y** is typed on the keyboard. This jump causes the Batch Output routine to execute. When you are ready to actually send the data, type

CTRL

addr1 . addr2 **Y** **RETURN**

where "addr1" is the hexadecimal starting address of the data, and "addr2" is the hexadecimal ending address of the data. For example, if we wanted to send the information that is stored in memory from address \$2000 through address \$3FFF, we would type

CTRL

2000.3FFF **Y** **RETURN**

As soon as the return is typed, the data from address 2000 to address 3FFF will be sent through the Serial Interface from the APPLE to the external receiving device.

2. Batch Input

When in the Monitor, type

3F8: 4C 3D C9 **RETURN**

This prepares the APPLE to jump to hexadecimal location \$C93D in the Serial Interface's Read-Only Memory when a control Y is typed on the keyboard. This jump causes the Batch Input routine to execute. When you are ready to actually receive the data, type

CTRL

addr1 . addr2 Y RETURN

where "addr1" is the hexadecimal starting address in which the incoming data will be stored, and "addr2" is the hexadecimal ending address in which the incoming data will be stored. For example, if we wanted to receive data from an external device, and store it in our APPLE's memory from address \$4000 through address \$5FFF, we would type

CTRL

4000.5FFF Y RETURN

As soon as return is typed, the serial data can be sent by the external transmitting device to the Serial Interface. As it is received, the incoming data will be stored in your APPLE's memory from address 4000 through address 5FFF.

Note: when the Serial Interface is instructed to receive a batch move, the cursor on the receiving APPLE's TV screen disappears, and the Interface waits patiently until *all* the specified locations have been filled with received data. Then the cursor returns.

BATCH MOVES FROM BASIC

While it is easiest to use the Batch routines from the Monitor, it is also possible to do Batch Moves from BASIC. In the following discussion these definitions will hold:

BAL = Beginning Address Low
(the two rightmost digits of the 4-digit hexadecimal starting address for the move, converted to decimal)

BAH = Beginning Address High
(the two leftmost digits of the 4-digit hexadecimal starting address for the move, converted to decimal)

EAL = Ending Address Low
(the two rightmost digits of the 4-digit hexadecimal ending address for the move, converted to decimal)

EAH = Ending Address High
(the two leftmost digits of the 4-digit hexadecimal ending address for the move, converted to decimal)

Suppose you wish to send someone a picture from your APPLE's high-resolution screen (page 1). The memory for this screen lies between 8K and 16K, from hexadecimal address \$20000 to hexadecimal address \$40000. For the Beginning Address \$20000:

BAL = \$00 (hex) = 0 (decimal)
BAH = \$20 (hex) = 32 (decimal)

For the Ending Address \$40000:
EAL = \$00 (hex) = 0 (decimal)
EAH = \$40 (hex) = 64 (decimal)

1. Batch Output from BASIC

The following BASIC program does the same task that the Monitor Batch Output command did. See the discussion of the Monitor Batch Output for more details.

```
10 PR#1 : PRINT " "           (initializes Interface)
20 POKE 60, BAL : POKE 61, BAH (sets starting address)
30 POKE 62, EAL : POKE 63, EAH (sets ending address)
40 CALL -14015                (jumps to Output Routine at $C941)
50 PR#0                       (returns to normal TV output)
60 END
```

2. Batch Input from BASIC

In the Batch Output program, above, change line 40 to
40 CALL -14019 (jumps to Input Routine at \$C93D)

The resulting BASIC program does the same task that the Monitor Batch Input command did. See the Monitor Batch Input discussion for more details. Note that the IN#1 is *not* necessary for accepting input through the Serial Interface, because CALL -14019 deals directly with the Interface (not through the input and output "vectors" set by PR# and IN#). In fact the PR#1 in line 10 was necessary *only* to initialize the Interface;

10 CALL -16128
would have done as well.

VI APPENDIX: SERIAL INTERFACE TIMING

TABLE OF BAUD RATE QUANTUM NUMBERS

The following is a table that gives seventeen of the most commonly used **baud** rates, along with their quantum value (for POKEing) and percent error. Although only seventeen different **baud** rates are shown here, any integer from 0 through 255 may be POKE'd into the proper address (1144+s), and each will give a different **baud** rate.

Average APPLE II Frequency 1.0204842 MHz
 Period .979926 microseconds
 Jitter +139.7 nanoseconds every 65th cycle

Baud Rate Loop Quantum** 53 APPLE II Cycles (51.94 microseconds)

Baud Rate	Quantum No. (Hex)	(Dec)	Period (microsec.)	Actual Baud Rate	% Error
75	\$ 00	*** 0	13296	75.20	+ .28
90	\$ D6	214	11115	89.96	- .037
110*	\$ B0	176	9141	109.4	- .548
134.5*	\$ 90	144	7479	133.7	- .586
150	\$ 80	128	6648	150.4	+ .28
240	\$ 50	80	4155	240.67	+ .28
300*	\$ 40	64	3324	300.85	+ .28
450	\$ 2B	43	2234	447.74	- .51
600	\$ 20	32	1662	601.7	+ .28
900	\$ 15	21	1091	916.8	+1.86
1200*	\$ 10	16	831	1203.4	+ .28
1800	\$ 0B	11	571.3	1750.2	-2.78
2400*	\$ 8	8	415.5	2406.8	+ .28
3600	\$ 5	5	259.7	3850.59	+6.96
4800*	\$ 4	4	207.7	4813.6	+ .28
9600*	\$ 2	2	103.9	9627.2	+ .28
19200*	\$ 1	1	51.9	19254.4	+ .28

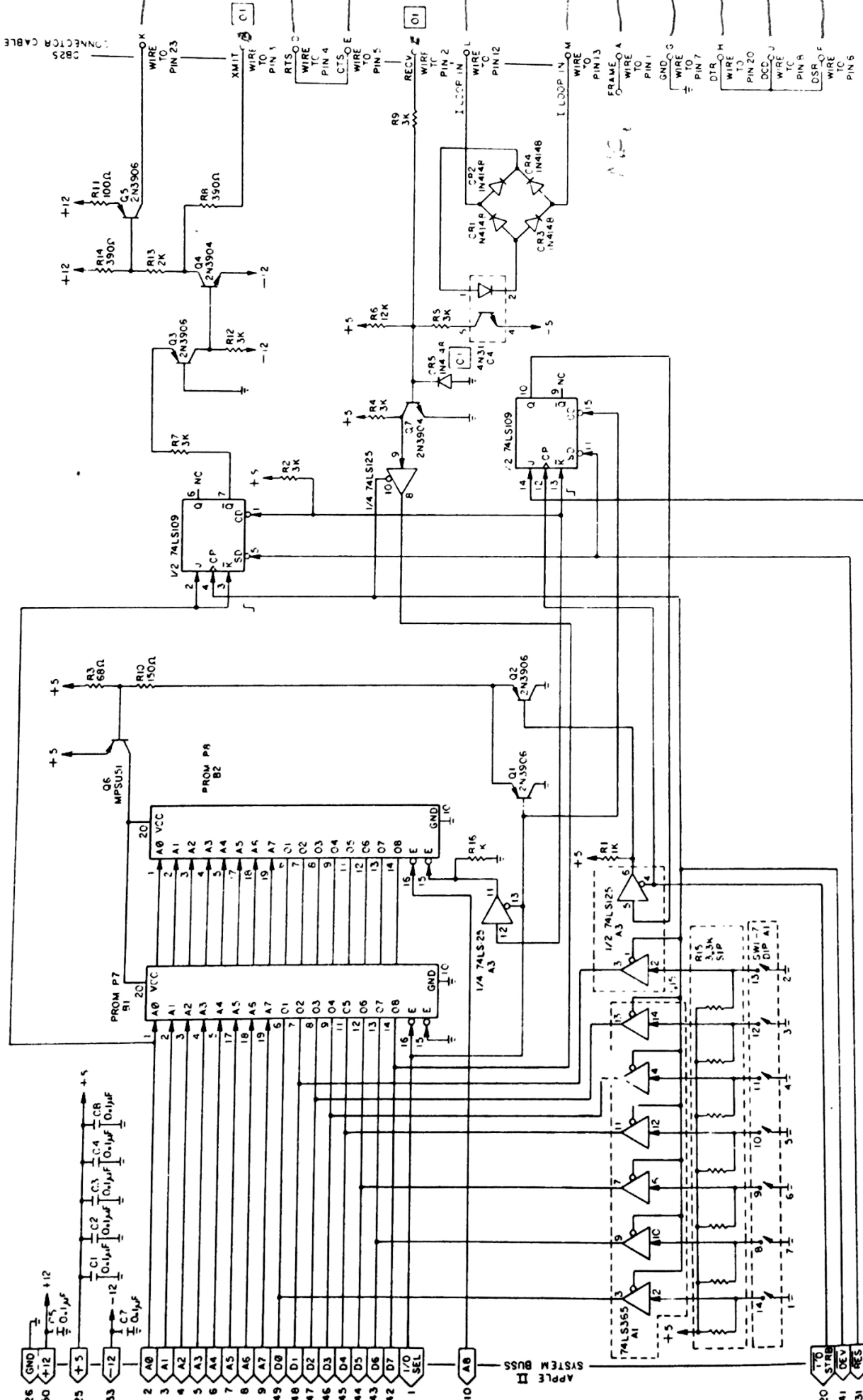
* **DIP** switch selectable

$$** \text{Quantum} = \frac{1}{(.00005194) * \text{Baud Rate}}$$

*** The quantum number zero is treated as 256 (\$100).

REV	ZONE	DESCRIPTION	DATE	INITIALS
01		040) PLS CHECK CTS (CRS WAS 1-14)		

23 4 3 2 1 7 2 8 6



NOTE: UNLESS OTHERWISE SPECIFIED

ITEM	QTY	PART NUMBER	REVISION
1	1	6-B-78	
2	1		
3	1		
4	1		
5	1		
6	1		
7	1		
8	1		
9	1		
10	1		
11	1		
12	1		
13	1		
14	1		
15	1		
16	1		
17	1		
18	1		
19	1		
20	1		
21	1		
22	1		
23	1		
24	1		
25	1		
26	1		
27	1		
28	1		
29	1		
30	1		
31	1		
32	1		
33	1		
34	1		
35	1		
36	1		
37	1		
38	1		
39	1		
40	1		
41	1		
42	1		
43	1		
44	1		
45	1		
46	1		
47	1		
48	1		
49	1		
50	1		
51	1		
52	1		
53	1		
54	1		
55	1		
56	1		
57	1		
58	1		
59	1		
60	1		
61	1		
62	1		
63	1		
64	1		
65	1		
66	1		
67	1		
68	1		
69	1		
70	1		
71	1		
72	1		
73	1		
74	1		
75	1		
76	1		
77	1		
78	1		
79	1		
80	1		
81	1		
82	1		
83	1		
84	1		
85	1		
86	1		
87	1		
88	1		
89	1		
90	1		
91	1		
92	1		
93	1		
94	1		
95	1		
96	1		
97	1		
98	1		
99	1		
100	1		

DESCRIPTION	REV	DATE
0029		
0030		
0031		
0032		
0033		
0034		
0035		
0036		
0037		
0038		
0039		
0040		
0041		
0042		
0043		
0044		
0045		
0046		
0047		
0048		
0049		
0050		
0051		
0052		
0053		
0054		
0055		
0056		
0057		
0058		
0059		
0060		
0061		
0062		
0063		
0064		
0065		
0066		
0067		
0068		
0069		
0070		
0071		
0072		
0073		
0074		
0075		
0076		
0077		
0078		
0079		
0080		
0081		
0082		
0083		
0084		
0085		
0086		
0087		
0088		
0089		
0090		
0091		
0092		
0093		
0094		
0095		
0096		
0097		
0098		
0099		
0100		

```

C100 1010 * * * * *
C100 1020 * * * * *
C100 1030 * SERIAL PRINTER / REMOTE INPUT
C100 1040 * INTELLIGENT INTERFACE FIRMWARE
C100 1050 *
C100 1060 * REVISION 3
C100 1070 * BY JAMES R. HUSTON JULY 21, 1978
C100 1080 * (F7-03, F8-01)
C100 1090 * * * * *
C100 1100 * * * * *
C100 1110 * * * * *
C100 1120 * ZERO PAGE EQUAS *
C100 1130 *
C100 1140 *
C100 1150 CH EQU $24 ;CURSOR HORIZONTAL POSITION
C100 1160 BASL EQU $28 ;BASE SCREEN ADDRESS POINTER
C100 1170 NCOUNT EQU $35 ;BIT COUNTER
C100 1180 TEMPX EQU $35 ;TEMPORARY BUFFER FOR $CN
C100 1190 CASE EQU $35 ;FOR LOWER-CASE MODES
C100 1200 CSWL EQU $36 ;LOW BYTE OF CHAR OUT VECTOR
C100 1210 CSWH EQU $37 ;HI BYTE OF CHAR OUT VECTOR
C100 1220 KSWL EQU $38 ;LOW BYTE OF CHAR IN VECTOR
C100 1230 KSWH EQU $39 ;HI BYTE OF CHAR IN VECTOR
C100 1240 AIL EQU $3C ;MONITOR ADDRESS BUFFER
C100 1250 CKSUM EQU $42 ;CHECK SUM BUFFER
C100 1260 COUNT EQU $43 ;BYTE COUNT BUFFER FOR BATCH
C100 1270 *
C100 1280 *
C100 1290 * GENERAL EQUATES *
C100 1300 *
C100 1310 FICK EQU $95 ;CONTROL-U
C100 1320 STACK EQU $100 ;SYSTEM STACK BLOCK
C100 1330 INBUFF EQU $200 ;SYSTEM INPUT BUFFER
C100 1340 KBD EQU $C000 ;KEYBOARD INPUT
C100 1350 KBDSTRE EQU $C010 ;KEYBOARD CLEAR
C100 1360 DEV EQU $C080 ;DEVICE ACCESS
C100 1370 ROMSW EQU $CFFF ;DISABLES CO-RESIDENT $C800 ROMS
C100 1380 *
C100 1390 *
C100 1400 * SLOT VARIABLES *
C100 1410 *
C100 1420 BRATE EQU $478-$CO ;THE BAUD QUANTUM NUMBER
C100 1430 STFBITS EQU $4F8-$CO ;THE NUMBER OF STOP BITS
C100 1440 STATUS EQU $578-$CO ;PARITY / CHECKSUM / MODES
C100 1450 COL EQU $5F8-$CO ;COLUMN POSITION COUNTER
C100 1460 OLDBYTE EQU $678 ;BUFFER FOR INPUT (LAST IN)
C100 1470 BYTE EQU $678-$CO ;BUFFER FOR INP/OUT DATA
C100 1480 NO EQU $6F8 ;BUFFER FOR DEVICE NUMBER ($NO)
C100 1490 PWDTH EQU $6F8-$CO ;PRINTER (FORMAT) WIDTH
C100 1500 PARITY EQU $778 ;BUFFER FOR PARITY CALC.
C100 1510 NBITS EQU $778-$CO ;THE NUMBER OF BATA BITS
C100 1520 MSL0T EQU $7F8 ;BUFFER FOR HI SLOT ADDR
C100 1530 FLAGS EQU $7F8-$CO ;FLAGS FOR ALL MODES.
C100 1540 *
C100 1550 *
C100 1560 * MONITOR SUBROUTINES *
C100 1570 *
C100 1580 WAIT EQU $FCAB ;DELAY SUBROUTINE
C100 1590 NXTA1 EQU $FCBA ;INCREMENTS A1 AND COMPARES TO A2
C100 1600 CROUT EQU $FD8E ;OUTPUTS A CARRAGE RETURN
C100 1610 COUT1 EQU $FDF0 ;CHARACTER OUT
C100 1620 PRERR EQU $FF2D ;OUTPUTS "ERR"
C100 1630 IORTS EQU $FF58 ;KNOWN "RTS" LOCATION
C100 1640 PAGE
C100 1650 *
C100 1660 *
C100 1670 DFLTENTR BIT IORTS ;SET THE V-FLAG (DEFAULT ENTRY)
C100 1680 BVS ENTRY ; BRANCH ALWAYS
C100 1690 SERIN SEC ;SERIAL INPUT ENTRY
C100 1700 BCC *
C100 1710 ORG *-1
C100 1720 SEROUT CLC ;SERIAL OUTPUT ENTRY
C100 1730 CLV
C100 1740 ENTRY PHP
C100 1750 SEI
C100 1760 STX TEMPX ;SAVE INPUT BUFFER INDEX
C100 1770 PWA ;SAVE REGISTERS
C100 1780 TXA
C100 1790 PWA
C100 1800 TYA
C100 1810 PWA
C100 1820 LDA ROMSW ;SWITCH OUT ALL CO-RESIDENT $C800 ROMS
C100 1830 JSR IORTS ;DETERMINE SLOT ADDRESS
C100 1840 TSX
C100 1850 LDA STACK,X ;RECOVER HIGH SLOT ADDRESS FROM STACK
C100 1860 STA MSL0T
C100 1870 ASL A ;CREATE $NO FROM SLOT ADDR ($CN)
C100 1880 ASL A
C100 1890 ASL A
C100 1900 ASL A
C100 1910 STA NO
C100 1920 TAY ;PUT $NO INTO REGISTER-Y ALSO
C100 1930 PLA
C100 1940 PLA
C100 1950 PLA ;RECOVER CHARACTER
C100 1960 PLP ;RECOVER STATUS FLAGS
C100 1970 TXS ;RESTORE STACK POINTER
C100 1980 LDX MSL0T ;PUT THE $CN NUMBER IN REG. X
C100 1990 PWA ;SAVE THE CHARACTER
C100 2000 BVC
C100 2010 JSR DODEF ;SET ALL SLOT DEPENDANT LOCATIONS
C100 2020 BCC SEROUT1 ;BRANCH IF NOT INPUT MODE
C100 2030 LDA SERINI
C100 2040 PWA ;SAVE FOR ESC TEST
C100 2050 PHP
C100 2060 BAI ;IGNORE OLD BYTE IF UPPER CASE
C100 2070 LDY TEMPX ;GET INPUT BUFFER INDEX VALUE.
C100 2080 BEQ GETIN ;DON'T MODIFY BUFFER IF 1ST CHAR.
C100 2090 DEY ; OR IF OLDBYTE DOESN'T MATCH
C100 2100 BNE INBUFF,Y ; WHAT'S IN THE INPUT BUFFER.
C100 2110 GETIN
C100 2120 ORA #$E0 ;MAKE IT LOWER CASE PROPER!
C100 2130 STA INBUFF,Y
C100 2140 GETIN JSR SHFTIN ;GO GET THE SERIAL INPUT.
C100 2150 PLP ;RESTORE CARRY AND INTERRUPT FLAGS
C100 2160 PLA ;(OLD BYTE)
C100 2170 FOR #$9B ;IS IT "ESC"
C100 2180 BNE DDCASE ;ALL INPUT FOLLOWING "ESC" MUST
C100 2190 CLC ; UPPER CASE.
C100 2200 DDCASE LDA #$DF ;BIT 5=0 IF UPPER CASE ALFA
C100 2210 BCC CAPSONLY ;BRANCH IF AFTER "ESC"
C100 2220 ORA FLAGS,X ;IF BIT 5=1 THEN LOWER CASE IS ENABLED
C100 2230 CAPSONLY STA CASE

```



```

C81B 68 3450 FLA
C81C 2A 3460 ROL A
C81D 2A 3470 ROL A ; DETERMINE WHICH COLUMN WIDTH
C81E 29 03 3480 AND #3
C820 88 3490 TAY ; TRANSFER 0-3 TO Y
C821 89 39 C9 3500 LDA WDTAB,Y ; GET WIDTH FROM TABLE
C822 90 38 06 3510 STA FWIDTH,X ; SAVE COLUMN WIDTH
C823 90 02 3520 LDA #2 ; SET THE NUMBER OF STP:BITS
C824 90 38 04 3530 STA STPBITS,X
C825 90 09 3540 LDA #9 ; AND THE DATA BITS
C826 90 88 06 3550 STA NBITS,X
C827 90 07 3560 LDA #7 ; LAST, THE PARITY STATUS
C828 90 88 04 3570 STA STATUS,X
C829 E4 37 3580 CFX CSWH ; SET CHARACTER OUT VECTOR
C830 D0 09 3590 BNE NOTOUT ; OR...
C831 A9 36 3600 CMF CSWL
C832 FD 05 3610 BEQ NOTOUT
C833 E5 36 3620 STA CSWL
C834 18 3630 NOTINP CLC ; (INDICATE OUTPUT)
C835 90 09 3640 RCC GOODDATA
C836 E4 39 3650 NOTOUT CFX ; SET KEY IN VECTOR.
C837 D0 09 3660 BNE NOTINP
C838 A9 05 3670 LDA #>SERIN
C839 85 38 3680 STA KSWL
C840 38 3690 PARERR SEC ; (INDICATE INPUT)
C841 60 3700 GOODDATA RTS
C842 3710 PAGE
C843 3720 ;
C844 3730 ;
C845 * * * * *
C846 * THIS IS THE SERIAL INPUT ROUTINE. ON ENTRY X=%CN.
C847 * REGISTERS A & Y MAY BE ANYTHING. BRATE (+%CN) MUST
C848 * CONTAIN THE BAUD RATE NUMBER, NBITS (+%CN) MUST CONTAIN
C849 * THE NUMBER OF BITS (INCLUDING THE START BIT), ONE STP
C850 * BIT IS ASSUMED. $NO MUST BE IN 'NO' (+%CN), BIT 1 OF
C851 * FLAGS INDICATES PARITY OPTION. (1=NO PARITY, 0=PARITY)
C852 * IF PARITY IS SELECTED, BIT 0 OF FLAGS INDICATES ODD OR
C853 * EVEN PARITY (0=EVEN PARITY, 1=ODD PARITY).
C854 * ON EXIT DATA IS IN BYTE (+%CN) (NOTE: FOR LESS THAN 8
C855 * BITS, THE REMAINING HIGH ORDER BITS ARE SET TO ONE. I.E.
C856 * IF NUMBER OF BITS IS SET TO 6, ONE START BIT AND 5 DATA;
C857 * THE RETURNED DATA WILL BE 11XXXXXX BINARY.).
C858 * X=%CN, AND Y=0. IF THE ROUTINE WAS INTERRUPTED BY THE
C859 * KEYBOARD OR THERE WAS A PARITY ERROR, THE CARRY FLAG IS
C860 * SET. ON ERROR OR KEYSTROKE: X=%CN, A & Y ARE UNDEFINED.
C861 * * * * *
C862 3910 SHFTIN SEI ; NO HARD INTERRUPTIONS, PLEASE
C863 3920 LDA STATUS,X ; PREPARE FOR PARITY/STOP BIT OPTIONS
C864 3930 AND #3 ; IF BIT ONE IS ZERO THEN PARITY, IF BIT ZERO IS
C865 3940 LSR A ; ZERO THEN EVEN PARITY
C866 3950 ROR PARITY ; INITIALIZE PARITY
C867 3960 PHA ; IF A=0 THEN PARITY IS EXPECTED
C868 3970 LDA NBITS,X
C869 3980 STA NCOUNT ; INITIALIZE BIT COUNT
C870 3990 LDY NO
C871 4000 SEC ; SET CARRY FOR START BIT
C872 4010 PCS GETBIT ; BRANCH ALWAYS
C873 4020 NXTIN LDY NO
C874 4030 DEC NCOUNT ; ARE ALL THE BITS IN YET?
C875 4040 BEQ CHKPAR ; YES, GET PARITY/STOP BIT...
C876 4050 CLC ; NO, MORE TO COME
C877 4060 GETBIT LDA DEV,Y
C878 4070 BCC SAVBIT ; BRANCH IF DATA
C879 4080 BPL GOTSTRT ; BRANCH IF START BIT
C880 4090 LDA KBD ; MAYBE THE KEYBOARD WANTS ATTENTION
C881 4100 BPL GETBIT ; NO, WAIT FOR THE START BIT
C882 4110 STA BYTE,X
C883 4120 BIT KBDSTRB ; CLEAR KEYBOARD
C884 4130 FLA ; YES, BETTER FIX THE STACK BEFORE RETURN
C885 4140 RTS ; THE KEYBOARD INTERRUPTED THE INPUT!
C886 4150 ;
C887 4160 GOTSTRT LDA BRATE,X
C888 4170 LDY #9 ; MUST CREATE A DELAY SUCH THAT DATA IS TAKEN
C889 4180 SWAIT ; AT THE CENTER OF THE SIGNAL.(BRANCH ALWAYS)
C890 4190 DEY
C891 4200 BNE SWAIT ; DONE WAITING AROUND?
C892 4210 SEC #1 ; YES, IT'S GET THAT DATA TIME
C893 4220 BEQ NXTIN ; NO, WAIT ANOTHER 78 CYCLE TIMES
C894 4230 LDY #E ; BRANCH ALWAYS
C895 4240 BNE SWAIT
C896 4250 ; SAVBIT TAY
C897 4260 ROL A ; SHIFT THE DATA INTO CARRY
C898 4270 ROR BYTE,X ; THEN PUT IT AWAY.
C899 4280 TYA
C900 4290 EOR PARITY ; UPDATE THE PARITY
C901 4300 STA PARITY
C902 4310 LDA BRATE,X ; GET THE BAUD DELAY NUMBER
C903 4320 SEC
C904 4330 WAIT1 SBC #1 ; TIME OUT THE BAUD RATE
C905 4340 BEQ NXTIN ; BRANCH IF DONE
C906 4350 LDY #9
C907 4360 WAIT53 DEY ; DELAY=53*(BRATE-1) CYCLES
C908 4370 BNE WAIT53
C909 4380 BEQ WAIT1
C910 4390 ;
C911 4400 CHKPAR FLA ; PARITY OR NO PARITY, THAT IS THE QUESTION
C912 4410 BNE MOD ; BRANCH IF NO PARITY
C913 4420 LDA DEV,Y ; GET THE PARITY BIT
C914 4430 EOR PARITY ; DOES THE ONE SENT MATCH THE CALCULATED?
C915 4440 BMI PARERR ; NOT IF THE MINUS FLAG IS ON
C916 4450 MOD LDA DEV,Y ; MAKE SURE THE SIGNAL IS HIGH BEFORE
C917 4460 BPL MOD ; RETURNING...
C918 4470 LDA #A
C919 4480 STA NBITS,X ; HOW MANY BITS TO MAKE A WHOLE BYTE?
C920 4490 TAY ; PUT THE LEFTOVER BIT COUNT IN REGISTER-Y
C921 4500 MOD1 DEY ; SHIFT IN A ONE?
C922 4510 BEQ GOODDATA ; NO, SEND THE DATA BACK TO CALLER
C923 4520 ROR BYTE,X ; YES, MAKE THE HIGH ORDER BIT A ONE.
C924 4530 SEC
C925 4540 BCS MOD1 ; BRANCH ALWAYS
C926 4550 ;
C927 4560 ;
C928 4570 OFI SERIAL3
C929 4580 ;
C930 4590 ;
C931 4600 * THE FOLLOWING ARE VARIOUS EXIT ROUTINES. *
C932 4610 ;
C933 4620 ;
C934 4630 VIDEO PLA
C935 4640 TAY
C936 4650 PLA
C937 4660 TAX
C938 4670 PLA

```



```

C981      5910 ;
C981      5920 ;
C981 20 4D C8 5930 MOVIN JSR SHFTIN ;GET THE INPUT
C984 30 21 5940 BMI ERR1
C986 8D 88 05 5950 LDA BYTE,X
C989 91 3C 5960 STA (A1L),Y ;PUT THE INPUT AWAY
C98B 45 42 5970 EOR CKSUM
C98D 85 42 5980 STA CKSUM ;UPDATE THE CHECKSUM
C98F 20 BA FC 5990 JSR NXTA1 ;INCREMENT RAM POINTER
C992 80 E2 6000 BCS FINISH
C994 E6 43 6010 INC COUNT ;UPDATE THE BYTE COUNT
C996 0D E9 6020 BNE MOVIN ;BRANCH IF BLOCK NOT FILLED
C998 28 6030 FLF
C999 80 80 6040 BCS BATCH ;BRANCH IF NO CHECKSUM
C99B 20 4D C8 6050 JSR SHFTIN ;GET THE CHECKSUM INPUT
C99E 8D 88 05 6060 LDA BYTE,X
C9A1 C5 42 6070 CMP CKSUM
C9A3 18 6080 CLC ;INDICATE CHECKSUM MODE
C9A4 F0 A5 6090 BEQ BATCH
C9A6 48 6100 PHA
C9A7 18 6110 ERR1 CLC ;INDICATE ERROR CONDITION
C9A8 90 CC 6120 BCC FINISH ;BRANCH ALWAYS TAKEN
C9AA 6130 ;
C9AA 6140 ;
C9AA 6150 ;
C9AA 6160 * * * * *
C9AA 6170 ; THIS IS THE SERIAL OUTPUT ROUTINE. ON ENTRY X=%CN
C9AA 6180 ; (N IS THE SLOT NUMBER), REGS A AND Y CAN BE ANYTHING.
C9AA 6190 ; THE DATA TO BE OUTPUT MUST BE IN BYTE (+%CN), IF DATA IS
C9AA 6200 ; LESS THAN 8 BITS (NBITS<9), THE LEFTOVER HIGH BITS MUST
C9AA 6210 ; BE SET TO ONES. (I.E. FOR 7 BITS: BYTE=1XXXXXX)
C9AA 6220 ; THE NUMBER OF BITS (INCLUDING THE START BIT) MUST BE IN
C9AA 6230 ; NBITS (+%CN), THE BAUD RATE (DELAY NUMBER) MUST BE IN
C9AA 6240 ; BRATE (+%CN), & NUMBER OF STOP BITS (+ PARITY BIT) IN
C9AA 6250 ; STPBITS (+%CN). %NO MUST BE IN 'NO' (+%CN).
C9AA 6260 ; ON EXIT A=0, X=%CN, AND Y=0. P-STATUS IS MODIFIED
C9AA 6270 * * * * *
C9AA 6280 ;
C9AA 78 6290 SHOUT SEI ;SET FOR NO HARD INTERRUPTIONS
C9AB 18 6300 CLC
C9AC 08 6310 FHP
C9AD 38 6320 SEC
C9AE 08 6330 FHP
C9AF BC 88 06 6340 LDY NBITS,X ;GET THE NUMBER OF BITS/WORD
C9B2 84 35 6350 STY NCOUNT ;STORE IT IN THE BIT COUNTER
C9B4 A9 01 6360 LDA #1
C9B6 3D 88 04 6370 AND STATUS,X ;FOR EVEN OR ODD PARITY
C9B9 0D F8 06 6380 ORA NO
C9BC 8D 78 07 6390 STA PARITY ;INITIALIZE PARITY
C9BF A9 00 6400 LDA #0
C9C1 1E 88 05 6410 ASL BYTE,X ;SHIFT FOR START BIT
C9C4 7E 88 05 6420 BITOUT ROR BYTE,X
C9C7 2A 6430 BITOUT ROL A ;BIT 0 OF BYTE -> BIT 0 OF ACCUMULATOR
C9CB 0D F8 06 6440 ORA NO
C9CC 29 01 6450 TAY ;COPY INTO REGISTER Y
C9CE 4D 78 07 6460 AND #1 ;STRIP ALL BUT THE DATA
C9D1 8D 78 07 6470 EOR PARITY ;UPDATE THE PARITY
C9D4 38 6480 STA PARITY ;AND SAVE IT AGAIN
C9D5 B9 81 C0 6490 FAROUT SEC ;THIS ENTRY POINT FOR STOPBITS AND PARITY
C9D8 BD 88 03 6500 LDA DEV+1,Y ;DATA IS SENT VIA BIT 0 OF THE ADDRESS BUS
C9DB E9 01 6510 BDLAY LDA BRATE,X ;GET THE BAUD NUMBER
C9DD F0 07 6520 BDLAY SBC #1
C9DF AD 09 6530 BDLAY1 LDY NXTOUT ;DELAY = 53 * (BAUD NUMBER-1) CYCLE TIMES
C9E1 88 6540 DEY
C9E2 D0 FD 6550 BNE BDLY1
C9E4 F0 F5 6560 BEQ BDLAY
C9E6 BC 38 04 6570 BDLAY1 LDY STPBITS,X ;BRANCH ALWAYS TAKEN
C9E9 C4 35 6580 LDY NCOUNT ;TIMING DURING DATA TRANSFER
C9EB D0 07 6590 DEC NCOUNT ;MORE BITS TO DO?
C9ED 28 6600 BNE BITOUT ;YES, DO ANOTHER ONE
C9EE 90 0E 6610 FLF ;NO, BUT WHAT ABOUT STOP BITS?
C9F0 6620 BCC RETSUB ;CARRY CLEAR, THEN DONE
C9F0 6630 ;
C9F0 6640 ; STOP BITS (AND PARITY IF SELECTED) YET TO DO
C9F0 6650 ;
C9F2 8D 88 04 6660 STY NCOUNT ;PUT THE NUMBER OF STOPBITS IN COUNTER
C9F5 29 02 6670 LDA STATUS,X ;NOW TO SEE IF A PARITY BIT IS TO BE SENT
C9F7 4A 6680 AND #2 ;IF ZERO THEN PARITY IS SENT, ELSE JUST
C9F8 0D 78 07 6690 LSR A
C9FB AD 08 6700 ORA PARITY ; STOP BITS ARE SENT.
C9FC D0 06 6710 TAY ;NOW Y=%NO+(STOP BIT OR PARITY BIT)
C9FE A8 6720 RETSUB TAY ;GO PUT IT OUT!
C9FF 60 6730 RETSUB TAY ;THEN SET THE ZERO FLAG
CA00 6740 RTS
6750 ;

```

LABELS

C9E1 BDLY1	0028 BASL	C948 BATCH	C9D8 BDLAY
0588 BYTE	C9C4 BITOUT	0388 BRATE	C931 BTAB
C8AA CHKPAR	C15E CAPSONLY	0035 CASE	0024 CH
FD0 SOUT1	0042 CKSUM	0038 COL	0043 COUNT
0036 CSWL	C1C8 CRLF	FD8E CROUT	CSWH
C800 DEF AULT	C1CA CTRL	C1C2 CTRLTST	0037 CURSOK
C157 DDCASE	C080 DEV	C10D DFLTENTR	C187 DLAY
C9A7 ERR1	C132 DODEF	C194 DONE	C1E4 ENTRY
0738 FLAGS	C8FE EXIT1	C923 EXIT2	C109 FINISH
C87F GOTSTRT	C868 GETBIT	C14D EXITN	C976 GOODATA
FF58 IORTS	0200 INBUFF	C8DC INF INSH	C84C INPUT
0039 KSWH	C8FC IVERSE	C000 KBD	C93D KBDSTR8
C8B5 MOD	0038 KSWL	C1E7 LFCBK	C010 MBE40
0078 MSL0T	C8C0 MXTOUT	C981 MOVIN	C816 MOVOUT
0033 NCOUNT	C8D8 NORMIO	06F8 NO	C954 NBITS
C843 NOTOUT	C135 NOTPICK	C919 NOTESC	0688 NXTINP
C1FB NXTCHR	C8E9 NXTIN	C167 NOUID	FC8A NXTA1
C19E OUT1	C941 OUTPUT	C9E6 PARERR	0678 OLD BYTE
C9D4 PAROUT	0095 PICK	C848 PRERR	0778 PARITY
C98D RETBATCH	C92F RETOUT	FF2D RETSUB	0638 PWDTH
C88F SAVBIT	C105 SERIN	C137 SERIN1	06FF R0MSW
Q100 STROUT1	C176 SETCH	C84D SHFTIN	C107 SEROUT
0438 STALC	0488 STATUS	C9AA STOP	C9AA SHOUT
C8C9 STPBIT'S	C884 SWAIT	0035 TEMPX	C916 STORFLGS
C939 VIDEO	FC68 WAIT	C89F WAIT1	C911 UPPER
			C8A5 WAIT53



10260 Bandley Drive
Cupertino, California 95014
(408) 996-1010