PHILIPS

P2000

PHILIPS

P2174
SERIAL INTERFACE (V24)
REFERENCE MANUAL

Data
Systems

PHILIPS

P2174

SERIAL INTERFACE (V24)
REFERENCE MANUAL

5103 991 37421

IMPORTANT

always ensure that the P2000 is switched off before inserting or removing the serial interface module.

inserting or removing the module while the power is on can damage it.

PREFACE

This manual explains how the Serial Interface module P2174 may be used with the P2000 to provide data communication. The interface on this module conforms to a widely-used standard, the V24 specifications (also known as RS232); the module is therefore suitable for a large number of applications, some of which are mentioned in this manual.

The Serial Interface module can be used provide data communication from the Microsoft BASIC system or from the UCSD system. In both cases, this may be done directly, using the intrinsic features of the system, or the appropriate ADACO (Asynchronous DAta COmmunication) Language Interface may be used. There are therefore four possible ways of using the Serial Interface:

- under Microsoft BASIC
- under UCSD
- under Microsoft BASIC with ADACO/B
- under UCSD with ADACO/U

There are two versions of the ADACO Language Interface, ADACO/U for the UCSD system, and ADACO/B for Microsoft BASIC. Both provide the same extensive range of facilities, which includes emulation (imitation) of different types of terminal, file transfer and mail transfer between two P2000s or one P2000 and a host computer. ADACO/U and ADACO/B are not the subject of this manual - refer instead to:

- P2367 ADACO/U Language Interface Reference manual
- P2317 ADACO/B Language Interface Reference manual.

This manual describes the use of the Serial Interface under Microsoft BASIC and the UCSD system without ADACO. Facilities for handling the Serial Interface are built-in to the UCSD system, and are described in detail in the UCSD documentation, so the information given here is brief, and reference is made to the UCSD manuals. The BASIC system, however, does not have such extensive built-in facilities, so more programming must be provided by the user. This manual therefore concentrates on the use of the Serial Interface from Microsoft BASIC without ADACO.

Page Layout

The pages in all P2000 manuals are arranged as follows:

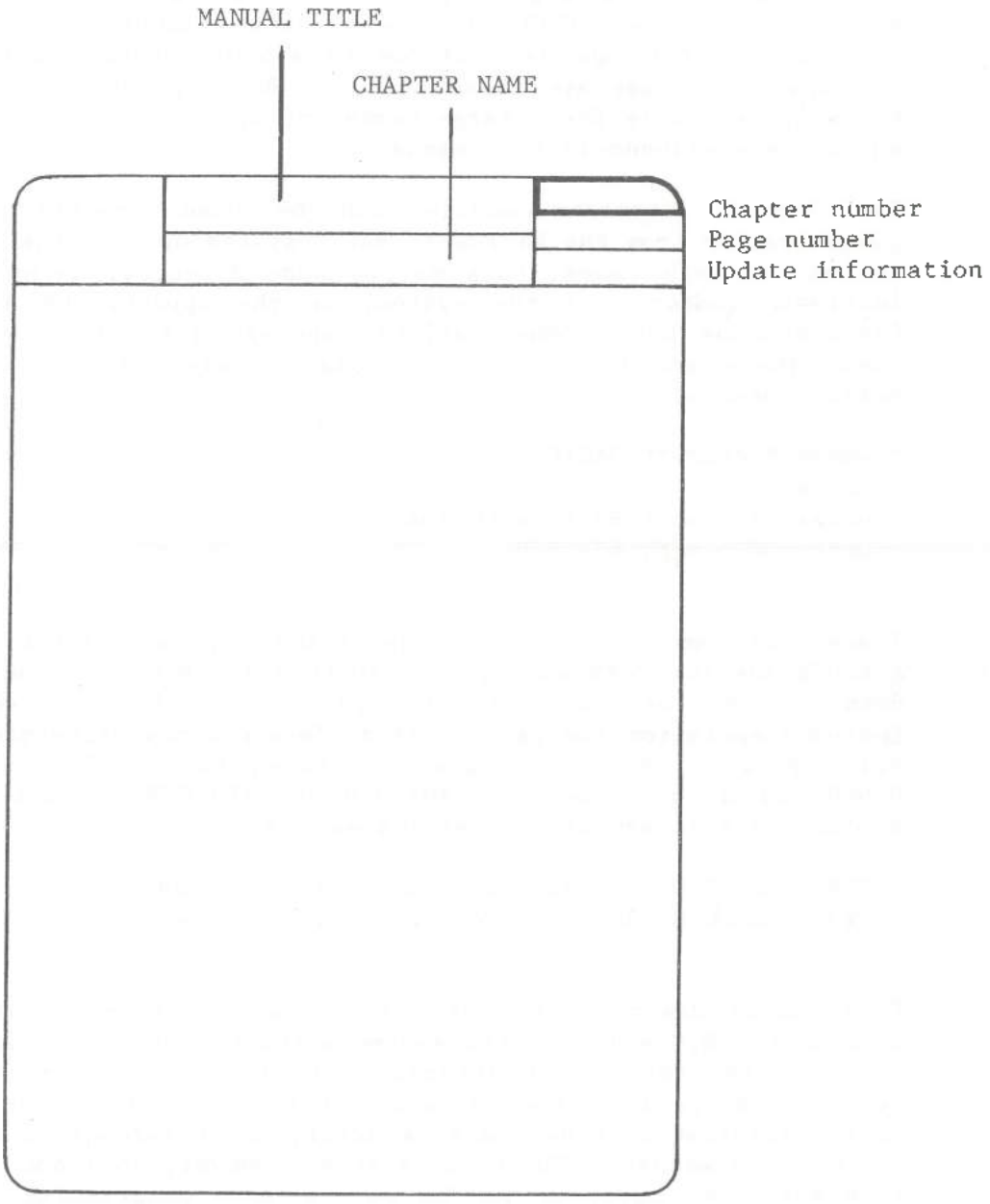


TABLE OF CONTENTS

1 THE SERIAL INTERFACE

This chapter defines some terms used in the rest of the manual, at the same time giving an overview of the Serial Interface module's function. Readers who are familiar with the V24 interface (also known as RS232) may wish to skip this and go on to Chapter 2.

1.1 GENERAL

The Serial Interface module provides communication between the P2000 and one other device, which may of course be another P2000. The two devices are connected by a cable; each device is fitted with an interface, which controls the data transmitted over the cable. The interface used in the Serial Interface module for the P2000 is known as V24 (also called RS232) - the other device must also incorporate, or be fitted with, a V24 interface. V24 is a popular and international standard so there is a wide range of compatible devices; The advantage of this standardised system is that any V24 device can be connected, regardless of its function or manufacturer.

The Serial Interface module can also be connected to a modem, which allows long-distance data communication via a telephone line. In fact, as far as the P2000 is concerned, the modem is just another device.

Communication between devices connected via the V24 interface may be in both directions - either one may send data (the 'transmitter'), and the other receive it (the 'receiver'). The communication is serial; that is, data is sent bit-by-bit on one line (in contrast with parallel communication, where several bits are sent simultaneously over several lines). And the communication is asynchronous; that is, the two devices are not always in step with each other, and must therefore get into step when data is to be communicated (in contrast with synchronous communication, when the devices keep in step with each other even when no data is being transmitted).

The P2000 uses a set of short programs to control the Serial Interface module - to instruct it to transmit data, or to receive data, and so on. These programs are known as 'drivers', or 'driver routines', and are usually written as Assembly language subroutines. If a BASIC program, for example, needs to use the Serial Interface, it will execute a call to the appropriate driver, and the driver will instruct the module to perform the required action.

1.2 DEFINITION OF TERMS

This section defines some standard terms used in data communication with the V24 interface. These terms occur throughout the rest of this manual, without further explanation.

1.2.1 Handshake

During data transmission, the transmitter and receiver must keep each other synchronised and informed of their status - for example, the receiver must tell the transmitter when it is ready for the next character of data. This is ensured through the use of a handshake procedure which is performed between receiver and transmitter.

1.2.2 Data Bits

The I/O module transmits data in the form of characters, each of which is represented by a number of data bits. The module allows the user to select how many bits represent each character. The number of bits used dictates the number of different characters which can be represented:

- with 5 data bits/character, 32 different characters - with 6 data bits/character, 64 different characters - with 7 data bits/character, 128 different characters - with 8 data bits/character, 256 different characters

The number of data bits per character should be chosen to match the requirements of the device with which you are communicating. If possible, seven data bits/character should be used, since this is compatible with the ASCII code.

1.2.3 Start Bit

Before sending the data bits for each character, the transmitter sends one extra bit to indicate to the receiver that a new character is beginning. This is known as the start bit.

1.2.4 Stop Bit(s)

At the end of each character of data transmitted, an extra signal is sent, to indicate the end of the character. The length of time for which this signal is sent is measured by the number of bits which could be sent in the same time. The Serial Interface module allows the user to select between:

- 1 stop bit per character
- 1.5 stop bits per character
- 2 stop bits per character

The value selected should, of course, be the same as that selected on the device with which you are communicating.

1.2.5 Transmission Rate

Data can be transmitted at different rates, which are measured in baud (bits per second). The higher the transmission rate, the more data characters are sent per second, but remember that each data character consists of between 7 and 12 bits, depending on the number of data, stop and parity bits.

The Serial Interface module provides a choice of transmission rates between 75 and 9600 baud. The rate selected should be the same as that of the attached device.

1.2.6 Parity, Framing and Overrun

A parity check is used by communicating devices to help ensure that data has not become corrupted (altered) during transmission. The transmitter performs a calculation on the data bits before sending the character, and sends one extra bit - the parity bit - after the data bits; this will be a one or a zero, depending on the result of the parity calculation. When the other device receives the character, it performs the same calculation; if the result does not agree with the parity bit, a parity error is reported.

Two forms of parity checking can be used, known as 'even' and 'odd'; and in some non-critical applications, parity checking may be left out if wished.

Two more error checks are performed by the receiver - framing and overrun. A framing error occurs when the character received does not conform to the specified 'frame' of start bit, data bits, parity bit and stop bits; that is, the receiver has got out of step with the transmitter. An overrun error occurs when the character received consists of more bits than were expected.

1.2.7 End-of-transmission

When all the data characters that are to be sent have been sent, some indication must be given that transmission is complete. There are two ways in which this can be done.

The most common method is for the transmitter to send a special 'end-of-transmission' character which is recognised by the receiver as indicating the end of data transmission. Different devices recognise different end-of-transmission characters; refer to the device manufacturer's specifications. The most usual are:

- CR (13 in the ASCII code)
- LF (10 in the ASCII code)
- ETX (3 in the ASCII code)
- CR followed by LF

Of course, the chosen end-of-transmission character cannot be used within the data.

The second method of indicating end of transmission is always to send the same number of characters. This fixed number of characters is known as a block; common sizes are 256, 512 or 1024 characters per block. The receiver will always expect one block at a time; if the transmitter has fewer characters than a full block to send, it must send extra characters to make up.

I.2.8 Time-out

When two devices are communicating via their Serial Interfaces, it may happen that one device is waiting to receive data, and the other has no data to send, or for some reason cannot send any - for example, if the connection between the devices is broken. In order to prevent the device waiting indefinitely, a time-out may be used. This is a specified time lapse for which one device will wait for the other. If no communication has taken place within the time-out period, the device will stop waiting.

Use of a time-out will prevent the P2000 from waiting indefinitely, so it can continue with some other task.

2 USING THE SERIAL INTERFACE WITH THE P2000

This chapter explains how to set up the Serial Interface module and the P2000 for data communication, when using Microsoft (Tape or Disk) BASIC without ADACO/B, or UCSD p-System without ADACO/U.

The module may also be used from Microsoft BASIC with the ADACO/B package, and on the UCSD system with the ADACO/U package. Refer to the ADACO documentation (listed in the Preface) for information on these packages.

2.1 EQUIPMENT

The following items of equipment are be required:

- P2174 Serial Interface module

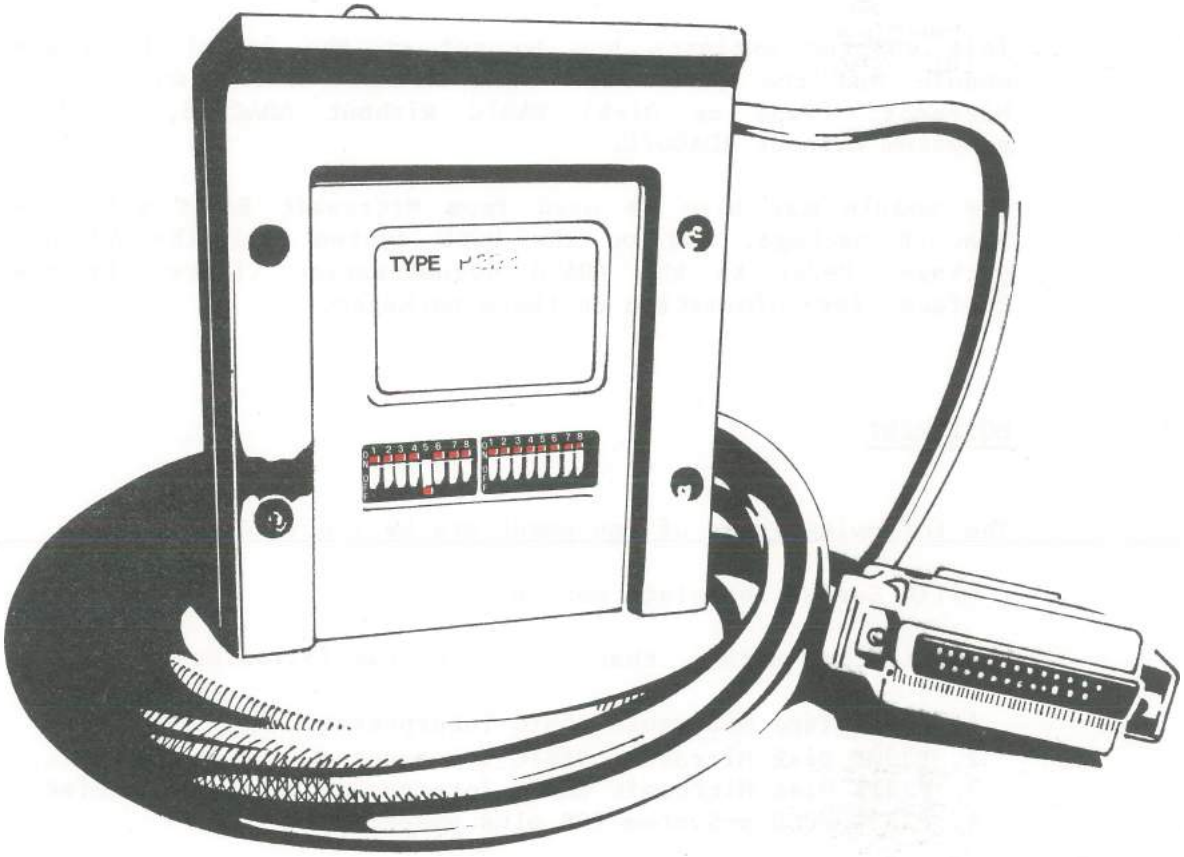
- application module, that is, one of the following

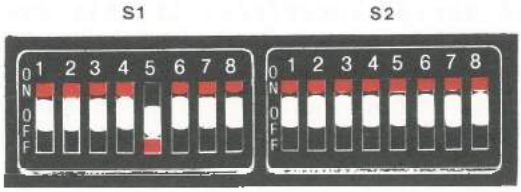
  1. P2305 Tape Microsoft BASIC Interpreter.
  2. P2306 Disk Microsoft BASIC Interpreter plus system disk.
  3. P2311 Disk Microsoft BASIC Interpreter plus system disk.
  4. P2351 UCSD p-System TSS plus system disk.

- the device with which you wish to communicate, and any items it may require

Ensure that the device with which you wish to communicate is fitted with a V24 Serial Interface. If this device is another P2000, it will require its own Serial Interface module, and its own Microsoft BASIC or UCSD module and disk. It is also possible to use one P2000 under Microsoft Disk BASIC and the other under Microsoft Tape BASIC, or one under Microsoft BASIC and one under UCSD.

Figure 1 - Serial Interface module (rear view)





2.2 SELECTION OF TRANSMISSION RATE

On the back of the Serial Interface module are two sets of eight 'dip' switches, as shown in Figure 1. These switches can be set on or off by pressing them with the tip of a ball-point pen. The lefthand set of switches, named S1-1 to S1-8, are used to select the transmission rate, as in the table below.

Switch	Transmission rate (baud)
S1-1	75
S1-2	150
S1-3	300
S1-4	600
S1-5	1200
S1-6	2400
S1-7	4800
S1-8	9600

Set the switch for the required transmission rate so that its red mark is at 'OFF', and all the others so that their red marks are at 'ON'. Do not use the module with more than one of S1-1 to S1-8 at 'OFF', it could be damaged. As an example, figure 1 shows the switch settings for a transmission rate of 1200 baud.

The transmission rates of both devices must both be the same, so set the other device according to the manufacturer's instructions. If the other device is also a P2000, set the S1 switches on its Serial Interface module as described above.

The S2 set of switches (the righthand set) is described in section 4.1. These switches should be left with their red marks at 'OFF',as shown in Figure 1.

2.3 CABLING

The Serial Interface module is fitted with approximately 2.5 metres of cable, terminated with a standard V24 25-pole male plug. The device with which you wish to communicate will probably be fitted with the corresponding 25-pole socket or female plug, in which case you can simply plug in to make the connection.

If, however, the other device also has a male plug (for example, if you are using two P2000's), you will need to make an interconnection between the two, using a length of 25-pole cable (CCITT-norm) with a V24 25-pole female plug at each end.This interconnection should be as shown in Appendix B.Cable and plugs of this type are readily available, for example from your P2000 dealer.

If the total length of cable separating the P2000 and the other device is in excess of 100 metres, errors in transmission may become more frequent, particularly when using poor quality cable. Factors which affect the clarity of long-distance transmission are the sensitivity of the V24 interface on the other device, the transmission rate, and the cable quality.

2.4 INSTALLATION

1. Follow sections 2.2 and 2.3.

2. Ensure the P2000-Keyboard is switched off.

3. Insert the application module in the front slot (marked '1'), in the usual way.

4. If using Microsoft Disk BASIC, or the UCSD p-System, insert the system disk in drive 1, but do not close the door yet.

5. Insert the Serial Interface module in the rear slot

```
**************************************************************
*                                                            *
*                       I M P O R T A N T                    *
*                                                            *
*          Always ensure that the P2000 is switched off      *
*       before inserting or removing the serial interface    *
*                          module.                           *
*                                                            *
*          Inserting or removing the module while the        *
*                    power is on can damage it.              *
*                                                            *
**************************************************************
```

6. Ensure that the other device is prepared according to the manufacturer's instructions, and connect it to the cable of the Serial Interface module.

   If the other device is a modem, prepare the modem, but do not attempt to establish the telephone link at this point. See section 2.6.

7. Switch on the P2000-Keyboard. (If using Microsoft Disk BASIC or the UCSD p-System, close the disk drive door when the red lamp goes on above the drive.) The system will load in the usual way.

You can now use the P2000 in exactly the same way you normally do; in addition, data communication can be performed via the Serial Interface, when the required parameters have been selected.

2.5 PARAMETER SELECTION

This section applies only to users with Microsoft BASIC, not to UCSD users.

The following parameters must be specified before the Serial Interface module is ready for data communication:

- number of data bits per character
- number of stop bits per character
- parity

(these terms are defined in Chapter 1). The parameters are specified by sending a 'control word' to the module before starting data communication. This control word is an integer value between 66 and 255, which is calculated in this way:

Start with 2, and to this add:

Data bits -	if 5 data bits/character,	add 0
	if 6 data bits/character,	add 4
	if 7 data bits/character,	add 8
	if 8 data bits/character,	add 12.
Stop bits -	if 1 stop bit/character,	add 64
	if 1.5 stop bits/character,	add 128
	if 2 stop bits/character,	add 192.
Parity -	if no parity is to be used,	add 0
	if odd parity is to be used,	add 16
	if even parity is to be used,	add 48.

The result of this addition should be a control word between 66 and 254. The way in which this value will be sent to the Serial Interface module is explained in the next chapters.

For example, if you wish to use 7 data bits/character, with 2 stop bits, and even parity:

start with	2
7 data bits/character	+8
2 stop bits/character	+192
even parity	+48
	250

This gives a control word of 250.

2.6 COMMUNICATION VIA A MODEM

This section applies if you are using the Serial interface to connect the P2000 to a modem. There are two ways in which modems may be used with the Serial Interface.

In the first case, two P2000s, each connected to a modem, with the modems connected to each other via a telephone line, can communicate with each other. In this case, there are no restrictions on the use of the Serial Interface module. The telephone link may be established at any time after the P2000s have been switched on.

In the second case, one P2000 with a modem is connected via a telephone line to another computer, typically a large time-sharing system, to use the facilities of the other computer. In this case, do not establish the telephone link until the program driving the Serial Interface module is running, because in most time-sharing systems, the computer will break the connection if no signal is received over the line from the P2000 within a short time-out period.

Most time-sharing systems require specific communication protocols to be followed by the P2000; the program which you are using to control the Serial Interface module must be written to perform these. Refer to the time-sharing system's documentation for details.

As an alternative, you may consider using one of the ADACO packages; these can be used to drive the Serial Interface module and provides emulation (imitation) of any required protocol. ADACO/U is used under the UCSD p-System; ADACO/B on the Microsoft BASIC system. Refer to the ADACO documentation listed in the Preface.

3 USING THE SERIAL INTERFACE FROM MICROSOFT BASIC

This chapter explains how the Serial Interface module can be used from a Microsoft BASIC (Tape or Disk) program without ADACO/B.

The driver routines described in this chapter have been written to be as concise as possible, and provide the simplest input/output functions. These will be adequate for users who want only to use the Serial Interface and are not concerned with the way it works; so this chapter does not attempt to give any explanation of why the drivers are written the way they are, or how they work.

Users who are interested to know how the module and drivers function, or who want to enhance the routines described here, should refer to Chapter 4, which gives a more detailed description of the routines and suggests possible additions.

The following routines use the "data terminal ready" and "clear to send" for a correct handshaking.

3.1 LOADING THE DRIVER ROUTINES

The driver routines are loaded into memory using the short BASIC program listed below. These program lines should be included at the start of any program which uses the Serial Interface. It is therefore a good idea to store the driver program as a file once you have typed it in.

The first line of the loading program is different in Tape BASIC and Disk BASIC, because the CLEAR statement has different syntax in these BASIC versions. In Tape BASIC, it is:

```
 10 CLEAR 255,&HCFFF
```

and in Disk BASIC it is:

```
 10 CLEAR ,&HCFFF,500
```

(The effect of these statements is the same.) The rest of the program is the same for both Tape and Disk BASIC:

```
 20 M=&HD000
 30 DATA 24,18,205,40,208,219,65,31,31,31
 40 DATA 48,249,26,211,64,19,16,243,24,15
 50 DATA 205,40,208,219,65,31,31,48,250,219
 60 DATA 64,18,19,16,244,62,80,211,65,201
 70 DATA 62,78,211,65,62,39,211,65,94,35
 80 DATA 86,213,225,70,35,94,35,86,201
 90 FOR I=M TO M + 58
100 READ A: C= C +  A
110 POKE I,A
120 NEXT I
130 B= (M + 40) AND &HFF: A= (M-B)/256 AND &HFF
140 N= M + 3: POKE N,B: POKE N + 1,A
150 N= M + 21: POKE N,B: POKE N + 1,A
160 POKE M + 41,78
170 IF C<>6050 THEN PRINT "ERROR IN DATA"
```

The DATA statement lists the Z-80 machine code instructions that make the up the drives (in decimal); a detailed listing and explanation of these instructions is given in section 3.5.

NOTE that the value 78 in line 160 is the control word as calculated in section 2.5. For example, if you wish to use 7 data bits/character, with 2 stop bits and even parity (control word= 250), this line should be:

```
160 POKE M + 41,250
```

3.2 SAVING THE DRIVER ROUTINES

Save the program in a file on tape (use CSAVE command) or disk (use SAVE command). Then whenever you are writing a BASICprogram which uses the Serial Interface, load the driver program into memory (use CLOAD or LOAD command), and add the rest of your program after it.

Once the program lines listed above have been executed, the driver routines are in memory, and can be called from any partof the program. There are two routines:

- receive a string
- transmit a string

The routines are described in the following sections.

3.3 RECEIVING

The receive routine takes characters which the module receivesfrom the other device, and assigns them to a string variable. The following statement should be given to specify the starting address of the routine (hexadecimal D000):

```
DEF USR1 = &HD000
```

After this, the routine may be called as many times as wished throughout the rest of the program. To call the routine, give the statements:

```
L$ = SPACE$(N)
X$ = USR1(L$)
```

Where L$ is the string variable in which the received characters are to be stored, and N is the number of characters to be received. X$ is a dummy variable, its value is not important.

Each time the receiver routine is called, use the SPACES function to set L$ to the correct length of spaces. This is important because the receiver routine will wait until N characters have been received. If fewer than N characters are received, the routine, and your program , will 'hang' (wait forever) - the only way to end the hang is to press the reset button, and that will also clear the program and the other device will transmit the corret number of characters; it can send extra space characters if necessary, to make up the full amount.

As an example, to receive 12 characters and store them in string L$, give these statements:

```
L$ = SPACE$(12)
X$ = USR1(L$)
```

3.4 TRANSMISSION

The transmit routine gives the characters in the string variable to the module, which sends them to the other device. The following statement should be given to specify the starting address of the routine (hexadecimal D002):

```
DEF USR2= &HD002
```

After this, the routine may be called as many times as wished throughout the rest of the program. To call the routine, give the statements:

```
L$= "string" + "."
X$= USR2(L$)
```

Where L$ is the string variable in which the characters to be transmitted are stored, and "string" are the characters to be transmitted; always assign the characters to the variable immediately before calling the transmitter routine and do ensure that the other device is ready to accept the number of characters being sent. X$ is a dummy variable, its value is not important. The character "." is a necessary dummy character. If the receiver expects strings of 12, the transmitter has to offer a string of 13 characters to the routine.

As an example, to transmit the characters "Time 01:45", give these statements:

```
L$= "time 01:45" + "."
X$= USR2(L$)
```

3.5 DESCRIPTION OF LOADER AND DRIVER

This section explains how the BASIC driver loading program described in section 3.1 works, and lists the elementary drivers in assembly code. It is not necessary to read this section before using the Serial Interface module; the information is provided for those people who wish to understand the drivers, or to adapt them.

3.5.1 Loading Program

Refer to the listing of the program in section 3.1.

Line 10 of the loading program set the highest address used by BASIC at hex CFFF; the first address usable for assembler subroutines is thus hex D000. Line 20 specifies that the driver routines are to be loaded starting at this address. You may change these if you wish.

In Tape BASIC, line 10 also specifies a maximum string space of 255 bytes (that is, 255 characters). In Disk BASIC, line 10 specifies a maximum of 500 bytes for the stack. You can change these if you want.

Lines 30-90 list the opcodes of the driver routines in decimal form. The opcodes and equivalent mnemonics are described in section 3.5.2. Lines 90 to 120 POKE the machine code instructions one-by-one into consecutive memory locations.

Line 130 calculates the starting address of the parameter-handling part of the drivers, which is shared by the receiver and transmitter; this address depends on the starting address specified in line 20. Lines 140 and 150 POKE this address to complete the opcode for the call instructions in receiver and transmitter.

Line 160 defines the control word used to initialize the Serial Interface (see section 2.5).

Line 170 checks the datastatement with the helps of a checksum, calculated in line 100.

If you write your own driver in machine code, you can use this program to load it into memory. Remember to modify the FOR loop to the correct number of instructions, and use lines 90 to 110, suitably modified, to deal with absolute addressing such as in a call instruction.

3.5.2 Driver Code

This section lists the complete code for the elementary drivers as it is given in the DATA statement of the BASIC loading program in section 3.1.

Perhaps reading this code will show you how straightforward assembly language is, and inspire you to have a go at writing some assembly language routines yourself!

```
hex  hex
addr opcode   label  mnemonic      comment
D000 1812            JR   RCEX     ;jump to read string
D002         ;
D002         TREX
D002 CD28D0          CALL INIT     ;connect and fetch
                                   ;parameter
D005         LOOP1
D005 DB41            IN   A,(41H)
D007 1F              RRA
D008 1F              RRA
D009 1F              RRA
D00A 30F9            JR   NC,LOOP1 ;wait until transmit
                                   ;enabled
D00C 1A              LD   A,(DE)   ;fetch character from
                                   ;string
D00D D340            OUT  (40H),A  ;send character
D00F 13              INC  DE       ;increase pointer in
                                   ;string
D010 10F3            DJNZ LOOP1    ;if not end of string
D012 180F            JR   EXIT     ;jump to disconnect
D014         ;
D014         RCEX
D014 CD28D0          CALL INIT     ;connect and fetch
                                   ;parameter
D017         LOOP2
D017 DB41            IN   A,(41H)
D019 1F              RRA
D01A 1F              RRA
D01B 30FA            JR   NC,LOOP2 ;wait until character
                                   ;arrives
D01D DB40            IN   A,(40H)  ;read character from
                                   ;interface
D01F 12              LD   (DE),A   ;place character into
                                   ;string
D020 13              INC  DE       ;increase pointer in
                                   ;string
D021 10F4            DJNZ LOOP2    ;if not end of string
D023         ;
```

Note: the code is continued on the next page.

This is a continuation of the code from the previous page.

```
hex  hex
addr opcode    label mnemonic       comment
D023           EXIT
D023           ; DISCONNECT SERIAL INTERFACE
D023 3E50            LD   A,80
D025 D341            OUT  (41H),A
D027 C9              RET            ;return to BASIC
D023           ;
D028           INIT
D028           ;
D028           ; CONNECT SERIAL INTERFACE
D028 3E4E            LD   A,78
D02A D341            OUT  (41H),A   ;fill mode word
D02C 3E27            LD   A,27H
D02E D341            OUT  (41H),A   ;fill command word
D030           ;
D030           ; GET STRING PARAMETER
D030 5E              LD   E,(HL)    ;fetch addr.of
D031 23              INC  HL        ;string descriptor
D032 56              LD   D,(HL)
D033 D5              PUSH DE        ;in
D034 E1              POP  HL        ;HL
D035           ;
D035 46              LD   B,(HL)    ;length of string in B
D036 23              INC  HL
D037 5E              LD   E,(HL)    ;address of string
D038 23              INC  HL        ;in
D039 56              LD   D,(HL)    ;DE
D03A C9              RET
```

4 USING THE SERIAL INTERFACE DIRECTLY

The first three sections of this chapter describe how the Serial Interface module functions; using this information, the rest of the chapter shows you how to write your own software drivers for the module. Drivers may be written in Z-80 assembler, Microsoft BASIC or another language; but in this chapter, the examples given are written in Assembler, and it is assumed that they will be called from a Microsoft BASIC program.

The drivers described in this chapter are the very simplest possible (known as 'minimal code'); they fill the three basic module-handling requirements - initialization, receiving one character, and transmitting one character. Simple drivers have the advantage of being very short subroutines, but when you are more familiar with the use of the Serial Interface module, you may wish to make some additions to the drivers; suggestions for these are also given in section 4.7.

4.1 COMMUNICATION WITH THE MODULE

The P2000 communicates with the Serial Interface module via four ports. The P2000 may send an eight-bit value to a port, or read an eight bit value in from the port. The ports are assigned these functions:

Port number (hexadecimal)	Function
40	Data port
41	Control port
61	Input address port
62	Switch S2 port

Data port - this is the port where the P2000 gives a character to be sent by the module to the other device; and from where the P2000 can read in a character received by the module from the other device.

Control port - this is the port where the P2000 gives commands to the module; and from where the P2000 can read the module's status.

Input address port - this port is reserved for use in connection with optional modifications which may be made to the Serial Interface module in the future, as indicated on the circuit diagram in Appendix C. It will not normally be necessary to use this port.

Switch S2 port - the P2000 can read the switch settings of switch S2 from this port as follows:

Switch	Corresponding bit of input address port
S2-1	7
S2-2	6
S2-3	5
S2-4	4
S2-5	3
S2-6	2
S2-7	1
S2-8	0

A switch with its red mark at 'OFF' registers as a bit '1'.

Switch S2 is the righthand 'dip' switch seen from the back of the module. (See figure on page 2.2)It is not used by the elementary drivers described in Chapter 3, but if you write your own drivers, they can read the switch if you want.

4.2 MODULE INSTRUCTIONS

The P2000 can give two types of instructions to the Serial Interface module - mode instructions and command instructions. Each instruction takes the form of one byte, sent to the Control port, hexadecimal 41.

4.2.1 Mode Instruction

The mode instruction is the same as the 'control value' described in Section 2.5. The bits of this instruction have the following indications:

Bit	7 6	5 4	3 2	1 0
	STOP	PARITY	DATA	n/a

STOP - Bits 7,6 — Number of stop bits/character

0 0	invalid
0 1	1 stop bit/character
1 0	1.5 stop bits/per character
1 1	2 stop bits/character

PARITY - Bits 5,4 — Parity

0 0	no parity
0 1	no parity
1 0	even parity
1 1	odd parity

DATA - Bits 3,2 — Number of data bits/character

0 0	5 data bits/character
0 1	6 data bits/character
1 0	7 data bits/character
1 1	8 data bits/character

n/a - Bits 1,0 — Control the internal speed of the module

1 0	Always use this bit setting

4.2.2 Command Instruction

This instruction sets the internal state of the module. The bits of this instruction have the following indications:

Bit	7	6	5	4	3	2	1	0
	n/a	IR	RTS	ER	n/a	RxEN	DTR	TxEN

n/a - bits 7 and 3 must both be 0.

IR - Internal reset
when 1, the module resets, and expects a new mode instruction.
when 0, the module continues normal operation.

RTS - Request to send
when 1, makes the RTS line high.
when 0, makes the RTS line low.

ER - Error reset
when 1, resets the FE, OE and PE error flags in the status byte.
when 0, the error flags remain as they are.

RxEN - Receiver enable
when 1, enables the module's receiver function.
when 0, disables the receiver function.

DTR - Data terminal ready
when 1, makes the DTR line high.
when 0, makes the DTR line low.

TxEN - Transmit enable
when 1, enables the module's transmit function.
when 0, disables the transmit function.

Before the module may be used, a mode instruction must be given. After that, command instructions are used to control the module's functioning; these command instructions must have bit IR=0. When a new mode is required (for example, if a different number of data bits/character is to be used), a command instruction with bit IR=1 is given, followed by the new mode instruction.

4.3 STATUS

The P2000 can test the status of the Serial Interface module at any time, by reading the status byte from the Control port (hexadecimal 41). The bits of the status byte have the following significance:

Bit	7	6	5	4	3	2	1	0
	DSR	n/a	FE	OE	PE	TxE	RxRDY	TxRDY

DSR - Data Set Ready
when 1, the DSR line is high (lamp on module on)
when 0, the DSR line is low (lamp on module off).

n/a - bit 6 will always be 0.

FE - Framing error
when 1, no framing error has been detected
when 0, a framing error has been detected in the last character received.

OE - Overrun error
when 1, an overrun error has been detected in the last character received.
when 0, no overrun error has been detected.

PE - Parity error
when 1, a parity error has been detected in the character received.
when 0, no parity error has been detected.

TxE - Transmitter empty
when 1, a character has just been transmitted, and the module's transmitter buffer is empty.
when 0, the transmitter buffer contains a character which has not yet been transmitted.

RxRDY - Receiver ready
when 1, the module's receiver buffer contains a character which has just been received. This character can be read in by the P2000 from the data port.
when 0, the receiver buffer is empty.

TxRDY - Transmitter ready
when 1, the module's transmitter buffer is ready to accept another character. The P2000 can send the next character to the data port.
when 0, the transmitter buffer is not ready for another character.

4.4 INITIALIZATION

Before the Serial Interface module can be used for data communication, it must be initialized with a mode instruction, and a command instruction, which are sent to the control port (hexadecimal 41). The initialization routine should be executed only once, before the module is used for data communication.

The minimal code for the initialization is given below (hexadecimal opcodes on the left):

```
3E4E        LD   A,4EH        ;mode instruction
D341        OUT  (41H),A      ;send to control port
3E27        LD   A,27H        ;command instruction
D341        OUT  (41H),A      ;send to control port
C9          RET               ;return
```

The mode instruction here is an example; it specifies 8 data bits/character, 1 stop bit, and no parity checking or generation. (You may like to check this by reference to section 4.2.1.) Any other valid mode instruction could be used.

The command instruction is not an example - this value must be used. It specifies: RTS line high, receiver function enabled, DTR line high, and transmitter function enabled.

Setting the DTR line high indicates to the other device that the module is ready for data communication. The other device should respond by setting the DSR line high, at which point the lamp on the module will light up. Communication can then take place.

4.5 RECEIVING

Once initialized, the module receives one character at a time. When it has received a character, the RxRDY bit of the status byte is set to one. The receiver driver tests the status byte (from the control port, hexadecimal 41) until there is a character, then reads the character from the data port (hexa- decimal 40).

The minimal code for the receiver is given below (hexadecimal opcodes on the left):

```
DB41    test: IN  A,(41H)    ;read status byte
1F            RRA            ;rotate right twice so
1F            RRA            ;bit 1 is in Carry bit
30FA          JR  NC,test    ;repeat if Carry bit=0
DB40          IN  A,(40H)    ;read the character
C9            RET            ;return
```

This routine leaves the character received in register A. To pass it back to a calling program, store it in the parameter whose address is given by the calling program, and then return. (Section 4.7.6 refers to parameter passing.)

4.6 TRANSMITTING

Once initialized, the module can transmit one character at time. When it is ready to transmit a character, the TxE bit of the status byte is set to one. The transmitter driver tests the status byte (from the control port, hexadecimal 41) until the module is ready, then sends the character to the data port (hexadecimal 40).

The minimal code for the transmitter is given below (hexa- decimal opcodes on the left):

```
DB41    test: IN  A,(41H)    ;read status byte
1F            RRA
1F            RRA
1F            RRA
30FB          JR  NC,test    ;repeat of Carry=0
7E            LD  A,(HL)     ;load character into A
D340          OUT (40H),A    ;send the character
C9            RET            ;return
```

This routine assumes that the address of the character to be transmitted is in register HL; the character is loaded into A to be output. The calling program will pass the character's address as a parameter. (Section 4.7.6 refers to parameter passing.)

4.7 MODIFICATIONS TO DRIVERS

A number of modifications can be made to the minimal driver routines; suggestions for some are given here. Look also at section 3.5.2, which lists the elementary drivers used from a BASIC program; you may prefer to modify these.

Programming these modifications in assembly language should not be difficult even for the beginner.

4.7.1 Time-out

The time-out principle is described in section 1.2.8. To prevent receiver or transmitter routines waiting indefinitely for the correct status, a time-out can be implemented by using a simple counting loop. The routines should return a status value to indicate that no data communication was performed before the time-out expired.

4.7.2 Block Transmit

The tranmsitter routine can be modified to transmit several characters instead of just one. The calling program would pass a string of characters to the transmitter routine, and these would be sent one by one; the routine would only return when all the characters had been transmitted (or the time-out had expired). The routine can find the length of the string as described in the parameter-handling part of the standard driver in section 3.5.2 (this applies only to Microsoft BASIC).

If using strings of different lengths, you may want to make the routine send an end-of-transmission character after sending the data characters, or to send extra blank characters, as described in section 1.2.7.

The elementary transmitter driver transmits one block of characters at a time; it is listed fully in section 3.5.2.

4.7.3 Block Receive

The receiver routine can be modified in a similar way to receive several characters. The calling program would pass a string to hold the characters, which would be received one by one; the routine would only return when all the characters had been received. The receiver must detect the end of transmission in some way - if a fixed number of characters is always sent, they can just be counted; alternatively, an end-of-transmission character may be used. Refer to section 1.2.7.

The elementary receiver driver receives one block of characters at a time; it is listed fully in section 3.5.2.

4.7.4 Transmission Errors

As explained in section 4.4, the status byte also provides an indication of parity, framing and overrun errors. The receiver routine can be modified to return a status value as a second parameter, so that the calling program is informed if there are any errors. The status value could be the status byte itself, or a special code, for example 1= ok, 0= not ok.

An alternative is for the receiver routine, if there is an error, to return a special character in place of the character received (since that character is incorrect anyway); this technique uses no extra parameter.

4.7.5 Polling

Sometimes, a program which is performing another task may need to know only if a character has been received by the Serial Interface module, but not what that character is. This may be done by reading the status byte at intervals, and testing bit 1, RxRDY, instead of calling the receiver routine. Repeatedly testing whether a character has arrived is known as 'polling'.

4.7.6 Parameter Passing

When writing your own drivers in assembler, to be called from a program in another language, you must consider the way in which parameters are passed to assembler subroutines. For Microsoft BASIC, this information is given in the Microsoft Tape BASIC and Microsoft Disk BASIC Reference manuals; Tape and Disk BASIC pass parameters in different ways.

In the UCSD system, parameters to assembler subroutines are usually passed on the stack. Refer to the UCSD p-System TSS Reference Manual for details.

You are recommended to refer to the appropriate manual before you start writing your own drivers or modifying the elementary drivers given in this manual, because the parameter-passing conventions must be taken into account.

5 USING THE SERIAL INTERFACE FROM THE UCSD SYSTEM

The UCSD system provides a wider range of device handling facilities than does Microsoft BASIC, so the use of the Serial Interface from UCSD is somewhat simpler. This chapter gives brief description of those facilities and their use, and refers the reader to the UCSD documentation for more information.

5.1 Volume Organisation

The UCSD system includes provision for a number of standard devices (also known as 'volumes' in UCSD terminology), including serial line input/output. This means the Serial Interface can be handled like any other volume, using the standard I/O functions; these are described in section 5.2. The Serial Interface is considered as two distinct devices:

- REMIN: , for serial line input (device number 7)
- REMOUT: , for serial line output (device number 8)

The device may be referred to by its name (the colon, ':', at the end is a part of the name), or by its number, when using the standard I/O functions. REMIN: is seen as a non-block-structured device which produces a stream of text, and REMOUT: as one which consumes a stream of text.

For further information, refer to:

- P2351 UCSD p-System Total Systems Set Reference Manual, Section 3.4 - Volume Organisation
- P2351 UCSD p-System Total Systems Set Internal Architecture Guide, Chapter 3 - Low-level I/O.

5.2 Device I/O Routines

To perform I/O on any device, the user of the system calls the appropriate I/O function of the language in use. These I/O calls are mapped by the Compiler and Operating System onto calls to the RSP/IO, the I/O part of the Runtime Support Package (that is, the part of the p-code Interpreter which contains native code devoted to input/output). The RSP/IO in turn calls the BIOS, the Basic Input/Output Subsystem, which controls the device operations themselves. This means the user need not be concerned with handshaking, parity and so on - BIOS performs all these driver functions.

This I/O hierarchy, and the use of the I/O facilities, are described fully in the P2351 UCSD p-System Total Systems Set Internal Architecture Guide, Chapter 3 - Low-level I/O. The reader is advised also to read that chapter.

The I/O routines in the RSP/IO are known as the Device I/O Routines. As explained, the Device I/O Routines are called when the user makes a language-level I/O call; that is, using the standard syntax of the language in question - for example, READLN in Pascal. Additionally, they may be called directly from the language; to the language user, they look like any other intrinsic routine. In both cases, when a unit or device must be specified as a parameter to the routine, REMIN: or REMOUT: should be specified as appropriate (the equivalent device number, 7 or 8, may be used instead).

The Device I/O Routines which are relevant to the Serial Interface are described very shortly below.

IORESULT

Returns an integer value which indicates the status of the last I/O request; that is, whether it terminated normally or not. Refer to the Internal Architecture Guide, Section 3.2.2 - IORESULT and Completion Codes, and to the TSS Reference Manual, section 7.14 - IORESULT function.

UNITBUSY

Returns a TRUE or FALSE (1 or 0) value indicating whether the the device is waiting for an I/O transfer to complete. Refer to the Internal Architecture Guide and to the TSS Reference Manual, section 7.34 - UNITBUSY function.

UNITCLEAR

Restores the device to its 'initial' state, and cancels all pending I/O calls to the device. Refer to the Internal Architecture Guide and to the TSS Reference Manual, section 7.35 - UNITCLEAR function.

UNITREAD

Performs input from the device. Refer to the Internal Architecture Guide and to the TSS Reference Manual, section 7.36 - UNITREAD procedure.

UNITWAIT

Waits for the device to complete the I/O operation in progress. Refer to the Internal Architecture Guide and to the TSS Reference Manual, section 7.38 - UNITWAIT procedure.

UNITWRITE

Performs input from the device. Refer to the Internal Architecture Guide and to the TSS Reference Manual, section 7.39 - UNITWRITE procedure.

APPENDIX A ASCII CODE

The American Standard Code for Information Interchange.

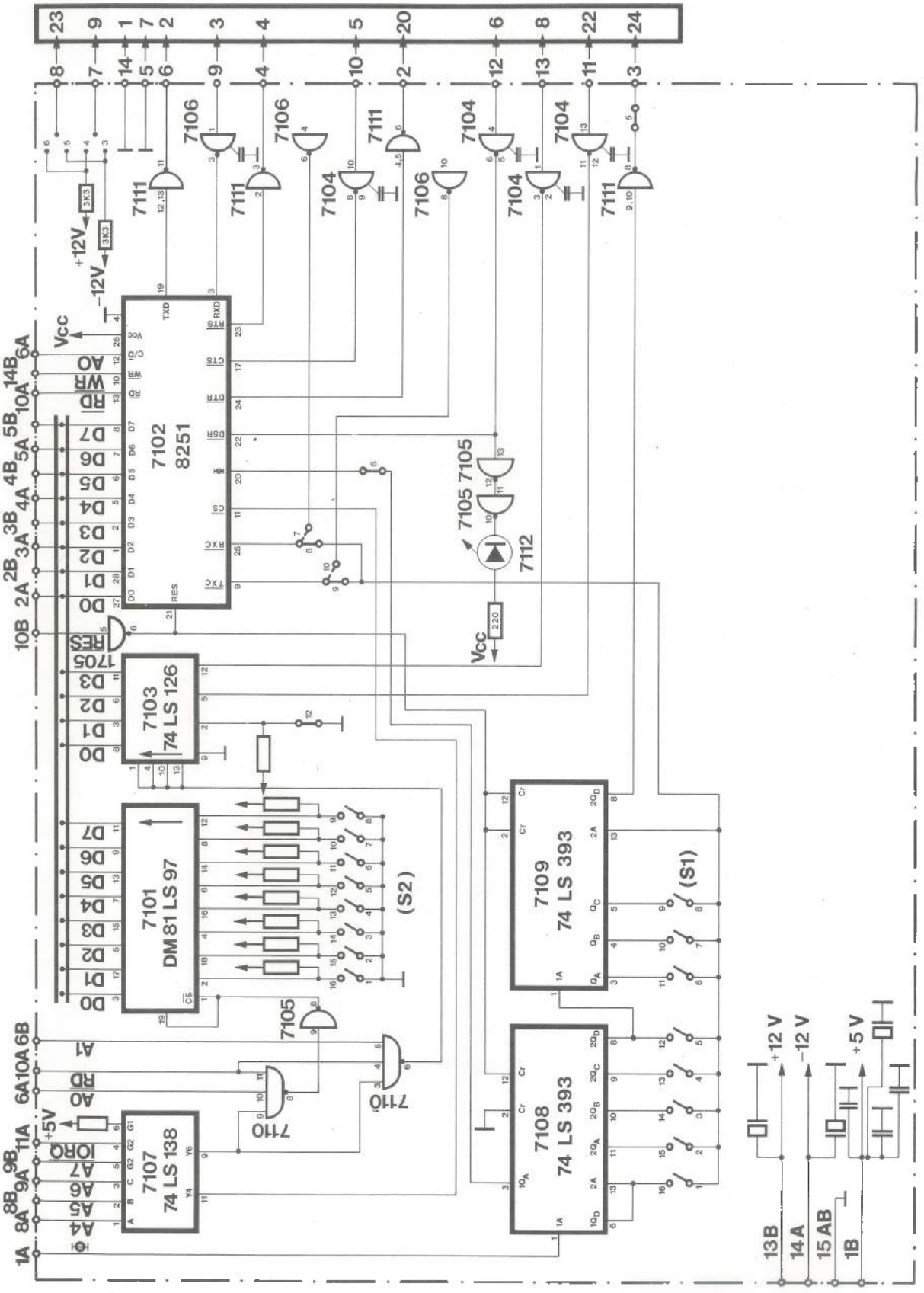
value	character	value	character	value	character	value	character
0	nul	32	sp	64	@	96	'
1	soh	33	!	65	A	97	a
2	stx	34	"	66	B	98	b
3	etx	35	£	67	C	99	c
4	eot	36	$	68	D	100	d
5	enq	37	%	69	E	101	e
6	ack	38	&	70	F	102	f
7	bel	39	'	71	G	103	g
8	bs	40	(	72	H	104	h
9	ht	41	)	73	I	105	i
10	lf	42	*	74	J	106	j
11	vt	43	+	75	K	107	k
12	ff	44	,	76	L	108	l
13	cr	45	-	77	M	109	m
14	so	46	.	78	N	110	n
15	si	47	/	79	O	111	o
16	dle	48	0	80	P	112	p
17	dc1	49	1	81	Q	113	q
18	dc2	50	2	82	R	114	r
19	dc3	51	3	83	S	115	s
20	dc4	52	4	84	T	116	t
21	nak	53	5	85	U	117	u
22	syn	54	6	86	V	118	v
23	etb	55	7	87	W	119	w
24	can	56	8	88	X	120	x
25	em	57	9	89	Y	121	y
26	sub	58	:	90	Z	122	z
27	esc	59	;	91	[	123	{
28	fs	60	<	92	\	124	|
29	gs	61	=	93	]	125	}
30	rs	62	>	94	^	126	~
31	us	63	?	95	_	127	del

APPENDIX B CONNECTION OF SERIAL INTERFACE MODULE TO DEVICE WITH MALE PLUG

The connections shown should be made when connecting the P2000 Serial Interface module to another P2000 and using the driver routines as described in chapter 3 and 4.

Device A			Device B	
Signal	Pole		Pole	Signal
Protective ground	1	----------	1	Protective ground
Transmit data	2	----------	3	Receive data
Receive data	3	----------	2	Transmit data
Request to send	4	----------	5	Clear to send
Clear to send	5	----------	4	Request to send
Data set ready	6	----------	20	Data terminal ready
Signal ground	7	----------	7	Signal ground
Data terminal ready	20	----------	6	Data set ready

APPENDIX C SERIAL INTERFACE MODULE CIRCUIT DIAGRAM



EIA PINNING

PIN 1: Protective ground
2: Transmit data (103)
3: Receive data (104)
4: Request to send (105)
5: Clear to send (106)
6: Data set ready (107)
7: Signal ground (102)
8: Carrier detect (109)
9: Select standby (116)
20: Data terminal ready (108/1, 108/2)
22: Calling indicator (125)
23: Data signalling rate selector (111)
24: Terminal equipment transmitter signal element timing (113)

SOFTWARE INTERFACE

UART-control-address: 41 H
UART-data-address: 40 H
DIP-switch-address: 62 H 8 Inputs
4-Bit input-address: 61 H Bit 1 short-circuit soldering spot
2 call indicator
3 carrier detect

BAUD-RATE ADJUSTMENT: SWITCH S1

Only 1 switch closed: 75 Bps
2 " " : 150 "
3 " " : 300 "
4 " " : 600 "
5 " " : 1200 "
6 " " : 2400 "
7 " " : 4800 "
8 " " : 9600 "

INPUT-PORT: SWITCH S2

Direct input of DIP-switches, a closed switch means 0 on
date bus: switch 1 corresponds with bit 7
2 " " bit 6
.
.
8 " " bit 0

INDEX

The references given in this index are to the section, not the page, where information on the indexed item can be found.

SERIAL INTERFACE REFERENCE MANUAL

12NC: 5103 991 37421

This publication comprises the following updates:

no updates

SERIAL INTERFACE REFERENCE MANUAL

12NC: 5103 991 37421

Including update(s).....

Originator:

Name .....

Address .....
.....
.....
.....

Comment (if possible, add a copy of the page(s) affected by the comment, marked with the proposed changes).

Please return this form to: Österr. Philips Industrie Ges.m.b.H.,
MAG-PERCO / MARKETING SUPPORT
A-1100 Wien, Triesterstr. 64,
A U S T R I A