CHAPTER 6

INPUT/OUTPUT GUIDE

- Introduction
- Output to the TV
- Output to Other Devices
- The Game Ports
- RS-232 Interface Description
- The User Port
- The Serial Bus
- The Expansion Port
- Z-80 Microprocessor Cartridge

INTRODUCTION

Computers have three basic abilities: they can calculate, make decisions, and communicate. Calculation is probably the easiest to program. Most of the rules of mathematics are familiar to us. Decision making is not too difficult, since the rules of logic are relatively few, even if you don't know them too well yet.

Communication is the most complex, because it involves the least exacting set of rules. This is not an oversight in the design of computers. The rules allow enough flexibility to communicate virtually anything, and in many possible ways. The only real rule is this: whatever sends information must present the information so that it can be understood by the receiver.

OUTPUT TO THE TV

The simplest form of output in BASIC is the PRINT statement. PRINT uses the TV screen as the output device, and your eyes are the input device because they use the information on the screen.

When PRINTing on the screen, your main objective is to format the information on the screen so it's easy to read. You should try to think like a graphic artist, using colors, placement of letters, capital and lower case letters, as well as graphics to best communicate the information. Remember, no matter how smart your program, you want to be able to understand what the results mean to you.

The PRINT statement uses certain character codes as "commands" to the cursor. The CRSR key doesn't actually display anything, it just makes the cursor change position. Other commands change colors, clear the screen, and insert or delete spaces. The RETURN key has a character code number (CHR$) of 13. A complete table of these codes is contained in Appendix C.

There are two functions in the BASIC language that work with the PRINT statement. TAB positions the cursor on the given position from the left edge of the screen, SPC moves the cursor right a given number of spaces from the current position.

Punctuation marks in the PRINT statement serve to separate and format information. The semicolon (;) separates 2 items without any spaces in between. If it is the last thing on a line, the cursor remains after the last thing PRINTed instead of going down to the next line. It suppresses

(replaces) the RETURN character that is normally PRINTed at the end of the line.

The comma (,) separates items into columns. The Commodore 64 has 4 columns of 10 characters each on the screen. When the computer PRINTs a comma, it moves the cursor right to the start of the next column. If it is past the last column of the line, it moves the cursor down to the next line. Like the semicolon, if it is the last item on a line the RETURN is suppressed.

The quote marks (" ") separate literal text from variables. The first quote mark on the line starts the literal area, and the next quote mark ends it. By the way, you don't have to have a final quote mark at the end of the line.

The RETURN code (CHR$ code of 13) makes the cursor go to the next *logical* line on the screen. This is not always the very next line. When you type past the end of a line, that line is linked to the next line. The computer knows that both lines are really one long line. The links are held in the *line link table* (see the memory map for how this is set up).

A logical line can be 1 or 2 screen lines long, depending on what was typed or PRINTed. The logical line the cursor is on determines where the RETURN key sends it. The logical line at the top of the screen determines if the screen scrolls 1 or 2 lines at a time.

There are other ways to use the TV as an output device. The chapter on graphics describes the commands to create objects that move across the screen. The VIC chip section tells how the screen and border colors and sizes are changed. And the sound chapter tells how the TV speaker creates music and special effects.

OUTPUT TO OTHER DEVICES

It is often necessary to send output to devices other than the screen, like a cassette deck, printer, disk drive, or modem. The OPEN statement in BASIC creates a "channel" to talk to one of these devices. Once the channel is OPEN, the PRINT# statement will send characters to that device.

EXAMPLE of OPEN and PRINT# Statements:

```
100 OPEN 4, 4: PRINT# 4, "WRITING ON PRINTER"
110 OPEN 3, 8, 3, "0:DISK-FILE,S,W": PRINT# 3, "SEND TO DISK"
120 OPEN 1, 1, 1, "TAPE-FILE": PRINT# 1, "WRITE ON TAPE"
130 OPEN 2, 2, 0, CHR$(10): PRINT# 2, "SEND TO MODEM"
```

The OPEN statement is somewhat different for each device. The parameters in the OPEN statement are shown in the table below for each device.

TABLE of OPEN Statement Parameters:

FORMAT: OPEN file#, device#, number, string

DEVICE	DEVICE#	NUMBER	STRING
CASSETTE	1	0 = Input 1 = Output 2 = Output with EOT	File Name
MODEM	2	0	Control Registers
SCREEN	3	0,1	
PRINTER	4 or 5	0 = Upper/Graphics 7 = Upper/Lower Case	Text Is PRINTed
DISK	8 to 11	2–14 = Data Channel	Drive #, File Name, File Type, Read/Write
		15 = Command Channel	Command

OUTPUT TO PRINTER

The printer is an output device similar to the screen. Your main concern when sending output to the printer is to create a format that is easy on the eyes. Your tools here include reversed, double-width, capital and lower case letters, as well as dot-programmable graphics.

The SPC function works for the printer in the same way it works for the screen. However, the TAB function does not work correctly on the printer, because it calculates the current position on the line based on the cursor's position on the screen, not on the paper.

The OPEN statement for the printer creates the channel for communication. It also specifies which character set will be used, either upper case with graphics or upper and lower case.

EXAMPLES of OPEN Statement for Printer:

```
OPEN 1, 4: REM UPPER CASE/GRAPHICS
OPEN 1, 4, 7: REM UPPER AND LOWER CASE
```

When working with one character set, individual lines can be PRINTed in the opposite character set. When in upper case with graphics, the cursor down character (CHR$(17)) switches the characters to the upper and lower case set. When in upper and lower case, the cursor up character (CHR$(145)) allows upper case and graphics characters to be PRINTed.

Other special functions in the printer are controlled through character codes. All these codes are simply PRINTed just like any other character.

TABLE of Printer Control Character Codes:

CHR$ CODE	PURPOSE
10	Line feed
13	RETURN (automatic line feed on CBM printers)
14	Begin double-width character mode
15	End double-width character mode
18	Begin reverse character mode
146	End reverse character mode
17	Switch to upper/lower case character set
145	Switch to upper case/graphics character set
16	Tab to position in next 2 characters
27	Move to specified dot position
8	Begin dot-programmable graphic mode
26	Repeat graphics data

See your Commodore printer's manual for details on using the command codes.

OUTPUT TO MODEM

The modem is a simple device that can translate character codes into audio pulses and vice versa, so that computers can communicate over telephone lines. The OPEN statement for the modem sets up the parameters to match the speed and format of the other computer you are communicating with. Two characters can be sent in the string at the end of the OPEN statement.

The bit positions of the first character code determine the baud rate, number of data bits, and number of stop bits. The second code is optional, and its bits specify the parity and duplex of the transmission. See the RS-232 section or your **VICMODEM** manual for specific details on this device.

EXAMPLE of OPEN Statement for Modem:

```
OPEN 1, 2, 0, CHR$(6): REM 300 BAUD
100 OPEN 2, 2, 0, CHR$(163) CHR$(112): REM 110 BAUD, ETC.
```

Most computers use the American Standard Code for Information Interchange, known as ASCII (pronounced ASK-KEY). This standard set of character codes is somewhat different from the codes used in the Commodore 64. When communicating with other computers, the Commodore character codes must be translated into their ASCII counterparts. A table of standard ASCII codes is included in this book in Appendix C.

Output to the modem is a fairly uncomplicated task, aside from the need for character translation. However, you must know the receiving device fairly well, especially when writing programs where your computer "talks" to another computer without human intervention. An example of this would be a terminal program that automatically types in your account number and secret password. To do this successfully, you must carefully count the number of characters and RETURN characters. Otherwise, the computer receiving the characters won't know what to do with them.

WORKING WITH CASSETTE TAPE

Cassette tapes have an almost unlimited capacity for data. The longer the tape, the more information it can store. However, tapes are limited in time. The more data on the tape, the longer the time it takes to find the information.

The programmer must try to minimize the time factor when working with tape storage. One common practice is to read the entire cassette data file into RAM, then process it, and then re-write all the data on the tape. This allows you to sort, edit, and examine your data. However, this limits the size of your files to the amount of available RAM.

If your data file is larger than the available RAM, it is probably time to switch to using the floppy disk. The disk can read data at any position on the disk, without needing to read through all the other data. You can write data over old data without disturbing the rest of the file. That's why the disk is used for all business applications like ledgers and mailing lists.

The PRINT# statement formats data just like the PRINT statement does. All punctuation works the same. But remember, you're not working with the screen now. The formatting must be done with the INPUT# statement constantly in mind.

Consider the statement PRINT# 1, A$, B$, C$. When used with the screen, the commas between the variables provide enough blank space between items to format them into columns ten characters wide. On cassette, anywhere from 1 to 10 spaces will be added, depending on the length of the strings. This wastes space on your tape.

Even worse is what happens when the INPUT# statement tries to read these strings. The statement INPUT# 1, A$, B$, C$ will discover no data for B$ and C$. A$ will contain all three variables, plus the spaces between them. What happens? Here's a look at the tape file:

```
A$="DOG" B$="CAT" C$="TREE"
PRINT# 1, A$, B$, C$

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
D O G              C  A  T                        T  R  E  E  RETURN
```

The INPUT# statement works like the regular INPUT statement. When typing data into the INPUT statement, the data items are separated, either by hitting the RETURN key or using commas to separate them. The PRINT# statement puts a RETURN at the end of a line just like the PRINT statement. A$ fills up with all three values because there's no separator on the tape between them, only after all three.

A proper separator would be a comma (,) or a RETURN on the tape. The RETURN code is automatically put at the end of a PRINT or PRINT # statement. One way to put the RETURN code between each item is to use only one item per PRINT# statement. A better way is to set a variable to the RETURN CHR$ code, which is CHR$(13), or use a comma. The statement for this is R$= "," : PRINT# 1, A$ R$ B$ R$ C$. Don't use commas or any other punctuation between the variable names, since the Commodore 64 can tell them apart and they'll only use up space in your program.

A proper tape file looks like this:

```
1 2 3 4 5 6 7 8 9 10 11 12 13
D O G , C A T , T R E  E  RETURN
```

The GET# statement will pick data from the tape one character at a time. It will receive each character, including the RETURN code and other punctuation. The CHR$(0) code is received as an empty string, not as a one character string with a code of 0. If you try to use the ASC function on an empty string, you get the error message **ILLEGAL QUANTITY ERROR**.

The line GET# 1, A$: A= ASC(A$) is commonly used in programs to examine tape data. To avoid error messages, the line should be modified to GET#1, A$: A= ASC(A$+ CHR$(0)). The CHR$(0) at the end acts as insurance against empty strings, but doesn't affect the ASC function when there are other characters in A$.

DATA STORAGE ON FLOPPY DISKETTES

Diskettes allow 3 different forms of data storage. Sequential files are similar to those on tape, but several can can be used at the same time. Relative files let you organize the data into records, and then read and replace individual records within the file. Random files let you work with data anywhere on the disk. They are organized into 256 byte sections called blocks.

The PRINT# statement's limitations are discussed in the section on cassette tape. The same limitations to format apply on the disk. RETURNs or commas are needed to separate your data. The CHR$(0) is still read by the GET# statement as an empty string.

Relative and random files both make use of separate data and command "channels." Data written to the disk goes through the data channel, where it is stored in a temporary buffer in the disk's RAM. When the record or block is complete, a command is sent through the command channel that tells the drive where to put the data, and the entire buffer is written.

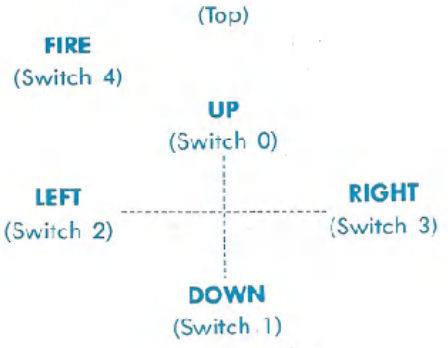
Applications that require large amounts of data to be processed are best stored in relative disk files. These will use the least amount of time and provide the best flexibility for the programmer. Your disk drive manual gives a complete programming guide to use of disk files.

THE GAME PORTS

The Commodore 64 has two 9-pin Game Ports which allow the use of joysticks, paddles, or a light pen. Each port will accept either one joystick or one paddle pair. A light pen can be plugged into Port A (only) for special graphic control, etc. This section gives you examples of how to use the joysticks and paddles from both BASIC and machine language.

The digital joystick is connected to CIA #1 (MOS 6526 Complex Interface Adapter). This input/output device also handles the paddle fire buttons and keyboard scanning. The 6526 CIA chip has 16 registers which are in memory locations 56320 through 56335 inclusive ($DC00 to $DC0F). Port A data appears at location 56320 (DC00) and Port B data is found at location 56321 ($DC01).

A digital joystick has five distinct switches, four of the switches are used for direction and one of the switches is used for the fire button. The joystick switches are arranged as shown:



These switches correspond to the lower 5 bits of the data in location 56320 or 56321. Normally the bit is set to a one if a direction is NOT chosen or the fire button is NOT pressed. When the fire button is

pressed, the bit (bit 4 in this case) changes to a 0. To read the joystick from BASIC, the following subroutine should be used:

```
10 FORK=0TO10:REM SET UP DIRECTION STRING
20 READDR$(K):NEXT
30 DATA"","N","S","","W","NW"
40 DATA"SW","","E","NE","SE"
50 PRINT"GOING...";
60 GOSUB100:REM READ THE JOYSTICK
65 IFDR$(JV)=""THENS0:REM CHECK IF A DIRECTION WAS
CHOSEN
70 PRINTDR$(JV);" ";:REM OUTPUT WHICH DIRECTION
80 IFFR=16THEN60:REM CHECK IF FIRE BUTTON WAS
PUSHED
90 PRINT"-----F-----I-----R-----E-----!!!":GOTO60
100 JV=PEEK(56320):REM GET JOYSTICK VALUE
110 FR=JVAND16:REM FORM FIRE BUTTON STATUS
120 JV=15-(JVAND15):REM FORM DIRECTION VALUE
130 RETURN
```

NOTE: For the second joystick, set JV = PEEK (56321).

The values for JV correspond to these directions:

JV EQUAL TO	DIRECTION
0	NONE
1	UP
2	DOWN
3	—
4	LEFT
5	UP & LEFT
6	DOWN & LEFT
7	—
8	RIGHT
9	UP & RIGHT
10	DOWN & RIGHT

A small machine code routine which accomplishes the same task is as follows:

```
1000 .PAGE (JOYSTICK.8/5)  JOYSTICK - BUTTON READ
ROUTINE
1010 ;
1020 ;AUTHOR - BILL HINDORFF
1030 ;
1040 DX=$C110
1050 DY=$C111
1060 *=$C200
1070 DJRR    LDA $DC00      ;    (GET INPUT FROM PORT
A ONLY)
1080 DJRRB   LDY #0         ;THIS ROUTINE READS AND
DECODES THE
1090         LDX #0         ;JOYSTICK/FIREBUTTON
INPUT DATA IN
1100         LSR A          ;THE ACCUMULATOR. THIS
LEAST SIGNIFICANT
1110         BCS DJR0       ;5 BITS CONTAIN THE
SWITCH CLOSURE
1120         DEY            ;INFORMATION. IF A SWITCH
IS CLOSED THEN IT
1130 DJR0    LSR A          ;PRODUCES A ZERO BIT. IF
A SWITCH IS OPEN THEN
1140         BCS DJR1       ;IT PRODUCES A ONE BIT.
THE JOYSTICK DIR-
1150         INY            ;ECTIONS ARE RIGHT, LEFT,
FORWARD, BACKWARD
1160 DJR1    LSR A          ;BIT3=RIGHT, BIT2=LEFT,
BIT1=BACKWARD,
1170         BCS DJR2       ;BIT0=FORWARD AND
BIT4=FIRE BUTTON.
1180         DEX            ;AT RTS TIME DX AND DY
CONTAIN 2'S COMPLIMENT
1190 DJR2    LSR A          ;DIRECTION NUMBERS I.E.
$FF=-1, $00=0, $01=1.
1200         BCS DJR3       ;DX=1 (MOVE RIGHT), DX=-1
(MOVE LEFT),
1210         INX            ;DX=0 (NO X CHANGE).
DY=-1 (MOVE UP SCREEN),
1220 DJR3    LSR A          ;DY=1 (MOVE DOWN SCREEN),
DY=0 (NO Y CHANGE).
1230         STX DX         ;THE FORWARD JOYSTICK
POSITION CORRESPONDS
1240         STY DY         ;TO MOVE UP THE SCREEN
AND THE BACKWARD
1250         RTS            ;POSITION TO MOVE DOWN
SCREEN.
1260 ;
1270 ;AT RTS TIME THE CARRY FLAG CONTAINS THE FIRE
BUTTON STATE.
1280 ;IF C=1 THEN BUTTON NOT PRESSED. IF C=0 THEN
PRESSED.
1290 ;
1300 .END
```

PADDLES

A paddle is connected to both CIA #1 and the SID chip (MOS 6581 Sound Interface Device) through a game port. The paddle value is read via the SID registers 54297 ($D419) and 54298 ($D41A). PADDLES ARE NOT RELIABLE WHEN READ FROM BASIC ALONE!!!! The best way to use paddles, from BASIC or machine code, is to use the following machine language routine. . . . (SYS to it from BASIC then PEEK the memory locations used by the subroutine).

```
1000
;*******************************************************
1010 ;* FOUR PADDLE READ ROUTINE (CAN ALSO BE USED
FOR TWO)
1020
;*******************************************************
1030 ;AUTHOR - BILL HINDORFF
1040 PORTA=$DC00
1050 CIDDRA=$DC02
1060 SID=$D400
1070 *=$C100
1080 BUFFER *=*+1
1090 PDLX *=*+2
1100 PDLY *=*+2
1110 BTNA *=*+1
1120 BTNB *=*+1
1130 *=$C000
1140 PDLRD
1150     LDX #1                    ;FOR FOUR PADDLES
OR TWO ANALOG JOYSTICKS
1160 PDLRD0                        ;ENTRY POINT FOR
ONE PAIR (CONDITION X 1ST)
1170     SEI
1180     LDA CIDDRA                ;GET CURRENT VALUE
OF DDR
1190     STA BUFFER                ;SAVE IT AWAY
1200     LDA #$C0
1210     STA CIDDRA                ;SET PORT A FOR
INPUT
1220     LDA #$80
1230 PDLRD1
1240     STA PORTA                 ;ADDRESS A PAIR OF
PADDLES
1250     LDY #$80                  ;WAIT A WHILE
1260 PDLRD2
1270     NOP
1280     DEY
1290     BPL PDLRD2
1300     LDA SID+25                ;GET X VALUE
1310     STA PDLX,X
1320     LDA SID+26                ;GET Y VALUE
1330     STA PDLY,X
```

```
1340     LDA PORTA                ;TIME TO READ
PADDLE FIRE BUTTONS
1350     ORA #$00                 ;MAKE IT THE SAME
AS OTHER PAIR
1360     STA BTNA                 ;BIT 2 IS PDL X,
BIT 3 IS PDL Y
1370     LDA #$40
1380     DEX                      ;ALL PAIRS DONE?
1390     BEL PDLRD1               ;NO
1400     LDA BUFFER
1410     STA CIDDRP               ;RESTORE PREVIOUS
VALUE OF DDR
1420     LDA PORTA+1              ;FOR 2ND PAIR -
1430     STA BTNB                 ;BIT 2 IS PDL X,
BIT 3 IS PDL Y
1440     CLI
1450     RTS
1460 .END
```

The paddles can be read by using the following BASIC program:

```
10 C=12*4096:REM SET PADDLE ROUTINE START
11 REM POKE IN THE PADDLE READING ROUTINE
15 FORI=0TO63:READA:POKEC+I,A:NEXT
20 SYSC:REM CALL THE PADDLE ROUTINE
30 P1=PEEK(C+257):REM SET PADDLE ONE VALLE
40 P2=PEEK(C+258):REM   "     "    TWO    "
50 P3=PEEK(C+259):REM   "     "   THREE   "
60 P4=PEEK(C+260):REM   "     "    FOUR   "
61 REM READ FIRE BUTTON STATUS
62 S1=PEEK(C+261):S2=PEEK(C+262)
70 PRINTP1,P2,P3,P4:REM PRINT PADDLE VALUES
72 REM PRINT FIRE BUTTON STATUS
75 PRINT:PRINT"FIRE A ";S1,"FIRE B ";S2
80 FORW=1TO50:NEXT:REM WAIT A WHILE
```

SHIFT CLR/HOME

```
90 PRINT"□":PRINT:GOTO 20:REM CLEAR SCREEN AND IO
AGAIN
95 REM DATA FOR MACHINE CODE ROUTINE
100 DATA162,1,120,173,2,220,141,0,193,169,192,141,
2,220,169
110 DATA128,141,0,220,160,128,234,136,16,252,173,
25,212,157
120 DATA1,193,173,26,212,157,3,193,173,0,220,9,128,
141,5,193
130 DATA169,64,202,16,222,173,0,193,141,2,220,173,
1,220,141
140 DATA6,193,88,96
```

LIGHT PEN

The light pen input latches the current screen position into a pair of registers (LPX, LPY) on a low-going edge. The X position register 19 ($13) will contain the 8 MSB of the X position at the time of transition. Since the X position is defined by a 512-state counter (9 bits), resolution to 2 horizontal dots is provided. Similarly, the Y position is latched in its register 20 ($14), but here 8 bits provide single raster resolution within the visible display. The light pen latch may be triggered only once per frame, and subsequent triggers within the same frame will have no effect. Therefore, you must take several samples before turning the pen to the screen (3 or more samples average), depending upon the characteristics of your light pen.

RS-232 INTERFACE DESCRIPTION

GENERAL OUTLINE

The Commodore 64 has a built-in RS-232 interface for connection to any RS-232 modem, printer, or other device. To connect a device to the Commodore 64, all you need is a cable and a little bit of programming.

RS-232 on the Commodore 64 is set-up in the standard RS-232 format, but the voltages are TTL levels (0 to 5V) rather than the normal RS-232 −12 to 12 volt range. The cable between the Commodore 64 and the RS-232 device should take care of the necessary voltage conversions. The Commodore RS-232 interface cartridge handles this properly.

The RS-232 interface software can be accessed from BASIC or from the KERNAL for machine language programming.

RS-232 on the BASIC level uses the normal BASIC commands: OPEN, CLOSE, CMD, INPUT#, GET#, PRINT#, and the reserved variable ST. INPUT# and GET# fetch data from the receiving buffer, while PRINT# and CMD place data into the transmitting buffer. The use of these commands (and examples) will be described in more detail later in this chapter.

The RS-232 KERNAL byte and bit level handlers run under the control of the 6526 CIA #2 device timers and interrupts. The 6526 chip gener-

ates *NMI (Non Maskable Interrupt)* requests for RS-232 processing. This allows background RS-232 processing to take place during BASIC and machine language programs. There are built-in hold-offs in the KERNAL, cassette, and serial bus routines to prevent the disruption of data storage or transmission by the NMIs that are generated by the RS-232 routines. During cassette or serial bus activities, data can NOT be received from RS-232 devices. But because these hold-offs are only local (assuming you're careful about your programming) no interference should result.

There are two buffers in the Commodore 64 RS-232 interface to help prevent the loss of data when transmitting or receiving RS-232 information.

The Commodore 64 RS-232 KERNAL buffers consist of two first-in/first-out (FIFO) buffers, each 256 bytes long, at the top of memory. The OPENing of an RS-232 channel automatically allocates 512 bytes of memory for these buffers. If there is not enough free space beyond the end of your BASIC program no error message will be printed, and the end of your program will be destroyed. **SO BE CAREFUL!**

These buffers are automatically removed by using the CLOSE command.

OPENING AN RS-232 CHANNEL

Only one RS-232 channel should be open at any time; a second OPEN statement will cause the buffer pointers to be reset. Any characters in either the transmit buffer or the receive buffer will be lost.

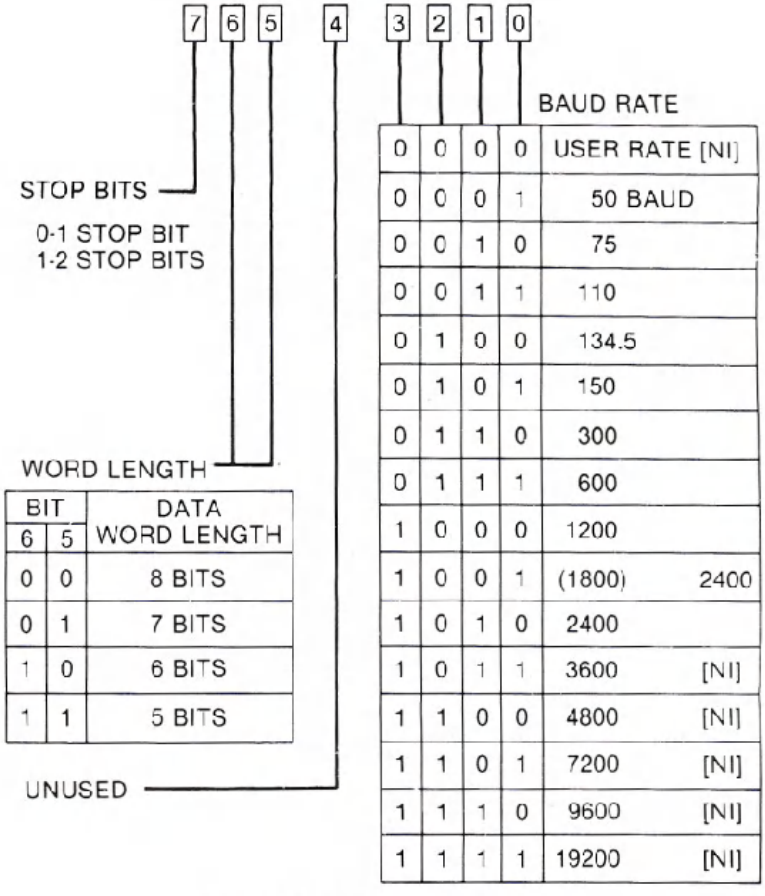
Up to 4 characters can be sent in the filename field. The first two are the control and command register characters; the other two are reserved for future system options. Baud rate, parity, and other options can be selected through this feature.

No error-checking is done on the control word to detect a non-implemented baud rate. Any illegal control word will cause the system output to operate at a very slow rate (below 50 baud).

BASIC SYNTAX:

OPEN lfn,2,0,"<control register><command register><opt baud low><opt baud high>"

lfn— The logical file number (lfn) then can be any number from 1 through 255. But be aware of the fact that if you choose a logical file number that is greater than 127, then a line feed will follow all carriage returns.



BIT 6	BIT 5	DATA WORD LENGTH
0	0	8 BITS
0	1	7 BITS
1	0	6 BITS
1	1	5 BITS

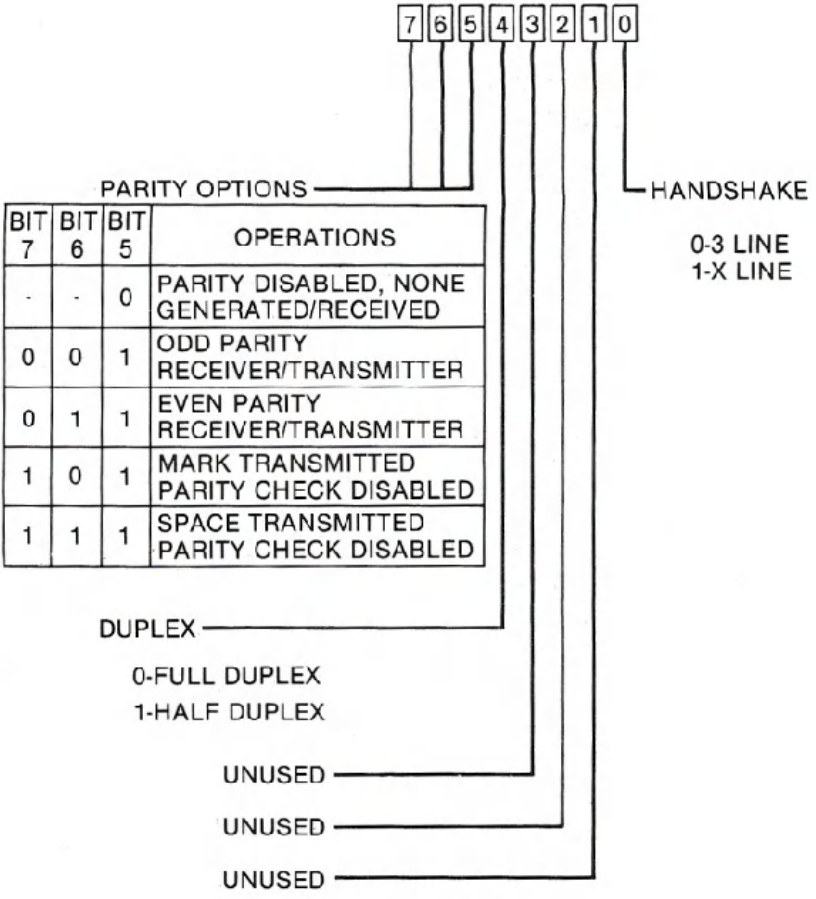
3	2	1	0		
0	0	0	0	USER RATE [NI]	
0	0	0	1	50 BAUD	
0	0	1	0	75	
0	0	1	1	110	
0	1	0	0	134.5	
0	1	0	1	150	
0	1	1	0	300	
0	1	1	1	600	
1	0	0	0	1200	
1	0	0	1	(1800)	2400
1	0	1	0	2400	
1	0	1	1	3600	[NI]
1	1	0	0	4800	[NI]
1	1	0	1	7200	[NI]
1	1	1	0	9600	[NI]
1	1	1	1	19200	[NI]

Figure 6-1. Control Register Map.

<control register> —Is a single byte character (see Figure 6-1, Control Register Map) required to specify the baud rates. If the lower 4 bits of the baud rate is equal to zero (0), the <opt baud low><opt baud high> characters give you a rate based on the following:

<opt baud low>=<system frequency/rate/2−100−<opt baud high>*256

<opt baud high>=INT((system frequency/rate/2−100)/256



BIT 7	BIT 6	BIT 5	OPERATIONS
-	-	0	PARITY DISABLED, NONE GENERATED/RECEIVED
0	0	1	ODD PARITY RECEIVER/TRANSMITTER
0	1	1	EVEN PARITY RECEIVER/TRANSMITTER
1	0	1	MARK TRANSMITTED PARITY CHECK DISABLED
1	1	1	SPACE TRANSMITTED PARITY CHECK DISABLED

Figure 6-2. Command Register Map.

The formulas above are based on the fact that:

system frequency = 1.02273E6 NTSC (North American TV standard)
= 0.98525E6 PAL (U.K. and most European TV standard)

<command register>—Is a single byte character (see Figure 6-2, Command Register Map) that defines other terminal parameters. This character is NOT required.

KERNAL ENTRY:

OPEN ($FFC0) (See KERNAL specifications for more information on entry conditions and instructions.)

> **IMPORTANT NOTE:** In a BASIC program, the RS-232 OPEN command should be performed before creating any variables or arrays because an automatic CLR is performed when an RS-232 channel is OPENed (This is due to the allocation of 512 bytes at the top of memory.) Also remember that your program will be destroyed if 512 bytes of space are not available at the time of the OPEN statement.

GETTING DATA FROM AN RS-232 CHANNEL

When getting data from an RS-232 channel, the Commodore 64 receiver buffer will hold up to 255 characters before the buffer overflows. This is indicated in the RS-232 status word (ST in BASIC, or RSSTAT in machine language). If an overflow occurs, then all characters received during a full buffer condition, from that point on, are lost. Obviously, it pays to keep the buffer as clear as possible.

If you wish to receive RS-232 data at high speeds (BASIC can only go so fast, especially considering garbage collects. This can cause the receiver buffer to overflow), you will have to use machine language routines to handle this type of data burst.

BASIC SYNTAX:

Recommended: GET#lfn, <string variable>
NOT Recommended: INPUT#lfn ,<variable list>

KERNAL ENTRIES:

CHKIN ($FFC6)—See Memory Map for more information on entry and exit conditions.

GETIN ($FFE4)—See Memory Map for more information on entry and exit conditions.

CHRIN ($FFCF)—See Memory Map for more information on entry and exit conditions.

NOTES:

If the word length is less than 8 bits, all unused bit(s) will be assigned a value of zero.

If a GET# does not find any data in the buffer, the character "" (a null) is returned.

If INPUT# is used, then the system will hang in a waiting condition until a non-null character and a following carriage return is received. Therefore, if the Clear To Send (CTS) or DataSsette Ready (DSR) line(s) disappear during character INPUT#, the system will hang in a RESTORE-only state. This is why the INPUT# and CHRIN routines are NOT recommended.

The routine CHKIN handles the x-line handshake which follows the EIA standard (August 1979) for RS-232-C interfaces. (The Request To Send (RTS), CTS, and Received line signal (DCD) lines are implemented with the Commodore 64 computer defined as the Data Terminal device.)

SENDING DATA TO AN RS-232 CHANNEL

When sending data, the output buffer can hold 255 characters before a full buffer hold-off occurs. The system will wait in the CHROUT routine until transmission is allowed or the RUN/STOP and RESTORE keys are used to recover the system through a WARM START.

BASIC SYNTAX:

CMD lfn—acts same as in the BASIC specifications.
PRINT#lfn,<variable list>

KERNAL ENTRIES:

CHKOUT ($FFC9)—See Memory Map for more information on entry and exit conditions.

CHROUT ($FFD2)—See Memory Map for more information on entry conditions.

IMPORTANT NOTES: There is no carriage-return delay built into the output channel. This means that a normal RS-232 printer cannot correctly print, unless some form of hold-off (asking the Commodore 64 to wait) or internal buffering is implemented by the printer. The hold-off can easily be implemented in your program. If a CTS (x-line) handshake is implemented, the Commodore 64 buffer will fill, and then hold-off more output until transmission is allowed by the RS-232 device. X-line handshaking is a handshake routine that uses multi-lines for receiving and transmitting data.

The routine CHKOUT handles the x-line handshake, which follows the EIA standard (August 1979) for RS-232-C interfaces. The RTS, CTS, and DCD lines are implemented with the Commodore 64 defined as the Data Terminal Device.

CLOSING AN RS-232 DATA CHANNEL

Closing an RS-232 file discards all data in the buffers at the time of execution (whether or not it had been transmitted or printed out), stops all RS-232 transmitting and receiving, sets the RTS and transmitted data (S_{out}) lines high, and removes both RS-232 buffers.

BASIC SYNTAX:

CLOSE lfn

KERNAL ENTRY:

CLOSE ($FFC3)—See Memory Map for more information on entry and exit conditions.

NOTE: Care should be taken to ensure all data is transmitted before closing the channel. A way to check this from BASIC is:

```
100 SS=ST: IF(SS=0 OR SS=8) THEN 100
110 CLOSE lfn
```

Table 6-1. User-Port Lines

(6526 DEVICE #2 Loc. $DD00–$DD0F)

PIN ID	6526 ID	DESCRIPTION	EIA	ABV	IN/ OUT	MODES
C	PB0	RECEIVED DATA	(BB)	S_{in}	IN	1 2
D	PB1	REQUEST TO SEND	(CA)	RTS	OUT	1*2
E	PB2	DATA TERMINAL READY	(CD)	DTR	OUT	1*2
F	PB3	RING INDICATOR	(CE)	RI	IN	3
H	PB4	RECEIVED LINE SIGNAL	(CF)	DCD	IN	2
J	PB5	UNASSIGNED	( )	XXX	IN	3
K	PB6	CLEAR TO SEND	(CB)	CTS	IN	2
L	PB7	DATA SET READY	(CC)	DSR	IN	2
B	FLAG2	RECEIVED DATA	(BB)	S_{in}	IN	1 2
M	PA2	TRANSMITTED DATA	(BA)	S_{out}	OUT	1 2
A	GND	PROTECTIVE GROUND	(AA)	GND		1 2
N	GND	SIGNAL GROUND	(AB)	GND		1 2 3

MODES:

1) 3-LINE INTERFACE (S_{in},S_{out},GND)
2) X-LINE INTERFACE
3) USER AVAILABLE ONLY (Unused/unimplemented in code.)

* These lines ore held high during 3-LINE mode.

```
[7] [6] [5] [4] [3] [2] [1] [0] (Machine Lang.—RSSTAT
 :   :   :   :   :   :   :   :—PARITY ERROR BIT
 :   :   :   :   :   :   :———FRAMING ERROR BIT
 :   :   :   :   :   :———————RECEIVER BUFFER OVERRUN BIT
 :   :   :   :   :———————————RECEIVER BUFFER—EMPTY
                             (USE TO TEST AFTER A GET#)
 :   :   :   :———————————————CTS SIGNAL MISSING BIT
 :   :   :———————————————————UNUSED BIT
 :   :———————————————————————DSR SIGNAL MISSING BIT
 :———————————————————————————BREAK DETECTED BIT
```

Figure 6-3. RS-232 Status Register.

NOTES:

If the BIT=0, then no error has been detected.

The RS-232 status register can be read from BASIC using the variable ST.

If ST is read by BASIC or by using the KERNAL READST routine the RS-232 status word is cleared when you exit. If multiple uses of the STATUS word are necessary the ST should be assigned to another variable. For example:

```
SR=ST: REM ASSIGNS ST TO SR
```

The RS-232 status is read (and cleared) only when the RS-232 channel was the last external I/O used.

SAMPLE BASIC PROGRAMS

```
10 REM THIS PROGRAM SENDS AND RECEIVES DATA
TO/FROM A SILENT 700
11 REM TERMINAL MODIFIED FOR PET ASCII
20 REM TI SILENT 700 SET-UP: 300 BAUD, 7-BIT ASCII,
 MARK PARITY,
21 REM FULL DUPLEX
30 REM SAME SET-UP AT COMPUTER USING 3-LINE
INTERFACE
100 OPEN 2,2,3,CHR$(6+32)+CHR$(32+128):REM OPEN
THE CHANNEL
110 GET#2,A$:REM TURN ON THE RECEIVER CHANNEL
(TOSS A NULL)
200 REM MAIN LOOP
210 GET B$:REM GET FROM COMPUTER KEYBOARD
220 IF B$<>"" THEN PRINT#2,B$;:REM IF A KEY
PRESSED, SEND TO TERMINAL
230 GET#2,C$:REM GET A KEY FROM THE TERMINAL
240 PRINT B$;C$;:REM PRINT ALL INPUTS TO COMPUTER
SCREEN
250 SR=ST: IF SR=0 OR SR=8 THEN 200:REM CHECK
STATUS, IF GOOD THEN CONTINUE
300 REM ERROR REPORTING
310 PRINT "ERROR: ";
320 IF SR AND 1 THEN PRINT "PARITY"
330 IF SR AND 2 THEN PRINT "FRAME"
340 IF SR AND 4 THEN PRINT "RECEIVER BUFFER FULL"
350 IF SR AND 128 THEN PRINT "BREAK"
360 IF (PEEK(673) AND 1) THEN 360:REM WAIT UNTIL
ALL CHARS TRANSMITTED
370 CLOSE 2: END
```

```
10 REM THIS PROGRAM SENDS AND RECEIVES TRUE ASCII
DATA
100 OPEN 5,2,3,CHR$(6)
110 DIM F%(255),T%(255)
200 FOR J=32 TO 64:T%(J)=J:NEXT
210 T%(13)=13:T%(20)=8:RV=18:CT=0
220 FOR J=65 TO 90:K=J+32:T%(J)=K:NEXT
230 FOR J=91 TO 95:T%(J)=J:NEXT
240 FOR J=193 TO 218:K=J-128:T%(J)=K:NEXT
250 T%(146)=15:T%(133)=16
260 FOR J=0 TO 255
270 K=T%(J)
280 IF K<>0THEN F%(K)=J:F%(K+128)=J
290 NEXT
300 PRINT " "CHR$(147)
310 GET#5,A$
320 IF A$=""OR ST<>0 THEN 360
330 PRINT " "CHR$(157);CHR$(F%(ASC(A$)));
340 IF F%(ASC(A$))=34 THEN POKE212,0
350 GOTO 310
360 PRINTCHR$(RV)" "CHR$(157);CHR$(146);:GET A$
370 IF A$<>""THENPRINT#5,CHR$(T%(ASC(A$)));
380 CT=CT+1
390 IF CT=8 THENCT=0:RV=164-RV
410 GOTO310
```

RECEIVER/TRANSMITTER BUFFER BASE LOCATION POINTERS

$00F7–RIBUF—A two-byte pointer to the Receiver Buffer base location.

$00F9–ROBUF—A two-byte pointer to the Transmitter Buffer base location.

The two locations above are set up by the OPEN KERNAL routine, each pointing to a different 256-byte buffer. They are de-allocated by writing a zero into the high order bytes ($00F8 and $00FA), which is done by the CLOSE KERNAL entry. They may also be allocated/de-allocated by the machine language programmer for his/her own purposes, removing/creating only the buffer(s) required. When using a machine language program that allocates these buffers, care must be taken to make sure that the top of memory pointers stay correct, especially if BASIC programs are expected to run at the same time.

ZERO-PAGE MEMORY LOCATIONS AND USAGE FOR RS-232 SYSTEM INTERFACE

$00A7–INBIT—Receiver input bit temp storage.
$00A8–BITCI—Receiver bit count in.
$00A9–RINONE—Receiver flag Start bit check.
$00AA–RIDATA—Receiver byte buffer/assembly location.
$00AB–RIPRTY—Receiver parity bit storage.
$00B4–BITTS—Transmitter bit count out.
$00B5–NXTBIT—Transmitter next bit to be sent.
$00B6–RODATA—Transmitter byte buffer/disassembly location.

All the above zero-page locations are used locally and are only given as a guide to understand the associated routines. These cannot be used directly by the BASIC or KERNAL level programmer to do RS-232 type things. The system RS-232 routines must be used.

NONZERO-PAGE MEMORY LOCATIONS AND USAGE FOR RS-232 SYSTEM INTERFACE

General RS-232 storage:

$0293–M51CTR—Pseudo 6551 control register (see Figure 6-1).
$0294–M51COR—Pseudo 6551 command register (see Figure 6-2).
$0295–M51AJB—Two bytes following the control and command registers in the file name field. These locations contain the baud rate for the start of the bit test during the interface activity, which, in turn, is used to calculate baud rate.
$0297–RSSTAT—The RS-232 status register (see Figure 6-3).
$0298–BITNUM—The number of bits to be sent/received.
$0299–BAUDOF—Two bytes that are equal to the time of one bit cell. (Based on system clock/baud rate.)

$029B–RIDBE—The byte index to the end of the receiver FIFO buffer.

$029C–RIDBS—The byte index to the start of the receiver FIFO buffer.

$029D–RODBS—The byte index to the start of the transmitter FIFO buffer.

$029E–RODBE—The byte index to the end of the transmitter FIFO buffer.

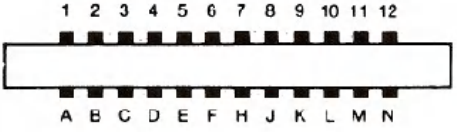
$02A1–ENABL—Holds current active interrupts in the CIA #2 ICR. When bit 4 is turned on means that the system is waiting for the Receiver Edge. When bit 1 is turned on then the system is receiving data. When bit 0 is turned on then the system is transmitting data.

THE USER PORT

The user port is meant to connect the Commodore 64 to the outside world. By using the lines available at this port, you can connect the Commodore 64 to a printer, a Votrax Type and Talk, a MODEM, even another computer.

The port on the Commodore 64 is directly connected to one of the 6526 CIA chips. By programming, the CIA will connect to many other devices.

PORT PIN DESCRIPTION



PORT PIN DESCRIPTION

PIN TOP SIDE	DESCRIPTION	NOTES
1	GROUND	
2	+5V	(100 mA MAX.)
3	RESET	By grounding this pin, the Commodore 64 will do a COLD START, resetting completely. The pointers to a BASIC program will be reset, but memory will not be cleared. This is also a RESET output for the external devices.
4	CNT1	Serial port counter from CIA #1 (SEE CIA SPECS).
5	SP1	Serial port from CIA #1 (SEE 6526 CIA SPECS).
6	CNT2	Serial port counter from CIA #2 (SEE CIA SPECS).
7	SP2	Serial port from CIA #1 (SEE 6526 CIA SPECS).
8	PC2	Handshaking line from CIA #2 (SEE CIA SPECS).
9	SERIAL ATN	This pin is connected to the ATN line of the serial bus.
10	9 VAC+phase	Connected directly to the Commodore 64 transformer (50 mA MAX.).
11	9 VAC−phase	
12	GND	
BOTTOM SIDE		
A	GND	The Commodore 64 gives you control over PORT B on CIA chip #1. Eight lines for input or output are available, as well as 2 lines for handshaking with an outside device. The I/O lines for PORT B are controlled by two locations. One is the PORT itself, and is located at 56577 ($DD01 HEX). Naturally you PEEK it to read an INPUT, or POKE it to set an OUTPUT. Each of the eight I/O lines can be set up as either an INPUT or an OUTPUT by setting the DATA DIRECTION REGISTER properly.
B	FLAG2	
C	PB0	
D	PB1	
E	PB2	
F	PB3	
H	PB4	
J	PB5	
K	PB6	
L	PB7	
M	PA2	
N	GND	

The **DATA DIRECTION REGISTER** has its location at 56579 ($DD03 hex). Each of the eight lines in the PORT has a BIT in the eight-bit DATA DIRECTION REGISTER (DDR) which controls whether that line will be an input or an output. If a bit in the DDR is a ONE, the corresponding line of the PORT will be an OUTPUT. If a bit in the DDR is a ZERO, the corresponding line of the PORT will be an INPUT. For example, if bit 3 of the DDR is set to 1, then line 3 of the PORT will be an output. A further example:

If the DDR is set like this:

BIT #: 7 6 5 4 3 2 1 0
VALUE: 0 0 1 1 1 0 0 0

You can see that lines 5,4, and 3 will be outputs since those bits are ones. The rest of the lines will be inputs, since those lines are zeros.

To PEEK or POKE the USER port, it is necessary to use both the DDR and the PORT itself.

Remember that the PEEK and POKE statements want a number from 0–255. The numbers given in the example must be translated into decimal before they can be used. The value would be:

$$2^5 + 2^4 + 2^3 = 32 + 16 + 8 = 56$$

Notice that the bit # for the DDR is the same number that = 2 raised to a power to turn the bit value on.

(16 = 2↑4=2×2×2×2, 8 = 2↑3=2×2×2)

The two other lines, FLAG1 and PA2 are different from the rest of the USER PORT. These two lines are mainly for HANDSHAKING, and are programmed differently from port B.

Handshaking is needed when two devices communicate. Since one device may run at a different speed than another device it is necessary to give the devices some way of knowing what the other device is doing. Even when the devices are operating at the same speed, handshaking is necessary to let the other know when data is to be sent, and if it has been received. The **FLAG1** line has special characteristics which make it well suited for handshaking.

FLAG1 is a negative edge sensitive input which can be used as a general purpose interrupt input. Any negative transition on the FLAG line will set the FLAG interrupt bit. If the FLAG interrupt is enabled, this will

cause an INTERRUPT REQUEST. If the FLAG bit is not enabled, it can be polled from the interrupt register under program control.

PA2 is bit 2 of PORT A of the CIA. It is controlled like any other bit in the port. The port is located at 56576 ($DD00). The data direction register is located at 56578 ($DD02.)

FOR MORE INFORMATION ON THE 6526 SEE THE CHIP SPECIFICATIONS IN APPENDIX M.

THE SERIAL BUS

The serial bus is a daisy chain arrangement designed to let the Commodore 64 communicate with devices such as the VIC-1541 DISK DRIVE and the VIC-1525 GRAPHICS PRINTER. The advantage of the serial bus is that more than one device can be connected to the port. Up to 5 devices can be connected to the serial bus at one time.

There are three types of operation over a serial bus—CONTROL, TALK, and LISTEN. A CONTROLLER device is one which controls operation of the serial bus. A TALKER transmits data onto the bus. A LISTENER receives data from the bus.

The Commodore 64 is the controller of the bus. It also acts as a TALKER (when sending data to the printer, for example) and as a LISTENER (when loading a program from the disk drive, for example). Other devices may be either LISTENERS (the printer), TALKERS, or both (the disk drive). Only the Commodore 64 can act as the controller.

All devices connected on the serial bus will receive all the data transmitted over the bus. To allow the Commodore 64 to route data to its intended destination, each device has a bus ADDRESS. By using this device address, the Commodore 64 can control access to the bus. Addresses on the serial bus range from 4 to 31.

The Commodore 64 can COMMAND a particular device to TALK or LISTEN. When the Commodore 64 commands a device to TALK, the device will begin putting data onto the serial bus. When the Commodore 64 commands a device to LISTEN, the device addressed will get ready to receive data (from the Commodore 64 or from another device on the bus). Only one device can TALK on the bus at a time; otherwise, the data will collide and the system will crash in confusion. However, any number of devices can LISTEN at the same time to one TALKER.

COMMON SERIAL BUS ADDRESSES

NUMBER	DEVICE
4 or 5	VIC-1525 GRAPHIC PRINTER
8	VIC-1541 DISK DRIVE

Other device addresses are possible. Each device has its own address. Certain devices (like the Commodore 64 printer) provide a choice between two addresses for the convenience of the user.

The SECONDARY ADDRESS is to let the Commodore 64 transmit setup information to a device. For example, to OPEN a connection on the bus to the printer, and have it print in UPPER/LOWER case, use the following:

```
OPEN 1,4,7
```

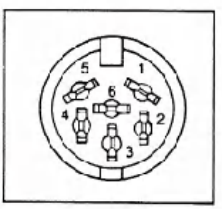
where,

1 is the logical file number (the number you PRINT# to),
4 is the ADDRESS of the printer, and
7 is the SECONDARY ADDRESS that tells the printer to go into UPPER/LOWER case mode.

There are 6 lines used in serial bus operation—3 input and 3 output. The 3 input lines bring data, control, and timing signals into the Commodore 64. The 3 output lines send data, control, and timing signals from the Commodore 64 to external devices on the serial bus.

SERIAL BUS PINOUTS

PIN	DESCRIPTION
1	SERIAL SRQ IN
2	GND
3	SERIAL ATN IN/OUT
4	SERIAL CLK IN/OUT
5	SERIAL DATA IN/OUT
6	NO CONNECTION



SERIAL SRQ IN: (SERIAL SERVICE REQUEST IN)

Any device on the serial bus can bring this signal LOW when it requires attention from the Commodore 64. The Commodore 64 will then take care of the device. (See Figure 6-4).

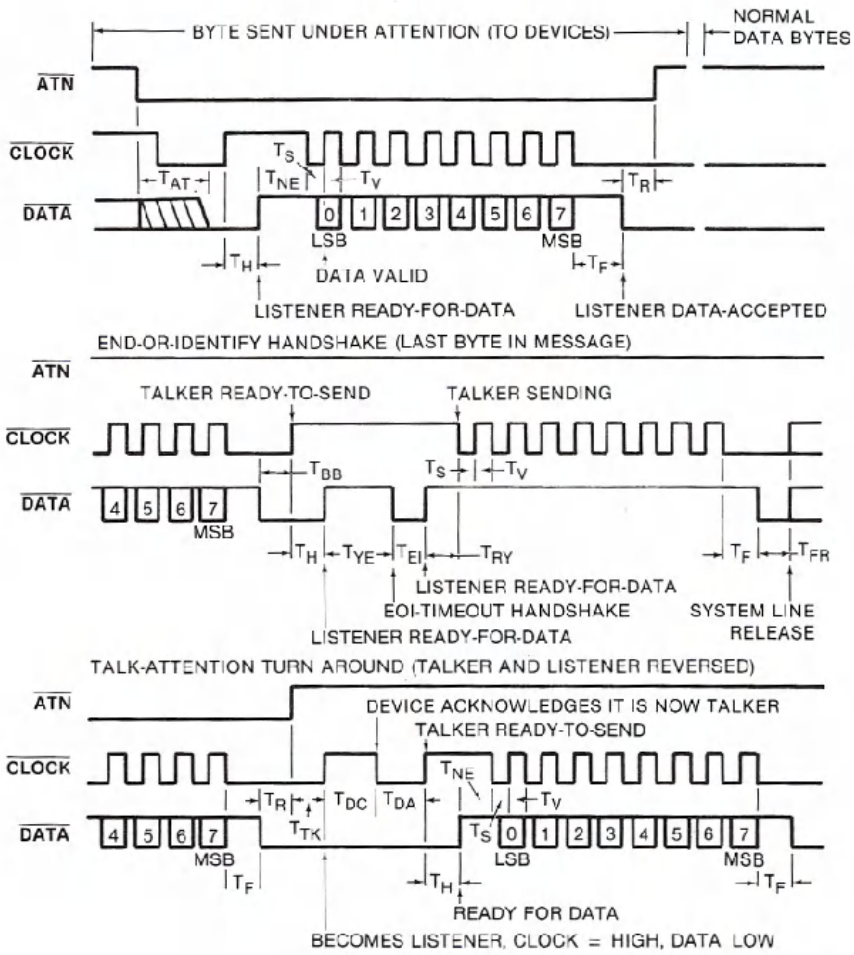
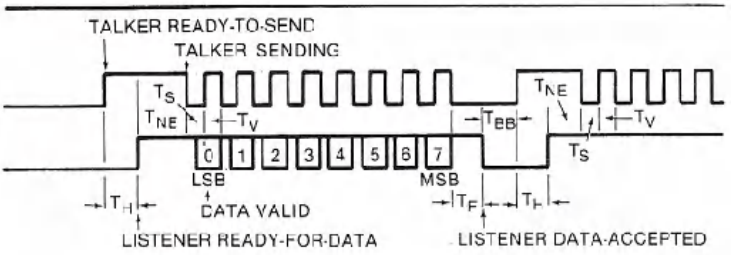


Figure 6-4. Serial

SERIAL ATN IN/OUT: (SERIAL ATTENTION IN/OUT)

The Commodore 64 uses this signal to start a command sequence for a device on the serial bus. When the Commodore 64 brings this signal LOW, all other devices on the bus start listening for the Commodore 64 to transmit an address. The device addressed must respond in a preset period of time; otherwise, the Commodore 64 will assume that the device addressed is not on the bus, and will return an error in the STATUS WORD. (See Figure 6-4).



SERIAL BUS TIMING

Description	Symbol	Min.	Typ.	Max.
ATN RESPONSE (REQUIRED)[1]	T_{AT}	—	—	1000μ
LISTENER HOLD-OFF	T_H	0	—	∞
NON-EOI RESPONSE TO RFD[2]	T_{NE}	—	40μs	200μs
BIT SET-UP TALKER[4]	T_S	20μs	70μs	—
DATA VALID	T_V	20μs	20μs	—
FRAME HANDSHAKE[3]	T_F	0	20	1000μs
FRAME TO RELEASE OF ATN	T_R	20μs	—	—
BETWEEN BYTES TIME	T_{BB}	100μs	—	—
EOI RESPONSE TIME	T_{YE}	200μs	250μs	—
EOI RESPONSE HOLD TIME[5]	T_{EI}	60μs	—	—
TALKER RESPONSE LIMIT	T_{RY}	0	30μs	60μs
BYTE-ACKNOWLEDGE[4]	T_{PR}	20μs	30μs	—
TALK-ATTENTION RELEASE	T_{TK}	20μs	30μs	100μs
TALK-ATTENTION ACKNOWLEDGE	T_{DC}	0	—	—
TALK-ATTENTION ACK. HOLD	T_{DA}	80μs	—	—
EOI ACKNOWLEDGE	T_{FR}	60μs	—	—

Notes:
1. If maximum time exceeded, device not present error.
2. If maximum time exceeded, EOI response required.
3. If maximum time exceeded, frame error.
4. T_V and T_{PR} minimum must be 80μs for external device to be a talker.
5. T_{EI} minimum must be 80μs for external device to be a listener.

Bus Timing.

SERIAL CLK IN/OUT: (SERIAL CLOCK IN/OUT)

This signal is used for timing the data sent on the serial bus. (See Figure 6-4).

SERIAL DATA IN/OUT:

Data on the serial bus is transmitted one bit at a time on this line. (See Figure 6-4.)

THE EXPANSION PORT

The expansion connector is a 44-pin (22/22) female edge connector on the back of the Commodore 64. With the Commodore 64 facing you, the expansion connector is on the far right of the back of the computer. To use the connector, a 44-pin (22/22) male edge connector is required.

This port is used for expansions of the Commodore 64 system which require access to the address bus or the data bus of the computer. Caution is necessary when using the expansion bus, because it's possible to damage the Commodore 64 by a malfunction of your equipment.

The expansion bus is arranged as follows:

The signals available on the connector are as follows:

NAME	PIN	DESCRIPTION
GND	1	System ground
+5 VDC	2	(Total USER PORT and CARTRIDGE devices can
+5 VDC	3	draw no more than 450 mA.)
$\overline{\text{IRQ}}$	4	Interrupt Request line to 6502 (active low)
R/$\overline{\text{W}}$	5	Read/Write
DOT CLOCK	6	8.18 MHz video dot clock
$\overline{\text{I/O1}}$	7	I/O block 1 @ $DE00–$DEFF (active low) unbuffered I/O
$\overline{\text{GAME}}$	8	active low ls ttl input
$\overline{\text{EXROM}}$	9	active low ls ttl input
$\overline{\text{I/O2}}$	10	I/O block 2 @ $DF00–$DFFF (active low) buff'ed ls ttl output

NAME	PIN	DESCRIPTION
$\overline{\text{ROML}}$	11	8K decoded RAM/ROM block @ $8000 (active low) buffered ls ttl output
BA	12	Bus available signal from the VIC-II chip unbuffered 1 ls load max.
$\overline{\text{DMA}}$	13	Direct memory access request line (active low input) ls ttl input
D7	14	Data bus bit 7
D6	15	Data bus bit 6
D5	16	Data bus bit 5
D4	17	Data bus bit 4
D3	18	Data bus bit 3
D2	19	Data bus bit 2
D1	20	Data bus bit 1
D0	21	Data bus bit 0
		} unbuffered, 1 ls ttl load max
GND	22	System ground
GND	A	
$\overline{\text{ROMH}}$	B	8K decoded RAM/ROM block @ $E000 buffered
$\overline{\text{RESET}}$	C	6502 RESET pin (active low) buff'ed ttl out/unbuff'ed in
$\overline{\text{NMI}}$	D	6502 Non Maskable Interrupt (active low) buff'ed ttl out, unbuff'ed in
$\phi 2$	E	Phase 2 system clock
A15	F	Address bus bit 15
A14	H	Address bus bit 14
A13	J	Address bus bit 13
A12	K	Address bus bit 12
A11	L	Address bus bit 11
A10	M	Address bus bit 10
A9	N	Address bus bit 9
A8	P	Address bus bit 8
A7	R	Address bus bit 7
A6	S	Address bus bit 6
A5	T	Address bus bit 5
A4	U	Address bus bit 4
A3	V	Address bus bit 3
A2	W	Address bus bit 2
A1	X	Address bus bit 1
A0	Y	Address bus bit 0
		} unbuffered, 1 ls ttl load max
GND	Z	System ground

Overbar means active low

Following is a description of some important lines on the expansion port:

Pins 1,22,A,Z are connected to the system ground.

Pin 6 is the DOT CLOCK. This is the 8.18-MHz video dot clock. All system timing is derived from this clock.

Pin 12 is the BA (BUS AVAILABLE) signal from the VIC-II chip. This line will go low 3 cycles before the VIC-II takes over the system busses, and remains low until the VIC-II is finished fetching display information.

Pin 13 is the DMA (DIRECT MEMORY ACCESS) line. When this line is pulled low, the address bus, the data bus, and the Read/Write line of the 6510 processor chip enter high-impedance state mode. This allows an external processor to take control of the system busses. This line should only be pulled low when the $\phi 2$ clock is low. Also, since the VIC-II chip will continue to perform display DMA, the external device must conform to the VIC-II timing. (See VIC-II timing diagram.) This line is pulled up on the Commodore 64.

Z-80 MICROPROCESSOR CARTRIDGE

Reading this book and using your computer has shown you just how versatile your Commodore 64 really is. But what makes this machine even more capable of meeting your needs is the addition of peripheral equipment. Peripherals are things like Datassette™ recorders, disk drives, printers, and modems. All these items can be added to your Commodore 64 through the various ports and sockets on the back of your machine. The thing that makes Commodore peripherals so good is the fact that our peripherals are "intelligent." That means that they don't take up valuable Random Access Memory space when they're in use. You're free to use all 64K of memory in your Commodore 64.

Another advantage of your Commodore 64 is the fact most programs you write on your Commodore 64 today will be upwardly compatible with any new Commodore computer you buy in the future. This is partially because of the qualities of the computer's Operating System (OS).

However, there is one thing that the Commodore OS can't do: make your programs compatible with a computer made by another company.

Most of the time you won't even have to think about using another company's computer, because your Commodore 64 is so easy to use. But for the occasional user who wants to take advantage of software that may not be available in Commodore 64 format we have created a Commodore CP/M® cartridge.

CP/M® is not a "computer dependent" operating system. Instead it uses some of the memory space normally available for programming to run its own operating system. There are advantages and disadvantages to this. The disadvantages are that the programs you write will have to be shorter than the programs you can write using the Commodore 64's built-in operating system. In addition, you can NOT use the Commodore 64's powerful screen editing capabilities. The advantages are that you can now use a large amount of software that has been specifically designed for CP/M® and the Z-80 microprocessor, and the programs that you write using the CP/M® operating system can be transported and run on any other computer that has CP/M® and a Z-80 card.

By the way, most computers that have a Z-80 microprocessor require that you go inside the computer to actually install a Z-80 card. With this method you have to be very careful not to disturb the delicate circuitry that runs the rest of the computer. The Commodore CP/M® cartridge eliminates this hassle because our Z-80 cartridge plugs into the back of your Commodore 64 quickly and easily, without any messy wires that can cause problems later.

USING COMMODORE CP/M®

The Commodore Z-80 cartridge let's you run programs designed for a Z-80 microprocessor on your Commodore 64. The cartridge is provided with a diskette containing the Commodore CP/M® operating system.

RUNNING COMMODORE CP/M®

To run CP/M®:

1) LOAD the CP/M® program from your disk drive.
2) Type RUN.
3) Hit the RETURN key.

At this point the 64K bytes of RAM in the Commodore 64 are accessible by the built-in 6510 central processor, OR 48K bytes of RAM are available for the Z-80 central processor. You can shift back and forth between these two processors, but you can NOT use them at the same time in a single program. This is possible because of your Commodore 64's sophisticated timing mechanism.

Below is the memory address translation that is performed on the Z-80 cartridge. You should notice that by adding 4096 bytes to the memory locations used in CP/M® $1000 (hex) you equal the memory addresses of the normal Commodore 64 operating system. The correspondence between Z-80 and 6510 memory addresses is as follows:

Z-80 ADDRESSES		6510 ADDRESSES	
DECIMAL	HEX	DECIMAL	HEX
0000–4095	0000–0FFF	4096–8191	1000–1FFF
4096–8191	1000–1FFF	8192–12287	2000–2FFF
8192–12287	2000–2FFF	12288–16383	3000–3FFF
12288–16383	3000–3FFF	16384–20479	4000–4FFF
16384–20479	4000–4FFF	20480–24575	5000–5FFF
20480–24575	5000–5FFF	24576–28671	6000–6FFF
24576–28671	6000–6FFF	28672–32767	7000–7FFF
28672–32767	7000–7FFF	32768–36863	8000–8FFF
32768–36863	8000–8FFF	36864–40959	9000–9FFF
36864–40959	9000–9FFF	40960–45055	A000–AFFF
40960–45055	A000–AFFF	45056–49151	B000–BFFF
45056–49151	B000–BFFF	49152–53247	C000–CFFF
49152–53247	C000–CFFF	53248–57343	D000–DFFF
53248–57343	D000–DFFF	57344–61439	E000–EFFF
57344–61439	E000–EFFF	61440–65535	F000–FFFF
61440–65535	F000–FFFF	0000–4095	0000–0FFF

To TURN ON the Z-80 and TURN OFF the 6510 chip, type in the following program:

```
10 REM THIS PROGRAM IS TO BE USED WITH THE Z80 CARD
20 REM IT FIRST STORES Z80 DATA AT $1000
(Z80=$0000)
30 REM THEN IT TURNS OFF THE 6510 IRQ'S AND ENABLES
40 REM THE Z80 CARD.  THE Z80 CARD MUST BE TURNED
OFF
50 REM TO REENABLE THE 6510 SYSTEM.
100 REM STORE Z80 DATA
110 READ B: REM GET SIZE OF Z80 CODE TO BE MOVED
120 FOR I=4096 TO 4096+B-1:REM MOVE CODE
130 READ A:POKE I,A
140 NEXT I
200 REM RUN Z80 CODE
210 POKE 56333,127 : REM TURN OF 6510 IRQ'S
220 POKE 56832,00  : REM TURN ON Z80 CARD
230 POKE 56333,129 : REM TURN ON 6510 IRQ'S WHEN
Z80 DONE
240 END
1000 REM Z80 MACHINE LANGUAGE CODE DATA SECTION
1010 DATA 18 : REM SIZE OF DATA TO BE PASSED
1100 REM Z80 TURN ON CODE
1110 DATA 00,00,00 : REM OUR Z80 CARD REQUIRES
TURN ON TIME AT $0000
1200 REM Z80 TASK DATA HERE
1210 DATA 33,02,245 : REM LD HL,NN (LOCATION ON
SCREEN)
1220 DATA 52 : REM INC HL (INCREMENT THAT LOCATION)
1300 REM Z80 SELF-TURN OFF DATA HERE
1310 DATA 62,01 : REM LD A,N
1320 DATA 50,00,206 : REM LD (NN),A :I/O LOCATION
1330 DATA 00,00,00 : REM NOP: NOP: NOP
1340 DATA 195,00,00 : REM JMP $0000
```

For more details about Commodore CP/M® and the Z-80 microprocessor look for the cartridge and the Z-80 Reference Guide at your local Commodore computer dealer.