The Commodore 64 Kernel and Hardware Revealed
Also by Nick Hampshire

*The Commodore 64 ROMs Revealed*
0 00 383087 X

*Advanced Commodore 64 Graphics and Sound*
0 00 383089 6

*Advanced Commodore 64 BASIC Revealed*
0 00 383088 8

*The Commodore 64 Disk Drive Revealed*
0 00 383091 8
Contents

Preface vii
1 Inside the Commodore 64 1
2 The Keyboard, Joysticks and Screen 15
3 Serial Communications 32
4 The Cassette Units 78
5 The User Port 127
6 Interrupts and Their Use 184
Index 193
Whether you program the CBM 64 in Basic or machine code, an understanding of how the kernel software works and how the system hardware functions is essential before writing many programs, particularly those involving connecting the CBM 64 to external devices. This book looks at the way the operating system and system hardware work and should be used in conjunction with Volume I of this series, *The Commodore 64 ROMs Revealed*, which gives the entire operating system kernel software source code.

A knowledge of how the operating system and hardware work enables one to perform many interesting functions. Notable amongst these is the high speed tape load and save routines, which allow the tape deck to operate at speeds equivalent to that on a 1541 disk drive. The whole area of program security is also covered. Without an understanding of the system software and hardware, the operation and use of the serial and RS232 ports is often quite mysterious.

This book is the product of many year's work on Commodore machines, and I am confident that it provides the most complete and useful set of information available from any one source. All serious programmers should find this an invaluable and constant reference book.

Nick Hampshire
# Chapter One

## Inside the Commodore 64

![Memory Architecture Map]

---

**Fig. 1.1.** Commodore 64 memory architecture map.
1.1 Commodore 64 design concept

The division of memory space into blocks in the Commodore 64 is shown in Fig. 1.1. The Commodore 64 is built around the VIC chip and the 6510 microprocessor. As with all microcomputers the Commodore 64 is designed to use the minimum number of chips. This is to reduce component and assembly costs.

Fig. 1.2. Commodore 64 schematics (Reproduced by courtesy of Commodore Business Machines (UK) Ltd).
The main computer circuit (see Fig. 1.2) consists of the 6510, the VIC chip with clock circuit and the eight 64K bit RAM chips. These chips are enough to make a computer circuit that can perform machine code instructions. Note that the VIC chip is required as it supplies most of the computer's RAM control signals and timing. However, to make a usable personal computer the 64 needs a few chips for I/O and sound, also a Basic interpreter and operating system in ROM.
The required input/output is supplied by two 6526 CIA chips. Together these two chips supply 4 timers used for IRQ timing, the tape system and serial. Their I/O ports are used for keyboard scanning, the user port, serial ports and VIC chip bank select. These chips also have serial ports and time of day clocks but these are not used in the CBM 64.

Lastly sound is generated by the SID chip. This has 3 voices each with 4
waveforms an envelope control and filters. The 2 channel analog to digital converter on this chip is connected to the joystick ports.

1.2 Chips in the Commodore 64

The Commodore 64 consists of a plastic case holding a double sided printed
Fig. 1.3. Printed circuit board assembly for the Commodore 64 (Reproduced by courtesy of Commodore Business Machines (UK) Ltd).
circuit board, a 66 key keyboard and assorted plugs and sockets, not forgetting the power switch and LED.

1.2.1 The PC board
The printed circuit board (see Fig. 1.3) has soldered onto one side of it, a few hundred assorted resistors, capacitors, ferrite beads, diodes, coils, 2 voltage regulators, a fuse and 31 integrated circuits (chips). The power supply box supplies the board with 5 volt dc regulated, 9 volt ac and a shielded ground line. The 5 V line is used to power most of the main chips but not the VIC chip or the clock circuit. The 5 V and 12 V for these chips is generated on the board using diodes and voltage regulators to convert the 9 volt ac supplied. This is done to limit interference from the VIC chip and clock circuit which are in a shielded can. (Do not run the computer with this can open as the lid of this provides the heat sink for the VIC chip.)

1.3 The main chips

1.3.1 6510 microprocessor (MPU)
The microprocessor (Fig. 1.4) is a version of the very common 6502. The main difference is that the 6510 has a 6 pin I/O port. In the CBM 64 this port is used for controlling memory configurations through the PLA chip, controlling output lines to the tape deck and sensing keys being pressed on the tape deck.

![Fig. 1.4. 6510 microprocessor (MPU).](image-url)
6510 Signals and Lines

- **Pin 1** (Φ1in): Clock in (from VIC chip)
- **Pin 2** (RDY): Ready. Processor waits in current state while this line is low. If this line is low and interrupts are enabled at the end of the current instruction cycle then an interrupt will be initiated.
- **Pin 4** (NMI): Non-maskable interrupt (negative edge sensitive input). When this line goes from high to low an interrupt will be initiated at the end of the current instruction.
- **Pin 5** (AEC): Address bus enable control. When this line goes low the processor frees the address bus for use by other chips (VIC in the 64).
- **Pin 6** (Vcc): Supply voltage (+5 V)
- **Pins 7–20 & 22, 23** (A0–A15): Address bus (enabled by AEC)
- **Pin 21** (GND): Ground (0 V)
- **Pins 24–29** (P5–P0): Processor I/O port
- **Pins 30–37** (D7–D0): Data bus
- **Pin 38** (R/W): Read/write. (Output: low flags for processor write.)
- **Pin 39** (Φ2in): Phase 2 clock input
- **Pin 40** (RES): Reset (Active low)

### 1.3.2 6526 complex interface adapter (CIA)

There are two of these chips in the Commodore 64. The first CIA#1 is used for...
scanning the keyboard and inputting from the cassette and serial port. The second CIA#2 supports the user port and most of the other serial port lines. It also is used as a latch for the VIC chip bank select value. CIA chip 2 has its IRQ output connected to the NMI line so that its timers can generate NMIs instead of IRQs. The chip select lines for the two chips are decoded from the processor address bus by the PLA chip to addresses \$DC00 & \$DD00 (Fig. 1.5).

### 6526 Signals and Lines

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GND Ground (0 V)</td>
</tr>
<tr>
<td>2-9</td>
<td>PA0-PA7 Data port A (Bi-directional data port)</td>
</tr>
<tr>
<td>10-17</td>
<td>PB0-PB7 Data port B (Bi-directional data port)</td>
</tr>
<tr>
<td>18</td>
<td>PC Handshaking output for port B</td>
</tr>
<tr>
<td>19</td>
<td>TOD Clock input for TOD clock</td>
</tr>
<tr>
<td>20</td>
<td>Vcc Supply voltage (5 V)</td>
</tr>
<tr>
<td>21</td>
<td>IRQ Interrupt request output</td>
</tr>
<tr>
<td>22</td>
<td>R/W Read/write. Input from processor (low for processor write)</td>
</tr>
<tr>
<td>23</td>
<td>CS Chip select. Low indicates a processor read or write to the CIA</td>
</tr>
<tr>
<td>24</td>
<td>FLAG Negative edge sensing input</td>
</tr>
<tr>
<td>25</td>
<td>2 Phase 2 clock input</td>
</tr>
<tr>
<td>26-33</td>
<td>D7-D0 Data bus (Bi-directional depending on R/W)</td>
</tr>
<tr>
<td>34</td>
<td>RES Reset (Active is low)</td>
</tr>
<tr>
<td>35-38</td>
<td>RS3-RS0 Register select. Connected to the lower order address lines to one of the 16 registers</td>
</tr>
<tr>
<td>39</td>
<td>SP Serial input. Not used for Commodore serial bus</td>
</tr>
<tr>
<td>40</td>
<td>CNT Pulse counter, serial clock. Not used for serial bus</td>
</tr>
</tbody>
</table>

![Fig. 1.6. 6581 sound interface device (SID).](image-url)
10 The Commodore 64 Kernal and Hardware Revealed

1.3.3 6581 sound interface device (SID)
SID is a music/sound effects generator for computer games. Its output goes to the modulator and audio/video socket, and has a sound input from a pin on this socket (see Fig. 1.6).

SID Signals and Lines

<table>
<thead>
<tr>
<th>Pins</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 &amp; 2</td>
<td>CAP1a-b</td>
</tr>
<tr>
<td>3 &amp; 4</td>
<td>CAP2a-b</td>
</tr>
<tr>
<td>7</td>
<td>R/W</td>
</tr>
<tr>
<td>8</td>
<td>CS</td>
</tr>
<tr>
<td>9-13</td>
<td>A0-A4</td>
</tr>
<tr>
<td>14</td>
<td>GND</td>
</tr>
<tr>
<td>15-22</td>
<td>D0-D7</td>
</tr>
<tr>
<td>23</td>
<td>POT Y</td>
</tr>
<tr>
<td>24</td>
<td>POT X</td>
</tr>
<tr>
<td>25</td>
<td>Vcc</td>
</tr>
<tr>
<td>26</td>
<td>EXT in</td>
</tr>
<tr>
<td>27</td>
<td>AUDIO</td>
</tr>
<tr>
<td>28</td>
<td>Vdd</td>
</tr>
</tbody>
</table>

1.3.4 6567-9 video interface chip (VIC)
This video display generator chip (Fig. 1.7) also produces most of the internal timing and control signals for the CBM 64, including the processor clock.

VIC generates its own address bus like the 6510. This is used to fetch display data from RAM and character ROM, but since the computer cannot have two completely separate address and data bus systems, VIC and the processor have to share them. 65xx series processors use the system buses only during phase 2 of the clock cycle. The VIC chip takes advantage of this and uses phase 1 of the clock ($\phi_2$ low $\phi_0$ high).

This chip has been given a higher internal bus priority than the 6510 processor. VIC can disable the 6510 and free the address bus for its own use during phase 2 by sending the lines AEC & BA low. The AEC line disables the 6510 address drivers so that its own can drive the address bus. The VIC chip can send the AEC line low during phase 1 and use the address bus without interfering with the processor's operation. The line BA is connected to the 6510's RDY (ready) pin. This can be set low during phase 1 and then held low causing the 6510 to pause at the end of its next read cycle. (This is ignored during 6510 write operations. VIC accessing memory with R/W low would not be desirable anyway!). BA will go low three cycles before AEC is used in phase 2. This ensures that all write operations have finished and avoids conflict with DMA (direct memory access) from any cartridge port device (Z80 card).

VIC also refreshes the dynamic RAM chips using its RAS line and its lower order address bus during phase 1.
VIC Signals and Lines

Pins 1–6 & 35–39: D6–D0 & D11–D7
- Data bus (D0–D7) are bi-directional and are used for register access and VIC memory fetches.
- D8–D11 are used for reading colour RAM.

Pin 8: IRQ
- Interrupt request output.

Pin 9: LP
- Light pen input.

Pin 10: CS
- Chip select.

Pin 11: R/W
- Read/write.

Pin 12: BA
- Bus available.

Pin 13: Vdd
- Supply voltage +12 V.

Pin 14: Colour
- Colour output.

Pin 15: S/LUM
- Sync/luminance.

Pin 16: AEC
- Address bus enable.

Pin 17: φ0
- Phase one clock out.

Pin 18: RAS
- Row address select. Dynamic RAM control signal, used for low order of multiplexed address and for refreshing.

Pin 19: CAS
- Column address select. Dynamic RAM control signal for high order address.

Pin 20: Vss
- Ground (0 V).

Pin 21: φCOLOUR
- Colour clock in 14–18 MHz.

Pin 22: φin
- Clock in 8 MHz.

- High order address output.

Pins 24–29: A0/A8–A5/A13
- Address lines A0–A13 multiplexed together. Gives address for VIC for memory fetches in output mode or register select in input mode.

Pin 40: Vcc
- Supply voltage 5 V.
1.3.5 Programmable logic array (PLA)
This is an array of logic gates programmed together at the time of manufacture to give most of the required logic circuits of the 64 (see Fig. 1.8). The chip has 16 inputs and 8 outputs. A very complicated logic table relates the outputs to the inputs. The pin names of this chip are I0–I15 and F0–F7!

![Fig. 1.8. Programmable logic array (PLA).](image)

The other main chips in the 64 are the RAM and ROM chips. The RAM comprises eight 4164 dynamic RAM chips. One chip gives 1 by 64K bits of memory, so the eight chips give 8 by 64K bits or 64K bytes. The 4164 has 8 address lines. The 16 bit address is multiplexed in (low byte first) and timed with the RAS and CAS lines. The main improvement with this chip over older dynamic RAMs is that it requires only a single 5 V supply instead of −5, +5 and +12 V for a 4116. The colour memory RAM is a 2114 chip; this is a 4 by 1024 bit static RAM.

The Basic and kernal ROMs are 8 bit by 8K ROMs. The character ROM is an 8 by 4K. These chips are Commodore's own manufacture and type. The 6510, 6526 CIA, VIC and SID chips are all manufactured by Commodore's chip manufacturing subsidiary, MOS Technology Inc. It is unfortunate that none of these devices appears to be second sourced and consequently replacements are either very difficult or impossible to obtain.

1.4 System logic and timing

Like all computers the Commodore 64 is a group of chips linked together by address and data buses. The main chips which are connected to the data and address buses are instructed to send to, take from or ignore the data bus by the
control system lines. There are, in addition, lines controlling the use of the address bus. These control lines are defined as follows:

1.4.1 Clock lines
\(\phi\) colour clock
This is the colour clock used by the VIC chip for generating colour signals. It is divided by part of the clock circuit and used as a reference for producing the VIC chip's dot clock.

Dot clock
This signal produced by the clock circuit is the clock input to the VIC chip. VIC uses this as the timing for producing pixels on the screen. Also VIC divides this signal by 8 and supplies it as the processor phase zero clock.

Phase 2 processor clock \(\phi_2\)
This clock line controls all 6510 read and write operations. It is produced by the 6510 from the VIC chip's \(\phi_0\) line. The 65xx series processors require only the system buses while this line is high (5 V).

1.4.2 Main system control signals
Read/Write R/W
If this line is low when a byte of memory or I/O device register is selected by the address bus, then the contents of the data bus will be transferred to the selected byte or register. If the line is high then the contents of the selected address are transferred onto the data bus.

Reset RES
This line is connected to all the main chips including the processor. On machine power up this line is held low for a few clock cycles to ensure the supply voltages have stabilised. This holds all chips in their reset state until they are ready.

Ready RDY
RDY is a processor input which, if low, causes the 6510 to pause at the end of the read cycle. It is used with the AEC line to disable the processor during phase 2 clock cycles for direct memory access.

Interrupt request IRQ
When this line is low it signals that one or more of the CIAs or VIC is requesting an interrupt service.

Non maskable interrupt NMI
When this line goes from high to low the processor will be interrupted at the end of the current instruction cycle. Only a change from high to low will cause an interrupt, so if this line is held low after an NMI it will disable future NMIs.

Bus available BA
When this line is low it flags that the VIC chip needs the system buses during phase 2. It disables the processor via the RDY line.

RAM control signals CAS RAS & CASRAM
The 4164 dynamic RAM chips have their 16 bit addresses fed to in two lots of 8
bits. This is because the 16 pin chip has only an 8 bit address bus. RAS, the row address and CAS, the column address are used to strobe in the low and high bytes. In the 64, CAS and RAM chip select are combined into CASRAM, so when this line is low it flags the high byte of address on the chip’s address pins and the chip is selected for read or write.
Chapter Two

The Keyboard, Joysticks and Screen

2.1 Keyboard

2.1.1 Keyboard hardware and software operation
The CBM 64 keyboard has a total of 66 keys, the layout of which is shown in Fig. 2.1. These 66 keys can be divided as follows:

1. RESTORE key; this is connected directly to the NMI line.
2. The left shift key and the shift lock are connected together.
3. All other 63 keys.

The main section of the keyboard thus has a total of 64 keys. These are organised electrically as an 8 by 8 matrix. The keyboard scanning is performed by the operating system software. The matrix is organised such that the columns are set as outputs by the scanning routine and the rows return a value if a key is pressed. These 8 row inputs and 8 column outputs from the keyboard matrix are connected to the computer via the CIA#1 I/O chip, where the output is via address $DC00 and the input is via address $DC01. The scanning routine loops through 8 times, and each time sends a different line on the output to a low state. It then reads the input port connected to the matrix row lines, which will return values for 8 keys (each key takes up one bit, $0 = \text{down}, 1 = \text{up}$). Therefore, looping 8 times through the scanning routine will look at each of the columns and return all keys which are pressed.

The keyboard is laid out as follows:

![Fig. 2.1. The keyboard layout.](image)
The scanning of the keyboard matrix, and the testing for depression of the RESTORE key, are all under software control. The entire processor time cannot be devoted to keyboard scanning, therefore scanning is initiated by a regular 1/60 second interrupt. Keyboard scanning is one of the functions of the IRQ interrupt servicing routine. The 1/60 second regular interrupt is generated by Timer A of CIA#1. The interrupt service routine starts at location $EA31 and the keyboard scanning portion at $EA87.

The keyboard scanning routine goes through a sequence of operations, the result of which is to place each input character into a special section of memory; the keyboard buffer. The sequence is as follows:

1. Check if key pressed; if not then exit from routine.
2. Initialise I/O ports of CIA#1 for keyboard scan and set pointers into keyboard character table I. Set character counter to 0.
3. Set one line of port A low and test for character input on port B by performing eight right shifts of the contents of port B register; if carry is clear then key present. Each shift increments key count; store key count in .Y.
4. Go back to step 3 and repeat for next column; if key found then continue.
5. Use key count value as index pointer into keyboard character table to get ASCII code corresponding to depressed key.
6. See if it is SHIFT or STOP key.
7. Evaluate shift function
   If SHIFT key then use table 2
   If CBM key then use table 3
   If CONTROL key then use table 4
8. Use key count value as index pointer into keyboard character table designated in step 7.
9. Check for repeat key operation.
10. Do repeat if required.
11. Put ASCII character obtained from keyboard character tables into the keyboard buffer; increment the pointer into the keyboard buffer.

The contents of the 10 character keyboard buffer are accessed on a first in first out basis by the INPUT and GET character routines. These routines take the first character in the keyboard buffer, decrement the buffer pointer and close up
the buffer by moving the contents down one byte thereby leaving space for new input characters.

The characters put into the keyboard buffer are removed by either the INPUT or GET kernel routines. Both these routines call a subroutine at $E5D4 which removes the first character and puts it in register Y then moves the whole buffer down by one byte. This routine is only called if at least one character is in the buffer.

**Warning:** Do not call the routine at $E5D4 when there are no characters in the keyboard buffer as this will crash the computer.

The GET character routine which is accessed by the kernel jumpblock at location $FFE4 (vectored at $032A) will return the Commodore ASCII code of the next character in the keyboard buffer in register .A. If no character was present, the value of zero is returned.

The INPUT routine, when called, will set the cursor flashing and will input characters from the keyboard buffer until a carriage return is found. Each character received is printed to the screen using the routine at $E716 and when a carriage return is found, the routine inputs the first character on the line from the screen and returns it in register .A. Subsequent calls to this routine will return one character at a time until they have all been returned. At this point, if the ASCII value of SHIFT/STOP is found, the LOAD/RUN combination is stored to the buffer replacing all characters following it. The routine is accessed via the kernel jumpblock at location $FFCF (vectored at $0324). A Basic program to emulate the keyboard scanning routine is given in Program 1.

```basic
1000 REM KEYBOARD SCAN SIMULATION PROGRAM
1010 REM ********************************************************************
1020 REM 1030 REM THIS BASIC PROGRAM SIMULATES
1040 REM THE IRQ SCANNING ROUTINE WITH
1050 REM A FULL SCREEN DISPLAY OF WHAT
1060 REM IS HAPPENING, THE ROUTINE FIRST
1070 REM WAITS FOR A KEY TO BE PRESSED
1080 REM AND THEN SCANS THROUGH TO PICK
1090 REM UP THE KEY(S). ANY KEY PRESSED
1100 REM WILL BE DISPLAYED AS A REVERSE
1110 REM KEY IN THE BOX LABELLED "KEY".
1120 REM
1130 REM YOU MUST HOLD DOWN A KEY UNTIL
1140 REM IT IS RECOGNISED.
1150 REM
1160 REM A KEYBOARD BUFFER IS KEPT AND
1170 REM THE ROUTINE WILL EXIT WHEN
1180 REM EITHER THE RETURN KEY IS FOUND
1190 REM OR WHEN THERE ARE TEN CHARACTERS
1200 REM IN THE BUFFER.
1210 REM
1220 P$="~(fI(fI.N~~~W
1230 DIM K$(71):FOR I=0 TO 71:READ K$(I):HEXT
1240 DIM K(3,64):FOR J=0 TO 2:FOR I=0 TO 64:K(J,I)=PEEK(69289+J*65+I):HEXTI,J
1250 FOR I=9 TO 64:K(3,I)=PEEK(69536+I):HEXTI
1260 PRINT "~
1270 PRINT "KEYBOARD SCAN SIMULATION"
1280 PRINT "OUTPUT: "1290 PRINT ":7DC00 INPUT :7DC81"
1290 PRINT ": BIT :76543210 bit :76543210"
1290 PRINT "I"
1300 PRINT ": "
1310 PRINT "J"
1320 PRINT "KEY : "
1330 FORI=1 TO4 :PRINTP$;FORJ=1 TO1 :PRINT":":NEXT
1340 PRINT ":" NEXT
```
The Commodore 64 Kernal and Hardware Revealed

1350 PRINT "CURRENT SHIFT : 0"
1370 PRINT "CURRENT KEY  : 64"
1350 PRINT " KEY BUFFER"
1400 POKE 56333,1:FON=0:T07:P2(I)=2:NEXT
1410 K€=0:CS=8:CK=64:VL=8:A=255:GOSUB 1700
1420 V=0:POKE 56332,1:GOSUB 1860:A=PEEK(56321)
1430 IF A=255 THEN 1420
1440 GOSUB 1700
1450 V=254:FOR VL=0 TO 7:POKE 56321,V:GOSUB 1660:DISPLAY OUT NEW
1460 A=PEEK(56321):GOSUB 17100 DISPLAY INPUT
1470 IF A=255 THEN 1420
1480 FOR I=0 TO 7:POKE 56321+I,K€:NEXT
1490 IF CS=3 THEN 1500
1500 IF A=255 THEN 1540
1510 POKE 53248+24,PEEK(53248+24):OR2
1520 POKE 53248+24,PEEK(53248+24):OR2
1530 GOTO 1560
1540 POKE 53248+24,PEEK(53248+24):AND253
1550 KT=CS:IF KT=3 THEN 3 THENENKT=3
1560 CC=K(KT,CK)
1570 IF CC=255 THEN 1650
1580 KB(BP)=CC:BP=BP+1
1590 IF CS=13 THEN 1640
1600 FOR I=0 TO BP-1:POKE 53248+I,KB(I):NEXT
1610 POKE 198, BP:POKE 56332, 129
1620 PRINT "###SUCCEEDED###SUCCEEDED###SUCCEEDED"
1630 END
1640 GOSUB 1900
1650 GOTO 1360
1660 REM
1670 REM DISPLAY OUTPUT VALUE BITWISE
1680 REM
1690 A=V:PRINT "###SUCCEEDED###SUCCEEDED###SUCCEEDED"
1700 REM
1710 REM DISPLAY INPUT VALUE BITWISE
1720 REM
1730 PRINT "###SUCCEEDED###SUCCEEDED###SUCCEEDED"
1740 FOR I=7 TO 0 STEP -1 :PRINT $;
1750 FOR J=7 TO 0 STEP -1:PRINT "#";:NEXT
1760 C=VL*8+I:FOR J=1 TO 7
1770 C$=MID$(KJ(C)+",J,1)
1780 PRINT "#";:IF A AND P2(I)=0 THEN PRINT "$";
1790 PRINTCS$":NEXT:NEXT:PRINT "$";:RETURN
1800 REM
1810 REM DISPLAY VALUE IN A BITWISE
1820 REM
1830 FOR I=7 TO 0 STEP -1
1840 C$="#";IF A AND P2(I)=0 THEN C$="$"
1850 PRINT "$";C$;":NEXT:RETURN
1860 REM
1870 REM CHECK FOR A KEY FROM THIS COLUMN
1880 REM
1890 FOR X=0 TO 7
1900 IF A AND P2(X)=0 THEN 1950
1910 NEXT
1920 PRINT "###SUCCEEDED###SUCCEEDED###SUCCEEDED"
1930 IF IN=4 THEN CK=X+VL*$GOTO 1910
1940 PRINT "$";CK;":NEXT:RETURN
1950 RETURN
1960 IF IN=4 THEN CK=X+VL*$GOTO 1910
1970 IF IN=3 THEN CK=X+VL*$GOTO 1910
1980 CS=CSORIN:GOTO 1910
1990 REM
2000 REM DISPLAY KEYBOARD BUFFER
2010 REM
2020 PRINT "###SUCCEEDED###SUCCEEDED###SUCCEEDED"
2030 FOR I=0 TO 9
2040 PRINT "$";KB(I):NEXT
2050 PRINT"*";
2060 IF BP=0 THEN 2080
2070 PRINTLEFT$("*",BP-1);"*
2080 RETURN
2100 REM
2110 REM MNEMONICS FOR KEY PRESS
2120 REM
2130 DATA DEL,RET,C.RT,F7,F1,F3,F5,C.IN
2140 DATA S,A,4,Z,S,E,L.SH
2160 DATA 5,R,D,6,C,F,T,X
2170 DATA 7,'r',G,8,B,H,U,
2180 DATA 9,I,J,0,M,K,O,N
2190 DATA +,P,
L,-,.,": "S",Of,
2200 DATA l,"CTRL2,SPC,CBM,Q,STOP
2210 DATA , , , , , ,

Program 1.

2.1.2 Modification of keyboard operation

Program 2 shows how a wedge can be made into the INPUT routine to give the function keys text definition. This follows the same principle as the expansion of the SHIFT/STOP character, with the exception that subsequent characters are not lost. Each function key definition can be up to 255 characters long and the definitions are stored behind the Basic ROM.

033C ! FUNCTION KEYS FOR THE 64.
033C !***********************************************************************
033C ! EACH KEY CAN HAVE A MAXIMUM
033C ! DEFINITION OF 255 CHARACTERS LONG
033C ! (MAXIMUM BASIC STRING LENGTH)
033C ! DEFINE A FUNCTION KEY USING:
033C !SYS 49163.N.DEFINITION
033C !
033C ! INITIALISE FUNCTION KEYS WITH:
033C !SYS 49152.
033C !
0300 8963 LDA #$FUNCTION
0302 82403 STA #$324
0305 82560 LDA #$FUNCTION
0307 82580 STA #$325
030A 60 RTS
030B !
030B 20F1B7 JSR $B7F1
030E 89E09 CPX #$09
0310 9005 BCC DEF1
0312 A20E JSR $AEFD
0314 6C0003 STA <FUNCTION
0317 !
0317 E000 DEF1 CPX #$00
0319 F0F7 BEQ ERROR
031C 4A DEX
031C 9A TXA
031D 48 PHA
031E 20F1DA JSR $AEFD
0321 209E9D JSR $A9E
0324 20A386 JSR $A6A3
0327 8D51C0 STA STLEN
032A 60 PLA
032B 9A ASL A
032C 9A TAX
032D 8D53C0 LDA POINT.X
0330 8524 STA #$24
0332 8D54C0 LDA POINT+1.X
The Commodore 64 Kernal and Hardware Revealed

C035 B525 STA $25
C037 A000 LDY $00
C039 B122 DEF2 LDA ($22),Y
C03B 9124 STA ($24),Y
C03D C0 CPY STLEN
C03E C51C0 CPY STLEN END OF STRING?
C041 F005 BEQ DEF4 YES
C043 C000 CPY #$00 END OF ROOM?
C045 D0F2 BNE DEF2 NOT YET
C047 60 DEF3 RTS
C048 C000 DEF4 CPY #$00 STRING LENGTH=256?
C04A F0FB BEQ DEF3 YES
C04C A900 LDA #$00 ZERO TERMINATOR
C04E 9124 STA ($24),Y STORE IT
C050 60 RTS
C051 00 STLEN BYT 0
C052 00 FUNCFLG BYT 0
C053 00B800 POINT WOR $B000,$BC00,$B900,$BD00
C055 00B400 WOR $B400,$BE00,$BD00,$BF00
C056  !
C057 A599 FUNCTION LDA $99 FROM KEYBOARD?
C058 F003 BEQ FUNC02 YES
C05A 4C57F1 JMP $F157 DO NORMAL
C05C  A5D3 FUNC02 LDA #$3 SAVE CURRENT CURSOR
C05E 85CA STA #$C COLUMN
C060 A5D6 LDA #$6 AND ROW
C062 95C9 STA #$9
C064 98 TYA
C066 48 PHA
C068 9A TXA
C06A 48 PHA
C06C A5D0 LDA #$0 SCREEN OR KEYBOARD?
C06E F006 BEQ FUNC04 KEYBOARD
C070 4C3AE6 JMP $E63A DO FOR SCREEN
C072 2016E7 FUNC03 JSR $E716 DISPLAY CHAR TO SCREEN
C074 AD52C0 FUNC04 LDA FUNCFLG DOING FUNCTION KEY?
C076 D066 BNE FUNC09
C078 A5C6 LDA #$6 ANY CHARs IN BUFFER?
C07A 85CC STA #$C IF NOT, BLINK CURSOR
C07C BD9202 STA #$292 AUTO SCROLL DOWN
C07E F0F2 BEQ FUNC04 REPEAT UNTIL CHAR
C080 78 SEI DISABLE KEYBOARD
C082 A5CF LDA #$C CURSOR BLINK?
C084 F08C BEQ FUNC05 NO
C086 A5CE LDA #$E RESTORE ORIG CHAR
C088 AE8702 LDX #$287 AND COLOUR
C08A 0000 LDY #$00
C08C 84CF STY #$CF SWITCH OFF BLINK
C08E 2013EA JSR #$E13 RESTORE CHAR
C08F 20B4E5 FUNC05 JSR #$E5B4 REMOVE CHAR
C091 C983 CMP #$83 RUN/STOP?
C093 D810 BNE FUNC07 NO
C095 A289 LDX #$89 COPY TEXT INTO BUFFER
C097 78 SEI
C099 B6C6 STX #$6
C09B ADE6EC FUNC06 LDA #$E66,X
C09D 9D7602 STA #$276,X
C09F CA DEX
C0A1 0DEF7 BNE FUNC06 REPEAT UNTIL DONE
C0A3 F0CA BEQ FUNC04 DONE
C0A4 C90D CMP #$00 CARRIAGE RETURN?
C0A6 0D03 BNE FUNC08 NO
C0A8 4C02E6 FUNCEXT JMP #$E602 END INPUT
C0AA  !
C0AC A5D4 FUNC08 LDX #$4 QUOTES?
C0EB D0BC BNE FUNC03 YES
To enable the function keys enter:

SYS 49152

To define a function key use:

SYS 49163,k#,def$

where k# is the number on the function key (without the ‘f’) and def$ is any string expression.

The text on the function keys will appear only if the function key character is removed by the input routine. This means that when using the GET command, the ASCII character for the function key is returned rather than a character from the text. The function key is not expanded if it is within quotes.

The following is an example definition of a function key:

A$="":FOR I=0 TO 79:A$=A$+CHR$(32)+CHR$(2.0):NEXT
SYS 49163,7,A$

This will set up a function on key 7 which deletes from the cursor position to the end of the line, leaving the cursor at the same position (space-delete 80 times).
2.2 Joysticks

There are two different types of joystick which can be connected to the CBM 64; a simple paddle switch joystick and a potentiometer or analog joystick. The switch joystick is widely used in games programs to move a cursor about the screen or to move an object. A switch joystick is primarily capable of only very simple directional input. It is, however, a very low cost device. The analog joystick is fairly expensive but is capable of far greater positional control. An interesting version of the analog joystick has started to appear in the form of low cost digitising pads which, when combined with the appropriate software, can produce some excellent computer art on the CBM 64.

2.2.1 The switch joystick

These joysticks are not part of the keyboard hardware but they are connected to the same lines on the CIA#1 chip: port 1 to the read line and port 2 to the write line:

```
$DC00:  Bits 7-5 Not used
  4  JOY2 fire button
  3  JOY2 east
  2  JOY2 west
  1  JOY2 south
  0  JOY2 north

$DC01:  Bits 7-5 Not used
  4  JOY1 fire button
  3  JOY1 east
  2  JOY1 west
  1  JOY1 south
  0  JOY1 north
```

As with the keyboard, both joysticks must be read assuming that when the bit is zero, the contact is made. Because port 1 is connected to the same line as the keyboard read, any switches on joystick 1 will affect the character read in. Program 3 demonstrates the operation of the switch joystick.

```
10 REM EXAMPLE OF READING THE JOYSTICK
20 REM
30 PRINT"#";
40 A=PEEK(56320):REM PORT 2, 56321 FOR PORT 1
50 F=0:IF (AAND16)=0 THEN F=1
60 E=0:IF (AAND8)=0 THEN E=1
70 W=0:IF (AAND4)=0 THEN W=1
80 S=0:IF (AAND2)=0 THEN S=1
90 N=0:IF (AAND1)=0 THEN N=1
100 IF F THEN RUN
110 IF E THEN PRINT"*";
120 IF W THEN PRINT"|";
130 IF S THEN PRINT"|";
140 IF N THEN PRINT"|";
150 GOTO 40
```

Program 3.

2.2.2 Potentiometer joystick

A potentiometer joystick consists of two potentiometers mounted at right
angles to each other in a mechanism which allows the joystick when moved to change the wiper position of either one or both of the potentiometers. One potentiometer registers the potentiometer movement in the X axis and the other in the Y axis. The rotational movement of either potentiometer is divided by the computer into 255 divisions. With the joystick centered vertically the X and Y potentiometers will both have a value of 128. The position of the joystick can thus be mapped in terms of 2D graph coordinates.

The two potentiometers are connected to the SID chip. SID has two analog inputs, and the two analog lines from each joystick port are multiplexed onto each input using a 4066 quad analog switch. The 4066 switching is controlled by lines PA6 and PA7 on CIA#1. Program 4 allows the input of values from two joysticks using the USR command.

```
20BF1 USR JSR $B1BF !FLOAT TO FIXED
20B55 LDA #$65 !LOW BYTE
20293 AND #3
202A8 TAY
20248 LDX #$40
20252 AND #2 !PORTS 0,1 OR 2,3
2026C BNE P00R1
20288 LDA #$80
20294 BEQ LOOP
202A5 P00R1
202B0 LDX #$80
202B5 STX #$DC00 !USES #DC00
202B9 LDX #$90
202C4 DEX !DELAY FOR A/D
202C8 LOOP BPL LOOP !CONVERTER
202DD TYA
202E5 AND #1
202F4 TRA
20310 BCD $D419,X !READ PORT
20316 CLI
20322 LDA #$0 !ZERO HIGH BYTE
20327 JMP $B391 !FIXED TO FLOAT & EXIT
```

Program 4.

### 2.3 The screen

#### 2.3.1 The hardware

The screen display on the Commodore 64 is created and controlled by one chip; the VIC II (video interface controller 6567/9). A detailed description of the VIC II hardware can be found in Chapter 1.

#### 2.3.2 The screen display operating system software

None of the wide range of potential features of the VIC II chip are implemented by the software of the 64 with the exception that on power-up the default screen and border colours are set up, and the case bit is toggled. The kernel software to control the text screen is split into two sections; print a character to the screen,
and scroll the screen. The routine to print a character to the screen is located at $E716. This routine prints the character in register .A to the screen taking into account colour control codes, etc. This routine does several tasks before the character is printed; these tasks are shown in the flow chart in Fig. 2.2.

The flow chart in Fig. 2.3 shows how the screen scrolls. This routine can be called from Basic with SYS 59626.

Readers interested in the addition of extra commands which utilise the capabilities of the VIC II chip should consult the companion volume in this series, *Advanced Commodore 64 Graphics and Sound*. 
The Keyboard, Joysticks and Screen

Display non control char

Fig. 2.2. Character output flowchart.
Handle char $<128$

Fig. 2.2. (contd.)
Test for colour or lower case

Handle colour codes

LOAD .X WITH 15

RETURN

LOAD COLOUR VAL WITH .X

RETURN

Fig. 2.2. (contd.)
Handle char #>127

AND CHAR WITH $7F

IS CHAR #? ($7F)

YES

CHAR # = $5E

CONTROL CHAR? < $20

YES

SHIFT RETURN? ($0D)

YES

NO

QUOTES OPEN?

YES

NO

INSERT? ($14)

YES

NO

ANY INSERTS?

YES

NO

OR CHAR WITH $4F

12

OR CHAR WITH $8F

1F

UPPER CASE? ($8E)

YES

NO

SET VIC CHIP TO UPPER CASE

6

Fig. 2.2. (contd.)
The Keyboard, Joysticks and Screen

Fig. 2.2. (contd.)
Operate on carriage return

Delete a character

7

ZERO # INSERTS
RESET QUOTES
FLAG, SET REVERSE OFF, MOVE CURSOR TO NEXT LINE

OFF BOTTOM?

NO

YES

SCROLL SCREEN UP 1 OR 2 LINES

6

8

FIRST CHAR ON LINE?

YES

MOVE 1 CHAR BACK

NO

DECREASE COLUMN BY 1

TOP OF SCREEN?

6

6

POSITION AT TOP LEFT OF SCREEN

6

Figure 2.2 (contd.)
Scroll screen up

$E8\ EA$

SAVE OFF
SA and EA

DECREASE
ROW #
POINT TO
SECOND LINE

24
LINES
DONE?

YES

NO

DECREASE
LINE #
BY 1

STORE
SPACES IN
BOTTOM LINE

SCROLL
LINE UP 1
LINE

DECREASE
LINE #
BY 1

STORE
SPACES IN
BOTTOM LINE

SCROLL LINE
LINK HIGH BITS
DOWN 1 BYTE.
SET BOTTOM LINE
AS NOT LINKED

YES

TOP
LINE
LINKED?

NO

INCREASE
ROW #

CONTROL
KEY
DOWN?

YES

NO

PAUSE LOOP
CLEAR KEY-
BOARD BUFFER

RESTORE
SA and EA

EXIT
(RTS)

Fig. 2.3. Screen scrolling flowchart.
The CBM 64 has two different forms of serial communications capability; these are the Commodore serial interface and the RS232 interface. The Commodore serial interface is designed to allow the CBM 64 to be connected to other Commodore peripherals, in particular the 1541 disk drive. The RS232 serial interface is a simplified version of an industry standard communications interface widely used when connecting computers to printers and modems. Unfortunately the RS232 interface does not conform to the correct industry standards and therefore requires a small additional circuit to make it function properly.

### 3.1 Commodore serial bus

The Commodore serial bus connects the Commodore 64 to its peripherals such as a disk drive and printers. This serial system has an effective speed of 3000 baud. This is not a true baud rate but is given just for comparison with the 300 baud normal cassette or 3600 baud for the high speed tape system in Chapter 4. A speed of 3000 baud is adequate for communicating with printers but makes the 1541 disk drive a little slow. The serial bus uses 5 lines including the ground line.

#### 3.1.1 Commodore serial bus lines

**Serial service request**

Input: This enables a serial device to generate an IRQ in the 64. (No CBM 64 firmware support is available for this feature.)

**Signal ground**

This is a common ground line for serial devices. It is for signal reference and shielding the cable.

**Serial attention**

Input and output: Normally the CBM 64 can use this line only as an output. The 64 pulls this line low when sending command bytes to serial devices. It instructs all serial devices to listen for a command.
Serial Communications

Serial clock and serial data lines
These two lines are both inputs and outputs. The current talking device uses these lines to send data and clock signals. Together these lines carry all data and perform the required handshaking.

The serial bus signals are produced in the CBM 64 by port A of CIA chip 2. The following table shows the line connections.

<table>
<thead>
<tr>
<th>Serial Data</th>
<th>Serial Clock</th>
<th>Serial Attn</th>
<th>SRQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>in on PA7</td>
<td>in on PA6</td>
<td>in to user port pin 9</td>
<td>in to CIA chip 1 FLAG pin</td>
</tr>
<tr>
<td>out on PA5</td>
<td>out on PA4</td>
<td>out on PA3</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 3.1 shows the serial port line driver. The serial port lines are driven by a 7406 inverting buffer/driver chip with its outputs tied to 5 V with 1K resistors (Fig. 3.2). The 7406 was chosen for its open collector outputs. An open collector

![Fig. 3.1. Serial port line driver.](image)

7406 inverting buffer/driver chip with its outputs tied to 5 V with 1K resistors (Fig. 3.2). The 7406 was chosen for its open collector outputs. An open collector

![Fig. 3.2. 7406 schematics (one gate).](image)
output can only drain current, not source it. So when the output of a 7406 gate is low it can pull the serial line low but when the output is high the serial line has to be pulled to 5 V by the 1K resistor. The 64 uses this by having the 5 V state as the release state i.e. available for use by other devices. When a line is in the released state the open collector outputs on another serial device can pull the line low.

The 7406 is an inverting buffer/driver, so all clock, data and attention signals sent are transmitted inverted. Therefore release or line high is sent as a zero but received as a logic one.

Only one device on the serial bus can talk at any one time but any number can listen. The Commodore 64 controls which device talks and which listens by commands sent with the attention line low (true). A timing diagram for serial operation is shown in Fig. 3.3.

1. Command byte sent under attention.

*Fig. 3.3. Serial bus timing.*
2. Normal bytes on serial.

---

**Fig. 3.3. (contd.)**
3. Last byte of a message (EOI handshake).

![Diagram of ATN, CLOCK, DATA signals with timing labels: Talker ready to send, Listener ready, Releases data, Listener accepted data, Pulls data low, TNE < 0.2 ms.]

*Fig.3.3. (contd.)*
4. Listener becomes talker.

Fig. 3.3. (contd.)
3.1.2 Serial commands

LISTEN  Command device to listen
TALK    Command device to talk
UNLSN   Commands all listening devices to unlisten
UNTLK   Commands talking device to stop talking (UNTALK)

3.2 Serial ROM routines

<table>
<thead>
<tr>
<th>LOC</th>
<th>CODE</th>
<th>LINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td></td>
<td>;*************************************************************************</td>
</tr>
<tr>
<td>0000</td>
<td></td>
<td>; SERIAL SYSTEM</td>
</tr>
<tr>
<td>0000</td>
<td></td>
<td>;*************************************************************************</td>
</tr>
<tr>
<td>0000</td>
<td></td>
<td>.LIB KDECLARE</td>
</tr>
<tr>
<td>0000</td>
<td>$=90</td>
<td>*</td>
</tr>
<tr>
<td>0090</td>
<td></td>
<td>;</td>
</tr>
<tr>
<td>0090</td>
<td></td>
<td>;KERNEL VARIABLES</td>
</tr>
<tr>
<td>0090</td>
<td></td>
<td>;</td>
</tr>
<tr>
<td>0090</td>
<td>STATUS ***+1</td>
<td>;I/O OPERATION STATUS BYTE</td>
</tr>
<tr>
<td>0091</td>
<td>STKEY ***+1</td>
<td>;STOP KEY FLAG</td>
</tr>
<tr>
<td>0092</td>
<td>SVXT ***+1</td>
<td>;TEMPORARY</td>
</tr>
<tr>
<td>0093</td>
<td>VERCK ***+1</td>
<td>;LOAD OR VERIFY FLAG</td>
</tr>
<tr>
<td>0094</td>
<td>C3PO ***+1</td>
<td>;IEEE BUFFERED CHAR FLAG</td>
</tr>
<tr>
<td>0095</td>
<td>BSOUR ***+1</td>
<td>;CHAR BUFFER FOR IEEE</td>
</tr>
<tr>
<td>0096</td>
<td>SYNO ***+1</td>
<td>;CASSETTE SYNC #</td>
</tr>
<tr>
<td>0097</td>
<td>XSAV ***+1</td>
<td>;TEMP FOR BASIN</td>
</tr>
<tr>
<td>0098</td>
<td>LDNDO ***+1</td>
<td>;INDEX TO LOGICAL FILE</td>
</tr>
<tr>
<td>0099</td>
<td>DFLTIN ***+1</td>
<td>;DEFAULT INPUT DEVICE #</td>
</tr>
<tr>
<td>009A</td>
<td>DFLT0 ***+1</td>
<td>;DEFAULT OUTPUT DEVICE #</td>
</tr>
<tr>
<td>009B</td>
<td>PRTY ***+1</td>
<td>;CASSETTE PARITY</td>
</tr>
<tr>
<td>009C</td>
<td>DPSW ***+1</td>
<td>;CASSETTE DIPROLE SWITCH</td>
</tr>
<tr>
<td>009D</td>
<td>MSFLG ***+1</td>
<td>;DS MESSAGE FLAG</td>
</tr>
<tr>
<td>009E</td>
<td>PTR1 ***+1</td>
<td>;CASSETTE ERROR PASS1</td>
</tr>
<tr>
<td>009F</td>
<td>T1 ***+1</td>
<td>;TEMPORARY 1</td>
</tr>
<tr>
<td>00E0</td>
<td>PTR2</td>
<td>;CASSETTE ERROR PASS2</td>
</tr>
<tr>
<td>00E1</td>
<td>T2 ***+1</td>
<td>;TEMPORARY 2</td>
</tr>
<tr>
<td>00A0</td>
<td>TIME ***+3</td>
<td>;24 HOUR CLOCK IN 1/60TH SECONDS</td>
</tr>
<tr>
<td>00A3</td>
<td>R2D2</td>
<td>;SERIAL BUS USAGE</td>
</tr>
<tr>
<td>00A3</td>
<td>FCHTR ***+1</td>
<td>;CASSETTE STUFF</td>
</tr>
<tr>
<td>00A4</td>
<td>BSOUR1</td>
<td>;TEMP USED BY SERIAL ROUTINE</td>
</tr>
<tr>
<td>00A4</td>
<td>FIAT ***+1</td>
<td></td>
</tr>
<tr>
<td>00A5</td>
<td>COUNT</td>
<td>;TEMP USED BY SERIAL ROUTINE</td>
</tr>
<tr>
<td>00A5</td>
<td>CNTDN ***+1</td>
<td>;CASSETTE SYNC COUNTDOWN</td>
</tr>
<tr>
<td>00A6</td>
<td>BUFPT ***+1</td>
<td>;CASSETTE BUFFER POINTER</td>
</tr>
<tr>
<td>00A7</td>
<td>INBIT</td>
<td>;RS-232 RCVR INPUT BIT STORAGE</td>
</tr>
<tr>
<td>00A7</td>
<td>SHCNL ***+1</td>
<td>;CASSETTE SHORT COUNT</td>
</tr>
<tr>
<td>00A8</td>
<td>BICL</td>
<td>;RS-232 RCVR BIT COUNT IN</td>
</tr>
<tr>
<td>00A8</td>
<td>KER ***+1</td>
<td>;CASSETTE READ ERROR</td>
</tr>
<tr>
<td>00A9</td>
<td>RINONE</td>
<td>;RS-232 RCVR FLAG FOR START BIT CHECK</td>
</tr>
<tr>
<td>00A9</td>
<td>REZ ***+1</td>
<td>;CASSETTE READING ZEROS</td>
</tr>
<tr>
<td>00AA</td>
<td>RIDATA</td>
<td>;RS-232 RCVR BYTE BUFFER</td>
</tr>
<tr>
<td>00AA</td>
<td>RFLD ***+1</td>
<td>;CASSETTE READ MODE</td>
</tr>
<tr>
<td>00AB</td>
<td>RIPRTY</td>
<td>;RS-232 RCVR PARITY STORAGE</td>
</tr>
<tr>
<td>00AC</td>
<td>SHCNH ***+1</td>
<td>;CASSETTE SHORT CNT</td>
</tr>
<tr>
<td>00AD</td>
<td>SAL ***+1</td>
<td></td>
</tr>
<tr>
<td>00AE</td>
<td>SAH ***+1</td>
<td></td>
</tr>
<tr>
<td>00AF</td>
<td>EAL ***+1</td>
<td></td>
</tr>
<tr>
<td>00B0</td>
<td>EAH ***+1</td>
<td></td>
</tr>
<tr>
<td>00B0</td>
<td>CMPD ***+1</td>
<td></td>
</tr>
<tr>
<td>00B1</td>
<td>TEMP ***+1</td>
<td></td>
</tr>
<tr>
<td>00B2</td>
<td>TAPE1 ***+2</td>
<td>;ADDRESS OF TAPE BUFFER</td>
</tr>
<tr>
<td>00B4</td>
<td>BITTS</td>
<td>;RS-232 TRNS BIT COUNT</td>
</tr>
<tr>
<td>00B4</td>
<td>SNR1 ***+1</td>
<td></td>
</tr>
</tbody>
</table>
Serial Communications

```assembly
; Variables for Screen Editor

00B5   LOC_CODE  ; RS-232 TRNS NEXT BIT TO BE SENT
00B5   NXTBIT   
00B6   DIFF     ; RS-232 TRNS BYTE BUFFER
00B6   RODATA   
00B7   PRP      ; LENGTH CURRENT FILE N STR
00B7   FNLEN    
00B8   SA       ; CURRENT FILE LOGICAL ADDR
00B8   LA       
00B9   FA       ; CURRENT FILE 2ND ADDR
00B9   CURRENT_FILE.Primary.Addr
00BA   FNADR    ; CURRENT FILE PRIMARY ADDR
00BA   ADDR_CURRENT_FILE.NAME.Str
00BB   ROPRTY   ; RS-232 TRNS PARITY BUFFER
00BD   OCHAR    ; CASSETTE READ BLOCK COUNT
00BE   FSBLK    ; CASSETTE MANUAL/CONTROLLED SWITCH
00BF   MYCH     ; CASSETTE LOAD TEMPS (2 BYTES)
00C0   CANI     ; VARIABLES FOR SCREEN EDITOR
00C1   TAPO     
00C1   STAL     
00C2   STAH     
00C3   MEMUSS   ; KEY SCAN INDEX
00C3   TMP2     
00C5   ; INDEX TO KEYBOARD Q
00C5   ; RVS FIELD ON FLAG
00C6   LSTX     ; CHAR BEFORE CURSOR
00C6   NDX      ; ON/OFF LINK FLAG
00C7   RVS      ; POINTER TO ROW
00C8   INDEX    ; CURSOR BLINK ENAB
00C8   ; COUNT TO TURGLE CUR
00C9   LSXP     ; CHAR BEFORE CURSOR
00CA   LSTP     ; CURSOR BLINK ENAB
00CB   SFDX     ; POINTER TO COLUMN
00CC   BLNSW    ; QUOTE SWITCH
00CD   BLNCT    ; CURSOR BLINK ENAB
00CE   GDBLN    ; COUNT TO TURGLE CUR
00CF   BLNON    ; CHAR BEFORE CURSOR
00D0   CSW      ; ON/OFF BLINK FLAG
00D1   FNT      ; INPUT VS GET FLAG
00D3   FNTR     ; INPUT VS GET FLAG
00D4   FNTR     ; INPUT VS GET FLAG
00D5   TSW      ; INPUT VS GET FLAG
00D6   LDMX     ; INPUT VS GET FLAG
00D7   TBLX     ; INPUT VS GET FLAG
00D8   DATA     ; INPUT VS GET FLAG
00D9   INSR1    ; KEY SCAN INDEX
00D9   LDTB1    ; KEY SCAN INDEX
00D9   ; 40/80 LINE FLAGS
00DF   LIMITMP  ; 40/80 LINE FLAGS
00E3   USER     ; TEMPORARY FOR LINE INDEX
00F5   KEYTAB   ; SCREEN EDITOR COLOUR IF
00F7   ; KEYSCAN TABLE INDIRECT
00F7   ; RS-232 Z-FAGE
00F7   ;
00F7   KIBUF    ; RS-232 INPUT BUFFER POINTER
00F7   ROBUF    ; RS-232 INPUT BUFFER POINTER
00F8   FREQZP   ; FREE KERNEL ZERO PAGE
00FF   BASZPT   ; LOCATION ($00FF) USED BY BASIC
0100   ;
0100   x=100    ;
0101   x=$200   ;
0200   BUF      ; BASIC/MONITOR BUFFER
0259   ; TABLES FOR OPEN FILES
0259   ;
0259   LAT      ; LOGICAL FILE NUMBERS
0259   ; PRIMARY DEVICE NUMBERS
026D   SAT      ; SECONDARY ADDRESSES
0277   ; SYSTEM STORAGE
0277   KEYD     ; IRQ KEYBOARD BUFFER
0281   MEMSTR   ; START OF MEMORY
```
MEMSIZ $***+2 ;TOP OF MEMORY
TIMOUI $***+1 ;IEEE TIMEOUT FLAG
;
;SCREEN EDITOR STORAGE
;
COLOR $***+1 ;ACTIVE COLOUR NIBBLE
GOCOL $***+1 ;ORIGINAL COLOUR BEFORE CURSOR
HIBASE $***+1 ;BASE LOCATION OF SCREEN (TOP)
XMAX $***+1
RPTFLG $***+1 ;KEY REPEAT FLAG
KOUNT $***+1
DELAY $***+1
;
;RS-232 STORAGE
;
M26CTR $***+1 ;6526 CONTROL REGISTER
M26CDR $***+1 ;6526 COMMAND REGISTER
M26AJB $***+2 ;NON STANDARD (BITTIME/2-100)
RSSIAT $***+1 ;RS-232 STATUS REGISTER
BUINUM $***+1 ;NUMBER OF BITS TO SEND (FAST RESPONSE)
BAUDOFF $***+2 ;BAUD RATE FULL BIT TIME
;
;RECEIVER STORAGE
;
RIDBE $***+1 ;INPUT BUFFER INDEX TO END
RIDBS $***+1 ;INPUT BUFFER POINTER TO START
;
;TRANSMITTER STORAGE
;
ROUBS $***+1 ;OUTPUT BUFFER INDEX TO START
ROUDE $***+1 ;OUTPUT BUFFER INDEX TO END
;
IRQTMP $***+2 ;HOLDS IRQ DURING TAPE OPS
;
;
;PROGRAM INDIRCETS(10)
==$0300 ;KERNAL/OS INDIRCETS (20)
CINV $***+2 ;IRQ RAM VECTOR
CBINV $***+2 ;BRK INSIR RAM VECTOR
NMINV $***+2 ;MMI RAM VECTOR
IOFEN $***+2 ;INDIRECTS FOR CODE
ICLOSE $***+2 ;CONFORMS TO KERNAL SPEC 8/19/80
ICHKN $***+2
ICKOUT $***+2
ILRCH $***+2
IBASIN $***+2
IBSOUT $***+2
ISTUP $***+2
IUTIN $***+2
ICLAL $***+2
USKCMD $***+2
ILOAD $***+2
ISAVE $***+2


שכר $0300+60 ;SAVESEP
CBUF $***+192 ;CASSETTE DATA BUFFER
CFC $***+192
CFC $***+480
CBMSN $***+999 ;64 SCREEN

;RAMLOC

<table>
<thead>
<tr>
<th>LOC CODE</th>
<th>LINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>*=D000</td>
</tr>
<tr>
<td>D000</td>
<td>VICREG *=+47</td>
</tr>
<tr>
<td>D02F</td>
<td>;VIC REGISTERS</td>
</tr>
<tr>
<td>D02H</td>
<td>*=D400</td>
</tr>
<tr>
<td>D400</td>
<td>SIDREG *=+29</td>
</tr>
<tr>
<td>D410</td>
<td>;SID REGISTERS</td>
</tr>
<tr>
<td>D410</td>
<td>*=D800</td>
</tr>
<tr>
<td>D800</td>
<td>CRMCOL *=+99Y</td>
</tr>
<tr>
<td>DBE7</td>
<td>;64 COLOUR NIBBLES</td>
</tr>
<tr>
<td>DBE7</td>
<td>;I/O DEVICES</td>
</tr>
<tr>
<td>DBE7</td>
<td>*=DC00</td>
</tr>
<tr>
<td>DC00</td>
<td>COLM ;KEYBOARD MATRIX</td>
</tr>
<tr>
<td>DC00</td>
<td>D10FA *=+1</td>
</tr>
<tr>
<td>DC01</td>
<td>ROWS ;KEYBOARD MATRIX</td>
</tr>
<tr>
<td>DC01</td>
<td>D10FB *=+1</td>
</tr>
<tr>
<td>DC02</td>
<td>D10DRA *=+1</td>
</tr>
<tr>
<td>DC03</td>
<td>D10DRB *=+1</td>
</tr>
<tr>
<td>DC04</td>
<td>D11AL *=+1</td>
</tr>
<tr>
<td>DC05</td>
<td>D11AH *=+1</td>
</tr>
<tr>
<td>DC06</td>
<td>D11BL *=+1</td>
</tr>
<tr>
<td>DC07</td>
<td>D11BH *=+1</td>
</tr>
<tr>
<td>DC08</td>
<td>D1T0D1 *=+1</td>
</tr>
<tr>
<td>DC09</td>
<td>D1T0D2 *=+1</td>
</tr>
<tr>
<td>DC0A</td>
<td>D1T0D3 *=+1</td>
</tr>
<tr>
<td>DC0B</td>
<td>D1T0D4 *=+1</td>
</tr>
<tr>
<td>DC0C</td>
<td>D1I0DB *=+1</td>
</tr>
<tr>
<td>DC0D</td>
<td>D1ICR *=+1</td>
</tr>
<tr>
<td>DC0E</td>
<td>D1ICKA *=+1</td>
</tr>
<tr>
<td>DC0F</td>
<td>D1ICRB *=+1</td>
</tr>
<tr>
<td>DC10</td>
<td>*=D200</td>
</tr>
<tr>
<td>D200</td>
<td>6526 (NMI)</td>
</tr>
<tr>
<td>D201</td>
<td>D2DF *=+1</td>
</tr>
<tr>
<td>D202</td>
<td>D2DDRA *=+1</td>
</tr>
<tr>
<td>D203</td>
<td>D2DDRB *=+1</td>
</tr>
<tr>
<td>D204</td>
<td>D2T0AL *=+1</td>
</tr>
<tr>
<td>D205</td>
<td>D2T0AH *=+1</td>
</tr>
<tr>
<td>D206</td>
<td>D2T0BL *=+1</td>
</tr>
<tr>
<td>D207</td>
<td>D2T0BH *=+1</td>
</tr>
<tr>
<td>D208</td>
<td>D2T0D1 *=+1</td>
</tr>
<tr>
<td>D209</td>
<td>D2T0D2 *=+1</td>
</tr>
<tr>
<td>D20A</td>
<td>D2T0D3 *=+1</td>
</tr>
<tr>
<td>D20B</td>
<td>D2T0D4 *=+1</td>
</tr>
<tr>
<td>D20C</td>
<td>D2I0DB *=+1</td>
</tr>
<tr>
<td>D20D</td>
<td>D2ICR *=+1</td>
</tr>
<tr>
<td>D20E</td>
<td>D2ICRA *=+1</td>
</tr>
<tr>
<td>D20F</td>
<td>D2ICRB *=+1</td>
</tr>
<tr>
<td>D210</td>
<td>;TAPE BLOCK TYPES</td>
</tr>
<tr>
<td>D210</td>
<td>;END OF TAPE</td>
</tr>
<tr>
<td>D210</td>
<td>EOT *=5</td>
</tr>
<tr>
<td>D210</td>
<td>BLF =1 ;BASIC LOAD FILE</td>
</tr>
<tr>
<td>D210</td>
<td>BDF =2 ;BASIC DATA FILE</td>
</tr>
<tr>
<td>D210</td>
<td>PLF =3 ;FIXED PROGRAM TYPE</td>
</tr>
<tr>
<td>D210</td>
<td>BDFH =4 ;BASIC DATA FILE HEADER</td>
</tr>
<tr>
<td>D210</td>
<td>BUFSIZE =192 ;BUFFER SIZE</td>
</tr>
<tr>
<td>D210</td>
<td>;TAPE ERROR TYPES</td>
</tr>
<tr>
<td>D210</td>
<td>SFERR =16</td>
</tr>
<tr>
<td>D210</td>
<td>CKERR =32</td>
</tr>
<tr>
<td>D210</td>
<td>SBERR =4</td>
</tr>
<tr>
<td>D210</td>
<td>LBERR =8</td>
</tr>
<tr>
<td>D210</td>
<td>;SCREEN EDITOR CONSTANTS</td>
</tr>
</tbody>
</table>
| D210     | ;
42 The Commodore 64 Kernal and Hardware Revealed

LOC CODE LINE

DD10 DD10 DD10 DD10 DD10 DD10 DD10 DD10 DD10 DD10
LLEN =40 LLEN2 =90 NUNES =2~j BLUE =6 LTF.LUE =14 CR =$0 MAXCHR =80 N~JRAF' .END

TALK

Entry point: $FFB4

Function: Command serial device to talk (transmit data)

Input parameters: .A device number

Output parameters: None

Registers used: .A

Error messages:
Device not present (returned in STatus var. $90) – attention not acknowledged by data low within 1 ms
Frame error (in ST) – no data accepted response (data low) within 1 ms of last bit of byte being sent.

Description: This routine ORs the device number in the .A register with $40. Before the command is sent the single character serial buffer is checked for being empty. If it is not the character in it is sent (with end message marker (EOI)). After this the attention line is set low (bit 3 set in chip 2 port A (assembler label D2DPA in listing). Then the command byte is sent with the attention line held low.

LOC CODE LINE

DD10 DD09 DD09 DD09 DD09 DD09 DD09 DD09 DD09 DD09
0010 009 009 009 009 009 009 009 009 009
LISTEN

Entry point: $FFB1
Function: Command serial device to listen

Input parameters: .A device number

Output parameters: None

Registers used: .A

Error messages:
Device not present (returned in STatus var. $90) – attention not acknowledged by data low within 1 ms
Frame error (in ST) – no data accepted response (data low) within 1 ms of last bit of byte being sent

Description: This routine ORs the device number in the .A register with $20. Before the command is sent the single character serial buffer is checked for being empty. If it is not the character in it is sent with an EOI handshake to mark it as the last byte of its message. After this the attention line is set low. Then the command byte is sent with the attention line held low. This routine includes the main routine to send a byte to the serial bus. This is done as follows:

L842  –  Set clock low
         Set data high
         Delay 1 ms
L859  –  Set data high (released)
         Set clock line high
         If EOI no handshake required then L850
          ; Wait with clock high for End Or Identify handshake
         Wait for data high
         Wait for data low
          ; That is end of hold, until serial device is ready
L850  –  Wait for data high
         Set clock low
         Put 8 in counter
L848  –  If data not high framing error
         Get next bit of byte to send
          ; Low bit first
         If bit is zero then set data low
         Set clock high ; flag bit
         Sort pause
         Set data high and clock low
         Decrease bit counter
         Go to L848 if not all sent
         Set timer for 1 ms
L855  –  Has timer expired?
         If so then L847 (framing error)
         If data not low go back to L855
         Exit
The Commodore 64 Kernal and Hardware Revealed

LOC CODE LINE

ED0C 09 20 L966 URA M$20 ;MAKE ADDR LISTEN
ED0E 20 A4 F0 JSR $F0A4 ;PROTECT FROM RS323 NMI
ED11 48 L980 PHA
ED12 24 94 BIT C3P0 ;CHAR IN BUFFER?
ED14 10 0A BPL L864 ;NO
ED16 1 ;SEND BUFFERED CHAR
ED16 38 SEC ;SET EOI FLAG
ED17 66 A3 ROR R202
ED19 20 40 ED JSR L859 ;SEND LAST CHAR
ED1C 46 94 LSR C3P0 ;BUFFER CLEAR
ED1E 46 A3 LSR R202 ;CLEAR EOI FLAG
ED20 68 L864 PLA ;TALK/LISTEN ADDR
ED21 05 95 STA BSOUR
ED23 78 ;SEI
ED24 20 97 EE JSR L844
ED27 C9 3F CMP #03F
ED29 D0 03 BNE L839
ED2B 20 85 EE JSR L875
ED2E AD 00 DD L839 LDA D2DFA ;ASSERT ATTENTION
ED31 09 0B ORA M$08
ED33 BD 00 DD STA D2DFA
ED36 1 ;
ED36 78 L842 SEI ;SET CLOCK LINE LOW
ED37 20 8E EE JSR L843
ED3A 20 97 EE JSR L844
ED3D 20 B3 EE JSR L846 ;DELAY 1 MS
ED40 78 L859 SEI ;DISABLE IRQ
ED41 20 97 EE JSR L844 ;MAKE SURE DATA IS RELEASED
ED44 20 A9 EE JSR L854 ;DATA SHOULD BE LOW
ED47 80 64 BCS L856
ED49 20 85 EE JSR L875 ;CLOCK LINE HI
ED4C 24 A3 BIT R202 ;EOI FLAG TEST
ED4E 10 0A BPL L850
ED50 1 ;DO EOI
ED50 20 A9 EE L840 JSR L854 ;WAIT FOR DATA HI
ED53 90 FB BCC L840
ED55 20 A9 EE L849 JSR L854 ;WAIT FOR DATA LO
ED58 80 FB BCS L849
ED5A 20 A9 EE L850 JSR L854 ;WAIT FOR DATA HI
ED5D 90 FB BCC L850
ED5F 20 8E EE JSR L843 ;SET CLOCK LU
ED62 1 ;
ED62 ;SET TO SEND DATA
ED62 1 ;
ED62 10 08 STA COUNT ;COUNT 8 BITS
ED64 85 A5 LDA M$08 ;COUNT BUS
ED66 20 00 DD L848 LDA D2DFA ;DEBOUNCE BUS
ED69 CD 00 DD CMP D2DFA
ED6C D0 F8 BNE L848
ED6E 0A ASL A
ED6F 90 3F BCC L847 ;DATA MUST BE HI
ED71 66 95 ROR BSOUR ;NEXT BIT INTO CARRY
ED73 80 05 BCS L851
ED75 20 A0 EE JSR L841
ED77 12 03 LDA M$08
ED79 80 05 BCS L851
ED7F 20 97 EE L851 JSR L844
ED87 20 85 EE L853 JSR L875 ;CLOCK HI
ED8D EA NOP
ED8E EA NOP
ED8F EA NOP

0920 URA M$20 ;MAKE ADDR LISTEN
JSR $F0A4 ;PROTECT FROM RS323 NMI
PHA
BIT C3P0 ;CHAR IN BUFFER?
BPL L864 ;NO
;SEND BUFFERED CHAR
SEC ;SET EOI FLAG
ROR R202
JSR L859 ;SEND LAST CHAR
LSR C3P0 ;BUFFER CLEAR
LSR R202 ;CLEAR EOI FLAG
L864 PLA ;TALK/LISTEN ADDR
STA BSOUR
SEI
JSR L844
CMP #03F
BNE L839
JSR L875
L839 LDA D2DFA ;ASSERT ATTENTION
ORA M$08
STA D2DFA
;SET CLOCK LINE LOW
JSR L843
JSR L844
JSR L846 ;DELAY 1 MS
L859 SEI ;DISABLE IRQ
JSR L844 ;MAKE SURE DATA IS RELEASED
JSR L854 ;DATA SHOULD BE LOW
BCS L856
JSR L875 ;CLOCK LINE HI
BIT R202 ;EOI FLAG TEST
BPL L850
1 ;DO EOI
L840 JSR L854 ;WAIT FOR DATA HI
BCC L840
L849 JSR L854 ;WAIT FOR DATA LO
BCS L849
L850 JSR L854 ;WAIT FOR DATA HI
BCC L850
JSR L843 ;SET CLOCK LU
;SET TO SEND DATA
LDA M$08 ;COUNT 8 BITS
L848 LDA D2DFA ;DEBOUNCE BUS
CMP D2DFA
BNE L848
ASL A
BCC L847 ;DATA MUST BE HI
ROR BSOUR ;NEXT BIT INTO CARRY
BCS L851
JSR L841
BNE L851
JSR L844
JSR L875 ;CLOCK HI
NOP
NOP
NOP
Entry point: $FF93

Function: Send secondary address after listen

Input parameters: Secondary address in .A register ORed with $60

Output parameters: None

Registers used: .A

Error messages:
Device not present (returned in STatus var. $90) – attention not acknowledged by data low within 1 ms
Frame error (in ST) – no data accepted response (data low) within 1 ms of last bit of byte being sent

Description: The secondary address is stored in the serial buffer and then sent to listening devices. Next the attention line is released (set high).
The Commodore 64 Kernal and Hardware Revealed

Entry point: $FF96

Function: Send secondary address after talk

Input parameters: Secondary address in .A register

Output parameters: None

Registers used: .A

Error messages:
Device not present (returned in STatus var. $90) – attention not acknowledged by data low within 1 ms
Frame error (in ST) – no data accepted response (data low) within 1 ms of last bit of byte being sent

Description: The secondary address is loaded into the serial buffer and then sent to the serial bus. This routine then waits for the new talking device to acknowledge it is the new talker by changing the clock line. This is done as follows:

Hold data low
Set attention high (release)
Set clock high
Then wait for clock to go low

```
EDC7 85 95 L860 STA BSOUR ;BUFFER CHAR
EDC7 20 36 ED JSR L842 ;SEND SA
EDC7 78 L970 SEI ;DISABLE IRQ
EDCD 20 A0 EE JSR L841 ;DATA LINE LO
EDDO 20 B0 ED JSR L983:
EDD3 20 85 EE JSR L875 ;CLOCK LINE HI
EDD6 20 A9 EE L960 JSR L854 ;WAIT FOR CLOCK LO
EDD9 30 FB BMI L960
EDDB 58 CLI ;DONE
EDDC 60 RTS
```
**CIOUT**

Entry point: $FFA8

Function: Send byte to serial bus

Input parameters: Byte to send in .A

Output parameters: None

Registers used: .A

Error messages:
Device not present (returned in STatus var. $90) – attention not acknowledged by data low within 1 ms
Frame error (in ST) – no data accepted response (data low ) within 1 ms of last bit of byte being sent

Description: Any character in the serial buffer is sent to the serial port. Then the current character is stored in the buffer.

```
EDDD
EDDD
EDDD
EDDD 24 94 L861 BIT C3F0 ;BUFFERED CHAR?
EDDF 30 05 BMI L947 ;YES, SEND LAST
EDE1 39 SEC ;NO
EDE2 66 94 ROR C3F0 ;SET BUFFERED CHAR FLAG
EDE4 D0 05 BNE L862 ;ALWAYS
EDE6 48 L949 PHA ;SAVE CURRENT CHAR
EDE7 20 40 ED JSR L859 ;SEND LAST CHAR
EDE9 68 PLA ;RESTORE CURRENT
EDEB 85 95 L862 STA BSOUR ;BUFFER IT
EDED 18 CLC ;GOOD EXIT
EDEE 60 RTS
```

**UNTLK**

Entry point: $FFAB

Function: Send command UNTALK

Input parameters: None

Output parameters: None

Registers used: .A

Error messages:
Device not present (returned in STatus var. $90) – attention not acknowledged by data low within 1 ms
Frame error (in ST) – no data accepted response (data low) within 1 ms of last bit of byte being sent

**Description:** This routine sends the $5F under attention to serial bus. This tells the current talking to stop. After a delay this routine ends by releasing clock and data lines.

**LOC CODE LINE**

```
EDEF  ;******************************
EDEF  ;SEND UNTALK.
EDEF  ;THIS ROUTINE SENDS AN 'UNTALK' TO THE SERIAL
EDEF  ;BUS. IT WILL TELL ALL DEVICES IN TALK
EDEF  ;MODE TO STOP TALKING (SENDING DATA).
EDEF  ;******************************
EDEF  7B  L863 SEI
EDEF  20 8E EE  JSR L843
EDEF  AD 00 DD  LDA D2DF  ;PULL ATN
EDEF  09 00     ORA #$00
EDEF  BD 00 DD  STA D2DF
EDEF  A$ 5F     LDA #$5F  ;UNTALK
EDEF  2C  ;BYTE $2C  ;SKIP NEXT COMMAND
```

**UNLSN**

**Entry point:** $FFAE

**Function:** Send command UNLISTEN

**Input parameters:** None

**Output parameters:** None

**Registers used:** .A

**Error messages:**
Device not present (returned in STatus var. $90) – attention not acknowledged by data low within 1 ms
Frame error (in ST) – no data accepted response (data low ) within 1 ms of last bit of byte being sent

**Description:** This routine sends the $3F under attention to serial bus. This tells the current listening devices to stop. After a delay this routine ends by releasing clock and data lines.

**LOC CODE LINE**

```
EDEF  ;******************************
EDEF  ;SEND UNLISTEN.
EDEF  ;THIS ROUTINE SENDS AN 'UNLISTEN' TO THE SERIAL
EDEF  ;BUS. IT WILL TELL ALL DEVICES IN LISTEN MODE TO STOP LISTENING.
EDEF  ;******************************
EDEF  A$ 3F  L1006 LDA #$3F  ;UNLISTEN COMMAND
EE00  20 11 ED  JSR L900  ;SEND
EE03  ;RELEASE ALL LINES
EE03  ;RELEASE ATN
EE06  ;
```
Entry point: $FFA5

Function: Input byte from serial port

Input parameters: None

Output parameters: Character in .A

 Registers used: .A

Error messages:
Read timeout (in ST) – no clock low response within 0.2 ms of data being released

Description: This routine gets a byte from serial bus and returns it in the .A register. It does this as follows:

L865 - Zero COUNT
Release clock line
Wait for clock to go high

L866 - Set timer to 256 ms
Release data

L872 - If timer expired go to L868
If clock still high go back to L872
Otherwise go to L870, to read byte

L868 - If COUNT non zero flag read timeout in ST and exit via a
routine to release lines
Otherwise assume EOI
; Handshake EOI
Set data low
Pause and release data
Flag EOI in ST
Increase COUNT
Go back to L866, to wait for clock
; Get a byte

L870 - Set COUNT for 8 bits
L869 - Wait for clock to go high
Get next bit of byte from data line
Wait for clock to go low
Decrease COUNT
If COUNT not zero go back to L869 to get next bit
Acknowledge byte by sending data low
Check EOI flag in ST
; EOI flags end of message
If set then delay and release data
Exit this byte read in .A

LOC CODE LINE

EE13 78 L865 SEI ;DISABLE IRQ
EE14 A9 00 STA COUNT
EE15 A9 05 JSR L875 ;RELEASE CLOCK LINE
EE18 20 09 EE L943 JSR L854 ;WAIT FOR CLOCK HI
EE1E 10 FB BPL L943
EE20 A9 01 L866 LDA #01 ;SET TIMER B FOR 256 U9
EE22 00 07 STA D17B
EE25 A9 19 LDA #19
EE27 00 0F STA D1CRB
EE2A 20 97 EE JSR L844
EE2D A0 00 DC LDA D1CR
EE30 A0 00 DC L872 LDA D1CR
EE33 29 02 AND #02 ;CHECK THE TIMER
EE35 D0 07 BNE L868 ;RAN OUT
EE37 20 A9 EE JSR L854 ;CHECK THE CLOCK LINE
EE39 30 F4 BMI L872 ;NOT YET
EE3C 10 18 BPL L870 ;YES
EE3E ;
EE3E A5 A5 L868 LDA COUNT ;CHECK FOR ERROR
EE40 F0 05 BEQ L867
EE42 A9 02 LDA #02
EE44 4C B2 ED JMP L852 ;ST=2, READ TIME OUT
EE47 ;
EE47 20 A0 EE L867 JSR L841 ;DATA LINE LO
EE4A 20 05 EE JSR L875 ;DELAY, SET DATA HI
EE4D A9 40 LDA #40
EE4F 20 1C EE JSR $FE1C ;OR AN EOI BIT INTO ST
EE52 E6 A5 INC COUNT AND AGAIN FOR ERROR CHECK
EE54 D0 CA BNE L866
EE56 ;
EE56 A9 08 L870 LDA #08 ;SET UP COUNTER
EE58 B5 A5 STA COUNT
EE5A 20 00 DD L869 LDA D20PA ;WAIT FOR CLOCK HI
EE5D CD 00 DD CMP D20PA ;DEBOUNCE
EE60 D0 F8 BNE L869
EE62 0A ASL A
EE63 10 F5 BPL L869
EE65 66 A4 ROR BSOUR1 ;ROTATE DATA IN
EE67 00 00 DD L873 LDA D20PA ;WAIT FOR CLOCK LO
EE6A CD 00 DD CMP D20PA ;DEBOUNCE
EE6D D0 F8 BNE L873
EE6F 0A ASL A
EE70 30 F5 BMI L873
EE72 C6 A5 DCC COUNT
EE74 D0 E4 BNE L873 ;MORE BITS
EE76 20 A0 EE JSR L841 ;DATA LO
3.3 General routines

All routines change only the .A register.

Set clock high, set clock low, set data high & set data low all just set or unset a bit in port A of CIA chip 2. Note that the bit is set to send a line low.

Debounce CIA routine first loops until a consistent value is read from port A of the CIA. It then sets the carry flag to the state of the data line, and the sign flag to the state of the clock line.

The 1 millisecond delay is a software delay loop lasting approx 1 ms.

```
EEK5  ;SET CLOCK LINE HI (INVERTED)
EEK5  ;
EEK5 AD 00 DD L875 LDA D2DFA
EEK8  29 EF AND $8EF
EEK9  60 RTS
EEKB  ;SET CLOCK LINE LO (INVERTED)
EEKB  ;
EEKB AD 00 DD L843 LDA D2DFA
EEK1  09 10 ORA $810
EEK3  60 RTS
EEK6  ;SET DATA LINE HI (INVERTED)
EEK6  ;
EEK6 AD 00 DD L844 LDA D2DFA
EEK9A  29 DF AND $8DF
EEK9B  60 RTS
EEKA0  ;SET DATA LINE LO (INVERTED)
EEKA0  ;
EEKA0 AD 00 DD L841 LDA D2DFA
EEK3  09 20 ORA $820
EEK5  60 RTS
EEKB  ;DEBOUNCE THE PIA
EEKB  ;
EEKB AD 00 DD L854 LDA D2DFA
EEKC  60 RTS
EEKD  ;DELAY 1 MS
EEKD  ;
EEKB  8A L846 TXA
```
52 The Commodore 64 Kernal and Hardware Revealed

```
EEB4 A2 B0     LDX #B8
EEB6 CA       LB45  DEX
EEB7 D0 FD    BNE LB45
EEB9 AA      TAX
EEB0 A0       RTS
EEB1 .END     .LIB KSER2
```

GETIN

Entry point: $FFE4

Function: Get a character from the current input device

Input parameters: None

Output parameters: .A holds character, CARRY clear

Registers used: .A

Error messages: .A

Description: For serial devices, GETIN is redirected to BASIN.

```
LOC  CODE       LINE

 EEBB        *=F13E
 F13E ;
 F13E ;**************GETIN -- GET CHARACTER FROM CHANNEL.
 F13E ;* CHANNEL IS DETERMINED BY DFLTN.
 F13E ;* IF DEVICE IS 0, KEYBOARD QUEUE IS
 F13E ;* EXAMINED AND A CHARACTER REMOVED IF
 F13E ;* AVAILABLE. IF QUEUE IS EMPTY, Z
 F13E ;* FLAG IS RETURNED SET. DEVICES 1-31
 F13E ;* RETURNED IN .A. IF ZERO, NULL CHAR.
 F13E ;**************
 F13E ;
 F13E A5 99   NGETIN LDA DFLTN  ;CHECK DEVICE
 F140 D0 08   BNE L924  ;NOT KEYBOARD
 F142 A5 C6   LDA NDX  ;QUEUE INDEX
 F144 F0 0F   BEQ L944  ;NOTHING THERE, EXIT
 F146 78     SEI
 F147 4C B4 E5   JMP #$E5B4  ;REMOVE A CHAR
 F14A C9 02   L924   CMP #02  ;RS-232?
 F14C D0 18   BNE L927  ;NO, USE BASIN
 F14E 84 97   L926   STY XSAVE  ;SAVE .Y, USED IN RS-232
 F150 20 B6 F0   JSR #$066
 F153 A4 97   L944   CLC  ;RESTORE .Y
 F155 18     CLC  ;GOOD RETURN
 F156 60     RTS
```

BASIN

Entry point: $FFCF
**Function:** Get a character from the current input device

**Input parameters:** None

**Output parameters:** .A holds character, CARRY clear

**Registers used:** .A

**Error messages:** None

**Description:** If the status from the last character read was ≠Ø (EOF), the character 13 (carriage return) is returned with CARRY clear. Otherwise one byte is read using the ACPTTR routine.

```assembly
LOC CODE LINE
F157   A5 99 NBASIN LDA DFLTN ;CHECK DEVICE
F157   D0 08 BNE L927 ;NOT KEYBOARD
F157   B5 03 ;INPUT FROM KEYBOARD
F158   A5 D3 LDA FNTR ;SAVE CURRENT:
F158   85 CA STA LSTP ;CURSOR COLUMN,
F158   A5 D6 LDA TBLX
F158   85 C9 STA LSXP ;LINE NUMBER
F163  4C 32 E6 JMP $E632 ;BLINK CURSOR UNTIL RETURN
F166  C9 03 L927 CMP $#03 ;SCREEN?
F168  D0 09 BNE L928 ;NO
F16A  85 D0 STA CRSW ;FAKE CARRIAGE RETURN
F16C  A5 D5 LDA LNMX ;ENDED;
F16E  85 CB STA INDEX ;ON THIS LINE
F170  4C 32 E6 JMP $E632 ;PICK UP CHARACTERS
F173  ;
F173  80 38 L928 BCS L939 ;DEVICES>3
F175  C9 02 CMP $#02 ;RS-232?
F177  F0 3F BEQ $F188 ;YES
F179  ;----------
F179  #=$F1AD
F1AD  ;
F1AD  ;INPUT FROM SERIAL BUS
F1AD  ;
F1AD  A5 90 L939 LDA STATUS ;STATUS FROM LAST
F1AF  F0 04 BEQ L941 ;O.K.
F1B1  A9 00 L932 LDA $#0D ;BAD, ALL DONE
F1B3  18 L946 CLC ;VALID DATA
F1B4  60 L945 RTS
F1B5  ;
F1B5  4C 13 EE L941 JMP L865 ;GOOD, HANDSHAKE
F1B8  ;----------
```
**BSOUT**

**Entry point:** $FFD2

**Function:** Output the character stored in .A to the current output device.

**Input parameters:** .A holds character

**Output parameters:** .A holds same character, CARRY clear

**Registers used:** None

**Error messages:** None

**Description:** This routine just jumps to the send buffered character to serial routine.

```
LOC  CODE  LINE

F1BB  *=F1CA
F1CA
F1CA ;************************************************************************
F1CA ;* BSOUT -- OUTPUT CHAR STORED IN .A TO
F1CA ;* CHANNEL DETERMINED BY VARIABLE DFLTO:
F1CA ;* 0 --- INVALID
F1CA ;* 1 --- CASSETTE
F1CA ;* 2 --- RS-232
F1CA ;* 3 --- SCREEN
F1CA ;* 4-31 --- SERIAL BUS
F1CA ;************************************************************************
F1CA
F1CA 48  NBSOUT PHA ;RESERVE .A
F1CB A5 9A  LDA DFLTO ;CHECK DEVICE
F1CD C9 03  CMP #$03  ;SCREEN?
F1CF D0 04  BNE L933 ;NO
F1D1 68  PLA ;YES, RESTORE .A
F1D2 4C 16 E7  JMP #$E716 ;PRINT TO SCREEN
F1D5
F1D5 90 04  L933  BCC #$1DB  ;DEVICE 1 OR 2
F1D7 ;
F1D7 ;PRINT TO SERIAL BUS
F1D7 ;
F1D7 68  PLA
F1DB 4C 0D ED  JMP L061
F1DB
F1DB ;-------------------------
F1DB
F1DB .LIB KSER3
```

**CHKIN**

**Entry point:** $FFC6

**Function:** Set a previously OPENed file for input.

**Input parameters:** .X holds the logical file number of the OPENed file.
Output parameters:
CARRY clear – OK
CARRY set – error, error number in .A

Registers used: .A, .X

Error messages:
File not open – if the logical file number in .X is not in the LFN table
Device not present – if bit 7 of ST is set, the device did not respond to the TALK command

Description: This routine first checks that the LFN in .X has a reference in the LFN table. If not, the message File not open is sent. The device referenced by the LFN is told to TALK and a secondary address is sent (if present). After sending the TALK secondary address, the device is shifted over to listener. If bit 7 of the STATUS byte (ST) is set, the message Device not present is sent.

LOC CODE LINE

F10B  *=F20E
F20E  ;
F20E  ;******************************************************************************
F20E  ;* CHKIN -- OPEN CHANNEL FOR INPUT.
F20E  ;* THE NUMBER OF THE LOGICAL FILE TO
F20E  ;* BE OPENED FOR INPUT IS PASSED IN .X.
F20E  ;* CHKIN SEARCHES THE LOGICAL FILE TO
F20E  ;* LOOK UP DEVICE AND COMMAND INFO.
F20E  ;* ERRORS ARE REPORTED IF THE DEVICE WAS
F20E  ;* NOT OPENED FOR INPUT, (E.G. CASSETTE
F20E  ;* WRITE FILE), OR THE LOGICAL FILE HAS
F20E  ;* NO REFERENCE IN THE TABLES. DEVICE 0,
F20E  ;* (KEYBOARD), AND DEVICE 3 (SCREEN),
F20E  ;* REQUIRE NO TABLE ENTRIES AND ARE
F20E  ;* HANDLED SEPARATELY.
F20E  ;******************************************************************************
F20E  
F20E  20 0F 3 E5CHIN JSR LI000 ;FILE OPENED?
F211 F0 03 BEQ L950 ;YES
F213 4C 01 F7 JMP L1009 ;NO, FILE NOT OPEN
F216 20 1F 3 F950 JSR L1002 ;GET FILE INFO
F219 A5 EA LDA FA
F21B F0 16 BEQ L963 ;KEYBOARD
F21D ;
F21D  C9 03 CMP $03 ;SCREEN?
F21F F0 12 BEQ L963 ;YES, DONE
F221 B0 14 BCS L961 ;SERIAL
F223 C9 02 CMP $02 ;RS-232?
F225 D0 03 BNE L958 ;NO, MUST BE TAPE
F227 4C 4D F0 JMP $F04D ;RS-232
F22A ;
F22A  ;CHECK FOR INPUT FILE ON TAPE
F22A ;
F22A  A6 B9 L958 LDX SA ;CHECK SECONDARY AD
F22C E0 60 CFX $60 ;INPUT?
F22E F0 03 BEQ L963 ;YES
F22F 4C 0A F7 JMP L971 ;NO, NOT INPUT FILE
F233 B9 99 L963 STA DFLTN ;SET INPUT
F235 18 CLC ;GOOD RETURN
F236 60 RTS
F237 ;
The Commodore 64 Kernel and Hardware Revealed

56

LOC CODE LINE
F237 ;A SERIAL DEVICE MUST TALK
F237 ;
F237 AA L961 TAX ;SAVE DEVICE #
F238 20 09 ED JSR L836 ;TALK
F238 45 E9 LDA SA ;SECOND?
F23D 10 06 BPL L962 ;YES, SEND IT
F23E 20 CC ED JSR L970 ;NO, LET GO
F242 4C 40 F2 JMP L967
F245 20 C7 ED L962 JSR L860 ;SEND TALK SA
F248 8A L967 TXA
F249 24 90 BIH STATUS ;DID IT LISTEN?
F24B C7 06 JMP L955 ;YES
F24D 4C 07 F7 JMP L1026 ;DEVICE NOT PRESENT

CHKOUT

Entry point: $FFC9

Function: Set a previously OPENed file for output.

Input parameters: .X holds the logical file number of the OPENed file.

Output parameters:
CARRY clear – OK
CARRY set – error, error number in .A

Registers used: .A, .X

Error messages:
File not open – if the logical file number in .X is not in the LFN table
Device not present – if bit 7 of ST is set, the device did not respond to the LISTEN command

Description: This routine first checks that the LFN in .X has a reference in the LFN table. If not, the message File not open is sent. The device referenced by the LFN is told to LISTEN and a secondary address is sent (if present). If bit 7 of the STATUS byte (ST) is set, the message Device not present is sent.

LOC CODE LINE
F250 ;**********************************************************************
F250 ;* CHKOUT -- OPEN CHANNEL FOR OUTPUT.
F250 ;* THE NUMBER OF THE LOGICAL FILE TO
F250 ;* BE OPENED FOR OUTPUT IS PASSED IN .X.
F250 ;* CHKOUT SEARCHES THE LOGICAL FILE TO
F250 ;* LOOK UP DEVICE AND COMMAND INFO.
F250 ;* ERRORS ARE REPORTED IF THE DEVICE WAS
F250 ;* NOT OPENED FOR INPUT, (E.G. KEYBOARD)
F250 ;* OR THE LOGICAL FILE HAS NO REFERENCE
F250 ;* IN THE TABLES. DEVICE 3 (SCREEN)
F250 ;* REQUIRES NO TABLE ENTRY AND IS
F250 ;* HANDLED SEPARATELY.
F250 ;**********************************************************************
Entry point: $FFC3

Function: Close a logical file.

Input parameters: .A holds the logical file number to close.

Output parameters: CARRY clear.


Error messages: None.

Description: The LFN table is checked for the file to be closed. If the file is not open the routine exits, otherwise the device is told to listen and then unlisten and the file entry is removed from the table.
The Commodore 64 Kernel and Hardware Revealed

LOC    CODE    LINE

F291    ;                   ;************************************************************
F291    ;* CLOSE -- CLOSE LOGICAL FILE.
F291    ;* THE LOGICAL FILE NUMBER OF THE
F291    ;* FILE TO BE CLOSED IS PASSED IN .A.
F291    ;* KEYBOARD, SCREEN, AND FILES NOT OPEN
F291    ;* PASS STRAIGHT THROUGH. TAPE FILES
F291    ;* OPEN FOR WRITE ARE CLOSED BY DUMPING
F291    ;* THE LAST BUFFER AND CONDITIONALLY
F291    ;* WRITING AN END OF TAPE BLOCK. SERIAL
F291    ;* FILES ARE CLOSED BY SENDING A CLOSE
F291    ;* FILE COMMAND IF A SECONDARY ADDRESS
F291    ;* WAS SPECIFIED IN ITS OPEN COMMAND.
F291    ;************************************************************
F291    ;
F291    20 14 F3 NCLOSE JSR L957 ;LOOK UP FILE
F294    F0 02 BEQ L992 ;FOUND
F296    1B CLC ;ELSE RETURN
F297    40 RTS
F29B    20 1F F3 L982 JSR L1002 ;GET FILE DATA
F29B    B8 TXA ;SAVE TABLE INDEX
F29C    4C PHA
F29D    A5 BA LDA FA ;CHECK DEVICE W
F29F    F0 50 BEQ L997 ;KEYBOARD, DONE
F2A1    C9 03 CMP H$03 ;SCREEN?
F2A3    F0 4C BEQ L997 ;YES, DONE
F2A5    B0 47 BCS L997 ;SERIAL
F2A7    C9 02 CMP H$02 ;RS-232?
F2A9    D0 1D BNE $F2C0 ;NO, MUST BE TAPE
F2A8    ;--------------------
F2A8    L991 JSR L1001
F2A8    ;REMOVE FILE ENTRIES FROM TABLES
F2F1    ;
F2F1    68 L9 form PLA ;GET TABLE INDEX
F2F2    AA L986 TAX
F2F3    C6 98 DEC LDNTND
F2F5    E4 98 CPX LDNTND ;IS IT AT END?
F2F7    F0 14 BEQ L989 ;YES, DONE
F2F9    A4 98 LDY LDNTND ;NO, SHIFT LAST ENTRY
F2FB    B9 59 02 LDA LAT,Y
F2FE    9D 59 02 SIA LAT,X ;INTO DELETED ENTRY'S
F301    B9 63 02 LDA FAT,Y
F304    9D 63 02 SIA FAT,X
F307    B9 6D 02 LDA SAT,Y
F30A    9D 6D 02 SIA SAT,X
F30B    1B L999 CLC ;GOOD EXIT
F30E    60 RTS
F30F    ;FIND FILE ENTRY
F30F    ;
F36F    A9 00 L1000 LDA H$00
F311    B5 90 SIA STATUS
F313    B0 TXA
F314    A6 98 L957 LDX LDNTND
F316    CA L984 DEX
F317    30 15 BMI L960
F319    DD 59 02 CMP LAT,X
F31C    D0 F8 BNE L984
F31E    60 RTS
F31F    ;
F31F    ;FETCH TABLE ENTRIES
F31F    ;
F321    BD 59 02 L9002 LDA LAT,X
F322    B5 B8 SIA LA
Entry point: $FFE7

Function: Close all logical files.

Input parameters: None

Output parameters: None

Registers used: .A, .X

Error messages: None

Description: The number of files open is zeroed and the CLRCH routine is entered.

Entry point: $FFCC

Function: Abort any serial I/O files and reset default I/O.

Input parameters: None

Output parameters: None

Registers used: .A, .X

Error messages: None
Description: The output device is checked and if it is serial, the command UNLISTEN is sent to it. The input device is then checked and if that is serial the command UNTALK is sent to it. The input device is then set to 0 (keyboard) and the output device is set to 3 (screen).

LOC CODE LINE

F333  ;
F333  ;----------------------------------------------
F333  ;* CLRCH -- CLEAR CHANNELS.
F333  ;* UNLISTEN OR UNTALK SERIAL.
F333  ;* DEVICES, BUT LEAVE OTHERS ALONE.
F333  ;* DEFAULT CHANNELS ARE RESTORED.
F333  ;----------------------------------------------
F333  ;
F333 A2 03  NCLRCH LDS @03
F335 E4 9A  CPX DFLTO ;OUTPUT CHANNEL SERIAL?
F337 B0 03  BCS L1001 ;NO
F339 20 FE ED  JSR L1006 ;YES, UNLISTEN
F33C E4 99  L1001 CPX DFLTN ;INPUT CHANNEL SERIAL?
F33E 00 03  BCS L1003 ;NO
F340 20 EF ED  JSR L863 ;YES, UNTALK
F343 B6 9A  L1003 STX DFLTN ;OUTPUT CHANNEL=3
F345 A9 00  LDA @000
F347 B5 99  STA DFLTN ;INPUT CHANNEL=0
F349 60  RTS
F34A .END
F34A .LIB KSER6

Entry point: $FFC0

Function: Open a logical file.

Input parameters:
$B7  – Length of text string to send with OPEN command (filename)
$B8  – Logical file number
$B9  – Secondary address
$BA  – Device number
$BB/$BC  – Pointer to filename

Output parameters:
CARRY clear – OK
CARRY set – error, error number in .A


Error messages:
File open – if the file number in $B8 is equal to any entry in the LFN table
Too many files – if the LFN table already has 10 entries
Device not present – if the device in $BA did not respond to the LISTEN command
Description: The LFN table is checked to see if the file number already exists (file open), and if so exits with error. The number of files open is then checked for ten (too many files), and if so exits with error. Otherwise, the file entry is submitted to the file tables (with the secondary address ORed with $60) and the number of files open incremented.

If there is no secondary address (>127), OPEN then exits. If there is no filename, OPEN exits. Otherwise, the device to be opened is told to LISTEN and the secondary address is sent. If bit 7 of ST is set, the routine exits with a device not present error. The filename is then sent one byte at a time and an UNLISTEN command is sent to the device.
The Commodore 64 Kernal and Hardware Revealed

LOAD/VERIFY

Entry point: $FFD5

Function: Load or verify a file from serial to RAM.

Input parameters:

- $B7 – Length of text string to send with OPEN command (filename)
- $B8 – Logical file number
- $B9 – Secondary address
- $BA – Device number
- $BB/$BC – Pointer to filename
- .A – Load (0)/verify ($\neq 0$) flag
- $Y/.X$ – Alternative load address (only if $B9=0$)

Output parameters:

- OK – CARRY clear
- $Y/.X$ end address
- Error – CARRY set
- .A error number


Error messages:

- Missing filename – if length of filename is zero
- File not found – if attempting to read the first byte gives a framing error
Break – if the stop key was pressed
Verify – if on verifying, the file does not match the memory contents

Description: The alternative load address is stored away. If the filename length is zero, a missing filename error is produced. Otherwise, the message ‘searching for ...’ is printed to the screen and the file is opened (with SA=$60). The device is commanded to TALK and the first byte is read in and stored to the load address low. If bit 1 of ST is set (file not found), the file is closed and LOAD exits with error in .A. Another byte is read and stored in the load address high. If the original secondary address was zero, the load address is replaced by the alternative load address.

The message ‘loading’ is printed to the screen and each byte is read in until an end of file (bit 6 of ST) is encountered or the stop key is pressed (break). With each byte, it is either stored to memory or compared with memory and if different, bit 4 of ST is set. The address is bumped by 1 and the next byte handled.

When EOF has been found, the .X and .Y registers are loaded with the end address, CARRY is cleared and LOAD exits.
LOC CODE LINE

F4CB A5 BA LDA FA ; TALK, ESTABLISH CHANNEL
F4CD 20 09 ED JSR L836 ; TELL IT TO LOAD
F4D0 A5 B9 LDA SA
F4D2 20 C7 ED JSR L860 ; GET FIRST BYTE
F4D5 20 13 EE JSR L865
F4DB 85 AE STA EAL
F4DA A5 90 LDA STATUS ; ERROR?
F4DC 4A LSR A
F4DD 4A LSR A
F4DE B0 50 BCS L1058 ; FILE NOT FOUND
F4E0 20 13 EE JSR L865
F4E3 85 AF STA EAH
F4E5 8A TXA ; ORIG SA=0?
F4E6 D0 08 BNE L1048 ; NO
F4EB A5 C3 LDA MEMUSS ; YES, SET ALT
F4EA 85 AE STA EAL ; LOAD ADDRESS
F4EC A5 C4 LDA MEMUSS+1
F4EE 85 AF STA EAH
F4F0 20 D2 F5 L1048 JSR L1070 ; 'LOADING'
F4F3 A9 FD L1051 LDA #0FD ; MASK OFF TIMEOUT
F4F5 25 90 AND STATUS
F4F7 85 90 STA STATUS
F4F9 20 EI FF JSR #FFE1 ; STOP KEY?
F4FC 00 03 BNE L1055 ; NO
F4FE 4C 33 F6 JSR L1084 ; 'BREAK'
F501 20 13 EE L1055 JSR L865 ; GET BYTE
F504 AA TAX
F505 A5 90 LDA STATUS ; TIMEOUT?
F507 4A LSR A
F508 4A LSR A
F509 B0 EB BCS L1051 ; YES, TRY AGAIN
F50B 8A TXA
F50C A4 93 LDY VERCK ; VERIFY?
F50E F0 0C BEQ L1053 ; NO, LOAD IT
F510 A0 00 LDY #000
F512 D1 AE CMP (EAL),Y ; VERIFY IT
F514 F0 08 BNE L1056 ; U.K.
F516 A9 10 LDA #SPERR ; NO, VERIFY ERROR
F518 20 1C FE JSR #FE1C ; UPDATE STATUS
F51B 2C .BYT #2C ; SKIP STORE
F51C 91 AE L1053 STA (EAL),Y
F51E 66 AE L1056 INC EAL ; INCREMENT STORE ADDR
F520 D0 02 BNE L1057
F522 8A AF INC EAH
F524 24 90 L1057 BIT STATUS ; END OF INPUT?
F526 50 CB BVC L1051 ; NO, CARRY ON
F528 20 EF ED JSR L863 ; CLOSE CHANNEL
F52B 20 42 F6 JSR L1081 ; CLOSE FILE
F52E 90 79 BCC L1067 ; ALWAYS
F530 4C 04 F7 L1058 JMP L959 ; FILE NOT FOUND
F533 -------------------------------
F533 # $F5A9
F549 18 L1067 CLC ; GOOD EXIT
F5AA ;
F5AA A6 AE LDX EAL
F5AC A4 AF LDY EAH
F5AE 60 L1059 RTS
F5AF ;
F5AF ; PRINT 'SEARCHING [FOR NAME]' 
F5AF EAL
F5AF A5 9D L1062 LDA MSGFLG ; PRINT IT?
F5B1 10 1E BPL L1071 ; NO
F5B3 A0 0C LDY #00C ; 'SEARCHING'
F5B5 20 2F F1 JSR #F12F
F5BA A5 B7 LDA FNLEN
Serial Communications

Entry point: $FFD8

Function: SAVE a section of memory to serial device.

Input parameters:
$B7  - Length of text string to send with open command (filename)
$B8  - Logical file number
$B9  - Secondary address
$BA  - Device number
$BB/$BC  - Pointer to filename
.A  - Pointer to zero page save address
.Y/.X  - End of save address
.(A)  - Page zero indirect start of save

Output parameters:
CARRY clear  - OK
CARRY set  - error, error number in .A


Error messages:
Missing filename – if the length of the filename is zero
Break – if the stop key was pressed during SAVE

Description: The length of the filename is checked and if zero (missing filename), exits with error. The file is opened, the message ‘saving ...’ is printed
The Commodore 64 Kernel and Hardware Revealed

to the screen and the device is told to LISTEN. The save address (low followed by high) is sent to the device. The start address is compared with the end address and if reached the file is closed and the routine exits with CARRY clear.

One byte of the file is sent to the device and the stop key is tested. If the stop key was not pressed, the start address is bumped by 1 and the routine loops back to the address comparison. If the stop key was pressed, the file is closed and the routine exits with CARRY set.

LOC   CODE   LINE

F50D  ;
F50D  *************
F50D  ; SAVE MEMORY FUNCTION.
F50D  ;* Saves to Cassette or Serial
F50D  ;* Devices >=4 to 31 as selected by
F50D  ;* Variable FA.
F50D  ;* Start of Save is indirect at .A
F50D  ;* End of Save is .X, .Y
F50D  ;* Use SEITFS & SETNAA before this routine
F50D  ;*************************@
F50D  
F50D  B6 AE L1072 STX EAL
F50D  B4 AF NSAVE LDA FA
F50E  AA STY EAH ; SET UP START
F50E  B3 00 LDA $00,X
F50E  B5 C1 STA STAL
F50E  B5 01 LDA $01,X
F50E  B5 C2 STA STH
F50E  6C 32 03 JMP (ISAVE)
F50E  A5 0A JSR SETLFS
F50E  D0 03 BNE L1075
F50F  4C 13 F7 L1242 JMP L1049 ;BAD DEVICE
F50F  C9 03 L1075 CMP $003 ; SERIAL?
F50F  F0 F9 BEU L1242 ;SCREEN, BAD DEVICE
F50F  90 5F BCC $F659 ; NO, 1FE
F50F  A9 61 LDA $61
F50F  B5 09 STA SA
F50F  A4 B7 LDY FNLEN
F50F  D0 03 BNE L1074
F50F  4C 10 F7 JMP L974 ;MISSING FILE NAME
F50F  20 B5 F3 L1074 JSR L1021 ;OPEN
F50F  20 BF F6 JSR L1087 ; 'SAVING'
F50F  A5 0A LDA FA
F50F  20 OC ED JSR L966 ;LISTEN
F50F  A5 B9 LDA SA
F50F  20 B9 ED JSR LB71 ;LISTEN SA
F50F  A0 00 LDY $00
F50F  20 0E FB JSR $F8BE
F50F  A5 AC LDA SAL
F50F  20 DD ED JSR LB61
F50F  A5 0D LDA SAH
F50F  20 DD ED JSR LB61
F50F  20 D1 FC L1077 JSR $FC01 ;COMPARE START TO END
F50F  20 16 BCS L1002 ; HAVE REACHED END
F50F  20 B1 AC LDA ($AL),Y
F50F  20 DD ED JSR LB61
F50F  20 E1 FF JSR $FFE1 ; STOP KEY?
F50F  D0 07 BNE L1054 ; NO
F50F  20 42 F6 L1094 JSR L1081 ; YES, CLOSE
F50F  A9 00 LDA $00
F50F  38 SEC
F50F  60 RTS
F50F  20 DB FC L1054 JSR $FCDB ; INCREMENT CURRENT ADDR
F50F  D0 E5 BNE L1077
Error handler

Entry point:
$F6FB – (1) too many files
$F6FE – (2) file open
$F701 – (3) file not open
$F704 – (4) file not found
$F707 – (5) device not present
$F70A – (6) not input file
$F70D – (7) not output file
$F710 – (8) missing filename
$F713 – (9) bad device

Function: To flag an error and print it if output is enabled.

Input parameters: None

Output parameters: CARRY set, error number in .A


Error message: I/O error #(number) – if bit 6 of MSGFLG is set

Description: At each entry point, the .A register is loaded with the value in brackets. The routine CLRCH is called and if bit 6 of MSGFLG (output enable) is clear, CARRY is set and .A holds the error number upon exit. If bit 6 is set, the message ‘I/O error #’ is printed to the screen and the number is converted to ASCII and printed. CARRY is set and .A is reloaded with the error number.
**The Commodore 64 Kernel and Hardware Revealed**

Loc Code Line

F69B *E8F6FB*
F6FB ;**********************************************************************
F6FB ;* ERROR HANDLER.*
F6FB ;* PRINTS KERNAL ERROR MESSAGE IF BIT 6
F6FB ;* OF MSGFLAG IS SET. RETURNS WITH ERROR
F6FB ;* # IN .A AND CARRY SET.
F6FB ;**********************************************************************
F6FB:
F6FB A9 01 L1097 LDA #01 ;100 MANY FILES
F6FB 2C .BYT $2C
F6FB A9 02 L1011 LDA #02 ;FILE OPEN
F6FB 2C .BYT $2C
F700 A9 03 L1009 LDA #03 ;FILE NOT OPEN
F703 2C .BYT $2C
F706 A9 04 L959 LDA #04 ;FILE NOT FOUND
F709 2C .BYT $2C
F710 A9 05 L1026 LDA #05 ;DEVICE NOT PRESENT
F700 2C .BYT $2C
F700 A9 06 L971 LDA #06 ;NOT INPUT FILE
F703 2C .BYT $2C
F706 A9 07 L965 LDA #07 ;NOT OUTPUT FILE
F709 2C .BYT $2C
F710 A9 08 L974 LDA #08 ;MISSING FILE NAME
F712 2C .BYT $2C
F713 A9 09 L1049 LDA #09 ;BAD DEVICE #
F715 48 PHA ;ERROR # ON STACK
F716 26 CC FF JSR #FFCC ;RESTORE I/O
F719 A0 00 LDY #00
F71B 24 9D BIT MSGFLAG ;PRINT ERROR?
F71D 50 0A BVC L1010 ;NO
F71F 26 2F F1 JSR #12F ;PRINT 'I/O ERROR #'
F722 68 PLA
F723 48 PHA
F724 09 30 ORA #30 ;MAKE ERROR # ASCII
F726 26 D2 FF JSR #FFD2 ;PRINT IT
F729 68 L1018 PLA
F72A 3B SEC
F72B 60 RTS
F72C ;----------------------------------
F72C .END .END

**Symbol Table**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOON</td>
<td>0292</td>
</tr>
<tr>
<td>BDF</td>
<td>#002</td>
</tr>
<tr>
<td>BIT1S</td>
<td>#004</td>
</tr>
<tr>
<td>BLN5W</td>
<td>#001</td>
</tr>
<tr>
<td>BUF</td>
<td>#006</td>
</tr>
<tr>
<td>CAS1</td>
<td>#016</td>
</tr>
<tr>
<td>CINV</td>
<td>#014</td>
</tr>
<tr>
<td>COLM</td>
<td>#286</td>
</tr>
<tr>
<td>CRSW</td>
<td>#000</td>
</tr>
<tr>
<td>D1DDRB</td>
<td>#003</td>
</tr>
<tr>
<td>D1100D</td>
<td>#00C</td>
</tr>
<tr>
<td>D11TLB</td>
<td>#006</td>
</tr>
<tr>
<td>D11D4D</td>
<td>#008</td>
</tr>
<tr>
<td>D2DDRB</td>
<td>#003</td>
</tr>
<tr>
<td>D210DD</td>
<td>#00C</td>
</tr>
<tr>
<td>D21TLB</td>
<td>#006</td>
</tr>
<tr>
<td>D210D4D</td>
<td>#008</td>
</tr>
<tr>
<td>DFL1U</td>
<td>#09A</td>
</tr>
<tr>
<td>EAL</td>
<td>#00A</td>
</tr>
<tr>
<td>F1RT</td>
<td>#004</td>
</tr>
<tr>
<td>F8BLK</td>
<td>#00E</td>
</tr>
<tr>
<td>BAD</td>
<td>#100</td>
</tr>
<tr>
<td>BASZFT</td>
<td>#0FF</td>
</tr>
<tr>
<td>BAUDOF</td>
<td>#0299</td>
</tr>
<tr>
<td>BITCI</td>
<td>#00B</td>
</tr>
<tr>
<td>BITNUM</td>
<td>#0298</td>
</tr>
<tr>
<td>BLNCT</td>
<td>#00C</td>
</tr>
<tr>
<td>BLN6N</td>
<td>#00F</td>
</tr>
<tr>
<td>BSDT</td>
<td>#095</td>
</tr>
<tr>
<td>BSDUR1</td>
<td>#0A4</td>
</tr>
<tr>
<td>BVSZ</td>
<td>#00F</td>
</tr>
<tr>
<td>CBM4COL</td>
<td>D000</td>
</tr>
<tr>
<td>CMFO</td>
<td>D000</td>
</tr>
<tr>
<td>CNTDN</td>
<td>D000</td>
</tr>
<tr>
<td>CR</td>
<td>00D</td>
</tr>
<tr>
<td>DICATION</td>
<td>DCE</td>
</tr>
<tr>
<td>D1CRB</td>
<td>D0F</td>
</tr>
<tr>
<td>D1DDRA</td>
<td>D02</td>
</tr>
<tr>
<td>D1DFB</td>
<td>D01</td>
</tr>
<tr>
<td>D1D0R</td>
<td>D00</td>
</tr>
<tr>
<td>D10TAL</td>
<td>D04</td>
</tr>
<tr>
<td>D1T0D2</td>
<td>D09</td>
</tr>
<tr>
<td>D210R</td>
<td>D04</td>
</tr>
<tr>
<td>D21AH</td>
<td>D05</td>
</tr>
<tr>
<td>D21AL</td>
<td>D06</td>
</tr>
<tr>
<td>D2T0D2</td>
<td>D09</td>
</tr>
<tr>
<td>D2T0D3</td>
<td>D0A</td>
</tr>
<tr>
<td>D3FD6</td>
<td>D00</td>
</tr>
<tr>
<td>DFPSW</td>
<td>D09C</td>
</tr>
<tr>
<td>DFL1N</td>
<td>#099</td>
</tr>
<tr>
<td>DFL1N</td>
<td>#099</td>
</tr>
<tr>
<td>DFL1N</td>
<td>#099</td>
</tr>
<tr>
<td>EAH</td>
<td>#06F</td>
</tr>
<tr>
<td>FA</td>
<td>#08A</td>
</tr>
<tr>
<td>FAT</td>
<td>#0263</td>
</tr>
<tr>
<td>FNADR</td>
<td>#0B8</td>
</tr>
<tr>
<td>FNLEN</td>
<td>#0B7</td>
</tr>
<tr>
<td>FREDKZP</td>
<td>#0BF</td>
</tr>
<tr>
<td>GDCOL</td>
<td>D028</td>
</tr>
<tr>
<td>HIBASE</td>
<td>D028</td>
</tr>
<tr>
<td>HIBASE</td>
<td>D028</td>
</tr>
<tr>
<td>SYMBOL</td>
<td>VALUE</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>IBASIN</td>
<td>0324</td>
</tr>
<tr>
<td>IBSOUT</td>
<td>0326</td>
</tr>
<tr>
<td>ICHKIN</td>
<td>031E</td>
</tr>
<tr>
<td>ICKOUT</td>
<td>0320</td>
</tr>
<tr>
<td>ICLALL</td>
<td>032C</td>
</tr>
<tr>
<td>ICLOSE</td>
<td>031C</td>
</tr>
<tr>
<td>ICLRC</td>
<td>0322</td>
</tr>
<tr>
<td>IEGTIN</td>
<td>032A</td>
</tr>
<tr>
<td>ILOAD</td>
<td>0330</td>
</tr>
<tr>
<td>IHBIT</td>
<td>0067</td>
</tr>
<tr>
<td>INDX</td>
<td>00CB</td>
</tr>
<tr>
<td>IINSRT</td>
<td>0081</td>
</tr>
<tr>
<td>IPRTY</td>
<td>032B</td>
</tr>
<tr>
<td>KEYD</td>
<td>0277</td>
</tr>
<tr>
<td>KEYLOG</td>
<td>028F</td>
</tr>
<tr>
<td>KEYTAB</td>
<td>00F5</td>
</tr>
<tr>
<td>KOUNT</td>
<td>02BB</td>
</tr>
<tr>
<td>L1000</td>
<td>F30F</td>
</tr>
<tr>
<td>L1001</td>
<td>F33C</td>
</tr>
<tr>
<td>L1002</td>
<td>F31F</td>
</tr>
<tr>
<td>L1003</td>
<td>F343</td>
</tr>
<tr>
<td>L1004</td>
<td>F303</td>
</tr>
<tr>
<td>L1005</td>
<td>F351</td>
</tr>
<tr>
<td>L1006</td>
<td>F36E</td>
</tr>
<tr>
<td>L1007</td>
<td>F304</td>
</tr>
<tr>
<td>L1008</td>
<td>F305</td>
</tr>
<tr>
<td>L1009</td>
<td>F312</td>
</tr>
<tr>
<td>L1010</td>
<td>F302</td>
</tr>
<tr>
<td>L1011</td>
<td>F303</td>
</tr>
<tr>
<td>L1012</td>
<td>F305</td>
</tr>
<tr>
<td>L1013</td>
<td>F312</td>
</tr>
<tr>
<td>L1014</td>
<td>F305</td>
</tr>
<tr>
<td>L1015</td>
<td>F312</td>
</tr>
<tr>
<td>L1016</td>
<td>F302</td>
</tr>
<tr>
<td>L1017</td>
<td>F303</td>
</tr>
<tr>
<td>L1018</td>
<td>F305</td>
</tr>
<tr>
<td>L1019</td>
<td>F312</td>
</tr>
<tr>
<td>L1020</td>
<td>F305</td>
</tr>
<tr>
<td>L1021</td>
<td>F312</td>
</tr>
<tr>
<td>L1022</td>
<td>F305</td>
</tr>
<tr>
<td>L1023</td>
<td>F312</td>
</tr>
</tbody>
</table>

**Serial Communications**

69
3.4 RS232 serial communications

The CBM 64 is able to communicate with peripheral devices, known as an RS232 I/O port. The name RS232 simply refers to an industry standard form of serial communication for computing devices. A serial I/O port can consist of as few as three lines, an output or transmit line, an input or receive line and a common ground line. The data is transmitted or received as a stream of pulses; a single byte becomes a string of eight pulses.

Although a serial port can have just three lines, other lines are frequently used to transfer control information. The 64 is able to receive and generate such control signals to implement a full 'X line' interface as well as the simple '3 line' interface. Whichever implementation is used all the lines are connected to I/O port B of CIA#2 (user port). The RS232 routines inside the 64 also use two other lines; PA2 on port A and FLAG which is connected to the NMI line. Normally an RS232 interface card will be used to connect between the user port and a standard RS232 connector. The card will also provide buffering and a higher drive voltage. For communications using the simple 3 line mode an interface card can easily be constructed using a couple of buffer/driver ICs. The RS232 line normally transmits data using a 12 volt signal, however, and providing cables are kept short it will work with a 5 volt signal. The standard RS232 connector is shown in Fig. 3.4. The function and pin assignment of each of these lines is as follows:

<table>
<thead>
<tr>
<th>CIA line #</th>
<th>RS232 pin #</th>
<th>CIA Abv</th>
<th>EIA In/Out</th>
<th>Modes</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>GND</td>
<td>1</td>
<td>A</td>
<td>GND AA</td>
<td>1,2</td>
<td>Protective ground</td>
</tr>
<tr>
<td>FLAG</td>
<td>3</td>
<td>B</td>
<td>SIN BB</td>
<td>1,2</td>
<td>Received data</td>
</tr>
<tr>
<td>PB0</td>
<td>3</td>
<td>C</td>
<td>SIN BB</td>
<td>1,2</td>
<td>Connected to FLAG</td>
</tr>
<tr>
<td>PB1</td>
<td>4</td>
<td>D</td>
<td>RTS CA</td>
<td>Out</td>
<td>Request to send</td>
</tr>
<tr>
<td>PB2</td>
<td>2</td>
<td>E</td>
<td>DTR CD</td>
<td>Out</td>
<td>Data terminal ready</td>
</tr>
<tr>
<td>PB3</td>
<td>18</td>
<td>F</td>
<td>RI CE</td>
<td>In</td>
<td>Ring indicator</td>
</tr>
<tr>
<td>PB4</td>
<td>8</td>
<td>H</td>
<td>DCD CF</td>
<td>In</td>
<td>Received line signal</td>
</tr>
<tr>
<td>PB5</td>
<td>Not assigned</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PB6</td>
<td>5</td>
<td>K</td>
<td>CTS CB</td>
<td>In</td>
<td>Clear to send</td>
</tr>
<tr>
<td>PB7</td>
<td>6</td>
<td>L</td>
<td>DSR CC</td>
<td>In</td>
<td>Data set ready</td>
</tr>
<tr>
<td>PA2</td>
<td>2</td>
<td>M</td>
<td>SOUT BA</td>
<td>Out</td>
<td>Transmitted data</td>
</tr>
<tr>
<td>GND</td>
<td>7</td>
<td>N</td>
<td>GND AB</td>
<td>2,3</td>
<td>Signal ground</td>
</tr>
</tbody>
</table>

Modes:
1 Three line interface (note RTS and DTR are both held high during this mode)
2 X line interface
3 User only, not implemented in the CBM 64 code
The implementation of the RS232 port on the 64 is very interesting since it involves the use of software (originally used on the VIC 20 with very few modifications for the 64) to emulate a hardware device (that was never used). This device was called the 6551 universal asynchronous receiver and transmitter or UART. Like the other I/O chips, it was intended that the 6551 functions were to be controlled by registers at specific memory locations. The software uses the same principle because when it was written for the VIC 20, Commodore
intended to replace the software with the 6551 when it became available, so there
would be complete compatibility.

The pseudo registers are located in various parts of the variable storage area
at the bottom of CBM 64 memory. Besides the registers, the RS232 routines
require two 256 byte buffers; one for received data and one for data to be
transmitted. The 512 bytes of memory occupied by these buffers are located at
the top of available RAM memory, and the starting address of the two buffers is
stored in four register bytes. The two most important registers are the control
and command registers. These determine the exact operation of the RS232 port.
They can be summarised as follows:

3.4.1 RS232 control register – Hex $0923 Decimal 659

The function of the control register (Fig. 3.5) is to set the speed of data
transmission and reception and set the number of bits needed to transmit each
character. The speed at which data is input or output is called the baud rate, and

![Table of BAUD RATE and USER RATE values]

the value assigned to this is the number of bits per second. If the baud rate is set
to 300 baud, and each character is transmitted as the eight character bits plus
one stop bit and one parity bit – a total of ten bits – then 300 characters will be
transmitted every second. The selected baud rate depends on the specifications
of the device communicating with the 64 via the RS232 port – check the manual.
of the device before setting this value. Bits 5, 6 and 7 control the number of bits needed to transmit or receive data between the 64 and a peripheral. The number of bits per character plus the number of stop bits depends on the peripheral.

3.4.2 RS232 command register – Hex $\$294 Decimal 66\$

The command register (Fig. 3.6) controls the mode for data transmission and reception. Bit 0 sets the mode; a 3 line mode or an X line mode. Bit 4 sets the duplex mode as follows:

Full duplex – simultaneous transmission and reception of data
Half duplex – alternate transmission and reception of data

![Fig. 3.6. Function of bits in the CBM 64 RS232 command register.]

Bits 5, 6 and 7 determine the nature of the parity bit and whether the mark or space is transmitted. The parity bit is transmitted after the data bits and has an error checking function. The choice of whether the parity is disabled or set to odd or even depends on the communicating peripheral. The mark/space setting determines whether a logic ‘1’ is transmitted as a zero voltage or a positive voltage.

3.4.3 RS232 status register – Hex $\$297 Decimal 663

The function of each bit of the status register is shown in Fig. 3.7. The memory locations and pseudo registers of the RS232 routines are as follows:
The Commodore 64 Kernal and Hardware Revealed

Fig. 3.7. Function of bits in the CBM 64 RS232 status register.

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PARITY ERROR BIT</td>
<td>FRAMING ERROR BIT</td>
<td>RECEIVER BUFFER OVERRUN BIT</td>
<td>Unused</td>
<td>CTS SIGNAL MISSING BIT</td>
<td>Unused</td>
<td>DSR SIGNAL MISSING BIT</td>
</tr>
</tbody>
</table>

RS-232 STATUS REGISTER — $02A1

3.4.4 RS232 system routine entry points

$EEB1 — entry for NMI continue routine
$EED7 — calculate parity
$EF00 — count stop bits
$EF06 — entry to start of byte transmission
3.5 Using the RS232 port

3.5.1 Opening an RS232 channel

Basic syntax: OPEN If,2,Ø,“(control register) (command register)”

The syntax coding is as follows:

If – normal logical file ID (1–255). If If>127 then line feed follows carriage return.

(control register) – an ASCII character equivalent to the required bit setting of the control register. Example: to set baud rate to 300 and transmit 7 bit code use CHR$(6+32) – this sets bits 1, 2 and 5 to '1' and leaves the rest at '0'.

(command register) – an ASCII character equivalent to the required bit setting of the command register. Example: to set the output to mark parity and full duplex use CHR$(32+128) – this sets bits 5 and 7 to '1' and leaves the rest at '0'.

Entry point: $FFCØ
Notes on usage: Only one RS232 channel should be open at any time. Since the OPEN statement resets the buffer pointers, a second OPEN will destroy any data in the buffers set up in the first OPEN. The OPEN RS232 channel command should be used before any variable DIM statements; failure to do this will cause wiping of data. This is because the OPEN RS232 channel command performs an automatic CLR before allocating the 512 bytes at the top of memory used for the two RS232 buffers.

3.5.2 Receiving data from an RS232 channel

Basic syntax: GET #If,(string variable)

If – logical file ID used in OPEN RS232 channel command.

Entry points:
$FFC6 – set channel for input. Handles full X line implementation according to EIA standard RS232C interfaces. The RTS, CTS, and DCD lines are implemented when the CBM 64 is designated as a data terminal device.

$FFE4 – get character from buffer.

Notes on usage: Received data is put into the 64's 256 byte internal receiver buffer set up during the OPEN routine. Data input is under control of the 6526 timers and NMIs, and is performed in the background during the running of a Basic program. This is done by having the RS232 data input line connected to the FLAG handshake line, and input on FLAG will generate an NMI system interrupt. The use of NMI interrupts is the reason why the cassette and serial bus should not be used during RS232 data communications. The NMI will call the serial data input routines whenever data is present on the RS232 input. These routines will place the received data into the 256 byte receiver buffer located at the top of RAM memory. If the input data has a word width less than eight bits then all unused bits will be filled with zero.

The receiver buffer is organised as a first in first out – FIFO – buffer. The buffer removes the necessity for Basic to wait for data input before processing each byte of data. Instead the Basic program can take data from the buffer when it needs it rather than when it is presented. Basic accesses the buffer using the GET# command to transfer a single byte of data into a Basic variable. If there is no data in the buffer then the GET# command will return with a null character. If the buffer should overflow then all characters are lost. An overflow condition is indicated by bit 2 in the RS232 status register being set. An overflow condition will frequently result if an attempt is made to input data at fairly high data rates using Basic. This is because Basic is normally slow and the use of the GET# command with string concatenation will give rise to frequent garbage collects. Machine language routines are best used for data rates above the normal 300 baud.

3.5.3 Transmitting data to an RS232 channel

Basic syntax: CMD If PRINT#lf,(variable list)

lf – logical file ID set up in the OPEN command.

Entry points:
$FFC9 – set channel for output. This handles X line handshaking for the
implementation of an EIA standard RS232 interface. The RTS, CTS, and DCD lines are implemented with the 64 as a data terminal.

$FFD2 – output character to channel.

Notes on usage: When either one of the two Basic commands is used data is first transferred from the assigned string or memory block to the 256 byte transmitter buffer. From here it is output to the RS232 channel using the format and baud rate assigned in the OPEN command. Data output is transparent to the operation of Basic since the timing is done by the 6526 timers and output of each byte initiated by an NMI system interrupt. As with data input on the RS232, the cassette or serial port should not be used during data transmission otherwise interrupt conflicts will occur. There is no carriage return delay implemented by the output channel, therefore a normal RS232 printer cannot correctly output the data unless some form of internal buffering or other hold-off is implemented by the printer. If a CTS handshake is implemented (in the X line mode) then the 64 buffer will fill, and output will not occur until transmission is allowed by an input on CTS.

3.5.4 Closing an RS232 data channel

Basic syntax: CLOSE If
If – logical file ID set up in the OPEN channel command.

Entry point:
$FFC3 – close logical file

Notes on usage: Closing the RS232 file causes all the data in the buffers to be discarded, stops data transmitting or receiving, sets the RTS and SOUT lines high, and deallocates the memory area used for the RS232 buffers. Closing an RS232 file will also allow the cassette or serial ports to be used. Before closing the channel care should be taken to ensure that all data in the buffer is transmitted. This can be done by checking the status (ST variable) is=0 and that bit 0 of the RS232 enable register at location 673 ($02A1) is set to logic 1. If both are true then there is still data in the buffer.
Chapter Four

The Cassette Units

4.1 The cassette hardware

The CBM 64 has a single external cassette unit which is used for program and data storage. The cassette deck is connected to the CBM 64 by six lines - write, read, motor, sense and two power lines; ground and +5 volts. The connections are shown in Fig. 4.1. The cassette is controlled by I/O lines from the CIA chip and the processor I/O register. The source of each of the cassette control lines is shown in Fig. 4.2. The cassette motor power supply lines are connected to the processor chip via a three transistor driver, used to boost the power and voltage, allowing the motor to be driven directly. The output to the motor is an

<table>
<thead>
<tr>
<th>PIN #</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-1</td>
<td>GND</td>
</tr>
<tr>
<td>B-2</td>
<td>+5V</td>
</tr>
<tr>
<td>C-3</td>
<td>CASSETTE MOTOR</td>
</tr>
<tr>
<td>D-4</td>
<td>CASSETTE READ</td>
</tr>
<tr>
<td>E-5</td>
<td>CASSETTE WRITE</td>
</tr>
<tr>
<td>F-6</td>
<td>CASSETTE SWITCH</td>
</tr>
</tbody>
</table>

Fig. 4.1. The allocation and function of pins on the cassette connector.

Fig. 4.2. The cassette circuit and its connection to the 6522 chips.
unregulated +9 volts at a power rating of up to 100 mA. The cassette deck motor can be turned on and off by toggling line 5 of the processor I/O register:

POKE 1, PEEK(1) AND 191 turns the motor on
POKE 1, PEEK(1) OR 64 turns it off

Great care should be taken not to alter the status of bits 1 and 2 of location I when using this command, since these control the memory configuration of the machine. The sense line input, line 4 of the processor I/O register, is connected to a switch on the cassette deck which senses when the play, rewind or fast forward buttons have been pressed. The switch is only required to sense the pushing of the play button during a read or write to tape routine; this is done by a subroutine at $F82E. If either the rewind or fast forward button is pressed accidentally instead of the play button, the system will be unable to tell the difference and will act as if the play button has been pressed. For a similar reason during a record routine the record button must be pressed before the play button, since recording will start as soon as the sense switch is closed by pressing the play button.

The cassette read line is connected to the negative edge sensitive serial input line of CIA#1 and the cassette write line to line 3 of the processor I/O register. During a read operation the operating system uses the setting of the CIA#1 interrupt flag to detect transitions on the cassette read line. The functioning of the read and write lines is controlled entirely by the operating system, the only hardware required being signal amplification and pulse shaping circuitry. These circuits are contained on a small PC board within the cassette deck, their function being to give correct voltage and current to the record head and amplify the input from the read head to give a 5 volt square wave output, able to produce an interrupt on the FLAG or CB1 lines.

4.2 Cassette operation

In normal usage the cassette deck is assigned an I/O device number. The cassette is device number 1, and the number of the device currently being used is stored in location 186. The device number together with the logical file number and the secondary address is used when saving or retrieving data files from the cassette deck. The logical file number can be any number from 1 to 255 and is used to allow multiple files to be kept on the same device. It is of little use with cassette tape and is intended primarily for use with floppy disk units. It is usual to have the logical file number the same as the device number; the logical file number of the current file is stored in location 184. The secondary address is important since it determines the operational mode of the cassette; the current secondary address is stored in location 185, the normal default value being zero. If the secondary address is zero then the tape is opened for a 'read' operation, if it is set to 1 then it is opened for a 'write' operation, and if 2 then it is opened for a 'write' with an end of tape header being forced when the file is closed.

The CBM 64 operating system is configured to allow two different types of file to be stored on cassette: program files and data files. These names are rather
misleading, however, since a program can be stored as a data file and data can be stored as a program file. The difference between these two file types is not in their application but in the way the contents of the machine's memory are recorded. Instead of program and data files we must look upon them as binary and ASCII files.

4.3 Binary files

A binary file is usually used to store programs, since a binary file is created by the operating system to store the contents of memory between a starting location and an end location. It is called a binary file because it stores on tape the binary value in each memory location within the assigned memory area. Basic statements are stored in memory using tokens. The use of tokens means that Basic commands are not stored in the same manner as they are listed on the display or were entered on the keyboard. They are instead stored in memory in a partly encoded form. Being partly encoded, a binary file is a quicker and more efficient way of storing programs. Binary files are essential when saving and loading machine code programs.

The starting address from which a binary file will be saved is stored in locations 172 and 173. These locations are loaded by the SAVE routine with the memory location at which the SAVE will begin. Normally they will be set to 01 and 08, thereby pointing to the start of the Basic text area at 2049. They can be altered by the SAVE routine to point to any location in memory. The end address of the area of memory to be saved is stored in locations 174 and 175. Normally when saving a Basic program these are set to the address of the double zero byte terminating link address. The end address can be altered to any desired location. To change either of these addresses one cannot use the normal SAVE routine since this automatically initialises these locations. Instead one must write a small machine code initialisation routine incorporating the desired operating system subroutines. By default a SAVE command will write a binary file and a LOAD command will read a binary file.

4.4 ASCII files

An ASCII file is normally used to store data but it can be used to store programs (see the MERGE procedure). The format is the same as that displayed on the screen or entered on the keyboard. ASCII files are created or read almost exclusively by instructions from within a Basic program. A binary file is created or read mostly by direct instructions, though the LOAD and SAVE instructions can be used within a program.

An ASCII file must first be opened with an OPEN statement. This specifies the logical file, device number, secondary address and filename. The operating system interprets these parameters and allows the user to read or write the file to the specified device. Data is written to an ASCII file on a particular device with a command to PRINT to the specified logical file number, and data is read by a READ from logical file command.
Whereas a binary file is loaded with the contents of successive memory locations, an ASCII file is loaded with a string of variables. Storing these would require the tape to be turned on and off repeatedly, storing a few bytes of data at a time. The CBM 64 overcomes this by having a 192 byte tape buffer into which all data to be written to or read from tape is loaded. Only when this buffer is full is the tape motor turned on. Data is stored on tape in blocks of 192 bytes, and since the motor is turned on and off between blocks a two second interval is left between blocks to allow the motor to accelerate and decelerate. The beginning of the 192 character buffer starts at address 828. The pointer to the start of the buffer is located at addresses 178 and 179. The number of characters in a buffer is stored in location 166. These locations can be used by the programmer to control the amount of space left in a data file. If, having opened a file on cassette, the command POKE 166,191 is executed, then the contents of the tape buffer even if empty are loaded onto the tape. If records are kept in multiples of 191 bytes we can very easily keep null or partially filled records allowing future data expansion.

4.5 Recording method

Whether the file being stored is binary or ASCII the recording method used is the same, involving an encoding method unique to Commodore and designed to ensure maximum reliability of recording and playback. Each byte of data or program is encoded by the operating system using pulses of three distinct audio frequencies. These are: long pulses with a frequency of 1488 Hz, medium pulses at 1953 Hz and short pulses at 2840 Hz. All these pulses are square waves with a mark space ratio of 1:1. One cycle of a medium frequency is: 256 microseconds in the high state and 256 microseconds in the low state. The operating system takes about 9 milliseconds to record a byte of data consisting of the eight data bits, a word marker bit and an odd parity bit. The data bits are either ones or zeros and are encoded by a sequence of medium and short pulses: a ‘1’ is one cycle of a medium length pulse followed by one cycle of a short length pulse, and ‘0’ is one cycle of a short length pulse followed by one cycle of a medium length pulse. Each bit consists of two square wave pulse cycles, one short and one medium, with a total duration of 864 microseconds. The waveform timing is shown in the diagram in Fig. 4.3.

The ‘odd parity’ bit is required for error checking and is encoded like the eight data bits using a long and short pulse. Its state is determined by the contents of the eight data bits. The word marker separates each byte of data and signals to the operating system the beginning of each byte. The word marker is encoded as one cycle of a long pulse followed by one cycle of a medium pulse (see Fig. 4.3).

Since a byte of data is recorded in just 8.96 milliseconds, a 192 byte block of data in an ASCII file should be recorded in just over 1.7 seconds. However, on timing such a recording we find it takes 5.7 seconds. There are two causes for this discrepancy in timing. Firstly, to reduce the possibility of audio dropouts the data is recorded twice. Secondly, a two second interrecord gap is left between each record of 192 bytes.
The Commodore 64 Kernal and Hardware Revealed

The extensive use of error checking techniques is one reason why the tape system on the CBM 64 is slow but also quite reliable compared with that available on many other popular computers. There are two levels of error checking. The first divides the data into blocks of eight bytes and then computes a ninth byte, the checksum digit. The checksum is obtained by adding the eight data bytes together; the checksum is the least significant byte of the result. On reading the tape, if one bit in the eight bytes is dropped and a zero becomes a one or vice versa, the checksum can be used to detect this error. To do this the same procedure to calculate the check digit is performed, but the result will be different from that stored in byte nine – the check digit of that block computed when the tape was recorded. The second level of error checking involves recording each block of data twice. This allows errors detected by the check digit to be corrected during the second reading of the 192 byte data block. By recording the data twice a verification can be performed by comparing the contents of the two blocks; this will highlight the few errors not detected by the checksum.

The use of pulse sequences rather than two frequencies as in a standard FSK recording has a great advantage since it allows the operating system to compensate easily for variations in recording speed. Normally a hardware phase locked loop circuit would be used to lock the system onto the correct frequencies coming from the tape head. The CBM 64, however, uses software to perform this process. A ten second leader is written on the tape before recording of the data or program commences. This leader has two functions; first it allows the
tape motor to reach the correct speed, and secondly the sequence of short pulses written on the leader is used to synchronise the read routine timing to the timing on the tape. The operating system can thus produce a correction factor which allows a very wide variation in tape speed without affecting reading.

The system timing used to perform both reading and writing is very accurate, based as it is on the crystal controlled system clock and Timer 1 and Timer 2 of CIA#2. Interrecord gaps are used only in ASCII files and their function is to allow the tape motor time to decelerate after being turned off and accelerate to the correct speed when turned on prior to a block read or write. Each interrecord gap is approximately two seconds long and is recorded as a sequence of short pulses in the same manner as the ten second leader. There is also a gap between blocks. When the first block of 192 bytes is recorded it is followed by a block end marker which consists of one single long pulse followed by 50+ cycles of short pulses. Then the second recording of the 192 block starts, which is identical to the first block.

The first record written on the tape after the ten second leader in both ASCII and binary files is a 192 character file header block. The file header contains the name of the file, the starting memory location, and the end location. In an ASCII file these addresses are the beginning and end of the tape buffer; in a binary file they point to the area of memory in which the program is to be stored.

The filename can be up to 187 bytes long. The length of the filename is stored in location 183. When read it is compared with the requested filename in the LOAD or OPEN command; if the name is the same the operating system will read the file, if different then it will search for the next ten second interfile gap and another header block. The filename is stored during a read or write operation in a block memory, the starting address of which is stored in locations 187 and 188. On completion of the operation these are reset to point to a location in the operating system. The starting location is normally set to the beginning of the user memory area, address 2049, however it can be changed to point to any location – a method employed when recording programs in machine code using the monitor. The starting address is pointed to by the contents of locations 172 and 173. The end address is stored in locations 174 and 175. Normally this is the highest byte of memory occupied by the program, however it can be altered to point to any address providing it is greater than the start address.

4.6 Cassette operating system routines

The CBM 64 kernal contains a whole series of routines for handling data transfer between the processor and the cassette unit. The following sections describe these routines and how they can be used. These descriptions are accompanied by annotated source code listings of each routine (consult volume 1 of this series, *The Commodore 64 ROMs Revealed*, for the full kernal source code listing). The variable declaration file for these routines is to be found in Chapter 5.
Protect cassette/serial from RS232 NMI interrupts

Entry point: $F0A4

Function: This routine checks location $02A1 to see if RS232 communications are enabled. If they are not then this location contains a zero and the routine exits to allow serial or cassette operation to commence. If RS232 communications are enabled then the routine goes into a loop waiting for the RS232 NMI interrupt to reset location $02A1 to $03, thereby signalling its completion. As soon as this happens the Timer A interrupt is disabled, the flag at $02A1 is set to zero and the routine exits. The reason for this routine is to prevent an NMI interrupt from occurring during cassette or serial operation thereby causing data loss.

Input parameters: $02A1 – if non zero then RS232 enabled

Output parameters: None

Registers used: .A is used but is pushed to the stack by the instruction at $F0A4 and then retrieved at the end of the routine by $F0BB.

Routine source code:

<table>
<thead>
<tr>
<th>LOC</th>
<th>CODE</th>
<th>LINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D010</td>
<td>$9A4</td>
<td>Entry point: $F0A4</td>
</tr>
<tr>
<td>$F0A4</td>
<td>;</td>
<td></td>
</tr>
<tr>
<td>$F0A4</td>
<td>;PROTECT SERIAL/CASSETTE FROM RS-232 NMI'S</td>
<td></td>
</tr>
<tr>
<td>$F0A4</td>
<td>;</td>
<td></td>
</tr>
<tr>
<td>$F0A4</td>
<td>4B</td>
<td>L921PHA;SAVE A</td>
</tr>
<tr>
<td>$F0A5</td>
<td>AD A1 02</td>
<td>LDA $02A1;RS232 ENABLES?</td>
</tr>
<tr>
<td>$F0A8</td>
<td>F0 11</td>
<td>BEQ L923;NO</td>
</tr>
<tr>
<td>$F0AA</td>
<td>AD A1 02</td>
<td>L83B</td>
</tr>
<tr>
<td>$F0AD</td>
<td>27 03</td>
<td>AND $03</td>
</tr>
<tr>
<td>$F0AF</td>
<td>D0 F9</td>
<td>BNE L83B</td>
</tr>
<tr>
<td>$F0B1</td>
<td>A9 10</td>
<td>LDA $010;DISABLE FLAG</td>
</tr>
<tr>
<td>$F0B3</td>
<td>BD 0D DD</td>
<td>STA DZICK</td>
</tr>
<tr>
<td>$F0B6</td>
<td>A9 00</td>
<td>LDA $000</td>
</tr>
<tr>
<td>$F0B8</td>
<td>BD A1 02</td>
<td>STA $02A1</td>
</tr>
<tr>
<td>$F0B8</td>
<td>60</td>
<td>L923PLA;ALL DONE</td>
</tr>
<tr>
<td>$F0BC</td>
<td>60</td>
<td>RTS</td>
</tr>
<tr>
<td>$F0BD</td>
<td>;</td>
<td></td>
</tr>
</tbody>
</table>

Cassette error message output

Entry point:
$F12B – tests direct mode flag first
$F12F – displays message to screen

Function: This routine outputs a message to the screen concerning cassette
operation. The first entry point tests the contents of location $9D to see if the output is in direct or run mode. The second entry point performs the actual message output, the choice of message being determined by the value in the $Y index register. The messages used by this routine are stored in the area of memory immediately above this routine starting at $F0BD.

**Input parameters:**

$Y index register contains message number $9D – direct mode flag; if the high bit of $A is set and the contents of location $9D are non zero, the required message is printed.

**Output parameters:** None

**Routine source code:**

```assembly
LOC CODE LINE

F0BD ;ERROR MESSAGES
F08D ;
F0BD 0D $.01 .BYT $D,'I/O ERROR',+$A3
F0BE 49 $F
F0CB $A3
F0C9 0D $.05 .BYT $D,'SEARCHING',$A0
F0CA 33 $43
F0D3 $A0
F0D4 46 $4F $52 $.06 .BYT 'FOR',$A0
F0D7 $A0
F0DB 0D $.07 .BYT $D,'PRESS PLAY ON TAP',+$C5
F0D9 50 $52
F0EA $C5
F0EB 50 $52 $.08 .BYT 'PRESS RECORD & PLAY ON TAP',+$C5
F105 $C5
F106 0D $.10 .BYT $D,'LOADING',$C7
F107 4C $4F
F10D $C7
F10E 0D $.11 .BYT $D,'SAVING',$A0
F10F 53 $41
F115 $A0
F116 0D $.12 .BYT $D,'VERIFYING',$C7
F117 56 $45
F11F $C7
F120 0D $.14 .BYT $D,'FOUND',$A0
F121 46 $4F
F126 $A0
F127 0D $.16 .BYT $D,'OK',$8D
F128 4F $4B
F12A $8D
F12B ;PRINT MESSAGE TO SCREEN ONLY IF
F12B ;OUTPUT ENABLED
F12B ;
F12B 24 $9D L922 BIT MSGFLG ;PRINTING MESSAGES?
F12D 10 0D BPL L925 ;NO
F12F 09 BD F0 L1073 LDA MS1,Y
F132 $0B
F133 29 $7F PHP
F135 29 $D2 FF JSR $FFD0
F138 $C8
F139 28
F13A 10 $F3 BPL L1073
F13C 18 L925 CLC
F13D 60 RTS
F13E .END .LIB KTape1
```
Load RAM function

Entry point: $F49E

**Function:** This routine loads from cassette or a serial bus device (with a device number between 4 and 31 where this device number is stored in location $BA$) into the memory starting at the LOAD address in the file if the secondary address is greater than 0, or at the specified address if the secondary address is 0.

**Input parameters:**
- $BA$ – device number
- $B9$ – secondary address
- $.X$ – LOAD address lo if secondary address is zero
- $.Y$ – LOAD address hi if secondary address is zero
- $.A$ – if $= 0$ then load, $\neq 0$ then verify

**Output parameters:**
- $.X$ – return high LOAD address hi
- $.Y$ – return high LOAD address

**Routine source code:**

```
LOC CODE LINE
F13E  ==$F49E
F49E  
F49E  ;******************************************************
F49E  ; LOAD RAM FUNCTION.
F49E  ;* LOADS FROM CASSETTE OR SERIAL BUS
F49E  ;* DEVICES >4 TO 31 AS DETERMINED BY
F49E  ;* CONTENTS OF VARIABLE FA, VERIFY FLAG
F49E  ;* IN .A.
F49E  ;* ALT LOAD IF SA=0, NORMAL SA=1
F49E  ;* .X, .Y LOAD ADDRESS IF SA=0
F49E  ;* .A=0 PERFORMS LOAD,<>0 IS VERIFY.
F49E  ;* HIGH LOAD RETURN IN .X,.Y
F49E  ;* USE SETLFS & SETNAM BEFORE THIS ROUTINE
F49E  ;******************************************************
F49E  
F49E  86 C3 L990 STX MEMUSS ;LO ALT START
F4A0  84 C4 STY MEMUSS+1 ;HI ALT START
F4A2  6C 30 03 JMP (ILDAD)
F4A5  85 93 wLOAD STA VERCK ;STORE VERIFY FLAG
F4A7  A9 00 LDA $00
F4A9  B5 70 STA STATUS
F4AB  A5 8A LDA FA ;CHECK DEVICE #
F4AD  D0 03 BNE L1046
F4AF  4C 13 F7 L1241 JMP L1049 ;KEYBOARD, BAD DEVICE
F4B2  C9 93 L1046 CMP $03 ;SCREEN?
F4B4  F0 F9 BEQ L1241 ;YES
F4B6  90 7B BCC L1050 ;TAPE
F4BB  ;---------------------------
F4BB  
F4BB  ==$F030
F530  4C 04 F7 L1058 JMP L059 ;FILE NOT FOUND
F533  
F533  ;***** LOAD FROM TAPE
F533  
```
The Cassette Units

```
LOC   CODE     LINE   ;TAPE?
F333  4A     L1050    LSR A       ;YES
F334  B0     BCS L1047  ;NO, BAD DEVICE
F336  4C 13  F7   JMP L1049  
F338  20  D0  F7   L1047    JSR L1104  ;SET TAPE POINTERS
F33C  B0  03    BCS L1060
F33E  4C 13  F7   JMP L1049  ;DEALLOCATED
F341  20  17  F8   L1060    JSR L918  ;'PRESS PLAY ON TAPE'
F344  B0  6B    BCS L1059  ;STOP KEY?
F346  20  AF  F5   JSR L1062  ;'SEARCHING'
F349  A5  B7    L1061    LDA FNLEN  ;NAME?
F34B  F0  09    BEQ L1066  ;NO, LOAD FIRST PROG
F34D  20  EA  F7   JSR L1108  ;YES, FIND A FILE
F350  90  0B    BCC L1063  ;FOUND
F352  F0  5A    BEQ L1059  ;STOP KEY
F354  B0  DA    BCS L1068  ;NO, END OF TAPE
F356  20  2C  F7   L1066    JSR L1098  ;FIND ANY HEADER
F359  F0  33    BEQ L1059  ;STOP KEY
F35B  B0  D3    BCS L1058  ;NO HEADER
F35D  A5  90    L1063    LDA STATUS  ;MUST HAVE GOT HEADER RIGHT
F35F  29  10  AND #SPWRK
F361  38  8C
F362  D0  4A    BNE L1059  ;IS BAD
F364  E0  01    CPX #BLF  ;MOVEABLE?
F366  F0  11    BEQ L1068  ;YES
F368  E0  03    CPX #PLF  ;PROGRAM
F36A  D0  DD    BNE L1061  ;NO, TRY FOR NEXT
F36C  A0  01    L1064    LDA #$01  ;FIXED LOAD
F36E  B1  B2    LDA (TAPE1),Y  ;ADDRESS IN BUFFER
F370  83  C3    STA MEMUSS  ;IS LOAD ADDRESS
F372  CB  36
F373  B1  B2    LDA (TAPE1),Y
F375  85  C4    STA MEMUSS+1
F377  B0  04    BCS L1065
F379  A5  B9    L1068    LDA SA  ;MONITOR LOAD?
F37B  D0  EF    BNE L1064  ;YES, FIXED TYPE
F37D  A0  03    L1065    LDA #$03  ;TAPEA-TAPESTA
F37F  B1  B2    LDA (TAPE1),Y
F381  A0  01    LDY #$01
F383  F1  B2    SBC (TAPE1),Y
F385  AA  8A    TAY  ;LO TO .X
F386  A0  04    LDY #$04
F38B  B1  B2    LDA (TAPE1),Y
F38A  A0  02    LDY #$02
F38C  F1  B2    SBC (TAPE1),Y
F38E  A8  8A    TAX  ;HI TO Y
F38F  18  7C    CLC  ;EA=STA+(TAPEA-TAPESTA)
F390  B0  8A    TYA
F391  65  C3    ADC MEMUSS
F393  85  AE    STA EAL
F395  9B  TYA
F396  A5  C4    ADC MEMUSS+1
F39B  85  AF    STA EAH
F39A  A5  C3    LDA MEMUSS  ;SET UP START ADDRESS
F39C  85  C1    STA STA
F39E  A5  C4    LDA MEMUSS+1
F3A0  B5  C2    STA STA
F3A2  20  4A  F8   JSR L1070  ;'LOADING'
F3A5  20  4A  F8   JSR L940  ;LOAD TAPE BLOCK
F3A8  24  8B
F3A9  18    L1067    CLC  ;GOOD EXIT
F3AA  
F3AA  ;SET UP END ADDRESS
F3AA  
F3A9  A6  AE    LDX EAL
F3AC  A4  AF    LDY EAH
F3AE  60    L1059    RTS
F3AF  
```
Print tape loading messages

Entry points:
$F5AF - print 'searching [for filename]'
$F5C1 - print filename
$F5D2 - print loading/verifying

Function: The function of these three routines is simply to display the appropriate messages on the screen when loading a program or file from tape.

Input parameters:
$9D - flag to indicate whether 'searching [for filename]' is printed; if high bit is set then message is printed
$B7 - filename length
$BB - filename address
$93 - loading/verifying flag; if = 0 then loading, otherwise verifying

Output parameters: None

Routine source code:

LOC    CODE    LINE
F5AF    ;PRINT 'SEARCHING FOR [NAME]'  
F5AF    
F5AF    A 9 9D   L1062  LDA MSGFLG   ;PRINT IT?
F5B1    10 1E   BPL L1071   ;NO
F5B3    A0 0C   LDY #MS5-MS1   ;'SEARCHING'
F5B5    20 2F F1  JSR L1073
F5BB    A5 87   :LDA FNLEN
F5BA    F0 15   BEQ L1071
F5BC    A0 17   LDY #MS6-MS1   ;'FOR'
F5BE    20 2F F1  JSR L1073
F5C1    
F5C1    ;PRINT FILENAME
F5C1    
F5C1    A4 B7   L1022  LDY FNLEN   ;NAME LENGTH
F5C3    F0 0C   BEQ L1071   ;NO NAME
F5C5    A0 00   LDY #000
F5C7    B1 B8   L1091  LDA (FMADK),Y
F5C9    20 02 FF  JSR #FFD2
F5CC    CB FF   INY
F5CD    C4 B7   CPY FNLEN
F5CF    D0 F6   BNE L1091
F5D1    60 1071  RTS
F5D2    
F5D2    ;PRINT LOADING/VERIFYING
F5D2    
F5D2    A0 49   L1070  LDY #MS10-MS1   ;ASSUME 'LOADING'
F5D4    A5 93   LDA VERCK   ;CHECK FLAG
F5D6    F0 02   BEQ L1052   ;'YES, LOADING'
F5D8    A0 59   LDY #MS21-MS1   ;'VERIFYING'
F5DA    4C 2B F1  L1052  JMP L922
F5DD    .END
F5DD    .LIB KTAP2
Save memory function

Entry point: $F5DD

Function: A specified block of memory is saved by this routine onto cassette or a serial device with a device number between 4 and 31. This routine must be preceded by the routine at $FFBA which sets logical first and secondary addresses and at $FFBD which sets up the filename.

Input parameters:
.A - indirect pointer to start of memory area to be saved
.X - end of SAVE lo
.Y - end of SAVE hi
$BA - device number

Output parameters: None

Routine source code:

LOC CODE LINE

F5DD ;*******************************************************************************
F5DD ;* SAVE MEMORY FUNCTION.
F5DD ;* SAVES TO CASSETTE OR SERIAL
F5DD ;* DEVICES >=4 TO 31 AS SELECTED BY
F5DD ;* VARIABLE FA.
F5DD ;* START OF SAVE IS INDIRECT AT .A
F5DD ;* END OF SAVE IS .X,.Y
F5DD ;* USE SETLFS & SETNAM BEFORE THIS ROUTINE
F5DD ;*******************************************************************************
F5DD :
F5DD B6 AE L1072 STX EAL ;STORE END ADDRESS
F5DF B4 AF STY EAH ;SET UP START
F5E1 AA TAX ;SET START
F5E2 B5 00 LDA $00,X
F5E4 B5 C1 STA STAHL
F5E6 B5 01 LDA $01,X
F5EB B5 C2 STA STAHI
F5EA 6C 32 03 JMP (ISAVE)
F5ED A5 BA NSAVE LDA FA
F5EF D0 03 BNE L1075
F5F1 4C 13 F7 L1242 JMP L1049 ;BAD DEVICE
F5F4 C9 03 L1075 CMP #03 ;SERIAL?
F5F6 F0 F9 BNE L1242 ;SCREEN, BAD DEVICE
F5FB 99 SF BCC L1085 ;NO, TAPE
F5FA ;----------------------------------
F5FA **=F659
F659 ;***** TAPE SAVE
F659 :
F659 4A L1085 LSR A ;RS-232?
F65A D0 03 BCS L1076 ;NO, MUST BE TAPE
F65C 4C 13 F7 JMP L1049 ;BAD DEVICE
F65F 20 D0 F7 L1076 JSR L1104 ;GET BUFFER ADDR
F662 90 8D BCC L1242 ;NOT ALLOCATED
F664 20 38 F8 JSR L1114
F667 B0 25 BCS L1090 ;STOP KEY
F669 20 BF F6 JSR L1087 ;'SAVING'
Print 'saving'

**Entry point:** $F68F

**Function:** Prints the message Saving [filename] on the screen. Note that this message can be output to another device such as a printer, but the following SAVE will give an error.

**Input parameters:** $9D – flag to indicate if message is to be printed; if high bit is not set then message is not printed.

**Output parameters:** None

**Routine source code:**

```
LOC CODE   LINE
F68C A2 03   LDX MPLF ;DECIDE TYPE TO SAVE
F68E A5 89   LDA SA ;1-PLF, 0-BLF
F679 29 01   AND $01
F672 D0 02   BNE L1086
F674 A2 01   LDX $8BF
F676 8A L1086 TXA
F677 20 6A F7 JSR L1099 ;WRITE HEADER BLOCKS
F67A B0 12   BCS L1090 ;STOP KEY
F67C 20 67 F8 JSR L952 ;WRITE PROGRAM BLOCKS
F67F 8A 0D   BCS L1090 ;STOP KEY
F681 A5 B9   LDA SA
F683 29 02   AND $02 ;WRITE END OF TAPE?
F685 F9 96   BEQ L1088 ;NO
F687 A9 05   LDA $005
F689 20 6A F7 JSR L1099 ;WRITE END TABLE BLOCKS
F68C 24 .BYT $24 ;SKIP COMMAND
F68D 18 L1088 CLC
F68E 60 L1090 RTS
```

Stop key servicing

**Entry point:** $F6ED

**Function:** This routine is included in this section because it is called by so many
of the other routines. The function of this routine is to check the stop key flag and if set then close any active I/O channels, flush the keyboard queue and return the machine to direct mode.

Input parameters: $91 – value of last keyboard row – contains 'stop' key

Output parameters:
Z flag – set if stop key depressed
.A – keys depressed from last keyboard row

Routine source code:

```
LOC CODE LINE

F69B ==F6ED
F6ED  
F6ED ;**************************
F6ED ;* STOP -- CHECK STOP KEY FLAG AND
F6ED ;* RETURN Z FLAG SET IF FLAG TRUE.
F6ED ;* ALSO CLOSES ACTIVE CHANNELS AND
F6ED ;* FLUSHES KEYBOARD QUEUE.
F6ED ;* ALSO RETURNS KEY DOWNS FROM LAST
F6ED ;* KEYBOARD ROW IN .A.
F6ED ;* SHOULD CALL UPDATE TIME BEFORE
F6ED ;* THIS.
F6ED ;**************************
F6ED
```

Error handler

Entry points:
$F6FB - too many files
$F6FE - file open
$F701 - file not open
$F704 - file not found
$F707 - device not present
$F70A - not input file
$F70D - not output file
$F710 - missing filename
$F713 - illegal device number

Function: This will display a designated error message from a list of nine cassette and serial I/O related messages. The table of actual error message texts is stored in locations $A19E to $A225.
Find any tape header

**Entry point:** $F72C

**Function:** This routine reads the tape device until one of the following two block types is found: ‘basic data file header’ or ‘basic load file’. The state of the carry flag indicates whether a header was found or not. Having found the header the message Found is displayed, followed by the filename from the header. A pause of 8.5 seconds is then generated before the routine exits to perform the rest of the load. This delay can be eliminated by pressing the CBM key.
Input parameters: None

Output parameters:
A = 0 if stop key pressed
Carry flag = clear = header found; set = header not found

Routine source code:

```assembly
F72C A5 93  L1098 LDA VERCK ;SAVE OLD VERIFY
F72E 4B  PHA
F72F 20 41 FO  JSR L1029 ;READ TAPE BLOCK
F732 68  PLA
F733 B5 93  STA VERCK ;RESTORE VERIFY
F735 B0 32  BCS L1191 ;READ TERMINATED
F737 A0 00  LDY #$00
F739 B1 B2  LDA (TAPE1),Y ;GET HEADER TYPE
F73B C9 05  CMP #000 ;END OF TAPE?
F73D F8 2A  BEQ L1101 ;YES
F73F C9 00  CMP #MPLF ;BASIC DATA FILE?
F741 F0 00  BEQ L1027 ;YES
F743 C9 03  CMP #NPLF ;FIXED LOAD FILE?
F745 F0 04  BEQ L1027 ;YES
F747 C9 04  CMP #MBDFH ;BASIC LOAD FILE?
F749 D0 E1  BNE L1098 ;NO, TRY AGAIN
F74B AA  L1027 TAX ;FILE TYPE IN .X
F74C 24 70  BIT #SGFLG ;PRINT MESSAGE?
F74E 10 17  BPL L1102 ;NO
F750 A0 63  LDY #MG17-MS1 ;'FOUND'
F752 20 2F F1  JSR L1073
F755 A9 95  LDY #$05
F757 B1 B2  L1100 LDA (TAPE1),Y ;OUTPUT COMPLETE
F759 29 D2 FF  JSR #$F02 ;FILENAME
F75C CB  INY
F75D C0 15  CPY #$15
F75F D0 F6  BNE L1100
F761 A5 A1  LDA TIME+1 ;WAIT FOR 0.5 SECONDS
F763 20 E0 E4  JSR #$E4E0 ;OR FOR THE CBM KEY
F766 EA  NOP
F767 1B  L1102 CLC ;SUCCESS
F768 88  DEY
F769 60  L1101 RTS
```

Write tape header

Entry point: $F76A

Function: This routine first pushes the program start and end addresses onto the
stack and then blanks the tape buffer memory area and sets up a tape header with all the requisite information being stored in the correct position in the tape buffer. The tape buffer contents are then written to tape and the start and end addresses restored off the stack.

**Input parameters:** All tape header variables and filename

**Output parameters:** .A – tape SAVE error flag

**Routine source code:**

```
LOC    CODE   LINE

F76A   B5 9E   L1099 STA T1
F76C   20 D0 F7 JSR L1104 ;GET BUFFER ADDRESS
F76F   90 5E   BCC L1106 ;NOT ALLOCATED
F771   A5 C2   LDA STA1 ;PREVIOUS START AND CODES
F773   48     PHA ; ADDRESSES
F774   A5 C1   LDA STA1
F776   48     PHA
F777   A5 AF   LDA EA1
F779   48     PHA
F77A   A5 AE   LDA EAI
F77C   48     PHA
F77D   A9 RF   LDY #BUFFSZ-1 ;BLANK TAPE BUFFER
F77F   A9 20   LDA #%20 ;SPACE CHAR
F781   91 B2   STA (TAPE1),Y
F783   B8     DEY
F784   D0 FB   BNE L998 ;BLOCK TYPE IN HEADER
F786   A5 9E   LDA T1
F788   91 B2   STA (TAPE1),Y
F790   A5 AE   LDA EAL
F792   91 B2   STA (TAPE1),Y
F794   C8     INY ;END ADDRESS IN HEADER
F795   A5 AE   LDA EAL
F797   91 B2   STA (TAPE1),Y
F799   C8     INY
F79A   A5 AF   LDA EA1
F79C   91 B2   STA (TAPE1),Y
F79E   C8     INY ;FILENAME IN HEADER
F79F   B4 9F   STY T2
F7A1   A0 00   LDY #$00
F7A3   B4 9E   STY T1
F7A5   A4 9E   L1105 LDY T1
F7A7   C4 B7   CPY FNLEN
F7A9   F0 0C   BEQ L1107
F7AB   B1 B8   LDA (FNADR),Y
F7AD   A4 9F   LDY T2
F7AF   91 B2   STA (TAPE1),Y
F7B1   E6 9E   INC T1
F7B3   E6 9F   INC T2
F7B5   D0 EE   BNE L1105
F7B7   20 D7 F7 JSR L995 ;SET UP START & END
F7BA   A9 69   LDA #$69 ;ADDR OF HEADER & SET
F7BC   85 AB   STA SHCNH ; TIME FOR LEADER
```
Return buffer address

**Entry point:** $F7D7

**Function:** This routine is in two parts; the first tests if the tape buffer is allocated and the second calculates the start and end address pointers which are required by the SAVE routines.

**Input parameters:** None

**Output parameters:**
- $C1$ – start address lo
- $C2$ – start address hi
- $AE$ – end address lo
- $AF$ – end address hi

**Routine source code:**

```
F7D0 00          ;RETURN BUFFER ADDRESS
F7D2 64 B3      LDY TAPE1+1
F7D4 C0 92      CPY #$02  ;ALLOCATED?
F7D6 60          RTS  ;CARRY CLEAR, DE-ALLOCATED
F7D7 20 D0 F7 L1104 JSR L1089 ;GET PTR TO CASSETTE
F7DA 8A          TXA
F7DB 85 C1      STA EAL  ;SAVE START LOW
F7DC 69       CLC
F7DE 69 C0  ADC #BUFSZ ;COMPUTE POINTER TO END
F7E0 85 AE      STA EAL  ;SAVE END LOW
F7E2 98          TYA
F7E4 85 C2      STA STAHI ;SAVE START HI
F7E5 69 00    ADC #000  ;COMPUTE POINTER TO END
F7E7 85 AF      STA EAH  ;SAVE END HIGH
F7E9 40          RTS
```

Find correct file on tape
The Commodore 64 Kernel and Hardware Revealed

Entry point: $F7EA

Function: This routine searches for a program header on tape. Having found a header the filename is compared with that specified (if the contents of location $B7 are zero then the first program encountered is loaded). If the program name in the header is not the same as that specified then the routine searches for the next header. It should be noted that this routine only compares the header filename for the number of characters in the filename specified in the LOAD/VERIFY command, thus if the filename Test is specified in the LOAD command but the header contains the filename Testing, then the routine will take this as a positive match and load Testing.

Input parameters: $B7 – length of current filename string

Output parameters: None

Routine source code:

```
LOC CODE LINE

F7EA 2B 2C F7 L1108 JSR L1098 ;FIND ANY HEADER
F7ED B0 10 8CS L1111 ;FAILED
F7EF A0 05 LDY #$05 ;CHECK NAME
F7F1 84 9F STY T2 ;OFFSET TO HEADER
F7F3 A0 00 LDY #$00
F7F5 84 9E STY T1 ;OFFSET TO NAME
F7F7 C4 B7 L1024 CPY FNLEN ;COMPARE THIS MANY
F7F9 F0 10 BEQ L1112 ;DONE
F7FB B1 BB LDA (FNADR),Y
F7FD A4 9F LDY T2
F7FF D1 B2 CMP (TAPE1),Y
F801 D0 E7 BNE L1108 ;WRONG FILENAME
F803 E6 9E INC T1
F805 E6 9F INC T2
F807 A4 9E LDY T1
F809 D0 EC BNE L1024 ;ALWAYS
F80B 10 L1112 CLC ;SUCCESS
F80C 60 L1111 RTS
F80D .END .LIB KTAPE5
```

Miscellaneous cassette support routines

Entry points:
$F80D – increase pointer in tape buffer
$F817 – wait for play switch
$F82E – test cassette switch
$F838 – check for record and play
$F841 – read header block
The Cassette Units

$F84A - read LOAD block entry

Routine source code:

<table>
<thead>
<tr>
<th>LOC</th>
<th>CODE</th>
<th>LINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>F800</td>
<td>;INCREASE POINTER IN TAPE BUFFER</td>
<td></td>
</tr>
<tr>
<td>F800</td>
<td>20 D0 F7</td>
<td>L1110 JSR L1104</td>
</tr>
<tr>
<td>F810</td>
<td>E6 A6</td>
<td>INC BUFPT</td>
</tr>
<tr>
<td>F812</td>
<td>A6 A6</td>
<td>LDY BUFPT</td>
</tr>
<tr>
<td>F814</td>
<td>C0 C0</td>
<td>CPY #BUFSZ</td>
</tr>
<tr>
<td>F816</td>
<td>60</td>
<td>RTS</td>
</tr>
<tr>
<td>F817</td>
<td>;WAIT FOR PLAY SWITCH</td>
<td></td>
</tr>
<tr>
<td>F817</td>
<td>20 2E F8</td>
<td>L938 JSR L1116</td>
</tr>
<tr>
<td>F81A</td>
<td>F8 1A</td>
<td>BEQ L1113</td>
</tr>
<tr>
<td>F81C</td>
<td>A0 1B</td>
<td>LDX #$07-MS1</td>
</tr>
<tr>
<td>F81E</td>
<td>20 2F F1</td>
<td>L1020 JSR L1073</td>
</tr>
<tr>
<td>F821</td>
<td>20 D0 F8</td>
<td>L1117 JSR L1120</td>
</tr>
<tr>
<td>F822</td>
<td>20 2E F8</td>
<td>JSR L1116</td>
</tr>
<tr>
<td>F827</td>
<td>D0 F8</td>
<td>BNE L1117</td>
</tr>
<tr>
<td>F829</td>
<td>A0 6A</td>
<td>LDX #$01-MS1</td>
</tr>
<tr>
<td>F82C</td>
<td>4C 2F F1</td>
<td>JMP L1073</td>
</tr>
<tr>
<td>F830</td>
<td>20 17 F8</td>
<td>JSR L940</td>
</tr>
<tr>
<td>F83A</td>
<td>20 1F</td>
<td>LDA #$F9</td>
</tr>
<tr>
<td>F840</td>
<td>7B</td>
<td>SEI</td>
</tr>
<tr>
<td>F859</td>
<td>A9 00</td>
<td>LDA #$00</td>
</tr>
<tr>
<td>F85A</td>
<td>85 90</td>
<td>STA STATUS</td>
</tr>
<tr>
<td>F85B</td>
<td>85 93</td>
<td>STA VERCK</td>
</tr>
<tr>
<td>F85C</td>
<td>20 D7 F7</td>
<td>JSR L995</td>
</tr>
<tr>
<td>F84A</td>
<td>20 17 F8</td>
<td>L940 JSR L938</td>
</tr>
<tr>
<td>F84B</td>
<td>80 1F</td>
<td>BCS L1109</td>
</tr>
<tr>
<td>F84F</td>
<td>7B</td>
<td>SEI</td>
</tr>
<tr>
<td>F859</td>
<td>A9 00</td>
<td>LDA #$00</td>
</tr>
<tr>
<td>F85A</td>
<td>85 90</td>
<td>STA STATUS</td>
</tr>
<tr>
<td>F85B</td>
<td>85 93</td>
<td>STA VERCK</td>
</tr>
<tr>
<td>F85C</td>
<td>85 90</td>
<td>STA PTR2</td>
</tr>
<tr>
<td>F85D</td>
<td>85 9C</td>
<td>STA DPSW</td>
</tr>
</tbody>
</table>

Write memory
The Commodore 64 Kernel and Hardware Revealed

Entry point:
$F864 – write tape buffer
$F867 – write memory between start address and end address

Function: The first entry point sets up the addresses to SAVE the tape buffer. The second entry point is to the main tape write routine. This routine writes the contents of memory between the previously determined start and end addresses onto tape. This routine calls up several small routines located at $FBA6.

Input parameters:
$C1 – start address lo
$C2 – start address hi
$AE – end address lo
$AF – end address hi

Output parameters: None

Routine source code:

LOC  CODE  LINE
F864  ;
F864  ;**************************************
F864  ;* WRITE TAPE BUFFER
F864  ;**************************************
F864  
F864  ;SET UP TO SAVE TAPE BUFFER
F864  :
F864  20 D7 F7 L1069 JSR L995 ;BUFFER
F867  ;**************************************
F867  ;WRITE MEMORY BETWEEN STA1,STAH
F867  ;AND EAL,EAH AS A BLOCK
F867  ;**************************************
F867  
F867  A9 14 L952. LDA $14 ;BETWEEN BLOCK SHORTS
F869  85 AB STA SHCNH
F868  29 3B F8 L1089 JSR L1114 ;'PRESS RECORD...'
F86E  80 6C L1109 BCS L1115 ;STOP KEY
F870  78 SEI
F871  A9 82 LDA $$82 ;ENABLE T2 IRQ
F873  A2 08 LDX $$08 ;POINT IRQ VECTOR TO WRITE
F875  ;
F875  ;START TAPE OPERATION ENTRY POINT
F875  :
F875  A0 7F L1118 LDD $$7F ;KILL UNWANTED IRQ
F877  BC 0D DC STY D1ICR
F87A  BD 0D DC STA D1ICR ;ENABLE WANTED
F87D  AD 0E DC LDA D1CRa
F880  69 19 ORA $$19
F882  BD 0F DC STA D1CRb
F885  29 91 AND $$91
F887  BD A2 02 STA $$02A2
F88A  20 A4 F0 JSR L921 ;WAIT FOR RS232
F88D  AD 11 09 LDA VICREG+17 ;BLANK SCREEN
F890  29 EF AND $$EF
F892  BD 11 09 STA VICREG+17
F895  AD 14 03 LDA CINV ;MOVE IRQ TO IRQ TEMP
F898  BD 09 02 STA IRGMP ;FOR CASETTE OPS
F89B  AD 15 03 LDA CINV+1 ;
F89E  BD A0 02 STA IRGMP+1
F8A1  20 BD FC JSR L1195 ;CHANGE IRQ VECTOR
F8A4  A9 02 LDA $$02 ;FSBLK STARTS AT 2
F8A6  85 BE STA FSBLK
F8A8  20 97 FB JSR L1079 ;PREPARE LOCAL COUNTERS
The Cassette Units

LOC CODE LINE

F8A6 A5 01 LDA $01 ;TURN CASSETTE MOTOR ON
F8A8 29 1F AND #$1F
F8A9 85 01 STA $01
F8B1 85 C0 STA CAS1 ;FLAG INTERNAL CONTROL
F8B3 A2 FF LDX #$FF ;DELAY BETWEEN BLOCKS
F8B5 A0 FF L1119 LDY #$FF
F8B7 88 L1124 DEY
F8B8 D0 FD BNE L1124
F8BA CA DEX
F8BB D0 F8 BNE L1119
F8BD  ; ENABLE TAPE IRQ ROUTINES TO
F8BE ;START WRITE OPERATION
F8BF 58 CLI
F8BE AD A0 02 L1123 LDA IRQTMP+1 ;CHECK FOR IRQ VECTOR
F8C1 CD 15 03 CMP CINV+1 ;POINTING AT WRITE ROUTINE
F8C4 18 CLC
F8C5 F0 15 BEQ L1125 ;YES, RETURN
F8C7 20 D0 F8 JSR L1125 ;NO CHECK STOP
F8CA 20 8C F6 JSR $F68C ;UPDATE TIME
F8CD 4C 8E F8 JMP L1123 ;STAY IN LOOP
F8D0 20 E1 FF L1125 JSR $FFE1 ;TOP KEY DOWN?
F8D3 18 CLC ;ASSUME NOT
F8D4 D9 0B BNE L1120 ;CORRECT ASSUMPTION
F8D6 20 93 FC JSR L1192 ;STOP DOWN STOP TAPE
F8D9 38 SEC ;FAILED
F8DA 68 PLA ;BACK ON RTS
F8DB 68 PLA
F8DC A9 00 L1115 LDA $000 ;DISABLE IRQTMP
F8DE 80 A0 02 STA IRQTMP+1
F8E1 60 L1120 RTS

Set up time out watch for next dipole

Entry point: $F8E2

Function: This routine is used to detect read errors by checking the timing of each pulse pair (dipole); if the pulses are too long then a time out error is assumed.

Input parameters: .X – time out constant for particular dipole

Output parameters: None

Routine source code:

LOC CODE LINE

F8E2  ;SET UP TIMEOUT WATCH FOR NEXT DIPOLE
F8E2 86 81 L1126 STX TEMP ;TIMEOUT CONSTANT
F8E4 A5 80 LDA CINV ;CINV+5
F8E6 0A ASL A
F8E7 0A ASL A
F8EB 18 CLC
F8EB 65 80 ADC CINV
F8EB 18 CLC
The Commodore 64 Kernal and Hardware Revealed

Cassette read subroutines

Entry point: $F92C

Function: This is the main routine which reads data from the tape. The bulk of the routine performs the timing of the incoming pulses in order to decode the pulse type and to give a software servo loop, which adjusts the timing of the pulses to the speed of the cassette deck. To understand the timing of the pulses see the waveform diagrams in Fig. 4.3 plus the documentation accompanying the source code listing.

Input parameters: None

Output parameters: $B6 – tape read error

Routine source code:

```
LOC CODE LINE
F92C 65 B1 ADC TEMP ;ADJUST LONG BYTE COUNT
F92C 65 B1 STA TEMP
F92C A9 00 LDA #$00
F92C 24 80 BIT CMPO
F92C 30 01 BMI L1146 ;MINUS, NO ADJUST
F92C 2A ROL A
F92C 96 B1 L1146 ASL TEMP ;MULTIPLY CORRECTED
F92C 2A ROL A ;VALUE BY 4
F92C 06 B1 ASL TEMP
F92C 2A ROL A
F92C AA TAX
F92C AD 06 DC L1128 LDA D11BL ;WATCH OUT FOR ROLLOVER
F92C C9 16 CMP #$16 ;TIME FOR ROUTINE?
F92C 90 F9 BCC L1128 ;TOO CLOSE SO WAIT
F92C 85 B1 ADC TEMP ;CALCULATE AND
F92C 8D 04 DC STA D11AH ;STORE ADJUSTED TIME COUNT
F92C 8A TXA ;ADJUST FOR HI TIME COUNT
F92C 0D 07 DC ADC D11BH
F92C 8D 05 DC STA D11AH
F92C AD A2 02 LDA $02A2
F92C 8D 0E DC STA D1CRA
F92C 8D A4 02 STA $02A4
F92C AD 0D DC LDA D11CR
F92C 29 10 AND #$10
F92C F3 09 BEQ L1129
F92C 99 F9 LDA #$F9
F92C 48 PHA
F92C 49 A9 2A LDA #$2A
F92C 6A 48 PHA
F92C 4C 43 FF JMP $FF43
F92C 5A 00 L1129 CLI
F92C 60 60 RTS
F92C 00 00
F92C .END .LIB KTAPE6
```

Cassette read subroutines

Entry point: $F92C

Function: This is the main routine which reads data from the tape. The bulk of the routine performs the timing of the incoming pulses in order to decode the pulse type and to give a software servo loop, which adjusts the timing of the pulses to the speed of the cassette deck. To understand the timing of the pulses see the waveform diagrams in Fig. 4.3 plus the documentation accompanying the source code listing.

Input parameters: None

Output parameters: $B6 – tape read error

Routine source code:
The Cassette Units

`F92C  AE 07 DC  L1130  LDX DITBH  ;GET TIME SINCE LAST IRQ`

`F92C  AU FF  LDY #FF  ;COMPUTE COUNTER DIFF`

`F931  98  TYA`

`F932  ED 0a DC  SBC DITBL`

`F935  EC 07 DC  CPX DITBH  ;TIMER HIGH KOLLOVER?`

`F938  D0 F2  BNE L1130  ;YES, RECOMPUTE`

`F93A  86 B1  STX TEMP`

`F93C  AA  TAX`

`F93D  BC 06 DC  STY DITBL  ;RE-LOAD TIMER B`

`F940  BC 07 DC  STY DITBH`

`F943  A9 19  LDA #19`

`F945  BD 0F DC  STA DICTB`

`F948  AD 0B DC  LDA DICTC`

`F94B  AD 03 02  STA #02NJ`

`F94E  9B  TTA`

`F94F  E5 B1  SBC TEMP  ;CALCULATE HIGH`

`F951  06 B1  STX TEMP`

`F953  4A  LSR A  ;MOVE 2 BITS FROM`

`F954  66 B1  ROR TEMP  ;HIGH TO TEMP`

`F956  4A  LSR A`

`F957  66 B1  ROR TEMP`

`F959  A5 00  LDA CMPO  ;CALC MIN PULSE VALUE`

`F95B  18  CLC`

`F95C  69 0C  ADC #3C`

`F95E  C5 B1  CMP TEMP  ;PULSE LESS THAN MIN?`

`F960  B0 4A  BCS L1141  ;YES, NOISE`

`F962  A6 9C  LDX DPSW  ;NO, LAST BIT?`

`F964  F0 65  BEQ L1132  ;NO, CONTINUE`

`F966  4C 06 FA  JMP L1134  ;YES, FINISH BYTE`

`F969  ;`

`F969  A6 03  L1132  LDX PCNTR  ;9 BITS READ?`

`F96B  31 0B  BNE L1134  ;YES, GOTO ENDING`

`F96D  A2 00  LDX #00  ;SET BIT VAL TO ZERO`

`F96F  69 1F  ADC #10  ;ADD UP TO HALF WAY BETWEEN`

`F971  65 0E  ADC CMPO  ;SHIRT PULSE AND SYNC PULSE`

`F973  C5 B1  CMP TEMP  ;SHORT?`

`F975  B0 1C  BCS L1139  ;YES`

`F977  E8  INX  ;SET BIT VAL TO 1`

`F978  69 26  ADC #26  ;MOVE TO MIDDLE OF HIGH`

`F97A  65 B0  ADC CMPO`

`F97C  C5 B1  CMP TEMP  ;1?`

`F97E  B0 17  BCS L1137  ;YES`

`F980  6F 0C  ADC #2C  ;MOVE TO LONGLONG`

`F982  65 B0  ADC CMPO`

`F984  C5 B1  CMP TEMP  ;LONGLONG?`

`F986  99 03  BEQ L1136  ;GREATER THAN, ERROR`

`F988  4C 10 FA  L1134  JMP L1145  ;YES`

`F98B  ;`

`F98B  A5 B4  L1136  LDA SNSW1  ;NOT SYNCRONISED?`

`F98D  F0 10  BEQ L1141  ;NO, ERROR`

`F98F  85 A8  STA KER  ;YES, FLAG KER`

`F991  D0 19  BNE L1141  ;ALWAYS`

`F993  ;`

`F993  E6 09  L1139  INC REZ  ;COUNT REZ UP ON ZEROS`

`F995  B0 02  BCS L1138  ;ALWAYS`

`F997  ;`

`F997  C6 A9  L1137  DEC REZ  ;COUNT REZ DOWN ON ONES`

`F999  38  L1138  SEC  ;CALC ACTUAL VAL FOR COMPARE`

`F99A  E9 13  SBC #13  ;SUBTRACT INPUT VAL`

`F99C  E5 B1  SBC TEMP`

`F99E  65 92  ADC SVXT  ;ADD DIFF TO TEMP STORE`

`F9A0  85 92  STA SVXT  ;USED TO ADJUST SOFT SERVO`

`F9A2  A5 A4  LDA F1RT  ;FLIP DIPOLE FLAG`

`F9A4  49 91  EOR #01  ;SUBTRACT INPUT VAL`
LOC    CODE    LINE
F9AC  85 A4    STA $FIRK ;SECOND HALF OF DIPOLE
F9AC  F9 28    BEQ L1143 ;FIRST HALF SO STORE VAL
F9AC  86 D7    STX DATA ;
F9AC  ;
F9AC  A5 B4    L1141 LDA $NSW1 ;NO BYTE START?
F9AE  F9 22    BEQ L1150 ;YES, RETURN
F9B0  A0 A3 02    LDA #$2A3 ;TIMER AIRQ'D?
F9B3  29 01    AND #$01
F9B5  D0 05    BNE L1133 ;YES
F9B7  A0 A4 02    LDA #$2A4
F9BA  D0 16    BNE L1150 ;NO, EXIT
F9BC  A9 00    L1133 LDA #$00 ;SET DIPOLE FLAG FOR FIRST HALF
F9B0  85 A4    STA $FIRK
F9C0  8D A4 02    STA #$2A4
F9C3  A5 A3    LDA PCNT ;WHERE IN BYTE
F9C5  10 30    BPL L1148 ;STILL DOING DATA
F9C7  30 8F    BMI L1134 ;PROCESS PARITY
F9C9  ;
F9C9  A2 A6    L1144 LDX #$A6 ;SETUP FOR LONGLONG
F9CC  29 E2 F8    JSR L1126 ;EVEN PARITY?
F9CF  A5 98    LDA PRTY ;NO, SET ERROR
F9D0  D0 B9    BNE L1136
F9D2  4C 8C 02    L1150 JMP #$FEB ;RESTORE REGS AND RTI
F9D5  ;
F9D5  A5 92    L1143 LDA SUXT ;ADJUST SOFT SERVO?
F9D7  F0 07    BEQ L1149 ;NO
F9D9  30 03    BNE L1142 ;YES, MORE BASE TIME
F9DB  C6 B0    DEC CMP0 ;YES, LESS BASE TIME
F9DD  2C .BYT #2C ;SKIP NEXT
F9DE  E6 80    L1142 INC CMP0
F9E0  A9 00    L1149 LDA #$00 ;CLEAR DIFF FLAG
F9E2  85 92    STA SUXT
F9E4  E4 D7    CPX DATA ;CONSEC. LIKE VALS IN DIPOLE?
F9E6  D0 0F    BNE L1148 ;NO, PROCESS INFO
F9E8  BA     TXA ;YES, CHECK VALS
F9E9  D0 A0    BNE L1136 ;ONES, ERROR
F9EB  A5 A9    LDA REZ ;HOW MANY ZEROS?
F9ED  30 BD    BMI L1141 ;TOO MANY
F9EF  C9 10    CMP #$10 ;16?
F9F1  96 89    BCC L1141 ;NO, CONTINUE
F9F3  85 96    STA SYNO ;YES, FLAG SYNO
F9F5  80 BS    BCS L1141 ;ALWAYS
F9F7  ;
F9F7  BA     L1148 TXA ;MOVE READ DATA TO .A
F9F8  45 9B    EOR PRTY ;CALC PARITY
F9FA  85 9B    STA PRTY
F9FC  A5 B4    LDA SNSW1 ;REAL DATA?
F9FE  F0 D2    BEQ L1150 ;NO, FORGET
FA00  C6 A3    DEC PCNT ;DEC BIT COUNT
FA02  30 C5    BMI L1144 ;NEG, TIME FOR PARITY
FA04  46 D7    LSR DATA ;SHIFT BIT FROM DATA
FA06  66 BF    ROR $HCH ;INTO BYTE STORE
FA08  82 8A    LDX #$DA
FA0A  20 E2 F8    JSR L1126
FA0D  4C 8C 02    JMP #$FEBC ;RESTORE REGS AND RTI
FA10  ;
FA10  A5 96    L1145 LDA SYNO ;GOT BLOCK SYNC?
FA12  F9 04    BEQ L1140 ;NO
FA14  A5 84    LDA SNSW1 ;HAD REAL BYTE?
FA16  F9 07    BEQ L1151 ;NO
FA18  A5 A3    L1140 LDA PCNT ;END OF BYTE?
FA1A  30 03    BMI L1131 ;YES
FA1C  4C 97 F9    JMP L1137 ;NO, TREAT AS LONG
FA1F  ;
FA1F  46 B1    L1151 LSR TEMP ;ADJUST TIME OUT FOR
<table>
<thead>
<tr>
<th>LOC</th>
<th>CODE</th>
<th>LINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>FA21</td>
<td>A9 93</td>
<td>LDA #$93 ; LONGLONG PULSE VAL</td>
</tr>
<tr>
<td>FA23</td>
<td>3A</td>
<td>SEC</td>
</tr>
<tr>
<td>FA24</td>
<td>E5 B1</td>
<td>SBC TEMP</td>
</tr>
<tr>
<td>FA26</td>
<td>6B B0</td>
<td>ADC CMPD</td>
</tr>
<tr>
<td>FA28</td>
<td>0A</td>
<td>ASL A</td>
</tr>
<tr>
<td>FA29</td>
<td>AA</td>
<td>TAX ; SET TIME OUT FOR LAST BIT</td>
</tr>
<tr>
<td>FA2A</td>
<td>29 E2 F8</td>
<td>JSR L1126 ; SET BIT THROW AWAY FLAG</td>
</tr>
<tr>
<td>FA2D</td>
<td>E6 9C</td>
<td>INC DPSW ; BYTE SYNCHRONISED?</td>
</tr>
<tr>
<td>FA2F</td>
<td>A5 B4</td>
<td>LDA SNSET1</td>
</tr>
<tr>
<td>FA31</td>
<td>08 11</td>
<td>BNE L1152 ; YES, SKIP TO PASS CHAR</td>
</tr>
<tr>
<td>FA33</td>
<td>A5 96</td>
<td>LDA SYN0 ; IMROW OUT DATA UNTIL SYNC</td>
</tr>
<tr>
<td>FA35</td>
<td>F8 26</td>
<td>BE L1155 ; NO SYNC</td>
</tr>
<tr>
<td>FA37</td>
<td>B5 A8</td>
<td>STA RER ; FLAG DATA AS ERROR</td>
</tr>
<tr>
<td>FA39</td>
<td>A9 00</td>
<td>LDA #$00 ; KILL 16 SYNC FLAG</td>
</tr>
<tr>
<td>FA3B</td>
<td>85 B4</td>
<td>STA SNSET1 ; FLAG WE HAVE BYTE SYNC</td>
</tr>
<tr>
<td>FA44</td>
<td>A5 96</td>
<td>L1152 LDA SYN0 ; SAVE SYN0 STATUS</td>
</tr>
<tr>
<td>FA46</td>
<td>B5 B5</td>
<td>STA DIFF</td>
</tr>
<tr>
<td>FA48</td>
<td>F0 09</td>
<td>BNE L1153 ; NO BLOCK SYNC</td>
</tr>
<tr>
<td>FA4A</td>
<td>A9 00</td>
<td>LDA #$00 ; TURN OFF BYTE SYNC SWITCH</td>
</tr>
<tr>
<td>FA4C</td>
<td>B5 B4</td>
<td>STA SNSET1</td>
</tr>
<tr>
<td>FA4E</td>
<td>A9 01</td>
<td>LDA #$01 ; DISABE TIMER B IRQ</td>
</tr>
<tr>
<td>FA50</td>
<td>BD 00 DC</td>
<td>STA D11CR</td>
</tr>
<tr>
<td>FA52</td>
<td>05 B4</td>
<td>STA SNSET1</td>
</tr>
<tr>
<td>FA55</td>
<td>B5 BD</td>
<td>STA OCHAR ; COMBINE ERROR VALS</td>
</tr>
<tr>
<td>FA57</td>
<td>A5 A8</td>
<td>LDA RER</td>
</tr>
<tr>
<td>FA59</td>
<td>05 9A</td>
<td>ORA KEZ</td>
</tr>
<tr>
<td>FA5B</td>
<td>B5 86</td>
<td>STA PRP ; AND SAVE IN PRP</td>
</tr>
<tr>
<td>FA5D</td>
<td>4C BC FE</td>
<td>L1155 JMP #FEBC ; SET LAST BYTE</td>
</tr>
<tr>
<td>FA5F</td>
<td>00</td>
<td>STA OCHAR</td>
</tr>
<tr>
<td>FA61</td>
<td>20 97 F8</td>
<td>JSR L1154 ; FINISH BYTE, CLK FLAGS</td>
</tr>
<tr>
<td>FA63</td>
<td>B5 9C</td>
<td>STA DPSW ; GET BIT THROW AWAY FLAG</td>
</tr>
<tr>
<td>FA65</td>
<td>A2 DA</td>
<td>LDX #$DA ; INIT FOR NEXT DIFOL</td>
</tr>
<tr>
<td>FA67</td>
<td>20 E2 F8</td>
<td>JSR L1126 ; CHECK FOR LAST VAL</td>
</tr>
<tr>
<td>FA6A</td>
<td>A5 BE</td>
<td>LDA FSBLK</td>
</tr>
<tr>
<td>FA6C</td>
<td>F0 02</td>
<td>BEU L1135</td>
</tr>
<tr>
<td>FA6E</td>
<td>B5 A7</td>
<td>STA SHCNL</td>
</tr>
</tbody>
</table>

FA70
FA70 ;*******************************************************************************
FA70 ; BYTE HANDLER OF CASSETTE READ.
FA70 ; RER IS SET IF THE BYTE IS IN
FA70 ; ERROR. REZ IS SET IF THE INTERRUPT
FA70 ; PROGRAM IS READING ZERUS. RDFLG TELLS
FA70 ; WHAT WE ARE DOING. BIT 7 SAYS TO
FA70 ; IGNORE BYTES UNTIL. REZ IS SET, BIT 6
FA70 ; SAYS TO LOAD THE BYTE. OTHERWISE
FA70 ; RDFLG IS A COUNTDOWN AFTER SYNC. IF
FA70 ; VERCK IS SET WE DO A COMPARE INSTEAD
FA70 ; OF A STORE AND SET STATUS. FSBK
FA70 ; COUNTS THE TWO BLOCKS. PTR1 IS THE
FA70 ; INDEX TO THE ERROR TABLE FOR PASS1.
FA70 ; PTR2 IS THE INDEX TO THE CORRECTION
FA70 ; TABLE FOR PASS2.
FA70 ;*******************************************************************************
FA70 ; L1135 LDA #$0F ; TEST FUNCTION MODE
FA70 ; 24 AA BIT RDFLG ; NOT WAITING FOR ZEROS
FA74 | 16 17 | BPL L1159
FA76 | A6 B5 | LDA DIFF ; ZEROS YET?
FA78 | 00 8C | BNE L1156 ; YES, WAIT FOR SYNC
FA7A | A6 BE | LDX FSBLK ; PASS OVER?
FA7C | CA | DEX
FA7D | 00 0B | BNE L1138 ; HU
FA7F | A9 08 | LDA #MBERR
The Commodore 64 Kernal and Hardware Revealed

```
FA01 20 1C FE      JSR $FE1C ;YES, LONG BLOCK ERROR
FA04 D0 04        BNE L1158 ;ALWAYS
FA06 A9 00        L1156 LDA #$00 ;NEW MODE, WAIT FOR SYNC
FA08 BS AA        STA RDFLG ;FIRST MODE, WAIT FOR SYNC
FA0A 4C BC FE     L1158 JMP $FE8C ;EXIT, DONE
FA0D 70 11        L1159 BUS L1163 ;LOADING
FA0F D0 18        BNE L1162 ;SYNCING
FA11 A5 85        LDA DIFF ;HAVE BLOCK SYNC?
FA13 D0 F5        BNE L1158 ;YES, EXIT
FA15 A5 B6        LDA PKP ;FIRST BYTE IN ERROR?
FA17 D0 F1        BNE L1158 ;YES, EXIT
FA19 A5 A7        LDA SHCNL ;MOVE FSBLK TO CARRY
FA1B 4A           LSR A
FA1D A5 BD        LDA OCHAR ;SHOULD BE A HEADER COUNT CHAR
FA1F 30 03        BM1 L1157 ;NEG, FIRST BLOCK DATA
FA20 90 18        BCC L1161 ;EXPECTING FIRST BLOCK DATA
FA22 1B           CLC
FA24 B0 15        L1157 BCS L1161 ;EXPECTING 2ND BLOCK
FA26 29 0F        AND #$0F ;MASK OFF HIGH STORE HEADER
FA28 BS AA        STA RDFLG ;COUNT IN MODE FLAG
FA2A C6 AA        L1162 DEC RDFLG ;WAIT FOR REAL DATA
FA2B D0 DD        BNE L1158 ;REAL
FA2D A9 46        LDA #$46 ;NEXT UP IS REAL DATA
FA30 BS AA        STA RDFLG ;SET DATA MODE
FA32 20 8E FB     L1174 JSR L1161 ;SETUP ADDR POINTERS
FA34 A9 90        LDA #$90
FA36 BS AB        STA SHCNH
FA38 F0 D9        BEQ L1158 ;ALWAYS, EXIT
FA3A A9 00        L1161 LDA #$80 ;IGNORE BYTES MODE
FA3C BS AA        STA RDFLG
FA3E D0 CA        BNE L1158 ;ALWAYS
FA40 A5 BS        L1163 LDA DIFF ;END OF BLOCK?
FA42 F0 0A        BEQ L1160 ;YES
FA44 A9 94        LDA MSBERR ;SHORT BLOCK ERROR
FA46 20 1C FE     JSR $FE1C
FA48 A9 90        LDA #$00 ;FORCE RDFLG FOR ERROR
FA4A 4C 4A FB     JMP L1167
FA4C 29 D1 FC     L1160 JSR L1193 ;END OF STORE AREA?
FA4E 90 03        BCC L1164 ;NOT YET
FA50 4C 4B FB     JMP L1172 ;YES
FA52 A6 A7        L1164 LDL SHCNL ;WHICH PASS?
FA54 CA           DEX
FA56 F0 2D        BEQ L1169 ;SECOND
FA58 A5 93        LDA VERCK ;LOAD OR VERIFY?
FA5A F0 0C        BEQ L1166 ;LOADING
FA5C 90 99        LDY #$99 ;VERIFYING
FA5E A5 BD        LDA OCHAR
FA60 D1 AC        CMP (SAL),Y ;COMPARE
FA62 F0 04        BEQ L1166 ;GOOD, CONTINUE
FA64 A9 91        LDA #$01 ;BAD, FLAG
FA66 BS B6        STA PKP ;AS ERROR
FA68 ;STORE BAD LOCATIONS FOR 2ND PASS RE-TRY
FA6A ;
FA6C A5 B6        L1166 LDA PKP ;CHK FOR ERRORS
FA6E F0 48        BEQ L1171 ;NONE
FA70 A2 3D        LDL #$3D ;MAX OF 30
FA72 E4 4E        CPX PTR1 ;REACHED MAX?
FA74 90 3E        BCC L1173 ;YES, FLAG 2ND PASS
FA76 A6 4E        LDX PTR1 ;INDEX INTO BAD
FA78 A5 AD        LDA SAM ;AND STORE BAD LOC
FA7A 90 01 01     STA BAD+1.X ;IN BAD TABLE
FA7C A5 AC        LDA SAL
FA7E 90 09 01     STA BAD,X ;ADVANCE TO NEXT
FA80 EB           INX
FA82 EB           INX
FA84 86 9E        STX PTR1
FA86 4C 3A FB     JMP L1171 ;STORE CHAR
The Cassette Units

FB00
FB09
FB00
FB00
FB08 A6 9F L1169 LDX PTR2 ;DONE ALL IN TABLE?
FB04 E4 9E CPX PTR1
FB0C F0 35 BEQ L1170 ;YES
FB0E A5 AC LDA SAL ;NEXT IN TABLE?
FB10 DD 00 01 CMP BAD, X
FB13 D0 2E BNE L1170 ;NO
FB15 A5 AD LDA SAH
FB17 DD 01 01 CMP BAD+1, X
FB1A D0 27 BNE L1170 ;NO
FB1C E6 9F INC PTR2 ;FOUND NEXT ONE, ADVANCE
FB1E E6 9F INC PTR2
FB20 A5 93 LDA VERCK ;LOAD OR VERIFY?
FB22 F0 08 BEQ L1168 ;LOADING
FB24 A5 B0 LDA OCHAR ;VERIFYING
FB26 A0 00 LDY $00
FB28 01 AC CMP (SAL), Y
FB2A F0 17 BEQ L1170 ;O.K.
FB2C 06 INY ;Y=1
FB2D B4 B6 STY PRP ;FLAG IT AS AN ERROR
FB2F A5 B6 L1168 LDA VERCK ;SECOND PASS ERROR?
FB31 F0 07 BEQ L1171 ;NO
FB33 A9 10 L1173 LDA WSPERR
FB35 20 1C FE JSR $FE1C
FB38 D0 09 BNE L1170 ;ALWAYS
FB3A A5 93 L1171 LDA VERCK ;LOAD OR VERIFY?
FB3C D0 05 BNE L1170 ;VERIFY
FB3E A3 B6 STA (SAL), Y ;STORE CHARACTER
FB43 20 DB FC L1170 JSR L1080 ;NEXT ADDRESS
FB46 D0 43 BNE L1177 ;ALWAYS
FB48
FB4B A9 B0 L1172 LDA #$00 ;SET SKIP NEXT DATA
FB4A 85 AA L1167 STA R0FLG
FB4C 1B SEI
FB4D A2 01 LDX #$01
FB4F BE 0D DC STX U11CR
FB52 AE 0D DC LDX 011CR
FB55 A6 BE LDX FSBLK ;DEC FSBLK FOR NEXT PASS
FB57 CA DEX
FB5B 30 02 RML 1165 ;DONE, FSBLK=0
FB5A B6 BE STX FSBLK ; ELSE, NEXT
FB5C C6 A7 L1165 DEC SHCNL ;DEC PASS CALC
FB5F 80 03 BEQ L1175 ;ALL DONE
FB60 A5 9E LDA PTR1 ;FIRST PASS ERRORS?
FB62 D0 27 BNE L1177 ;YES, CONTINUE
FB64 85 BE STA FSBLK ;CLEAR FSBLK IF NO ERRORS
FB66 E0 23 BEQ L1177 ;ALWAYS, EXIT
FB68
FB6B 20 93 FC L1175 JSR L1192 ;READ IT ALL, EXIT
FB6C 20 91 FC JSR L1174 ;RESTORE SAL & SAH
FB6E A0 00 LDY #$00 ;SHCNH=0
FB70 84 AB STY SHCNH ; USED TO CALC PARITY BYTE
FB72
FB72 B1 AC L1176 LDA (SAL), Y ;CALC BLOCK BCC
FB74 45 AB EOR SHCNH
FB76 95 AB STA SHCNH
FB78 20 DB FC JSR L1080 ;BUMP ADDRESS
FB7B 20 D1 FC JSR L1193 ;AI END?
FB7E 9F 02 BCC L1176 ;NOT YET
FB80 A5 AC LDA SHCNH ;BCC CHAR MATCH?
The Commodore 64 Kernel and Hardware Revealed

Cassette write subroutines

Entry point: $FBA6

Function: These five routines are all required by the main write to tape routine at $F867.

Routine source code:

```
LOC  CODE   LINE

F8A6  :;
F8A6  ;***********************************************************************
F8A6  ; CASSETTE WRITE SUBROUTINES.
F8A6  ;* FSBLK IS BLOCK COUNTER FOR RECORD
F8A6  ;* = 0 SECOND DATA
F8A6  ;* = 1 FIRST DATA
F8A6  ;* = 2 FIRST HEADER
F8A6  ;***********************************************************************
F8A6  :
F8A6  ;TOGGLE WRITE BIT ACCORDING TO LSB
F8A6  ;IN OCHAR
F8A6  :
F8A6  A5 BD  L1122  LDA OCHAR  ;BIT TO WRITE INTO CARRY
F8A8  4A  LSR A
F8A9  A9 60  LDA #$60  ;ASSUME CARRY CLEAR (SHORT)
F8AB  90 02  BCC L1184  ;CORRECT
F8AD  A9 B0  L1185  LDA #$B0  ;SET LONG
F8AF  A2 00  L1184  LDX #$00  ;SET AND STORE TIME
F8B1  BD 00 DC  L1178  STA DITBL  ;LO BYTE
F8B4  AE 07 DC  STA DITBH  ;HI BYTE
F8B7  AD 00 DC  LDA D11CR  ;CLEAR IRQ
F8BA  A9 19  LDA #$19
F8BC  BD 0F DC  STA D1CRB  ;FORCE LOAD & START TIMER
F8BF  A5 01  LDA #$01  ;TOGGLE WRITE BIT
F8C1  49 08  EOR #$08
F8C3  85 01  STA #$01
```
LOC  CODE LINE

FBC5  29 08 AND #$08 ;LEAVE JUST WRITE BIT
FRC7  49 05 RTS
FBC8  38 L1181 SEC ;FLAG PRP FOR END OF BLOCK
FBC9  66 B6 ROR PRP
FBCB  30 3C BM1 L1183 ;ALWAYS
FBCD  ;
FBCD  ;TAPE WRITE INQ ENTRY
FBCD  ;
FBCD  A5 A8 WRfN LDA REK ;CHECK FOR ONE LONG
FBCF D0 12 BNE L1191
FBDA A9 10 LDA #$10 ;WRITE LONG BIT
FBDF A2 01 LDX #$01
FBD5 20 B1 FB JSR L1178
FBD8 D0 2F BNE L1183
FBDA E6 A8 INC RER
FBD5 05 B3 LDA PRP ;END OF BLOCK?
FBD6 10 29 BPL L1183 ;NO, CONTINUE
FBD9 4C 57 FC JMP L1194 ;YES, FINISH OFF
FBD3 A5 A9 L1191 LDA REZ ;CHECK FOR A ONE BIT
FBD5 D0 09 BNE L1180
FBD7 20 AD FB JSR L1185
FBD8 D0 10 BNE L1183
FBD6 E6 A8 INC REZ
FBD9 D0 19 BNE L1183
FBD0 20 A6 FB L1180 JSR L1122 ;WRITE
FBD3 D0 14 BNE L1183 ;ON BIT LOW, EXIT
FBD5 A5 A4 LDA FIRT ;FIRST OF DIPOLE?
FBDA 49 01 EOR #$01
FBDA 85 A4 STA FIRT
FBD8 F0 0F BFX L1179 ;DIPOLE DONE
FBD0 A5 BD LDA OCHAR ;FLIPS BIT FOR COMPLEMENTARY
FBD2 49 01 EOR #$01
FC01 85 BD STA OCHAR
FC03 29 01 AND #$01 ;TOGGLE PARITY
FC05 45 98 EOR PRTY.
FC07 85 98 STA PRTY
FC09 4C 8C FE L1183 JMP #FEB0 ;RESTORE REGS AND RTI
FCC0  ;
FCC0 46 BD L1179 LSR OCHAR ;NEXT BIT
FCE8 C6 A3 DEC PCNTR ;DEC COUNTER FOR # BITS
FC10 A5 A3 LDA PCNTR ;# BITS SENT?
FC12 F0 3A BEQ L1190 ;YES, DO PARITY
FC14 10 F3 BPL L1183 ;NO, SEND REST
FC16  ;
FC16 20 97 FB L1186 JSK L1079 ;CLEAN UP COUNTERS
FC19 38 CLI ;ALLOW INTERRUPTIONS TO NEST
FC1A A5 A5 LDA CNTDN ;WRITING HEADER COUNTER?
FC1C F0 12 BEQ L1189 ;NO
FC1E A2 00 LDX #$00 ;WRITE HEADER COUNTERS
FC20 06 D7 STX DATA ;CLEAR BCC
FC22 C6 A5 DEC CNTDN
FC24 A6 BE LDX FSBLK ;FIRST BLOCK HEADER?
FC26 E0 02 CPX #$02
FC28 D0 02 BNE L1196 ;NO
FC2A 09 80 ORA #$80 ;YES, MARK 1ST BLOCK HEADER
FC2C 85 BD L1196 STA OCHAR ;WRITE CHARs IN HEADER
FC2E D0 D9 BNE L1183
FC30 29 D1 FC L1189 JSR L1193 ;ADDR=END?
FC33 08 0A BCC L1188 ;NOT YET
FC35 D0 91 BNE L1181 ;MARK END
FC37 E6 AD INC SAH
FC39 A5 D7 LDA DATA ;WRITE BCC
FC3B 85 BD STA OCHAR
FC3D 80 CA BCS L1183 ;ALWAYS
The Commodore 64 Kernel and Hardware Revealed

Entry point:
$FC6A - tape IRQ entry
$FCBD - change IRQ vectors

Function: The first of these two routines performs the main IRQ loop for both tape LOAD and SAVE. It should be noted that all SAVE and LOAD operations are performed under IRQ as a background program. The second routine is used to change the IRQ vectors for different tape read and write operations.

Routine source code:

Tape IRQ

Entry point:
$FC6A - tape IRQ entry
$FCBD - change IRQ vectors

Function: The first of these two routines performs the main IRQ loop for both tape LOAD and SAVE. It should be noted that all SAVE and LOAD operations are performed under IRQ as a background program. The second routine is used to change the IRQ vectors for different tape read and write operations.

Routine source code:
The Cassette Units

FC88 20 BE FB  JSR L1174  ;SETUP FOR HEADER COUNT
FC88 A2 07  LDX #$09  ;SETUP FOR HEADER COUNT
FC88 B6 A5  STX CNTDN
FC8F B6 B6  STX PRP
FC91 D0 83  BNE L1186  ;ALWAYS
FC93 08  L1192 PHP  ;CLEAN UP IRQ AND
FC94 73  SEI  ;RESTORE PIA'S
FC95 AD 11 D0  LDA VICREG+17  ;RESTORE SCREEN
FC96 09 10  ORA #$10
FC97 BD 11 D0  STA VICREG+17
FC9D 20 CA FC  JSR L1121  ;TURN OFF MOTOR
FCA0 A9 7F  LDA #$7F  ;CLEAR INTERRUPTS
FCB2 BD 00 DC  STA DIICR
FCB4 20 DD FD  JSR #$D9D  ;RESTORE KEYBOARD IRQ
FCB6 AD A0 02  LDA IRTMP+1  ;RESTORE KEYBOARD INTERRUPT VECTOR
FCB8 F0 09  BEQ L1127  ;NO IRQ
FCB9 BD 15 03  STA CINV+1
FCBf AD 9F 02  LDA IRTMP
FCB3 BD 14 03  STA CINV
FCB6 28  L1127 PLP  ;END OF SUBROUTINE
FCB7 60  RTS

FC88 20 93 FC  L1190 JSR L1192  ;RESTORE SYSTEM IRQ
FC88 F0 97  BNE L1187  ;SAME FOR TAPE IRQ SO RTI
FCBD  ;***********************************************************************
FCBD  ;* SUBROUTINE TO CHANGE IRQ VECTORS.
FCBD  ;* ON ENTRY, .X = 8 WRITE ZEROS TO TAPE
FCBD  ;* 10 WRITE DATA TO TAPE
FCBD  ;* 12 RESTORE TO KEYSCAN
FCBD  ;* 14 READ DATA FROM TAPE
FCBD  ;***********************************************************************
FCBD  ;FCBD BD 93 FD  L1195 LDA #$FD93,X  ;MOVE IRQ VECTORS
FCCE BD 94 03  STA CINV  ;TO VECTOR TABLE
FCCE BD 94 FD  LDA #$FD94,X
FCCE BD 15 03  STA CINV+1
FCF9 60  RTS

FCCA  ;
FCCA A5 01  L1121 LDA #$01  ;TURN OFF CASSSETTE MITOR
FCCE 9Y 20  ORA #$20
FCCE 85 01  STA #$01
FCDD 60  RTS

FCB1  ;*******************************************
FCB1  ;* COMPARE START AND END UF LOAD/SAVE
FCB1  ;* ADDRESSES. SUBROUTINE CALLED BY
FCB1  ;* TAPE READ, SAVE, TAPE WRITE
FCB1  ;*******************************************
FCB1  ;
FCB1 38  L1193 SEC

FCB2 A5 AC  LDA SAL
FCB4 E5 4E  SBC EAL
FCB6 A5 AD  LDA SAH
FCB8 E5 AF  SBC EAH
FCFA 60  RTS

FCB8  ;BUMP ADDRESS POINTER SAL
FCB8  ;

FCB2 E6 8E  L1080 INC SAL
FCB3 D0 02  BNE L1083
FCB5 E6 AD  INC SAH

FCB1 60  L1083 RTS

FCB2  ;

FCB2 #=$FD9B

FCB5  ; TAPE IRQ VECTORS
Virtually all Commodore 64 software currently being marketed uses some form of fast loader. These fast loaders are given names like: Turbo (this was the first fast loader available), Pavload, Flash Load, etc. The origin of these fast loader routines is rather obscure since many of the software houses use the same loader routines. In this section we give the source code for two fast loaders and their associated SAVE routine; these have been used on several software products of Zifra Software Ltd. under the name of ZITload and ZIFRALoad.

A fast loader is a routine which replaces the existing LOAD and allows a program or data to be loaded from tape at about ten times the speed of a normal LOAD. This means that a tape can be as fast as a disk drive. A fast loader is achieved by simply changing the format of the pulse sequence which is stored onto the tape in order to allow a far greater density of information storage per inch of tape. In order to create a fast loader two programs are needed; a fast loader program which is a fairly short machine code routine loaded at the beginning of a LOAD operation and then auto run to LOAD the rest of the program and/or data which is stored in fast loader format. The second program which is required is a routine to SAVE a program in fast loader format, the fast SAVE.

The first major problem to be overcome in designing a fast loader is how each bit is stored on the tape. Each bit is stored on tape as a pulse which goes through a high-low transition (see Fig. 4.4). The length of the total pulse decides whether the bit is a 1 or 0. A short pulse is a zero and a long pulse is a one. The bit is flagged in the interrupt register on the falling edge of the pulse.

The loader is a machine code program which runs with the interrupts disabled, sets a timer to between the two lengths, and when the timer runs out the interrupt register is checked to see if the pulse came in or not. If the falling edge of the pulse generates an interrupt before the timer runs out then the pulse was a zero, otherwise it was a one. The bits are then rotated into a byte storage until 8 bits have been read, thereby loading a full byte.

Before any bytes can be read and stored, the loader must set itself to be in sync with the bits on the tape. This is done by writing a string of zero bits with a single one bit at every byte interval. The routine then tries to align itself by recognising the value of the byte. An example of a header byte for aligning would be the value 64, hex $40$ or in binary: 01000000. A series of these bytes is written as the header; only when this byte has been read in and recognised can the actual program can be read without risk of alignment errors.
Waveform written out

Waveform read back

\[ F = \text{Flag triggered on falling edge of pulse} \]

*Fig. 4.4. High speed tape waveform.*
The program is stored in different ways depending on how much protection it is desired to put in the program. The simplest way of formatting the file is to first save the two byte load address followed by the two byte end address and then the actual file. The final byte following the end of the file is a checksum that was calculated by the SAVE routine and is also calculated during loading. If the two values are the same, the LOAD was successful. The routine for this form of fast loader is given in Program 5.

Program 5

```assembly
033C  !  FAST TAPE SAVE FOR THE 64.
033C  !*****ASSIGN 3 pauses for screen wait*****
033C  !THIS ROUTINE WILL SAVE A PROGRAM
033C  !TO TAPE SO THAT WHEN LOADED BACK
033C  !IT WILL LOAD FASTER THAN THE
033C  !$901 DISK DRIVE.
033C  !AN OPTION FOR AUTO-RUN IS
033C  !INCLUDED.
033C  !
033C  C000  *  00
  C000  A90B    LDA  #3SAVVEC  !CHANGE SAVE VECTOR
  C002  8D3203  STA  #0332  ! TO GO TO NEW
  C005  A9C0    LDA  #3SAVVEC  ! SAVE ROUTINE
  C007  8D3303  STA  #0333
  C00A  60    RTS
  C008  48  SAVVEC  PHA
  C00C  A5BA    LDA  #$B  !GET DEVICE #
  C00E  C307    CMP  #$0F  !NUMBER 7?
  C010  F004    BEQ TSAVE  !YES
  C012  68  PLA
  C013  4CEDF5  JMP  $F5ED  !DO NORMAL SAVE
  C016  !
  C016  A5B9  TSAVE  LDA  #$B  !GET SEC. ADDR.
  C018  8D23C2  STA  RUNFLG  !FLAG FOR AUTO-RUN
  C01B  A00F  LDX  #$0D
  C01D  A920  LDA  #$20
  C01F  998C1 loop1 STA  FLNAME,Y
  C022  68  DEY
  C023  10FA  BPL  LOOP1
  C025  A4B7  LDY  #$A7  !GET FILENAME LENGTH
  C027  C011  CPY  #$11  !GREATER THAN 16?
  C029  9002  BCC  LOOP2  !NO
  C02B  A010  LDY  #$18  !ONLY 1ST 16 Chars
  C02D  88  LOOP2  DEY
  C02E  300B  BMI  TSAVE1  !GET FILENAME
  C030  B1B8  LDA  ($BB),Y
  C032  998C1  STA  FLNAME,Y  !STORE IT
  C035  4C2DC0  JMP  LOOP2  !DO NEXT CHAR
  C038  !
  C038  A047  TSAVE1  LDX  #$47  !GET LOADER BYTE
  C03A  B94C1  TSAVE2  LDA  LOADER,Y  !STORE IT TO SAVE
  C03D  998C02  STA  #2BC,Y
  C040  88  DEY
  C041  10F7  BPL  TSAVE2  !FOR ALL BYTES
  C043  A901  LDA  #$01
  C045  AA  TAX
  C046  A9  TAY
  C047  20AFF  JSR  $FFBA  !SET FILE DETAILS
  C04A  A99E  LDA  #$3E  !LENGTH OF FILENAME
  C04C  A28C  LDX  #FLNAME
  C04E  A0C1  LDY  #FLNAME
  C050  20BFF  JSR  $FFBD  !FILE NAME DETAILS
  C053  A900  LDA  #$00  !STOP NAME FROM BEING
  C055  850D  STA  #00
  C057  A902  LDA  #$02
```
The Cassette Units 113

C059 85FC STA $FC !SAVE START HI
C05B A9BC LDA #$BC !SAVE START LO
C05D 85FB STA #$FB !POINT TO START
C05F A9FB LDA #$FB !SAVE END LO
C061 A204 LDY #$04 !SAVE END HI
C063 A003 LDI #$03 !SAVE END HI
C065 20DF8 JSR #$DF8 !SAVE LOADER FILE
C068 A983 LDA #$83 !VECTOR
C06A 8D0203 STA #$02 !RESET WARM START
C06D A9A4 LDA #$A4 !VECTOR
C06F 8D0303 STA #$0303
C072 A901 LDA #$81 !VECTOR
C074 859D STA #$9D !VECTOR
C076 A000 LDI #$00 !VECTOR
C078 A200 LDX #$00 !VECTOR
C07A !
C07A THE FAST SAVE ROUTINE
C07A STARTS HERE.
C07A !
C07A 20CBC0 JSR WRTHDR !WRITE ALIGNMENT BYTES
C07D A52B LDA #$2B !GET START LO
C07F 48 PHA !
C080 20FAC0 JSR WRTBYT !WRITE IT
C083 A52C LDA #$2C !GET START HI
C085 48 PHA !
C086 20FAC0 JSR WRTBYT !WRITE IT
C089 A52D LDA #$2D !GET END LO
C08B 20FAC0 JSR WRTBYT !WRITE IT
C08E A52E LDA #$2E !GET END HI
C090 20FAC0 JSR WRTBYT !WRITE IT
C093 84FB STY #$FB !ZERO CHECKSUM
C095 A42B LDY #$2B !GET PAGE OFFSET
C097 A900 LDA #$00 !ZERO LO BYTE
C099 852B STA #$2B !ZERO LO BYTE
C09B B12B TSAVELOOP LDA($2B),Y !GET A BYTE
C09D 20FAC0 JSR WRTBYT !WRITE IT
C0A0 C8 INY !
C0A1 D002 NBE TSAVE3 !
C0A3 862C INC #$2C
C0A5 C42D TSAVE3 CPY #$2D !CHECK END OF SAVE
C0A7 A52C LDA #$2C !
C0A9 852E SBC #$2E
C0AB 96EE BCC TSAVELOOP !NOT YET
C0AD A5FB LDA #$FB !GET CHECKSUM
C0AF 20FAC0 JSR WRTBYT !WRITE IT
C0B2 200EC1 JSR WRTBIT !CLOSE OFF LAST BIT
C0B5 A91B LDA #$1B
C0B7 8D11D0 STA #$D1 !UNBLANK SCREEN
C0BA A937 LDA #$37
C0BC 8501 STA #$01 !STOP TAPE
C0BE 58 CLI !RESTART IRQ
C0BF 68 PLA !GET START HI
C0C0 852C STA #$2C !STORE IT
C0C2 68 PLA !GET START LO
C0C3 852B STA #$2B !STORE IT
C0C5 2084FF JSR #$FF !RESET I/O
C0C8 4C74A4 JMP #$474 !EXIT TO 'READY.'
C0CB !
C0CB 8506 WRTHDR LDA #$06 !BASIC ROM OUT &
C0CD 8501 STA #$01 !START TAPE
C0CF A90B LDA #$0B !
C0D0 8D11D0 STA #$11 !BLANK SCREEN
C0D4 CA HEADR1 DEX !BLANK SCREEN
C0D5 80FD BNE HEADR1 !PAUSE FOR TAPE
C0D7 88 DEY !TO GET TO FULL
C0DE D0FA BNE HEADR1 !SPEED
C0DA 78 SEI !DISABLE IRQ
C0DB A9A0 LDA #$A0 !INITIAL TIMER
C0DD 8D044D STA #$04 !VALUE FOR DELAY
C0E0 A900 LDA #$00 !
The Commodore 64 Kernel and Hardware Revealed

114

C0E2 8D05DD STA #D05D
C0E3 A913 LDA #$19
C0E7 8D0EDD STA #D0DE
C0EA A040 LDA #$40
C0E8 A940 HEADR2 JSR WRTBYT
C0EE 20FAC0 JSR WRTBYT
C0F1 80 STA $DDEE
C0F2 D0F8 BNE HEADR2
C0F4 A95A LDA #$5A
C0F5 20FAC0 JSR WRTBYT
C0F9 60 RTS
C0FA 85BD WRTBYT STA #BD
C0FC 45FB EOR #$F
C0FE 85FB STA #F
C100 A908 LDA #$08
C102 85A3 STA #A3
C104 26BD WBYTE1 ROL #B
C106 200EC1 JSR WRTBIT
C109 6A3 RTS
C10B D07 BNE WBYTE1
C110 60 RTS
C10E A240 WRTBIT LDX #$40
C110 8002 BCC WBIT1
C112 A290 LDX #$90
C114 8E04DD WBIT1 STX #DD04
C117 A900 LDA #$00
C119 8D05DD STA #D05D
C11C A901 LDA #$01
C11E 2C0DDD WB12 BIT #D0D
C121 F0FB BEQ WB12
C123 8501 LDA #$01
C125 4908 EOR #$08
C127 D501 STA #01
C129 EE20D0 INC #D020
C12C A919 LDA #$19
C12E 8D0EED STA #D0DE
C131 A901 LDA #$01
C133 2C0DDD WB13 BIT #D0D
C136 F0FB BEQ WB13
C138 8501 LDA #$01
C13A 4908 EOR #$08
C13C D501 STA #01
C13E 8D0EED STA #D0DE
C140 60 RTS
C144 ! THE LOADER STARTS HERE
C144 ! AD20D0 LOADER LDA #$D20
C147 85FE STA #F
C149 A944 LDA #$A4
C14B 8D0303 STA #$0303
C14E A903 LDA #$93
C150 8D0303 STA #$0302
C152 205103 JSR #D251
C155 A5FE LDA #F
C158 8D20D0 STA #$D020
C15B A937 LDA #$37
C15D 8501 STA #01
C15F 89 CLI
C160 A91B LDA #$1B
C162 8D11D0 STA #D111
C165 20BAFF JSR #$FF94
C168 A5FC LDA #$FC
C16A 55FB CMP #$F
C16D D015 BNE LODERR
C16E 2063A6 JSR #A663
C171 A5503 LDA #$3DE

!START TIMER
!STORE BYTE
!CHECKSUM
!LOOP FOR 8 BITS
!BIT INTO CARRY
!WRITE THE BIT
!NEXT BIT
!ASSUME ZERO BIT
!CORRECT ASSUMPTION
!SET FOR ONE BIT
!SET TIMER
!WAIT FOR TIMER
!TOGGLE WRITE BIT
!IN 6510 REGISTER
!SHOW IT IS WORKING
!FOR TIMER
!TOGGLE WRITE BIT
!IN 6510 REGISTER
!FOR TIMER
!START TIMER
!FAST LOAD THE FILE
!RESTORE BORDER COLOUR
!STOP THE TAPE
!RESTORE IRQ
!UNBLANK SCREEN
!RESET I/O
!DIFFERENT..ERROR
!CLR
!GET AUTO-RUN FLAG

115
The Cassette Units 115

C174 F00A BEQ EXIT !NO RUN
C176 206EA6 JSR #$A68E !SET CHARGED POINTER
C179 A900 LDA #$00 !FLAG RUN MODE
C17B 4CAEA7 JMP #$A7AE !RUN
C180
C180 6C0203 EXIT JMP ($0302) !WARM START
C183 | !
C183 A21D LODERR LDX #$1D !"LOAD ERROR"
C185 4C37A4 JMP #$A437 !SEND ERROR
C188 |
C188 B8E3 WOR #$E3B !ERROR LINK
C18A B0C2 WOR #$02BC !WARM START LINK
C18C |
C18C 202020 FLNAME TXT "*
C19C |
C19C | !16 SPACES
C19C |
C19C *==#0351

C19C |
C19C 200703 JSR #$0387 !READ HEADER
C19F 20B083 JSR #$03BA !READ A BYTE
C1A2 A8 TRAP !LOAD LO
C1A3 A900 LDA #$00 !LOAD HI
C1A5 85C1 STA #$C1
C1A7 20BA03 JSR #$03BA !READ A BYTE
C1AA 85C2 STA #$C2 !LOAD HI
C1AC 20BA03 JSR #$03BA !READ A BYTE
C1AF 85D2 STA #$2D !END LO
C1B1 20BA03 JSR #$03BA !READ A BYTE
C1B4 852E STA #$2E !END HI
C1B6 20BA03 TLOAD1 JSR #$03BA !READ A BYTE
C1B9 91C1 STA (#01),Y !STORE IT
C1BB 45FC EOR #$FC !CALCULATE CHECKSUM
C1BD 85FC STA #$FC
C1BF C8 INY
C1C0 D082 BNE TLOAD2 !CHECK END OF LOAD
C1C2 E6C2 INC #$C2
C1C4 C42D TLOAD2 CPY #$2D !CHECK END OF LOAD
C1C6 A5C2 LDA #$C2
C1C8 E52E SBC #$2E
C1CA 90EA BCC TLOAD1 !NOT YET
C1CC 20BA03 JSR #$03BA !READ CHECKSUM
C1CF 85FB STA #$FB
C1D1 60 RTS

C1D2 |
C1D2 *==#0357
C1D2 |
C1D2 A907 LDA #$07 !START TAPE
C1D4 85B1 STA #$01 !BLANK SCREEN
C1D6 A90B LDA #$0B !PAUSE FOR TAPE TO
C1DB 8D11D0 STA #$D01 !REACH FULL SPEED
C1DB CA RHEAD1 DEY
C1DC D0FD BNE RHEAD1 !DISABLE IRQ
C1DE 85 RIV
C1DF D0FA BNE RHEAD1 !SET TIMER HI
C1E1 73 SEI !ZERO CHECKSUM
C1E2 84FC STY #$FC
C1E4 8C05DD STY #$DD05 !SET TIMER HI
C1E7 A9F8 LDA #$F8 !SET TIMER LO
C1E9 8D4DD STA #$D40 !READ A BIT
C1EC A200 LDX #$00 !INTO BYTE
C1EE 20CA03 RHEAD2 JSR #$2C03
C1F1 263D ROL #$BD !ALIGNED?
C1F3 A5BD LDA #$B
C1F5 C940 CMP #$40 !NOT YET
C1F7 D0F5 BNE RHEAD2
C1F9 20BA03 RHEAD3 JSR #$03BA !READ A BYTE
C1FC C940 CMP #$40 !IS IT 64?
C1FE F0F9 BEQ RHEAD3 !YES
Another type of LOAD, which uses the same saver but is slower, is the interrupt loader. This method has the advantage of being able to LOAD with the screen on and a foreground program running whilst the main program is loaded. Loaders of this type are: Novaload and Micro Load. The difference with this type of LOAD is that an interrupt is created when a pulse is read by the tape recorder, and the timer is checked to find out whether the pulse was a zero or a one. The whole LOAD is done in the background allowing a foreground program to play music, run a clock, etc. The foreground program must check at regular intervals to see if the loader has flagged for the end of load. The example of a background LOAD in Program 6 has only a foreground program that is waiting for the end of LOAD flag to be set.
The Cassette Units

The Fast Save Routine

! THE FAST SAVE ROUTINE

! STARTS HERE.

! WRITE ALIGNMENT BYTES

! GET END LO

! GET START LO

! GET END LO

! GET END HI

! SET FILE DETAILS

! GET LOADER BYTE

! STORE IT TO SAVE

! STORE IT

! GET FILENAME LENGTH

! GREATER THAN 14?

! ONLY 1ST 14 CHAR

! DO NORMAL SAVE

! DO NEXT CHAR

! FOR ALL BYTES

! THE FAST SAVE ROUTINE

! STARTS HERE.

! WRITE IT

! WRITE IT

! ZERO CHECKSUM

! GET PAGE OFFSET

! ZERO LO BYTE

! GET A BYTE
The Commodore 64 Kernel and Hardware Revealed

C088 20F5C0 JSR WRTBYT !WRITE IT
C08B C8 INY
C09C D002 BNE TSAVE3
C09E E62C INC #2C
C0A0 C42D CPY #2D
C0A2 A52C LDA #2C
C0A4 E52E SBC #2E
C0A6 90EE BCC TSAVELOOP !NOT YET
C0AA A5FB LDA #$FB
C0AB 20F5C0 JSR WRTBYT !WRITE IT
C0AD 2003C1 JSR WRTBIT !CLOSE OFF LAST BIT
C0B0 A91B LDA #$1B
C0B2 8D11D0 STA #D011 !UNBLANK SCREEN
C0B5 A937 LDA #$37
C0B7 8D01 STA #01
C0B9 58 CLI !RESTART IRQ
C0BA 68 PLA !GET START HI
C0BB 852C STA #2C !STORE IT
C0BD 68 PLA !GET START LO
C0BE 852B STA #2B
C0C0 2084FF JSR #$FB4 !RESET I/O
C0C3 4C74A4 JMP #$A474 !EXIT TO 'READY.'!
C0C6
C0C6 A906 WRTHDR LDA #$06 !BASIC ROM OUT &
C0C9 8501 STA #01 !START TAPE
C0CA A90D LDA #$0D
C0CC 8D11D0 STA #D011 !BLANK SCREEN
C0CF CA HEADR1 DEX !PUSH FOR TAPE
C0D0 D0FD BNE HEADR1 !TO GET TO FULL
C0D2 88 DEY !SPEED
C0D3 D0FA BNE HEADR1
C0D5 78 SEI !DISABLE IRQ
C0D6 A9A0 LDA #$A0 !INITIAL TIMER
C0D8 8D04DD STA #D004 !VALUE FOR DELAY
C0DB A908 LDA #$08
C0DD 8D05DD STA #D005
C0DE A919 LDA #$19
C0E2 8D0EDD STA #D0AE !START TIMER
C0E5 A940 LDY #$40
C0E7 A940 HEADR2 LDA #$40 !$01000000 FOR
C0E9 20F5C0 JSR WRTBYT !ALIGNMENT
C0EC 38 DEY
C0ED D0F8 BNE HEADR2 !WRITE 64 OF THEM
C0EF A95A LDA #$5A !CHECK ALIGNMENT
C0F1 20F5C0 JSR WRTBYT !WRITE IT
C0F4 60 RTS
C0F5
C0F5 85BD WRTBYT STA #BD !STORE BYTE
C0F7 45FB EOR #$FB !CHECKSUM
C0F9 85FB STA #$FB
C0FB A908 LDA #$08 !LOOP FOR 8 BITS
C0FD 85A3 STA #A3
C0FF 26BD WBYTE1 ROL #BD !BIT INTO CARRY
C101 2003C1 JSR WRTBIT !WRITE THE BIT
C104 C6A3 DEC #A3
C106 D0F7 BNE WBYTE1 !DO NEXT BIT
C108 60 RTS
C109
C109 A270 WRTBIT LDX #$70 !ASSUME ZERO BIT
C10B 9002 BCC WBIT1 !CORRECT ASSUMPTION
C10D A2FF LDX #$FF !SET FOR ONE BIT
C10F 8E04DD WBIT1 STX #D004 !SET TIMER
C112 A900 LDA #$00
C114 8D05DD STA #D005
C117 A901 LDA #$01
C119 2C0DDD WBIT2 BIT #D00D !WAIT FOR TIMER
C11C F0FB BEQ WBIT2
C11E A501 LDA #$1
C120 4908 EOR #$08 ! TOGGLE WRITE BIT
C122 8501 STA #$1
The Cassette Units

C124 EE20D0 INC $D020  SHOW IT IS WORKING
C127 A913 LDA #$19  START TIMER
C129 B006D6 STA #$D0E  WAIT FOR TIMER
C12C A901 LDA #$11  TOGGLE WRITE BIT
C12E 2C00DD WRT3 BIT #$D0D
C131 F0F8 BEQ WRT3
C133 A501 LDA #$01 IN 6510 REGISTER
C135 4903 EOR #$08
C137 8501 STA #$01
C139 A919 LDA #$19
C13B B006D6 STA #$D0E  START TIMER
C13E 60 RTS
C13F ! THE LOADER STARTS HERE
C13F !
C13F A005 LOADER LDY #$05
C141 A920 LDA #$20 BLANK OUT 'READY.'
C143 995004 BLOOP STA #$0450,Y
C146 88 DEY
C147 10FA BPL BLOOP
C149 78 STA
C14A A905 LDA #$05
C14C 5501 STA #$01
C14E A91F LDA #$1F
C150 8D00DD STA #$D0D
C153 B006DC STA #$DC0D
C156 A006DC STA #$DC0D
C159 A006DC STA #$DC0D
C15C A963 LDA #$05
C15E B006DC STA #$DC0D
C161 A903 LDA #$03 ENABLE TAPE IRQ
C163 B005DC STA #$DC05
C166 A905 LDA #$05
C168 B006DC STA #$DC0D
C16B A951 LDA #$51
C16D B0FF STA #$FFFE
C170 A903 LDA #$03 LOAD ROUTINE
C172 B0FF STA #$FFFF
C175 A977 LDA #$77
C177 B0AFF STA #$FFFA
C17A A903 LDA #$03
C17C B0FF STA #$FFFE
C17F A900 LDA #$00 CLEAR LOADED FLAG
C181 8502 STA #$02
C183 58 CLI
C184 4C1D03 JMP #$03DD
C187 ! WAIT FOR END OF LOAD
C187 !
C187 A903 LDA #$03
C189 B00203 STA #$0202 VECTR
C18C A9A4 LDA #$A4
C18E B0003 STA #$03
C191 4C84FF JMP #$FF84
C194 B003 AB WOR #$E30B,#02AB
C198 !
C198 932A20 FLNAME TXT "C#"
C198 ! 14 SPACES
C198 A8 PBA IRQ ENTRY POINT
C19A 98 TYA
C19A 48 PBA
C19B A05DC LDA #$DC05
C19E A019 LIV #$19
C19E C00EDC STY #$DCE
C192 4902 EOR #$02
C195 4A LSR A
C196 4A LSR A
C197 26A9 ROL #$9
C199 A9A9 LDA #$9
C19B 9002 BCC BIT00
The Commodore 64 Kernel and Hardware Revealed

120

C1BD B00D BCS EXIT !NOT COMPLETE BYTE
C1BF C940 CMP ##40 !ALIGNMENT?
C1C1 D039 BNE EXIT !NO
C1C3 A916 LDA #15 !SET NEW ADDRESS
C1C5 8D6503 STA #8365
C1C8 A9FE EX1 LDA #FE !GET READY FOR
C1CA 85A9 STA #A9 !A NEW BYTE
C1CC AD0D0C EXIT LDA $DC0D !CLEAR IRQ
C1CF 68 PLA
C1D0 A8 TRAP !EXIT IRQ
C1D1 68 PLA
C1D2 40 RTI
C1D3 !
C1D3 C940 CMP ##40 !MORE ALIGNMENT?
C1D5 F0F1 BEQ EX1 !YES
C1D7 C95A CMP ##5A !FINAL CHECK?
C1D9 F007 BEQ BITGT2 !YES
C1DB A902 LDA #02 !GO BACK TO
C1DD 8D6503 STA #8365 !ALIGNMENT ROUTINE
C1E0 D066 BNE EX1
C1E2 A930 BITGT2 LDA #30 !SET NEW ADDRESS
C1E4 8D6503 STA #8365 !TO READ IN LOAD
C1E7 A900 LDA #00 !ADDRESSES, CLEAR
C1E9 85C1 STA #1 !CHECKSUM
C1EB F0DB BEQ EX1
C1ED !
C1ED 85FB STA #FB !STORE LOAD ADDRESS
C1EF EE3703 INC #0397 !INCREASE STORE
C1F2 AD9703 LDA #0397 !UNTIL 4 BYTES
C1F5 C9FF CMP ##FF !NOT YET
C1F7 D0CF BNE EX1 !STORE NEW ADDRESS
C1F9 A943 LDA #43 !FOR READING FILE
C1FB 8D6503 STA #8365
C1FE D0C3 BNE EX1
C200 !
C200 A000 LDY #$00 !STORE A BYTE
C202 91F8 STA ($FB),Y !CALCULATE CHECKSUM
C204 45C1 EOR #1
C206 85C1 STA #1
C208 EE0008 INC #0000 !SHOW IT IS WORKING
C20B 86F8 INC #$FB !INCREASE ADDRESS
C20D D062 BNE BITGT5 !STORE CHECKSUM
C20F 86FC INC #$FC !FLAG END OF LOAD
C211 A5FB BITGT5 LDA #$FB !RESET BRANCH TO
C213 C5FD CMP #$FD !ALIGNMENT
C215 A5FC LDA #$FC
C217 E5FE SBC $FE !END OF LOAD?
C219 90AD BCC EX1 !NOT YET
C21B A965 LDA #$65 !NEW ADDRESS FOR
C21D 8D6503 STA #8365 !CHECKSUM
C220 D0A6 BNE EX1
C222 !
C222 85C2 STA #C2 !STORE CHECKSUM
C224 A9FF LDA ##FF !FLAG END OF LOAD
C226 8502 STA #02 !RESET BRANCH TO
C228 A902 LDA #$02 !ALIGNMENT
C22A 8D6503 STA #8365
C22D A9FB LDA #$FB
C22F 8D9703 STA #0397 !WAIT FOR FILE
C232 D094 BNE EX1 !TO BACKGROUND LOAD
C234 !
C234 !*=$03DD
C234 !
C234 A502 PAUSE BEQ PAUSE
C236 85FC STA #C2 !WAIT FOR FILE
C238 A900 LDA #00 !TO BACKGROUND LOAD
C23A 8502 STA #02
C23C A907 LDA #07
C23E 8501 STA #01 !RESET KERNAL ROM
C240 20F302 JSR #02F3
4.7.1 Fast tape routines

Putting the theory into practice to create the fast loader routines is not difficult. The actual timing for the SAVE routine was not calculated from any theoretical formula but was obtained just by trial and error. The only guidelines were that the short pulse should be slightly shorter than half the long pulse, as the waveform of the pulse is evened out by the cassette hardware. The timing value used by the loader is just shorter than the time required before the long pulse reaches its falling edge.

There are two program listings in this section, one for each of the two types of LOAD. Each program will SAVE a Basic program to tape in its fast format and automatically put the fast loader routine into the filename where it is stored and, when loaded, will automatically start on the warm start vector. The routines are initialised by SYS(49152). A Basic program can be fast saved by using the SAVE command as normal but with a device number of 7, thus:

```
SAVE "PROGRAM",7
```

In addition the first kind of fast LOAD also makes use of the secondary address to auto run the program, thus:

```
SAVE "PROGRAM",7,1
```

will cause the program to auto run when loaded back. With both routines, when a program has been saved using one of these fast loader SAVE routines it is unnecessary to load anything before loading the program; it will load directly from the LOAD command.

An example of how fast these routines can be is shown by the following timing table. This was based on the time taken to load a 26.3K byte Basic program:

<table>
<thead>
<tr>
<th>Method</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk</td>
<td>1 minute</td>
</tr>
<tr>
<td>Method 2</td>
<td>1 minute</td>
</tr>
<tr>
<td>Normal tape</td>
<td>8 minutes</td>
</tr>
</tbody>
</table>

It should also be noted that the SAVE routines for the fast tape operation are considerably shorter than the normal tape routines which were analysed at the beginning of this chapter.

4.8 Causing programs to auto run from tape

The facility to have a program run automatically after completing its LOAD is a
nice feature to include, particularly if the program is intended for commercial sale. Adding this auto run feature to tape loaded programs is not difficult and considerably enhances a program's professionalism. Before saving a Basic program to tape, the secondary address is used to indicate whether the LOAD routine starts loading into the memory area from which it was saved, or starts loading at an address stored in the pointer to the start of Basic program storage variables. Thus if a program is saved with: SAVE“PROGRAM” it will commence loading wherever the pointer stored at $2B,$2C (decimal 43,44) indicates, regardless of where it was saved from. If, however, SAVE “PROGRAM”,1,1 is used the LOAD routine will load the program into the same locations from which it was saved.

The use of the secondary address is thus the main principle required for auto running. To auto run a program a short machine code loader is required; this is loaded first, and on loading will then take control of the computer. The only way to make this happen is to write over one of the operating system vectors in page 3 of memory, the top end of the stack, or the ‘Tape load IRQ’ vector temporary storage at location $029F/$02A0.

4.8.1 Page three vectors
There are plenty of vectors which can be used. The most commonly used is the Basic warm start vector (as in the fast load routines in the previous section). This is the easiest one to use since it can be set to point to the auto run routine in the vectors saved with the program, and then reset afterwards. In addition, use of this vector allows code to access the sprite 11 block.

Other vectors which can be used are:

- Input vector at $0324
- Output vector at $0326
- Abort 1/O vector at $032C

These three vectors will also cause control to be transferred to the routine after loading, but use of the sprite 11 block for code is impossible so the code must therefore be located in the filename. Problems can arise in using these vectors when saving, for example: the output vector is set up, as soon as the SAVE routine is called, and the computer will crash when it tries to print the message ‘PRESS RECORD & PLAY ON TAPE’. The way to overcome this is to add a bit of code into the SAVE routine which is called before the vector is set up:

LDA #$00
STA $9D !disable the message ‘saving’
JSR $F838 !wait for record and play

You can then set up the vector and save it.

4.8.2 The stack
A machine code program can be made to auto run by using the top 8 bytes of the stack. These locations are all set to a value of 2, and the machine code starts at location $0203. This method is not widely used, since it will only work on the majority of occasions when the machine is freshly powered up. There is one
advantage, however; if it does not auto run, there is less chance of the machine code being intact for prying eyes.

4.8.3 Tape IRQ save
With this vector you must have a SAVE routine which saves a program from one area of memory which will be loaded into another (see beginning of this section). All the auto run code must be located in the filename.

Having decided which vector to auto run and where to place the machine code loader, it is necessary to decide what the loader will do. The first function of any loader should be to get the kernal LOAD routine to stop printing messages. This will prevent the pause for CBM key when the next file is found. It may also be necessary to disable the RUN/STOP key (see next section). Other security methods that you can add into your loader are detailed in the next section of this chapter. Whichever vector is used to auto run, it must be reset to normal before running the main program. If page 3 vectors are used (IRQ save included), there are two ROM subroutines to use: $E453 for vectors from $0300 to $030B, or $FF8A for vectors between $0314 and $0333 (IRQ save changes $0314). The program can then be loaded. Depending on whether the program is in machine code or Basic, the auto run routine can either jump straight into the main program or cause the Basic program to run. Running a machine code program is straightforward, however there are several ways to initiate the running of a Basic program from a machine code routine:

a) Keyboard buffer
By storing the characters R, shift U, and carriage return into the keyboard buffer ($0277–$0280) and setting the number of characters to 3 ($C6), the Basic program will then run by: JMP ($0302). The problem with this is that, to be on the safe side, the screen should first be cleared or there is a possibility of a syntax error occurring.

b) Basic ROM routines
The second, and best, way of running a Basic program is to use the routines in the ROM. The code to run a program this way is shorter than that for the keyboard buffer method. In both types, the end address from the LOAD must be stored into locations $2D and $2E. The code for running a Basic program using the ROM routines is as follows:

```
JSR $A65C   !perform ‘CLR’
JSR $A68E   !reset charge pointers
JMP $A7AE   !execute the next statement
```

There is no need to store anything into the keyboard buffer or to clear the screen.

4.9 Tape security and anti piracy techniques
The greatest problem for anyone writing and/or selling commercial software is
illegal copying. This can lose the author a substantial proportion of the expected royalty. It has been estimated that often as many as two out of every three copies of a program in circulation are illegal pirate copies. The SAVE command makes pirating of unprotected programs so easy that it is essential to put some protection onto any commercial program. There is no absolutely secure way of protecting a piece of software on the 64. If someone has enough patience they can break any protection method and copy the program. Therefore, the main thing to concentrate on is making the job of breaking the protection as difficult or laborious as possible. The initial methods include disabling the RUN/STOP key, encoding the program before saving and decoding it on loading, etc.

To disable the RUN/STOP key is very simple:

```
LDA #<STOP   !SET RUN/STOP VECTOR
STA $0328   ! TO POINT TO NEW
LDA #>STOP   ! ROUTINE
STA $0329
.
.
STOP   LDA $91
RTS
```

Another method of disabling the STOP key is by altering the low byte, but the above method is the only totally reliable way. This disables both the normal stop from a Basic program and the STOP/RESTORE combination. If the program is a machine code game, then it is better to just disable the NMI vector. This can be done by changing the vector at $0318 to point to an RTI instruction ($FEC1). The NMI vector does not have to be disabled; it could be of use in the actual program (see Chapter 6).

Encryption of the program is useful as it will stop a pirate from loading the main file without the auto run part. Encryption means that a special SAVE routine is used to encode a program which is then totally indecipherable. The loading then decodes it so that it can run properly. The best way of encoding and decoding is to use one of the arithmetic commands in the 6510 instruction set. The most common and easiest to use is the EOR command. To do this take a key value, a number between 0 and 255, EOR it with a byte of the original code and store the result; this is the encrypted code. To restore the original code, simply take the stored encrypted code byte, EOR it with the key value and the original code is restored:

```
LDA STORE   !GET THE VALUE FROM MEMORY
EOR #$A1   !ENCODE IT
STA STORE   !STORE IT
```

This routine, when called the first time, will encode the byte. Call it a second time and the original value will be restored. Use the following routine to encode (or decode) a complete program where ($2B) is the start and ($2D) is the end plus 1:
The Cassette Units

LDA $2B  !SET START ADDRESS
STA $FB
LDA $2C
STA $FC
LDY #$00
LOOP  LDA ($FB),Y  !GET A BYTE
       EOR $FB  !ENCODE/DECODE IT
       STA ($FB),Y  !STORE IT
       INC $FB  !INCREMENT POINTER
       BNE CHECK
INC $FC
CHECK  LDA $FB  !CHECK END OF
       CMP $2D  !PROGRAM
       LDA $FC
       SBC $2E
       BCC LOOP  !NOT YET

It is not necessary to use location $FB for the EOR code, but whatever value is
used it must be the same on encoding as it is on decoding. With the advent of fast
tape formats, the need for this EOR encoding/decoding is nullified due to the
fact that the high speed loader must be present to be able to load the main
program.

Final security checks should be made on running the program to check
certain locations for the presence of known values. Obvious locations are the
cassette buffer (filename), device number (to check the last device used), etc.

4.9.1 Undocumented codes

All the previously mentioned methods can be displayed with the use of a
monitor, and so with a little detective work they can be understood by someone
intent on breaking the security. The use of some of the undocumented codes of
the 6502 within the program and its security makes the use of a monitor much
harder. On all 6502 microcomputers there are some instructions that don't
appear in most documentation. These codes are therefore not included in any of
the monitors available. The most useful of these codes are the multi-byte NOP
instructions. These instructions have the same effect as the normal NOP with
the exception that one or two bytes following are ignored. Using a two byte
NOP before a three byte instruction with the byte to be ignored as, for example,
$20 (JSR) or $4C (JMP) will result in the code looking like garbage upon
disassembly. 2 byte NOPs:

$04,$14,$34,$44,$54,$64,$74, and $F4

3 byte NOPs:

$0C,$1C,$3C,$5C,$7C,$DC, and $FC

Three byte NOPs are useful since the next two bytes could contain a 2 byte
The Commodore 64 Kernal and Hardware Revealed

instruction which will read well with the rest of the code but is in fact ignored. For example, on assembly:

```
BYT $44,$4C  !2 BYTE NOP
JSR $FFD5  !LOAD FILE
BYT $3C  !3 BYTE NOP
LDX #$00  !IGNORED
STX $2D  !STORE END LO
BYT $7C  !3 BYTE NOP
LDY #$00  !IGNORED
STY $2E  !STORE END HI
BYT $74,$20  !2 BYTE NOP
JMP $A474  !GOTO ‘READY.’
```

actually does:

```
JSR $FFD5
STX $2D
STY $2E
JMP $A474
```

but on disassembly it gives:

```
033C 44    ???
033D 4C 20 D5 JMP $D520
0340 FF    ???
0341 3C    ???
0342 A2 00 LDX #$00
0344 86 2D STX $2D
0346 7C    ???
0347 A0 00 LDY #$00
0349 84 2E STY $2E
034B 74    ???
034C 20 4C 74 JSR $744C
034F A4 xx LDY $xx
```

The byte xx has nothing to do with the code.

The subject of program protection and security methods is one which can be gone into in great depth but unfortunately it would be inadvisable to give more information than has been included since a knowledge of how to protect a program will also tell the intending pirate how to break that protection. Readers interested in adding protection and security to their programs should write to Zifra Software Ltd., 40 Bowling Green Lane, London EC1. Zifra have considerable experience and expertise in security and protection methods for both tape and disk which have been used on Zifra products.
Chapter Five
The User Port

5.1 The I/O ports and the 6526

The CBM 64 communicates with peripheral devices via five integrated circuits. The most important of the five is the 6510 microprocessor. This has a single eight line I/O port which is used principally to control memory bank switching but also some of the tape operations. The 6566 VIC chip controls the video display and has a light pen input. The sound output is generated by a 6581 SID chip, which also has four analog joystick inputs (see Fig. 5.1 for I/O connections on the CBM 64). The other two integrated circuits are 6526 complex interface adapters or CIAs, which are used to perform all the other I/O functions of the CBM 64. We can summarise the function of these two chips as follows:

- Keyboard input
- User port
- Cassette deck
- Serial I/O – used by the disk drive and printer
- RS232 I/O – for printers, modems etc.
- Joystick – simple switch type
- IRQ timing for real time clock and keyboard

The two CIA chips which are used to control all these functions have between them just 32 programmable I/O lines and 8 handshake lines; many of these lines are thus used by more than one of the above functions.

![Diagram of CBM 64 I/O ports]

1) Game I/O  2) Memory expansion  3) Audio and video  4) Serial I/O  5) Cassette  6) User port  7) Modulated TV output

Fig. 5.1. The position of the different CBM 64 I/O outputs.
The Commodore 64 Kernel and Hardware Revealed

SERIAL I/O

<table>
<thead>
<tr>
<th>PIN #</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SERIAL SRQ IN</td>
</tr>
<tr>
<td>2</td>
<td>GND</td>
</tr>
<tr>
<td>3</td>
<td>SERIAL ATN IN/OUT</td>
</tr>
<tr>
<td>4</td>
<td>SERIAL CLK IN/OUT</td>
</tr>
<tr>
<td>5</td>
<td>SERIAL DATA IN/OUT</td>
</tr>
<tr>
<td>6</td>
<td>NC</td>
</tr>
</tbody>
</table>

AUDIO/VIDEO

<table>
<thead>
<tr>
<th>PIN #</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LUMINANCE</td>
</tr>
<tr>
<td>2</td>
<td>GND</td>
</tr>
<tr>
<td>3</td>
<td>AUDIO OUT</td>
</tr>
<tr>
<td>4</td>
<td>COMP VIDEO</td>
</tr>
<tr>
<td>5</td>
<td>AUDIO IN</td>
</tr>
</tbody>
</table>

GAME I/O

Port #1

<table>
<thead>
<tr>
<th>PIN #</th>
<th>TYPE</th>
<th>NOTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>JOY0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>JOY1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>JOY2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>JOY3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>POT Y</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>LIGHT PEN/</td>
<td>MAX. 100mA</td>
</tr>
<tr>
<td></td>
<td>BUTTON A</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>+5V</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>POT X</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 5.1. (contd.)
An understanding of the two 6526 CIA interface chips is essential if all the features of the CBM 64 are to be used to the full, and a knowledge of these chips helps to explain some of the quirks of the system. The functioning of the chips is controlled by internal programmable registers, and there are 16 registers in each chip. These 32 registers (16 from each chip) are located in addressable memory space and are located at hex $OC00$ to $DOFF$ (decimal 56320 to 56831). They can thus be accessed from Basic using PEEK and POKE statements and from machine code using LDA and STA commands.

Of the 40 I/O lines output from the two CIA chips the user can directly connect equipment to, and control the functioning of or input from, 21 lines; the other 18 lines are used by the keyboard or memory bank select and are not therefore particularly usable; one line is not connected. All but two of the 1/0 lines on CIA #2 can be used, but only three of the lines on CIA #1. CIA #2 is thus used in all the examples in this section. The functions of each I/O line from the two CIA chips are shown in Fig. 5.2 and the electrical connections which allow the user to utilise some of the lines are shown in Fig. 5.3. Though these lines are all assigned particular functions the user is not confined to using a particular
I/O line for the function designated for that line. This is because all the I/O lines are under software control and it is not until the routines within the operating system which utilise that line for a particular function are called that that line is used. This flexibility allows the redefinition of I/O line function and is one of the most useful features of the CBM 64.

The 6526 is a very complex chip with sixteen different addressable registers. Each bit within these registers has a specific function, either as an input or an output or to control the operation of the 6526. The registers are of six basic types; I/₀, data direction, peripheral control, shift register, timers and timer control registers.

The 6526 can be functionally divided into two component parts. On one side are the connections to the processor; the processor interface. On the other side are the input/output lines; the peripheral interface. The main components of the processor interface are the eight bi-directional data lines. These are connected directly to the processor data bus and are used to transfer data between the CIA and the processor. As with any memory, the processor treats the 6526 as a sixteen byte block of memory. The direction of data transfer is controlled by the R/W line, the exact timing of a transfer being controlled by the φ₂ clock line. The individual registers are addressed by the register select lines connected to the bottom address lines A₀–A₃. The exact location of the 6526 within memory space is determined by decoding some of the address lines and connecting these to the chip select inputs. The registers of the 6526 will be accessed only if chip select CS is low. As with all the I/O chips the 6526 can generate a processor interrupt by pulling the IRQ line low. This occurs whenever an internal interrupt flag is set as a result of an input on one of the peripheral control lines.

The processor interface lines have seven basic functions which can be summarised as follows:

1) Phase two clock (φ₂) – data transfers between the 6526 and the processor take place only when the φ₂ clock is high. This clock also acts as a time base for the internal 6526 timers and shift register. On the CBM 64 the φ₂ clock is derived from the 6510 microprocessor chip which in turn is derived from the ϕ₀ clock produced by the VIC chip. The φ₂ clock has a frequency of 0.98 MHz on a PAL machine (UK version 64) and 1.02 MHz on an NTSC (US version 64).
2) Chip select line (CS) – the chip select input is connected to the decoding circuitry connected to the PLA chip.
3) Register select lines (RS₀, RS₁, RS₂, RS₃) – the four register select lines are connected to the processor address bus lines A₀ – A₃. This allows the register to select one of the sixteen registers in the 6526.
4) Read/write line (R/W) – the direction of data transfer between the 6526 and the processor is controlled by the R/W line. If R/W is high then a ‘read’ operation is performed and data is transferred from the 6526 onto the data bus. If R/W is low then a ‘write’ operation is performed and data currently on the data bus is loaded into the addressed register of the 6526.

5) Data bus (DB0 to DB7) – data is transferred between the processor and the 6526 via the eight bi-directional lines of the data bus. The internal data bus of the 6526 will only be connected to the processor data bus when the two chip select lines are enabled and the φ2 clock is high. The direction of data transfer will depend on the state of the R/W line and the register addressed on lines RS0 to RS3.

6) Reset (RES) – the reset line clears all the internal registers of the 6526 (except the timers and shift register) and sets them all at logic zero. The result is that all the interface lines are put in the input state, and timers, shift register and interrupts are all disabled. This is connected to the processor power up circuitry and is used only when the system is switched on (this line is accessible externally and since the system software can be changed its function could be modified).

7) Interrupt request (IRQ) – the interrupt request output from the 6526 is very important in the CBM 64. The IRQ line goes low whenever an internal interrupt flag is set and the corresponding interrupt enable flag is high. On CIA#2 the IRQ line is connected to the processor NMI interrupt line; the NMI line is also used to test the RESTORE key. On CIA#1 the IRQ line is connected to the processor IRQ line. The function of this line is to generate a regular 60 Hz interrupt which is used by the clock, I/O and keyboard routines; this interrupt is provided by Timer A in the CIA.

5.2 The peripheral interface lines

The peripheral interface lines on each 6526 CIA chip are divided into two I/O ports, each port having eight bi-directional I/O lines. The two ports share two handshaking and two serial control lines. The following is a brief description of the I/O buses and control lines of a 6526.

1) Peripheral ports A and B (PA0 – PA7) and (PB0 – PB7) – these ports consist of eight bi-directional lines each of which can be independently programmed under control of the data direction register to act as either an input or an output. The polarity of the lines defined as outputs is controlled by the contents of the output register. The internal control registers are used by the processor to control the modes of operation of the 6526. All lines represent a load of one standard TTL gate in the input mode and will drive two standard TTL loads in the output mode.

2) Handshaking lines (FLAG, PC) – the FLAG peripheral control line acts as interrupt input, and PC as handshake output for peripheral port B. Each line controls an internal interrupt flag with a corresponding interrupt enable bit. The various modes of operation are controlled by the processor via the internal control registers of the 6526.
5.3 Operation of the I/O ports

Three registers are required to access each of the eight line peripheral ports; they are a data direction register, an output register and an input register. Each port has a data direction register for specifying whether each of the eight lines acts as either an input or an output. A zero in a bit of the data direction register causes the corresponding peripheral line to act as an input. A one causes the line to act as an output.

Example: Set lines 0 to 3 as inputs and 4 to 7 as outputs on port B of CIA#2.

<table>
<thead>
<tr>
<th>I/O line number</th>
<th>Data direction</th>
<th>DDR contents if line = in</th>
<th>DDR contents for example</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>in</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>in</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>in</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>in</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>out</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>out</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>out</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>7</td>
<td>out</td>
<td>128</td>
<td>128</td>
</tr>
</tbody>
</table>

Total for example – 240

Command is POKE 56579, 240

Each peripheral line is connected to an input register and an output register. When a line is programmed to act as an output the voltage on that line is controlled by the corresponding bit in the output register. A ‘1’ in the output register causes the corresponding line to go high, and a ‘0’ causes it to go low.

Example: Output to port B of CIA#2 using the data direction set out in the previous example. Lines 4 and 7 are high and lines 5 and 6 are low.

<table>
<thead>
<tr>
<th>I/O line number</th>
<th>Data direction of line</th>
<th>High or low</th>
<th>Value of line in I/O reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>in</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>1</td>
<td>in</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>2</td>
<td>in</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>3</td>
<td>in</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>4</td>
<td>out</td>
<td>high</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>out</td>
<td>low</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>out</td>
<td>low</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>out</td>
<td>high</td>
<td>128</td>
</tr>
</tbody>
</table>

Total for example 144

Command is POKE 56577, 144
Reading one of the peripheral port registers causes the contents of the input register to be transferred onto the data bus. With input latching disabled the contents of the input registers will always reflect the data currently on all the peripheral port lines.

Example: Read the contents of the input lines of port B of CIA #2 set up in the example on data direction, and store the contents as variable A.

\[ A = \text{PEEK}(56577) \text{ AND } 15 \]

The AND 15 masks off the lines used as outputs; they must be removed since the current state of the output lines is stored in the input register. AND commands can then be used to determine which lines are high and which are low.

5.3.1 Registers used in the operation of the I/O ports

Register 1 Parallel port A I/O register
CIA#1 - Hex $DC00$ decimal 56320
CIA#2 - Hex $DD00$ decimal 56576

This register contains the contents of the input and output lines of port A of the 6526.

Register 2 Parallel port B I/O register with handshake control
CIA#1 - Hex $DC01$ decimal 56321
CIA#2 - Hex $DD01$ decimal 56577

This register contains the contents of the input and output lines of port B. It is identical to that of port A, except that this register has control over the handshake line PC. The PC line goes low for one clock cycle following either a read or write to the port B peripheral I/O register.

Register 3 Data direction register for port A
CIA#1 - Hex $DC02$ decimal 56322
CIA#2 - Hex $DD02$ decimal 56578

This register controls each of the eight lines on port A and determines whether they are acting as inputs or outputs. A one in any of the eight bits of this register sets the corresponding line into the output mode, and a zero puts it into the input mode.

Register 4 Data direction register for port B
CIA#1 - Hex $DC03$ decimal 56323
CIA#2 - Hex $DD03$ decimal 56579

This register controls each of the eight lines on port B and determines whether they are acting as inputs or as outputs. A one in any of the eight bits of this register sets the corresponding line into the output mode, and a zero puts it into the input mode.

5.3.2 Handshaking

The term handshaking is used to refer to signals which control or synchronise the transfer of data between the computer and another device. The 6526 has two handshaking lines for use on data transfers; the FLAG input and the PC output. The PC line is used to indicate that data is ready to be transmitted or received on port B. This is indicated by the PC line going low for one clock cycle following a
read or write of the port B data register. The FLAG line is an input which, on receiving a negative transition, will set the FLAG bit in the interrupt control register. The FLAG line can be used to detect the PC output from another 6526.

5.4 The interval timers and counters of the 6526

The 6526 has three internal interval timers (Timer A, Timer B, and TOD). One of these, TOD, is a 24 hour time of day clock. Timer A will also function as a counter of pulses input on one of the I/O lines. These timers are not only useful but are of vital importance to the operation of the CBM 64. It is these timers which are used to control the generation of the 60 Hz interrupt used to update the real time clock and scan the keyboard. They are also used to control the timing of I/O on the serial port, the RS232 port and the cassette. Since the CBM 64 interface uses two 6526 chips there are a total of six timers available for use by the system software. The timers are used in conjunction with the processor interrupts. The following table shows some of the functions of each timer plus the interrupt line affected:

<table>
<thead>
<tr>
<th>CIA#1 IRQ interrupt</th>
<th>Timer A</th>
<th>System 60 Hz interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Real time clock updating</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Keyboard scanning</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cassette read/write timing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>User programmable functions</td>
</tr>
<tr>
<td>Timer B</td>
<td></td>
<td>Cassette read/write timing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Serial port timing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>User programmable functions</td>
</tr>
<tr>
<td>TOD</td>
<td></td>
<td>User programmable functions</td>
</tr>
</tbody>
</table>

Note that Timer A is used for both updating the kernel software's real time clock and cassette timing; for this reason the real time clock loses whenever the cassette is used.

CIA#2 NMI interrupt

<table>
<thead>
<tr>
<th>Timer A</th>
<th>RS232 port I/O timing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>User programmable functions</td>
</tr>
<tr>
<td>Timer B</td>
<td>RS232 port I/O timing</td>
</tr>
<tr>
<td></td>
<td>User programmable functions</td>
</tr>
<tr>
<td>TOD</td>
<td>User programmable</td>
</tr>
</tbody>
</table>

5.4.1 Timer A and Timer B

Each interval timer consists of two eight bit latches and a sixteen bit counter. Each timer occupies two of the 6526 registers; register numbers 5 to 8. Their locations in the CBM 64 are as follows:
Register 5  -  Timer A low order byte
CIA#1  -  Hex $D04  decimal 56324
CIA#2  -  Hex $D04  decimal 56580
Register 6  -  Timer A high order byte
CIA#1  -  Hex $D05  decimal 56325
CIA#2  -  Hex $D05  decimal 56581
Register 7  -  Timer B low order byte
CIA#1  -  Hex $D06  decimal 56326
CIA#2  -  Hex $D06  decimal 56582
Register 8  -  Timer B high order byte
CIA#1  -  Hex $D07  decimal 56327
CIA#2  -  Hex $D07  decimal 56583

These registers are used to load values into the counters. After loading, the counter decrements at the system clock rate (0.98 MHz on PAL machines and 1.02 MHz on NTSC). Thus if the counter is loaded with its maximum value (all sixteen bits = 1 or decimal 65535) it will be decremented to zero in 0.0669 seconds. Upon reaching zero, an interrupt flag is set and one of the two interrupt lines will go low and generate a processor interrupt. The timer will thus disable any further interrups until the interrupt servicing routine reads the interrupt control register of the 6526. In addition, the timer can be instructed to invert the output level on one of the peripheral I/O lines each time it 'times out'. The modes of operation are controlled by reading or writing to the four timer registers plus the two control registers and the interrupt register. The two timers A and B can be linked to create a single 32 bit counter which is capable of creating long delays. Program 7 demonstrates the operation of a 32 bit timer.

10 POKE56583,13: POKE56582,13
20 POKE56581,255: POKE56580,255
30 POKE56590,16+1: POKE56591,64+16+8+1
35 T1=T1
40 IF(PEEK(56589) AND 2)<2 THEN 40
50 PRINT(TI-T1)/60

Program 7.

5.4.2 Time of day clock – TOD
This is a general purpose 24 hr am/pm timer with a 1 10 second resolution. This timer occupies four registers within the 6526. They are located and organised as follows in the CBM 64:

<table>
<thead>
<tr>
<th>Reg #</th>
<th>CIA#1 location</th>
<th>CIA#2 location</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>$D08</td>
<td>$D08</td>
<td>TOD 1/10 secs</td>
</tr>
<tr>
<td>9</td>
<td>$D09</td>
<td>$D09</td>
<td>TOD secs</td>
</tr>
<tr>
<td>10</td>
<td>$D0A</td>
<td>$D0A</td>
<td>TOD mins</td>
</tr>
<tr>
<td>11</td>
<td>$D0B</td>
<td>$D0B</td>
<td>TOD hours</td>
</tr>
</tbody>
</table>
Each of these registers stores its data as shown in the following table. It should be noted that the values are stored in BCD form rather than straight binary:

<table>
<thead>
<tr>
<th>Register</th>
<th>Byte</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 – TOD 1/10 sec</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>T8</td>
<td>T4</td>
<td>T2</td>
<td>T1</td>
<td></td>
</tr>
<tr>
<td>9 – TOD seconds</td>
<td>0</td>
<td>shi4</td>
<td>shi2</td>
<td>shi1</td>
<td>slo8</td>
<td>slo4</td>
<td>slo2</td>
<td>slo1</td>
<td></td>
</tr>
<tr>
<td>10 – TOD minutes</td>
<td>0</td>
<td>mhi4</td>
<td>mhi2</td>
<td>mhi1</td>
<td>mlo8</td>
<td>mlo4</td>
<td>mlo2</td>
<td>mlo1</td>
<td></td>
</tr>
<tr>
<td>11 – TOD hours</td>
<td>PM flag</td>
<td>0</td>
<td>0</td>
<td>hhi</td>
<td>hlo8</td>
<td>hlo4</td>
<td>hlo2</td>
<td>hlo1</td>
<td></td>
</tr>
</tbody>
</table>

shi, mhi and hhi are the decade portion of the respective second, minute or hour and slo, mlo and hlo are the unit portion.

The TOD clock is timed by a 50 or 60 Hz external clock pulse provided for the system interrupt timing (50 Hz is found on US machines and 60 Hz on UK machines). This external clock frequency must always be set to the correct value by setting the required bit in control register .A bit 7, to 0 = 60 Hz and 1 = 50 Hz. The TOD time registers can be preset by writing to them the required time values. This must, however, be done in the correct sequence. The TOD clock is stopped when a write is performed to the hours register. The clock will not start again until a write is performed to the 1/10 seconds register, thereby allowing accurate time setting.

When reading the contents of the TOD registers there is the problem of the register values changing whilst they are being read. This is overcome by latching all the register values on a read of the hours register. Whilst the values are latched the TOD clock will continue to count but will not affect the register values until the 1/10 seconds register is read; this disables the latches. Any of the registers (apart from hours) can be read without latching providing the problem of carry between registers is not important.

The TOD clock incorporates an alarm feature which causes an interrupt to be generated whenever the time reaches a preset value. The alarm is set by first setting bit 7 of control register .B to 1 and then writing the desired alarm time value into the four TOD registers. Having written the desired values into these registers, control register .B bit 7 should be reset to zero. Program 8 demonstrates the use of the TOD clock registers and the alarm feature.
The User Port

5.5 Serial data register (SDR)

One of the registers of the 6526, register 12, functions as a serial in/parallel out or parallel in/serial out shift register. The serial input or output from this register is connected to the SP pin on the chip and is designed to be used in conjunction with the CNT line to allow serial communications between 6526 chips. Data is clocked in or out of the shift register using either Timer A or the CNT pulses. In the input mode data is clocked in off the SP line on the rising edge of a clock pulse applied to the CNT line. Each pulse on the CNT line clocks in one bit of data, represented by the state of the SP line. After 8 CNT pulses the data in the shift register is transferred to the serial data register and an interrupt to the processor is generated.

In the output mode, data is loaded into the serial data register. Timer A is used to generate the timing rate at which data is clocked out. Timer A is first set into continuous running mode and data will be shifted out at half the timer underflow rate, the highest possible data rate being one quarter of the $f_2$ clock (245 KHz on a UK version of the 64 and 255 KHz on a US version). As soon as the data is written into the SDR transmission will commence, assuming that Timer A has already been set into continuous operation. Every time an underflow is generated by Timer A a CNT pulse is generated. A bit from the SDR data is shifted onto the SP line, becomes valid on the falling edge of the CNT pulse and remains valid until the next falling edge of a CNT pulse. After eight CNT pulses the entire contents of the SDR will have been transmitted on the SP line and an interrupt is then generated to indicate that data transmission
has been completed. On completion of transmission the CNT line will go high and the SP line will stay at the level of the last data bit transmitted. If a new byte of data is loaded into the SDR before the transmission of the previous byte has been completed, the 6526 will complete transmission of the current byte, and then instead of generating an interrupt, will carry on and transmit the second byte. In this manner continuous transmission can be achieved.

An important potential use for the serial line is in serial communications between computers and/or peripheral devices which also use the 6526. Program 9 shows how two or more CBM 64 computers can be connected via the CNT and SP lines, plus ground, to enable the transmission of programs and data between the two machines. This could easily be expanded to work with more machines.

```
0 REM RUN TO SEND FIRST
1 REM OR RUN70 TO RECEIVE FIRST
2 REM
5 INPUTA$:A$=A$+CHR$(13)
10 POKE56590,2:POKE56591,0
20 POKE56590,64+16+1
25 FORI=1TOLEN(A$)
26 J=ASC(MID$(A$,I,1))
30 POKE56590,J:PRINTCHR$(J);
50 IF (PEEK(56590)AND8)<8 THEN 50
60 NEXT I:POKE56590,0
65 IF (PEEK(56590)AND8)<3 THEN 65
70 POKE56590,PEEK(56590) AND (128+63)
80 POKE56590,0
90 IF (PEEK(56590)AND8)<3 THEN 90
100 IF (PEEK(56590)AND8)<8 THEN 100
105 IF (PEEK(56590)AND8)<3 THEN 105
110 GOTO5
10 POKE56579,1
20 POKE56326,255:POKE56327,255
30 POKE56335,32+16+1
40 PRINT"J"
50 PRINT"*65535-(PEEK(56326)+PEEK(56327)*256)
60 FORI=0TO100:NEXT
70 GETA$:IF A$="THEN70
80 POKE56577,1:POKE56577,0
90 GOTO 50
10 POKE56590,PEEK(56590) AND(128+63)
20 POKE56579,1
30 GETA$:IF A$="THEN30
40 POKE56577,0:POKE56577,1
50 IF (PEEK(56590)AND8)<8 THEN30
60 PRINTPEEK(56588):GOTO30
```

Program 9.

5.6 The interrupt control register (ICR)

The 6526 can generate interrupts in several different ways. There are altogether five sources of interrupts. The source of the interrupt is identified by examining which bit is set in the interrupt control register; this is register 13 of the 6526. The functions of the individual bits of this register are:
The ICR, in fact, consists of two separate registers; the interrupt flag register which is read only and the interrupt mask register which is write only. When an interrupt occurs the corresponding bit in the interrupt flag register is set, providing it has previously been enabled by the mask register; an interrupt to the processor is also generated. The mask register is used to control which function or functions can create an interrupt. An interrupt will occur only when the corresponding mask register bit is set. When enabling an interrupt the mask register, with bit 7 = 1, is set by writing the appropriate bit pattern to register 13, the ICR. If, when writing to the ICR, bit 7 is cleared then all mask bits set to one will be cleared whilst all mask bits set to zero will remain in their previous state. Any interrupt which is enabled by the mask register will set bit 7 of the ICR flag register and thereby cause the IRQ pin to go low and generate an interrupt. The interrupt is cleared by reading the interrupt flag register. The interrupts on the CBM 64 are examined in greater detail in the Chapter 6.

5.7 The 6526 control registers (CRA and CRB)

The two control registers of the 6526 are used, as their names imply, to control the actual functioning and modes of the timers and the serial port. Each bit in the two registers has a separate control function; they can be summarised as follows:

<table>
<thead>
<tr>
<th>Control register A</th>
<th>Bit</th>
<th>State</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>start Timer A</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>stop Timer A; in one shot mode this bit is reset on underflow</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>line PB6 functions normally as an I/O line</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0</td>
<td>pulse output on PB6 (only if bit 1 set)</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>toggle output on PB6 (only if bit 1 set)</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0</td>
<td>Timer A in continuous running mode</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1</td>
<td>Timer A in one shot mode</td>
</tr>
</tbody>
</table>
Control register A

<table>
<thead>
<tr>
<th>Bit</th>
<th>State</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
<td>no effect</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>forces the Timer A counter to be loaded from the Timer A latch</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>Timer A counts φ2 clock pulses</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Timer A counts positive transitions on the CNT line</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>SP line in input mode using external shift pulses on CNT line</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>SP line in output mode. CNT sources shift pulses</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>TOD clock timing pulse at 60 Hz (use on US machines)</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>TOD clock timing pulse at 50 Hz (use on UK machines)</td>
</tr>
</tbody>
</table>

Control Register B

<table>
<thead>
<tr>
<th>Bit</th>
<th>State</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>stop Timer B</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>start Timer B; in one shot mode this bit is reset on underflow</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>normal I/O operation on the PB7 line</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>output from Timer B appears on line PB7</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>pulse output on PB7 (only if bit 1 of CRB set)</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>toggled output on PB7 (only if bit 1 of CRB set)</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>Timer B in continuous running mode</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Timer B in one shot mode</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>no effect</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>forces the Timer B counter to be loaded from the Timer B latch</td>
</tr>
<tr>
<td>5,6</td>
<td></td>
<td>these two bits select one of four input modes for Timer B:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Timer B counts φ2 clock pulses</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Timer B counts positive CNT transitions</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Timer B counts Timer A underflow pulses</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Timer B counts Timer A underflow pulses while the CNT line is high</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>7</th>
<th>0</th>
<th>writing to TOD registers will set TOD clock</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>writing to TOD registers will set TOD alarm</td>
</tr>
</tbody>
</table>

5.8 Parallel interfacing

In some applications it is simply not possible to connect devices directly to the eight user port I/O lines. This is generally for one of two reasons. The first is that
the user port lines do not output enough power to drive the device; directly connecting the device to the user port could result in damage to the CIA chip. The second reason is that there are insufficient I/O lines for the application. The first type of problem can be overcome by using relays and opto-isolators; the second type by using multiplexing techniques to expand the available number of I/O lines.

The eight I/O lines from the CIA chip are in output mode only, capable of each driving the input of a single TTL chip. The simplest method of improving this drive capability is to put a buffer onto each output line; this will expand the drive capability of each line to 10 TTL chip inputs. This is adequate if the I/O lines are used just to control some TTL circuitry, but it is inadequate for controlling power devices such as motors, relays and lamps; for these a power driver circuit is needed. The simplest power driver circuit consists of a single transistor whose base is connected to the output of a buffer and whose outputs are connected on one side to the power supply and on the other to the device to be driven. It is, however, much easier to use one of the peripheral driver ICs like the SN75446. This chip is used in the circuit diagram in Fig. 5.4. The SN75446 is capable of driving devices requiring up to 50 volts and 400 mA. The circuit shows it driving two relays.

![Fig. 5.4. Dual relay control circuit.](image)

If there is any risk of a high voltage accidentally being present on one of the I/O lines it is advisable to protect the computer by using opto-isolators on lines. An opto-isolator simply consists of an optically coupled LED and photo transistor. An example of such a device which provides four separate opto-isolators which work on 5 volt lines is the ILQ74.

The most commonly needed I/O expansion is the requirement to test the state of a large number of input lines. This kind of I/O expansion is used when adding a special keyboard to the machine or testing switches in a security alarm system.
A circuit of this kind is shown in Fig. 5.5. It uses four 16 line to 4 line demultiplexers which convert the state of the 16 lines into a 4 bit binary value. This value is fed into four lines of the user port. The chip select line of each of the four demultiplexers is controlled by four output lines of the user port. Using this circuit the computer can scan the state of up to 64 input lines.

5.8.1 Parallel port application example
Program 10 is an example of how the parallel port can be used. This program is designed to allow disabled people to use a computer, both to write and use programs and to control various devices such as TV, radio, lamps etc. The program requires line 0 of the user port to be connected to a switch, the other side of which is connected to ground. The switch can be a simple microswitch usable with minimum finger pressure, or a suck/blow switch controllable by the user blowing into it. The other I/O lines are connected to devices which are to be controlled, and each line will require connection via a power driver circuit and relay, as in Fig. 5.4. A selection of possible applications is shown in the program.
The User Port

220 DATA N.O.R.P,J,#.OF
230 DATA C,D.H,M.X@IS
240 DATA B.W.Y,K.Ê.T0
250 DATA 0.2.4.6.8.Ô.IN
260 DATA1.3.5.7.9.ê.THAT
270 DATA ÷,÷,÷,÷,÷,ê÷ê
280 DATA ÷,÷,÷,÷,÷ê÷êê
290 DATA ÷,÷,÷,÷,÷ê÷êê
300 DATA ÷,÷,÷,÷,÷ê÷êê
310 DATA ÷,÷,÷,÷,÷ê÷êê
320 DATA ÷,÷,÷,÷,÷ê÷êê
330 DATA ÷,÷,÷,÷,÷ê÷êê
340 DATA ÷,÷,÷,÷,÷ê÷êê
350 DATA ÷,÷,÷,÷,÷ê÷êê
360 DATA ÷,÷,÷,÷,÷ê÷êê
370 DATA ÷,÷,÷,÷,÷ê÷êê
380 DATA ÷,÷,÷,÷,÷ê÷êê
390 DATA ÷,÷,÷,÷,÷ê÷êê
400 DATA ÷,÷,÷,÷,÷ê÷êê
410 DATA ÷,÷,÷,÷,÷ê÷êê
420 DATA ÷,÷,÷,÷,÷ê÷êê
430 DATA ÷,÷,÷,÷,÷ê÷êê
440 DATA ÷,÷,÷,÷,÷ê÷êê
450 DATA ÷,÷,÷,÷,÷ê÷êê
460 DATA ÷,÷,÷,÷,÷ê÷êê
470 DATA ÷,÷,÷,÷,÷ê÷êê
480 DATA ÷,÷,÷,÷,÷ê÷êê
490 DATA ÷,÷,÷,÷,÷ê÷êê
500 DATA ÷,÷,÷,÷,÷ê÷êê
510 DATA ÷,÷,÷,÷,÷ê÷êê
520 DATA ÷,÷,÷,÷,÷ê÷êê
530 DATA ÷,÷,÷,÷,÷ê÷êê
540 DATA ÷,÷,÷,÷,÷ê÷êê
550 DATA ÷,÷,÷,÷,÷ê÷êê
560 DATA ÷,÷,÷,÷,÷ê÷êê
570 DATA ÷,÷,÷,÷,÷ê÷êê
580 DATA ÷,÷,÷,÷,÷ê÷êê
590 DATA ÷,÷,÷,÷,÷ê÷êê
600 DATA ÷,÷,÷,÷,÷ê÷êê
610 DATA ÷,÷,÷,÷,÷ê÷êê
620 DATA ÷,÷,÷,÷,÷ê÷êê
630 DATA ÷,÷,÷,÷,÷ê÷êê
640 DATA ÷,÷,÷,÷,÷ê÷êê
650 DATA ÷,÷,÷,÷,÷ê÷êê
660 DATA ÷,÷,÷,÷,÷ê÷êê
670 DATA ÷,÷,÷,÷,÷ê÷êê
680 DATA ÷,÷,÷,÷,÷ê÷êê
690 DATA ÷,÷,÷,÷,÷ê÷êê
700 DATA ÷,÷,÷,÷,÷ê÷êê
710 DATA ÷,÷,÷,÷,÷ê÷êê
720 DATA ÷,÷,÷,÷,÷ê÷êê
730 DATA ÷,÷,÷,÷,÷ê÷êê
740 DATA ÷,÷,÷,÷,÷ê÷êê
750 DATA ÷,÷,÷,÷,÷ê÷êê
760 DATA ÷,÷,÷,÷,÷ê÷êê
770 DATA ÷,÷,÷,÷,÷ê÷êê
780 DATA ÷,÷,÷,÷,÷ê÷êê
790 DATA ÷,÷,÷,÷,÷ê÷êê
800 DATA ÷,÷,÷,÷,÷ê÷êê
810 DATA ÷,÷,÷,÷,÷ê÷êê
820 DATA ÷,÷,÷,÷,÷ê÷êê
830 DATA ÷,÷,÷,÷,÷ê÷êê
840 DATA ÷,÷,÷,÷,÷ê÷êê
850 DATA ÷,÷,÷,÷,÷ê÷êê
860 DATA ÷,÷,÷,÷,÷ê÷êê
870 DATA ÷,÷,÷,÷,÷ê÷êê
880 DATA ÷,÷,÷,÷,÷ê÷êê
890 DATA ÷,÷,÷,÷,÷ê÷êê
900 DATA ÷,÷,÷,÷,÷ê÷êê
910 DATA ÷,÷,÷,÷,÷ê÷êê
920 DATA ÷,÷,÷,÷,÷ê÷êê
930 DATA ÷,÷,÷,÷,÷ê÷êê
940 DATA ÷,÷,÷,÷,÷ê÷êê
950 DATA ÷,÷,÷,÷,÷ê÷êê
960 DATA ÷,÷,÷,÷,÷ê÷êê
970 DATA ÷,÷,÷,÷,÷ê÷êê
980 DATA ÷,÷,÷,÷,÷ê÷êê
990 DATA ÷,÷,÷,÷,÷ê÷êê
1000 DATA ÷,÷,÷,÷,÷ê÷êê
1010 DIM DF$(2.6),DS$(1.9.6),TV$(20),TV$(20)
1020 FORI=0 TO2:FORJ=0 TO6
1030 READ DF$(I,J)
1040 NEXTJ: NEXTI
1050 FORI=0 TO9
1060 FORJ=0 TO6
1070 FORK=0 TO6
1080 READ DA$(I,J,K)
1090 IF LEN DA$(I,J,K)=1 THEN DA$(I,J,K)=" "+DA$(I,J,K)" "
1100 NEXTK: NEXTJ
1110 FOR J=0 TO9
1120 FOR K=0 TO6
1130 READ TA$(I,J,K)
1140 NEXTK: NEXTJ
1150 NEXTI
1160 TA$(0.5.5)=CHR$(34)
1170 TA$(1.3.3)=CHR$(34)
1180 FOR I=0 TO20
1190 TV$(I)=" ":TV$(I)="  
1200 NEXTI
1210 UP=56578:UX=1
1220 VN=0: M=0: AR=" ": UC=1: ED=0: HP=0: TV=0: NY=0
1230 PD=70: PQ=50: FM=0
1240 GOSUB 20000
2000 PRINT "HOMEHOMEOFFICE";
2010 PRINT "HOMEHOMEOFFICE";
2020 PRINT "HOMEHOMEOFFICE"
2030 FOR J=0 TO9:PRINT " ";
2040 FOR J=0 TO9:PRINT DA$(M,I,J);:NEXTI
2050 NEXTJ: NEXTI
2060 PRINT "HOMEOFFICEFUNCTIONS"; "VARIABLES";
2070 PRINT "HOMEOFFICEFUNCTIONS";
2080 FOR I=0 TO9:PRINT " ";
2090 FOR J=0 TO9:PRINT TV$(I*7+J);
2100 NEXTJ: NEXTI
2110 PRINT "HOMEOFFICEFUNCTIONS"; "VARIABLES";
2120 PRINT "HOMEOFFICEFUNCTIONS";
2130 FOR I=0 TO9:PRINT " ";
2140 FOR J=0 TO9:PRINT DF$(I,J);
2150 NEXTJ: NEXTI
2160 NEXTI
3840 PRINT"DllillllljJ); DF$(II, J); 
3850 NEXT J 
3855 PRINT"DllillllljJ); DF$(II, J-1);: GOTO3200 
3860 PRINT"DllillllljJ); DF$(II, J); 
3870 SS=(I*I+J)+1 
3875 IF NY>0 GOTO7720 
3880 IF HP=2GOTO44000 
4000 IF SS>7GOTO4020 
4010 ONSGOTO4100.4300.4500,4700,4900,5100,5300 
4020 IF SS>14GOTO4040 
4030 ONS7GOTO5500,5700,5900,6100,6300,6500,6700 
4040 ONSSGOTO6900,7100,7300,7500,7700,7900,8100 
4100 REM SET TO LOWER CASE 
4110 UC=0:GOSUB2000:GOTO~2ee 
4300 REM BEGIN A SENTENCE 
4310 UC=3:GOSUB20000:A$=A$+" • ":GOT03008 
4500 REM DELETE CHARACTER 
4505 IFLEN(A$)>GOOTO3200 
4510 IFLEN(A$)>1GOTO4500 
4520 AS="":FL$="":GOTO3000 
4530 AS=LEFT$(A$,LEN(A$)-1):FL$="":GOTO3000 
4700 REM DELETE WORD 
4701 IFLEN(A$)>GOOTO3000 
4702 IFLEN(A$)>1GOTO4745 
4705 FORI=LEN(A$)+1TO2STEP-1 
4710 NEXTI 
4745 IFLEN(A$)=1ANDLEFT$(A$,1)<>" THENGOSUB4800 
4750 GOTO3000 
4760 CIFFLS="":FL$="":RETURN 
4900 REM NOT DEFINED 
4910 GOTO30200 
5100 REM NOT DEFINED 
5110 GOTO30200 
5300 REM CR-SENDRCR/LF TO PRINTER 
5310 OPENPR,PR 
5320 PRINTPR 
5330 CLOSE PR 
5340 GOTO3000 
5500 REM SET TO UPPER CASE 
5510 UC=1:GOSUB20000:GOTO30200 
5700 REM PARAGRAPH 
5710 OPENPR,PR 
5720 GOSUB21000:GOSUB20900 
5725 GOSUB20000 
5730 PRINTPR:PRINTPR:PRINTPR 
5740 AS="": 
5750 UC=3 
5760 CLOSE PR 
5770 GOTO3000 
5900 REM PRINT CONTENTS OF A$ 
5910 OPEN PR,PR 
5920 GOSUB21000:GOSUB20900 
5930 AS="": 
5940 CLOSEPR 
5950 GOTO3000 
6100 REM SEND LINE TO MICRO 
6110 OPEN MC,MC 
6120 PRINTMC,A$ 
6125 GOSUB20900 
6130 AS="": 
6140 CLOSEMC 
6150 GOTO3000 
6300 REM ADD A MAKE A$ THE NEXT VARIABLE 
6310 IF VN<21 THEN 6340 
6320 PRINT"ADVICE - NO ROOM FOR NEW VARIABLE"; 
6330 GOTO3000 
6335 IF A$="":THEN3000 

The User Port 145
The Commodore 64 Kernel and Hardware Revealed

6340 TV$(VN)=A$
6350 DV$(VN)=LEFT$(TV$(VN)+" ",5)
6360 DV$(VN)=LEFT$(DV$(VN),4)
6365 DV$(VN)=DV$(VN)+CHR$(32)
6370 GOSUB22000
6380 VN=VN+1
6390 GOTO30000
6500 REM SEND ASCII 3 TO MICRO <BREAK-IN>
6510 OPEN MC,M C
6520 PRINT#MC,CHR$(3);CHR$(3);
6530 CLOSE MC
6540 GOTO30000
6700 REM UNDEFINED
6710 GOTO 3200
6900 REM TEXT MODE
6910 M=0:GOTO2000
7100 REM PROGRAM MODE
7110 M=1:UC=1:GOSUB20000:GOTO2000
7300 REM USER MODE
7310 GOTO30000
7500 REM HELP SUB-SYSTEM
7510 IF VN=0 GOTO7710
7700 REM DELETE VARIABLE
7701 IF VN=0 GOTO3000
7703 PRINT"!!!ADVICE - NO VARIABLES TO REMOVE!";
7705 GOTO3000
7710 PRINT"!!!ADVICE - SELECT VARIABLE TO BE REMOVED!";
7713 VN=1
7715 GOTO3000
7720 REM UNKNOWN VARIABLE
7725 PRINT"!!!ADVICE - NO VARIABLE SELECTED! ";
7730 NV=0
7735 GOTO3000
7740 REM DELETE SELECTED VARIABLE
7745 PRINT"!!!ADVICE - O.K. ";
7750 FOR K=II*7+J TO 19
7755 DV$(K)=DV$(K+1)
7760 TV$(K)=TV$(K+1)
7765 NEXT K
7760 DV$(20)=" 
7770 TV$(20)="
7780 GOSUB 22000
7785 NV=0
7790 VN=VN-1
7795 GOTO3000
7900 REM UNDEFINED
7910 GOTO3200
8100 REM DELETE BUFFER
8110 GOSUB20000:R$="";GOTO3000
10000 PRINT"!!!ADVICE NO HELP IN THIS VERSION";
10005 HP=0
10010 GOTO3000
20000 PRINT"!!!ADVICE: UNDEFINED LC ";
20010 PRINT"!!!ADVICE: UNDEFINED UC* ";
20020 IF$(1,0)=" UC* ":DF$(0,0)=" LC "
20030 RETURN
20100 PRINT"!!!ADVICE: UNDEFINED LC* ";
20110 PRINT"!!!ADVICE: UNDEFINED UC ";
20120 IF$(1,0)=" UC* ":DF$(0,0)=" LC* 
20130 RETURN
20900 PRINT"!!!
20910 FORL=1TO10
20920 PRINT"
20930 NEXT
20940 RETURN
21000 PRINT#PR,A$ ;
21010 RETURN
22000 PRINT"!!!ADVICE: UNDEFINED ";
22010 FORI=0TO2:PRINT"
22020 FORJ=0TO6:PRINTDV$(I*7+J)";}
REM USER FRAME
30010 PRINT "J *** SELECT FUNCTION ***"
30020 PRINT
30030 PRINT "T.V. ON"
30040 PRINT "T.V. OFF"
30050 PRINT "B.B.C. 1"
30060 PRINT "B.B.C. 2"
30070 PRINT "I.T.V.";
30080 PRINT "RADIO ON"
30090 PRINT "RADIO OFF"
30100 PRINT "LAMP ON"
30110 PRINT "LAMP OFF"
30120 PRINT "DOWN FASTER"
30130 PRINT "DOWN SLOWER"
30140 PRINT "ACROSS FASTER"
30150 PRINT "ACROSS SLOWER"
30160 PRINT "RETURN"
30170 PRINT "RETURN";
30180 IF (PEEK <UP) AND UX) =0 GOTO 30180
30190 IF (PEEK <UP) AND UX) GOTO 30190
30200 PRINT "Muslim"
30210 FOR I = 1 TO 14
30220 PRINT "Muslim"
30230 FOR J = 1 TO PD
30240 IF (PEEK <UP) AND UX) GOTO 30300
30250 NEXT J
30260 PRINT "Muslim"
30270 NEXT I
30280 PRINT "Muslim"
30290 GOTO 30170
30300 PRINT "Muslim"
30310 IF I > 7 GOTO 30330
30320 ON I GOTO 30400, 30500, 30600, 30700, 30800, 30900, 31000
30330 ON I = 7 GOTO 31100, 31200, 31300, 31400, 31500, 31600, 31700
30400 REM TURN TV ON
30410 POKE UP, PEEK <UP) AND 251
30420 GOTO 30660
30500 REM TURN TV OFF
30510 POKE UP, PEEK <UP) OR 4
30520 GOTO 30170
30600 REM BBC1
30610 POKE UP, PEEK <UP) OR 3
30620 GOTO 30170
30700 REM BBC2
30710 POKE UP, PEEK <UP) OR 3) AND 254
30720 GOTO 30170
30800 REM ITV
30810 POKE UP, PEEK <UP) OR 3) AND 253
30820 GOTO 30170
30900 REM RADIO ON
30910 POKE UP, PEEK <UP) AND 247
30920 GOTO 30170
31000 REM RADIO OFF
31010 POKE UP, PEEK <UP) OR 8
31020 GOTO 30170
31100 REM LAMP ON
31110 POKE UP, PEEK <UP) AND 239
31120 GOTO 30170
31200 REM LAMP OFF
31210 POKE UP, PEEK <UP) OR 16
31220 GOTO 30170
31300 REM DOWN FASTER
31310 PD = PD - 10: IF PD < 30 THEN PD = 30
31320 GOTO 30170
31400 REM DOWN SLOWER
31410 PD = PD + 10
31420 GOTO 30170
31500 REM ACROSS FASTER
The Commodore 64 Kernel and Hardware Revealed

31510 PQ=PQ-19: IF PQ<30 THEN PQ=30
31520 GOTO38170
31600 REM ACROSS SLOWER
31610 PQ=PQ+10
31620 GOTO38170
31700 REM RETURN
31710 GOTO20000
40000 REM HELP WITH LETTERS
40100 GOSUB 209000
40200 PRINT"BY SELECTING "':DA$(M,I,J):" ;
40300 IF LEN(TA$(M,I,J))=0 THEN PRINT"" "NOTHING"
40400 IF LEN(TA$(M,I,J))=1 THEN PRINT"" THE CHARACTER "":DA$(M,I,J):" "
40500 IF LEN(TA$(M,I,J))>1 THEN PRINT"" THE STRING "":TA$(M,I,J):" "
40600 PRINT"IS ADDED TO THE BUFFER."
40700 GOTO490000
42000 REM HELP WITH VARIABLES
42100 GOSUB 299000:PRINT"; 
42200 IF LEN(TV$(I*7+J))=0 THEN PRINT"" "IS EMPTY;"
42300 PRINT"SEE +VAR AND -VAR."
42400 GOTO490000
42600 PRINT"BY SELECTING VARIABLE "':DV$(I*7+J):" THEN" 
42700 PRINT"T":TV$(I*7+J):" "IS ADDED"
42800 PRINT"TO THE BUFFER."
42900 GOTO490000
44000 REM HELP WITH FUNCTIONS
44100 GOSUB 209000:PRINT"; 
44200 IF SS=7 GOTO 44030
44300 ON SS GOTO450000,451000,452000,453000,454000,455000,456000 
44400 IF SS=14 GOTO 44050
44500 ON SS GOTO450000,451000,452000,453000,454000,455000,456000,457000,458000,459000 
45000 PRINT"; LC "FOLLOWING LETTERS WILL BE LOWER"
45100 PRINT"CASE. ALSO SEE / UC "." 
45200 GOTO490000
45300 PRINT"; ENDT END OF SENTENCE, ADDS A FULL"
45400 PRINT";STOP AND THREE SPACES TO THE BUFFER"
45500 PRINT"AND MAKES THE NEXT LETTER A CAPITAL."
45600 GOTO490000
45700 PRINT";DELCL DELETES THE MOST RECENT"
45800 PRINT"CHARACTER IN BUFFER."
45900 GOTO490000
45900 PRINT";DELEW DELETES THE MOST RECENT WORD"
46000 PRINT";IN THE BUFFER, NOT INCLUDING SPACES."
46100 GOTO490000
46200 PRINT";NOT DEFINED - CODE AT 4900"
46300 GOTO490000
46400 PRINT";NOT DEFINED - CODE AT 5100"
46500 GOTO490000
46600 PRINT";NL NEWLINE ON PRINTER, DOES NOT"
46700 PRINT";AFFECT BUFFER."
46800 GOTO490000
46900 PRINT"; UC FOLLOWING LETTERS WILL BE" 
47000 PRINT";UPPER CASE. ALSO SEE / UC "." 
47100 GOTO490000
47200 PRINT"; PARA END OF PARAGRAPH, PRINTS"
47300 PRINT";BUFFER, THREE NEW LINES, AND SETS"
47400 GOTO490000
47500 PRINT";BUFFER TO THREE SPACES."
47600 GOTO490000
47700 PRINT";SEND PRINTS CONTENTS OF BUFFER."
47800 GOTO490000
47900 PRINT";ERASES BUFFER AND TAKES A NEWLINE."
48000 GOTO490000
48100 PRINT"; ESC SENDS BUFFER TO 2ND. MICRO;"
48200 PRINT";TAND ESCAPE CHARACTER."
48300 GOTO490000
48400 PRINT";VAR SAVES THE CONTENTS OF THE BUFFER"
48500 PRINT";IN THE NEXT FREE VARIABLE LOCATION."
48600 PRINT";SHOWS FIRST FOUR CHARACTERS."
48700 GOTO490000
48800 PRINT"; TC SENDS CONTROL C (ASCII<3>) TO"
The User Port

5.9 Voice synthesis

Adding a voice synthesiser to the CBM 64 is both simple and cheap, and probably one of the easiest ways is to use the General Instrument SP0256 speech processor chip. This IC is connected directly to the user port and its output is simply fed via an amplifier to the audio input line on the SID chip and thence to the monitor or TV speaker. This circuit is shown in Fig. 5.6.

The SP0256 is an allophone speech generator which can be used to synthesise any English word by concatenating the individual speech sounds (phonemes) which comprise the word. Within this chip is a table of 64 different allophones and pauses; a full list of these is given in Table 5.1. These are accessed via the chip's six address lines. By having the user port connected directly to these six address lines we can generate any required allophone simply by outputting the correct address to these six lines. Normal speech contains between ten and twelve allophones per second, and consequently allophone synthesis is a very compact way of storing speech. The major advantage of allophone synthesis is that it can provide an unlimited vocabulary with fairly low storage requirements.

When using this circuit to generate speech it must be realised that the allophones do not necessarily correspond directly to the written letters and therefore a word must first be converted to its phonetic form. Thus because of
Fig. 5.6. Voice synthesiser circuit.
irregularities in spelling it is necessary to use the sounds of the word rather than
the letters when dealing with speech allophones. A second problem is the
segmentation of speech. This means that although we think of a spoken word as
consisting of a sequence of separate sounds which correspond to a letter name,
in fact speech sound is a continuously varying signal which cannot easily be
broken into discrete units. This accounts for the occasional problems of
intelligibility in allophone synthesised speech. A third problem is that the ear
will perceive the same acoustic signal differently depending on the sounds
which precede or follow it. Thus the initial p in ‘pop’ is different from the p in
‘spy’. An attempt is made to overcome some of these problems by the design of
the allophone sounds and by careful selection of allophones to describe a
particular word.

The individual sounds of language are called phonemes. In each language
these are slightly different. The SP0256 is designed to give English phonemes.
The phonemes can be divided into three different categories; consonants,
vowels, and speech sounds such as aspirants and pauses. Tables 5.2 to 5.6 show
the consonant and vowel phonemes and how allophones can be used. Program
11 shows how to use the voice synthesis circuit and allophones from Basic.
The Commodore 64 Kernal and Hardware Revealed

1 REM *************************************************
2 REM * SPEECH DATA TO USER PORT * 
3 REM *************************************************
4 REM 
5 POKE54272+24,15
10 DD=56576
20 POKE DD+3,127
35 READDD: IFD=-1 THEN END
40 POKE DD+1,D OR 64
50 POKE DD+1,D AND 63
51 IF PEEK(DD+1)<128 THEN S
60 GOTO 35
70 DATA 42,59,45,3
80 DATA 3,39,12,50,59,21,3
90 DATA 3,26,11,33,3
91 DATA 3,46,52,41,3
93 DATA 3,46,52,41,3
100 DATA 43,12,40,39,26,3
110 DATA 12,11,40,58,16,20,37,15,11,3
120 DATA 13,7,42,56,53,45,53,10,19,3,-1

Program 11.

Table 5.2. Examples of spelling irregularities. (Reproduced by courtesy of General Instruments).

<table>
<thead>
<tr>
<th>Vowels</th>
<th>meat, feet, Pete, people, penny</th>
<th>vein, foreign, deism, deicer, geisha</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consonants</td>
<td>ship, tension, precious, nation</td>
<td>although, gasly, cough</td>
</tr>
</tbody>
</table>

Table 5.3. Consonant phonemes of English (Reproduced by courtesy of General Instruments).

<table>
<thead>
<tr>
<th>Stops:</th>
<th>Labial¹</th>
<th>Labio-Dental²</th>
<th>Inter-Dental³</th>
<th>Alveolar⁴</th>
<th>Palatal⁵</th>
<th>Velar⁶</th>
<th>Glottal⁷</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voiceless</td>
<td>PP, BB</td>
<td>TT, DD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Voiced</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>KK, GG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fricatives:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Voiceless</td>
<td>WH, VV</td>
<td>SS, ZZ</td>
<td></td>
<td></td>
<td>SH, ZH*</td>
<td>HH</td>
<td></td>
</tr>
<tr>
<td>Voiced</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Affricates:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Voiceless</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CH, JH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Voiced</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nasals:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Voiced</td>
<td>MM</td>
<td>NN</td>
<td></td>
<td></td>
<td>NG*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resonants:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Voiced</td>
<td>WW</td>
<td>RR, LL</td>
<td></td>
<td></td>
<td>YY</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

¹ These do not occur in word-initial position in English.

1. Upper and Lower Lips Touch or Approximate
2. Upper Teeth and Lower Lip Touch
3. Tongue Between Teeth
4. Tip of Tongue Touches or Approximates Alveolar Ridge (just behind upper teeth)
5. Body of Tongue Approximates Palate (roof of mouth)
6. Body of Tongue Touches Velum (posterior portion of roof of mouth)
7. Glottis (opening between vocal cords)
Table 5.4. Vowel phonemes of English (Reproduced by courtesy of General Instruments).

<table>
<thead>
<tr>
<th>HIGH</th>
<th>FRONT</th>
<th>CENTRAL</th>
<th>BACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>YR</td>
<td></td>
<td></td>
<td>UW#</td>
</tr>
<tr>
<td>IY</td>
<td></td>
<td></td>
<td>UH*</td>
</tr>
<tr>
<td>IH*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MID</td>
<td>EY</td>
<td>ER</td>
<td>OW#</td>
</tr>
<tr>
<td>EH*</td>
<td>AX*</td>
<td></td>
<td>OY#</td>
</tr>
<tr>
<td>XR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOW</td>
<td>AE*</td>
<td>AW#</td>
<td>AO*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AY</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AR</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AA*</td>
<td>#</td>
</tr>
</tbody>
</table>

* SHORT VOWELS  # ROUNDED VOWELS

Table 5.5. Examples of words made from allophones (Reproduced by courtesy of General Instruments).

<table>
<thead>
<tr>
<th>Example</th>
<th>Allophone</th>
</tr>
</thead>
<tbody>
<tr>
<td>“daughter”</td>
<td>DD2-AO-TT2-ER1</td>
</tr>
<tr>
<td>“sister”</td>
<td>KK3-AX-LL-AY-DD1</td>
</tr>
<tr>
<td>“clown”</td>
<td>SS-SS-IH-SS-TT2-ER1</td>
</tr>
<tr>
<td>“cookie”</td>
<td>KK1-LL-AW-NN1</td>
</tr>
<tr>
<td>“letter”</td>
<td>KK3-UH-KK1-IY</td>
</tr>
<tr>
<td>“little”</td>
<td>LL-EH-TT2-ER</td>
</tr>
<tr>
<td>“uncle”</td>
<td>LL-IH-TT2-EL</td>
</tr>
<tr>
<td>“computer”</td>
<td>AX-NG-KK3-EL</td>
</tr>
<tr>
<td>“two”</td>
<td>KK1-AX-MM-PP1-YY1-UW1-TT2-ER</td>
</tr>
<tr>
<td>“extent”</td>
<td>EH-KK1-SS-TT2-EH-EH-NN1-TT2</td>
</tr>
<tr>
<td>“score”</td>
<td>TT2-UW2</td>
</tr>
<tr>
<td>“fur”</td>
<td>AX-LL-AR-MM</td>
</tr>
<tr>
<td></td>
<td>SS-KK3-OR</td>
</tr>
<tr>
<td></td>
<td>FF-ER2</td>
</tr>
</tbody>
</table>

Table 5.6. Guidelines for using the allophones (Reproduced by courtesy of General Instruments).

<table>
<thead>
<tr>
<th>Silence</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PA1 (10ms)</td>
<td>before BB, DD, GG, and JH</td>
</tr>
<tr>
<td>PA2 (30ms)</td>
<td>before BB, DD, GG, and JH</td>
</tr>
<tr>
<td>PA3 (50ms)</td>
<td>before PP, TT, KK, and CH, and between words</td>
</tr>
<tr>
<td>PA4 (100ms)</td>
<td>between clauses and sentences</td>
</tr>
<tr>
<td>PA5 (200ms)</td>
<td>between clauses and sentences</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Short Vowels</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>*/IH/</td>
<td>sitting, stranded</td>
</tr>
<tr>
<td>*/EH/</td>
<td>extent, gentlemen</td>
</tr>
<tr>
<td>*/AE/</td>
<td>extract, acting</td>
</tr>
<tr>
<td>*/UH/</td>
<td>cookie, full</td>
</tr>
<tr>
<td>*/AO/</td>
<td>talking, song</td>
</tr>
<tr>
<td>*/AX/</td>
<td>lapel, instruct</td>
</tr>
<tr>
<td>*/AA/</td>
<td>pottery, cotton</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Long Vowels</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>*/IY/</td>
<td>treat, people, penny</td>
</tr>
<tr>
<td>*/EY/</td>
<td>great, statement, tray</td>
</tr>
<tr>
<td>*/AY/</td>
<td>kite, sky, mighty</td>
</tr>
<tr>
<td>*/OY/</td>
<td>noise, toy, voice</td>
</tr>
<tr>
<td>*/UW1/</td>
<td>after clusters with YY: computer</td>
</tr>
<tr>
<td>*/UW2/</td>
<td>in monosyllabic words: two, food</td>
</tr>
<tr>
<td>*/OW/</td>
<td>zone, close, snow</td>
</tr>
<tr>
<td>*/AW/</td>
<td>sound, mouse, down</td>
</tr>
</tbody>
</table>
Table 5.6. Continued.

| R-Colored Vowels |  
|---|---|
| /ER1/ | letter, furniture, interrupt |
| /ER2/ | monosyllables: bird, fern, burn |
| /OR/ | fortune, adorn, store |
| /AR/ | farm, alarm, garment |
| /YR/ | hear, earring, irresponsible |
| /XR/ | hair, declare, stare |

| Resonants |  
|---|---|
| /WW/ | we, warrant, linguist |
| /RR1/ | initial position: read, write, x-ray |
| /RR2/ | initial clusters: brown, crane, grease |
| /LL/ | like, hello, steel |
| /EL/ | little, angle, gentlemen |
| /YY1/ | clusters: cute, beauty, computer |
| /YY2/ | initial position: yes, yarn, yo-yo |

| Voiced Fricatives |  
|---|---|
| /VV/ | vest, prove, even |
| /DH1/ | word-initial position: this, then, they |
| /DH2/ | word-final and between vowels: bathe, bathing |
| /ZZ/ | zoo, phase |
| /ZH/ | beige, pleasure |

| Voiceless Fricatives |  
|---|---|
| */FF/ | These may be doubled for initial position |
| */TH/ | and used singly in final position |
| */SS/ | shirt, leash, nation |
| /SH/ |  |
| /HH1/ | before front vowels: YR, IY, IH, EY, EH, XR, AE |
| /HH2/ | before back vowels: UW, UH, OW, OY, AO, OR, AR |
| /WH/ | white, whim, twenty |

| Voiced Stops |  
|---|---|
| /BB1/ | final position: rib; between vowels: fibber; in clusters: bleed, brown |
| /BB2/ | initial position before a vowel: beast |
| /DD1/ | final position: played, end |
| /DD2/ | initial position: down; clusters: drain |
| /GG1/ | before high front vowels: YR, IY, IH, EY, EH, XR |
| /GG2/ | before high back vowels: UW, UH, OW, OY, AX; and clusters: green, glue |
| /GG3/ | before low vowels: AE, AW, AR, AA, AO, OR, ER; in medial clusters: anger; and final position: peg |

| Voiceless Stops |  
|---|---|
| /PP/ | pleasure, ample, trip |
| /TT1/ | final clusters before SS: tests, its |
| /TT2/ | all other positions: test, street |
| /KK1/ | before front vowels: YR, IY, IH, EY, EH, XR, AY, AE, ER, AX; |
| | initial clusters: cute, clown, scream |
| /KK2/ | final position: speak; final clusters: task |
| /KK3/ | before back vowels: UW, UH, OW, OY, OR, AR, AO; |
| | initial clusters: crane, quick, clown, scream |

| Affricates |  
|---|---|
| /CH/ | church, feature |
| /JH/ | judge, injure |

| Nasal |  
|---|---|
| /MM/ | milk, alarm, ample |
| /NN1/ | before front and central vowels: YR, IY, IH, EY, EH, XR, AE, ER, AX, AW, AY, UW; final clusters: earn |
| /NN2/ | before back vowels: UH, OW, OY, OR, AR, AA |
| /NG/ | string, anger |

* These allophones can be doubled.
5.10 Analog interfacing

In digital systems two voltage levels are used to represent data, with the binary digit 0 represented by a low voltage and binary 1 by a high voltage. Digital data is therefore represented by a series of pulses. Analog signals have an infinite range of voltage levels and are therefore continuous rather than having discrete units of information. The difference between the two types of waveform is shown in Fig. 5.7. The digital equivalent of an analog signal is generated by regularly sampling the waveform, and on each sample converting the measured voltage into a binary value using an analog to digital conversion circuit. The sampling of a waveform is shown in Fig. 5.8. For a computer to be able to create a smooth analog waveform it must at regular intervals output a binary value, which is converted to an analog voltage by a digital to analog conversion circuit. The digital synthesis of an analog waveform is shown in Fig. 5.9.

![Typical digital signal](image1)

![Typical analog signal](image2)

*Fig. 5.7. Comparison of digital and analog waveforms (Reproduced by courtesy of Practical Electronics).*

![Digital sampling of an analog waveform](image3)

*Fig. 5.8. Digital sampling of an analog waveform (Reproduced by courtesy of Practical Electronics).*
Fig. 5.9. Digital synthesis of an analog waveform (Reproduced by courtesy of *Practical Electronics*).

### 5.11 Digital to analog conversion

The circuit used to convert a digital signal to an analog signal is quite simple involving the use of a weighted resistor network. This most commonly takes the form of an R, 2R, 4R, 8R etc. ladder, such as the one shown in Fig. 5.10. In operation each successively lower weighted input produces an output voltage which is exactly half that produced by the preceding input. The voltages from each input are summed at the combined output and a voltage developed which is the analog representation of the binary digital value.

Although a digital to analog conversion circuit can be constructed easily from resistors it is simpler and often more accurate to use one of the many D to A converter ICs. An example of such a D to A IC is the ZN425E. This chip is an 8 bit dual mode A/D and D/A converter, incorporating a voltage reference, resistor network, input switches and an 8 bit binary counter. A block diagram and pin out for this IC is shown in Fig. 5.11. The 8 bit counter is used with an external comparator IC to facilitate analog to digital conversion. The logic

Fig. 5.10. Digital to analog converter using R-2R register network (Reproduced by courtesy of *Practical Electronics*).
select pin determines which function, A/D or D/A, is being used. The on-chip 2.5 volt reference signal ensures that the conversion is very accurate (0.2% accuracy). This chip is very easy to connect to the processor; the circuit diagram for connecting it is shown in Fig. 5.12. It should be noted that the output from this chip is a voltage ranging between 0 volts and 2.5 volts (an increase of 1 in the output to the D to A therefore gives a corresponding increase of approximately 0.01 volts). A buffer amplifier is necessary to change this voltage range, particularly if a positive to negative ranging voltage is desired, and calibrate the output.

The ability to generate an analog signal from the computer has many applications, probably the most obvious is in the generation of simple or complex analog waveforms. The technique for outputting a repeating waveform of frequency is quite simple. The fundamental requirement is a waveform table, usually one page (256 bytes) long. This table contains a sequence of precalculated values which define the waveform shape. By repeatedly outputting all the values in the table, in sequence, to the digital to analog converter a regular waveform is generated. The frequency of the waveform can be varied either by varying the delay between outputting each
Fig. 5.12. Analog to digital and digital to analog conversion circuit.
The User Port

LOC CODE LINE
C291  .LIB  FUNC.SIMPLE
C291  ; ROUTINE TO OUTPUT A FUNCTION
C291  ; TABLE OVER THE USER PORT TO A
C291  ; DIGITAL TO ANALOG CONVERTER.
C291  ;
C291  ; THE FREQUENCY IS CONTROLLED
C291  ; BY EITHER THE TIMER VALUE OR
C291  ; INCREASING/DECREASING THE
C291  ; SAMPLING VALUE.
C291  ;
C291  20 FD AE  FUNC:JSR $AEFD  ; GET VALUE FOR
C291  20 8A AD  JSR $AD8A  ; TIMER A
C291  20 F7 B7  JSR $B7F7
C291  A9 D1  LDA #NMI  ; SET NMI VECTOR
C291  8D 19 03  STA $0319
C291  8D FA FF  STA $FFFA
C291  A9 C2  LDA #NMI
C291  8D 19 03  STA $0319
C291  8D FB FF  STA $FFFB
C291  A5 14  LDA #14
C291  8D 04 DD  STA $0004  ; STORE TIMER VALUE
C291  A5 15  LDA #15
C291  8D 05 DD  STA $0005
C291  A9 FF  LDA #FF
C291  8D 03 DD  STA $0003  ; USER PORT TO OUTPUT
C291  A9 0F  LDA #0F
C291  8D 18 DD  STA $0000+24  ; MAX SID VOLUME
C291  A9 00  LDA #00
C291  8D 01 C3  STA FLAG  ; POINTER TO TABLE
C291  AD 0D DD  LDA $0DD  ; CLEAR NMIS
C291  A9 31  LDA #S1  ; SET TIMER A NMI
C291  8D 00 DD  STA $0000
C291  A9 11  LDA #11  ; START TIMER A
C291  8D 0E DD  STA $0DDE
C291  60  RTS
C291  ;
C291  48  NMI  PHA  ; STORE REGISTERS
C291  8A  TXA
C291  48  PHA
C291  98  TYA
C291  48  PHA
C291  A9 01  LDA #01  ; CAUSED BY TIMER A?
C291  AD 0D DD  BIT $0DD
C291  DD 11  BNE SEND  ; YES
C291  DD 7F  LDA #7F  ; CLEAR ENABLED NMIS
C291  DF 0D DD  STA $0DD
C291  EE 2D 84 FF  JSR $FF84  ; RESET I/O
C291  EE 20 8A FF  JSR $FF8A  ; RESET KERNEL
C291  EE 68  NMEXIT PLA  ; PULL REGISTERS
C291  EE A8  TAY
C291  EE A8  PLA
C291  EE AA  TXA
C291  EE 68  PLA
C291  EE 40  RTI
C291  ;
C291  AE 01 C3  SEND  LDX FLAG  ; GET POINTER
C291  F1 BD 13 C3  LDA TABLE,X  ; GET BYTE
C291  F4 8D 01 DD  STA $0DD  ; SEND TO USER PORT
C291  FF 8A  TXA
C291  F8  18  CLC
C291  F9  63 01  STORE ADC #03  ; ADD SAMPLING VALUE
C291  FB 8D 01 C3  STA FLAG
C291  FF 4C 8E C2  JMP NMEXIT  ; EXIT NMI
C291  ;
C291  00  FLAG .BYTE 0
value or for higher frequencies by sampling the table rather than using every value, with the frequency being determined by the sampling step.

The machine code routine in Program 12 can be used to output a waveform using a predefined waveform table over a range of different frequencies, the frequency being controlled by the value stored in Timer A. This routine is designed such that the analog output is connected to the sound input of the SID chip so that the sound generated by the waveform can be heard. The Basic program in Program 13 is used in conjunction with Program 12 to create a waveform table, the waveform being created either from a mathematical function or drawn directly using either a joystick or the cursor keys. The resulting waveform is displayed on the screen using high resolution graphics. It should be noted that in order to obtain the graphics, the graphics routines in Program 14 should be present. These are integrated with the frequency generation routine so that they can both reside in memory together.

Program 12.

```
10 PRINT"JUSE:" 20 PRINT"1 - JOYSTICK"
30 PRINT"2 - KEYBOARD"
40 PRINT"3 - FUNCTION"
50 GETA$: IFA$="" THEN 50
60 A=VAL(A$): IFA$="" THEN 50
70 T=49939: SY$49627
110 ONAGOSUB 1000, 2000, 3000
120 GETA$: IFA$="" THEN 120
```
130 SYS49790
140 END
1600 REM
1610 REM INPUT WAVEFORM USING JOYSTICK
1620 REM
1630 X=0:Y=T(X)-27
1640 A=PEEK(56320):E=0:W=0:S=0:N=0
1650 IF (A AND 16) = 0 THEN RETURN
1660 IF (A AND 5) = 0 THEN E=1:GOTO 1110
1670 IF (A AND 4) = 0 THEN W=-1:GOTO 1110
1680 IF (A AND 2) = 0 THEN S=-1:GOTO 1110
1690 IF (A AND 1) = 0 THEN N=1:GOTO 1110
1700 GOTO 1040
1710 GOSUB 11100:GOTO 1040
2000 REM
2010 REM INPUT WAVEFORM USING KEYBOARD
2020 REM
2030 X=0:Y=T(X)-27
2040 GETA$:E=0:W=0:S=0:N=0
2050 IF A$="\r" THEN RETURN
2060 IF A$="L" THEN E=1:GOTO 2110
2070 IF A$="R" THEN S=-1:GOTO 2110
2080 IF A$="U" THEN N=1:GOTO 2110
2090 GOTO 2940
2100 GOSUB 11100:GOTO 2940
3000 REM
3010 REM PUT UP A WAVEFORM DEFINED IN
3020 REM FNF
3030 REM
3040 DEF FNF(Z)=128+(31-.122*Z)*SIN(Z*PI/128)12
3050 FORI=0TO255
3060 'I=FNF(I)
3070 POKET+Z,"':T(J)=
3080 NEXTI:RETURN
11100 IF E OR W THEN 11120
11110 IF (Y+S+N) > 199 THEN RETURN
11112 SYS49537,X,Y:Y=Y+S+N
11115 TXC=(Y+27):POKET+,X,(Y+27):SYS49537,X,Y:RETURN
11120 X=(X+E+W)AND255:Y=(Y+S+N)
11130 TXC=Y+27:POKET+,X,Y+27:RETURN

Program 13.
The Commodore 64 Kernel and Hardware Revealed

LOC CODE' LINE

C02E 85 14 STR #14
C030 A5 15 LDA #15
C032 69 00 ADC #$00
C034 C9 A0 CMP #$A0 ;WRAPAROUND?
C036 D0 02 BNE ZLOOP1 ;NO
C038 A9 78 LDA #$78
C03A 85 15 ZLOOP1 STA #15 ;INCREASE X
C03C E6 FD INC #FD ;COORDINATE BY 1
C03E D0 02 BNE ZLOOP2 ;COORDINATE BY 1
C040 E6 FE INC #FE
C042 A5 FE ZLOOP2 LDA #FE
C044 C9 01 CMP #$01 ;DONE 320 POINTS?
C046 D0 DA BNE ZLOOP ;NOT YET
C048 A5 FD LDA #FD
C04A C9 40 CMP #$40
C04C D0 DA BNE ZLOOP ;NO
C04E 60 RTS ;YES, EXIT
C04F
C050 85 5B ZPLOT STA #5B ;STORE THE Y
C051 A9 00 LDA #00 ;COORDINATE
C053 85 5C STA #5C
C055 A5 FD LDA #FD ;STORE THE X
C057 35 59 STA #59 ;COORDINATE
C059 A5 FE LDA #FE
C05B 85 5A STA #5A
C05D 4C 84 C1 JMP PLOTIT+3 ;PLOT THE POINT
C060 .END
C061 78 .LIB READ.SIMPLE
C063 D0 0B LDA #0B ;DISABLE IRQ
C065 8D 11 D0 STR #$D011
C066 A9 00 LDA #00 ;SET POINTER TO
C068 35 FD STA #FD ;START OF TABLE
C06A A9 78 LDA #$78
C06C 85 FE STA #FE
C06E A2 01 RLOOP1 LDX #$01 ;SAMPLE RATE
C070 20 92 C0 RLOOP2 JSR SETUP ;READ VALUE
C073 CA DEX
C074 D0 FA BNE RLOOP2 ;IGNORE SAMPLE
C076 A9 00 LDA #00
C078 91 FD STA ($FD),Y ;STORE THE VALUE
C07A A9 01 LDA #$01 ;INCREASE TABLE
C07C 18 CLC ;POINTER BY 1
C07D 65 FD ADC #FD
C07F 35 FD STA #FD
C081 A5 FE LDA #FE
C083 69 00 ADC #$30
C085 85 FE STA #FE
C087 C9 A0 CMP #$A0 ;END OF TABLE?
C089 D0 E3 BNE RLOOP1 ;NO
C08B A9 18 LDA #$18 ;RESTORE SCREEN
C08D 8D 11 D0 STR #$D011
C090 58 CLI ;ENABLE IRQ
C091 60 RTS
C092
C094 8D 03 DD STR #$D03 ;TO OUTPUT
C097 A9 FB LDA #$FB
C099 8D 02 DD STR #$D02 ;(PA2)
C09C A9 80 ATOD LDA #$80 ;SET A TO D
C09E D0 01 DD STR #$D01 ;START VALUE
C0A1 35 61 STR #61
C0A3 D0 01 DD ALOOP1 LDA #$D01 ;START MAIN LOOP
C0A6 05 61 ORA #61
C0A8 8D 01 DD STR #$D01
C0BAD NOP
C0C9 NOP
C0CD AD 00 DD LDA #$D00 ;INPUT FROM
The User Port

LOC  CODE  LINE

C0B0  6A  ROR A ; COMPARATOR INTO
C0B1  6A  ROR A ; CARRY
C0B2  6A  ROR A
C0B3  B0 0E  BCS LOOP2 ; VALUE TOO SMALL
C0B5  A5 61  LDA #61 ; VALUE TOO LARGE
C0B7  4D 01 DD  EOR #ID01 ; TRY HALF VALUE
C0BA  8D 01 DD  STA #ID01
C0BD  A5 61  LSR #61 ; DECREASE SEARCH STEP
C0BF  F0 0C  BEQ EXIT ; COMPLETE
C0C1  10 E0  BPL AL00P1 ; TRY AGAIN
C0C3  A5 61  LOOP2 LSR #61 ; DECREASE SEARCH STEP
C0C5  EA  NOP ; ADJUST TIMING
C0C6  EA  NOP
C0C7  EA  NOP
C0C8  EA  NOP
C0C9  A5 61  LDA #61 ; TRY AGAIN
C0CB  D0 D6  LDA #ID01 ; VALUE RETURNED IN
C0D0  60  RTS ; A

C0D1  .END
C0D2  .LIB DOT.SIMPLE
C0D1  ;
C0D1  ; ROUTINE TO CALCULATE LOCATION
C0D1  ; AND BIT(S) FROM THE X AND Y
C0D1  ; COORDINATES.
C0D1  ;
C0D1  A5 5A  DOT LDA #5A ; CHECK THAT X AND Y
C0D3  C9 00  CMP #$00 ; ARE WITHIN BOUNDS
C0D5  F0 0C  BEQ XOK
C0D7  C9 01  CMP #$01
C0D9  D0 06  BNE XER
C0DB  A5 59  LDA #59
C0DD  C9 40  CMP #$40
C0DF  9B 02  BCC XOK
C0E1  38  XER SEC ; TOO LARGE EXIT
C0E2  60  RTS
C0E3  A5 5C  XOK  LDA #5C
C0E5  D0 FA  BNE XER
C0E7  A5 5B  LDA #5B
C0EB  C3 C3  CMP #$C3
C0ED  B0 F4  BCS XER
C0EF  A9 C7  LDA #199
C0F0  38  SEC
C0F2  E5 5B  SBC #5B
C0F4  95 5B  STA #5B
C0F6  A5 59  LDA #59 ; CALCULATE THE BIT TO
C0F8  29 07  AND #$07 ; BE PLOTTED AS
C0F9  85 5E  STA #5E ; 7-(X AND Y)
C0FA  A9 07  LDA #$07
C0FC  38  SEC
C0FD  E5 5E  SBC #5E
C0FF  AA  TAX ; CALCULATE 2*#5E
C100  BD 79 C1  TOP2X,X
C102  85 5E  STA #5E
C105  ;
C105  ; CALCULATE INT(Y/8)*320
C105  ; AND STORE IN #57
C105  ;
C105  A5 5B  LDA #5B
C107  4A  LSR A
C109  4A  LSR A
C10B  4A  LSR A
C10D  0A  ASL A
C10F  AA  TAX
C110  BD 47 C1  MUL320,X
C112  85 57  STA #57
C114  BD 48 C1  MUL320+1,X
C114 85 58 STA $58
C116 ;
C116 ; ADD Y AND 7 TO $57
C116 ;
C116 A5 5B LDA $5B
C118 29 07 AND #$07
C11A 18 CLC
C11B 65 57 ADC $57
C11D 85 57 STA $57
C11F 90 02 BCC XMD70K
C121 E6 58 INC #58
C123 ;
C123 ; CALCULATE INT(X/8)
C123 ;
C123 A0 03 XMD70K LDY #$03
C125 66 5A DIV8 LSR $5A
C127 66 59 ROR $59
C129 88 DEY
C12A D0 F9 BNE DIV8
C12C ;
C12C ; CALCULATE INT(X/8)*8
C12C ;
C12C A0 03 LDV #$03
C12E 96 59 MUL8 ASL $59
C130 26 5A ROL $5A
C132 88 DEY
C133 D0 F9 BNE MUL8
C135 ;
C135 ; ADD INT(X/8)*8 INTO #57
C135 ;
C135 A5 57 LDA $57
C137 18 CLC
C138 65 59 ADC $59
C13A 95 57 STA $57
C13C A5 58 LDA $58
C13E 65 5A ADC $5A
C140 ;
C140 ; ADD #$E000 INTO #57
C140 ;
C140 18 CLC
C141 69 E0 ADC #$E0
C143 65 58 STA #$58
C145 18 CLC
C146 60 RTS
C147 00 00 MUL320 .WOR 0,320,640,960,1280
C149 40 01
C14B 00 02
C14D 00 03
C14F 00 05
C151 40 06 .WOR 1600,1920,2240,2560,2880
C153 00 07
C155 C0 08
C157 00 0A
C159 40 0B
C15B 00 0C .WOR 3200,3520,3840,4160,4480
C15D C0 0D
C15F 00 0F
C161 40 10
C163 00 11
C165 C0 12 .WOR 4800,5120,5440,5760,6080
C167 00 14
C169 40 15
C16B 00 16
C16D C0 17
C16F 00 19 .WOR 6400,6720,7040,7360,7680
C171 40 1A
C173 00 1B
LOC  CODE  LINE

C175  C0 1C
C177  00 1E
C179  01  TOP2X .BYT 1,2,4,8,16,32,64,128
C17A  02
C17B  04
C17C  08
C17D  10
C17E  20
C17F  40
C180  80
C181  .END
C181  .LIB  PLOT.SIMPLE
C181  ; ROUTINE TO PLOT A POINT
C181  ;
C181  20  98  C1  PLOTIT  JSR  GXY  ; GET X AND Y
C184  20  D1  C0  JSR  DOT
C187  90  01  BCC  PLOT1
C189  60  RTS
C18A  20  C9  C1  PLOT1  JSR  KEROUT  ;DISABLE IRQ
C18D  A0  00  LDY  #$00
C18F  B1  57  LDA  ($57),Y  ; OTHERWISE PLOT POINT
C191  45  5E  EOR  #$5E
C193  91  57  STA  ($57),Y
C195  4C  D1  C1  JMP  KERIN
C198  ;
C198  ; GET X AND Y VALUE
C198  ; INTO #$59 AND #$5B
C198  ;
C198  20  FD  AE  GXY  JSR  $AEFD
C19B  20  8A  AD  JSR  $AD8A  ; GET X
C19E  20  BF  B1  JSR  $B1BF  ; FIX IT
C1A1  A6  65  LDX  #$65
C1A3  A4  64  LDY  #$64
C1A5  9E  C7  C1  ; STX TX
C1A8  8C  C9  C1  ; STY TX+1
C1AB  20  FD  AE  JSR  $AEFD  ; CHECK '/,'
C1AE  20  8A  AD  JSR  $AD8A  ; GET Y
C1B1  20  BF  B1  JSR  $B1BF  ; FIX IT
C1B4  A6  65  LDX  #$65
C1B6  A4  64  LDY  #$64
C1B8  86  5B  STX  #$5B
C1BA  84  5C  STY  #$5C
C1BC  AD  C7  C1  LDA  TX
C1BF  85  59  STA  #$59
C1C1  AD  C9  C1  LDA  TX+1
C1C4  85  5A  STA  #$5A
C1C6  60  RTS
C1C7  00  00  TX  .WOR  0
C1C9  ;
C1C9  ;DISABLE KERNAL AND IRQ
C1C9  ;
C1C9  73  KEROUT  SEI
C1CA  A5  01  LDA  #$01
C1CC  29  FD  AND  #$FD  ; SWITCH OUT
C1CE  85  01  STA  #$01
C1D0  60  RTS
C1D1  ;
C1D1  ; ENABLE KERNAL AND IRQ
C1D1  ;
C1D1  48  KERIN  PHA
C1D2  A5  01  LDA  #$01
C1D4  09  02  ORA  #$02  ; SWITCH IN
C1D6  95  01  STA  #$01
C1D8  58  CLI
C1DA  68  PLA
C1DB  60  RTS
C1D8  .END
The Commodore 64 Kernel and Hardware Revealed

LOC CODE LINE

C1DB .LIB MODE.SIMPLE
C1DB
C1DB ; ROUTINE TO SET UP HIRES SCREEN
C1DB
C1DB 20 03 C2 JSR CLRMEM ; CLEAR HIRES SCREEN
C1DE 20 6A C2 JSR CLRSCN ; CLEAR VIDEO SCREEN
C1E1 ;
C1E1 ; GRAPH COMMAND ENTRY
C1E1 ;
C1E1 A9 3B GRAPH LDA #$3B
C1E3 8D 11 60 STA #$D011 ; SELECT BIT MAP MODE
C1E5 A9 3D LDA #$3D
C1E7 8D 18 D0 STA #$D018 ; CHOOSE HIRES SCREEN
C1E9 A9 C8 LDA #$C8
C1ED 8D 16 D0 STA #$D016 ; ELSE SET HIRES MODE
C1F0 AD 02 DD DONE LDA #$D002 ; SELECT BANK 2 FOR HIRES
C1F3 09 03 ORA #$03 ; SCREEN
C1F5 8D 02 DD STA #$D002
C1F7 AD 00 DD LDA #$D000
C1F9 29 FC AND #$FC
C1Fb 613 RTS
C203 ;
C203 ; CLG COMMAND ENTRY
C203 ;
C203 A0 00 CLRMEM LDY #$00 ; LOOP TO CLEAR HIRES
C205 98 TYA
C206 99 00 E0 LOOP STA #$E000,Y
C209 99 00 E1 STA #$E100,Y
C20C 99 00 E2 STA #$E200,Y
C20F 99 00 E3 STA #$E300,Y
C212 99 00 E4 STA #$E400,Y
C215 99 00 E5 STA #$E500,Y
C218 99 00 E6 STA #$E600,Y
C21B 99 00 E7 STA #$E700,Y
C21E 99 00 E8 STA #$E800,Y
C221 99 00 E9 STA #$E900,Y
C224 99 00 EA STA #$EA00,Y
C227 99 00 EB STA #$EBO0,Y
C22A 99 00 EC STA #$EC00,Y
C22D 99 00 ED STA #$ED00,Y
C230 99 00 EE STA #$EE00,Y
C233 99 00 EF STA #$EF00,Y
C236 99 00 F0 STA #$F000,Y
C239 99 00 F1 STA #$F100,Y
C23C 99 00 F2 STA #$F200,Y
C23F 99 00 F3 STA #$F300,Y
C242 99 00 F4 STA #$F400,Y
C245 99 00 F5 STA #$F500,Y
C248 99 00 F6 STA #$F600,Y
C24B 99 00 F7 STA #$F700,Y
C24E 99 00 F8 STA #$F800,Y
C251 99 00 F9 STA #$F900,Y
C254 99 00 FA STA #$FA00,Y
C257 99 00 FB STA #$FB00,Y
C25A 99 00 FC STA #$FC00,Y
C25D 99 00 FD STA #$FD00,Y
C260 99 00 FE STA #$FE00,Y
C263 99 F9 FE STA #$FEF9,Y
C266 88 DEY
C267 D0 9D INE LOOP
C269 60 RTS
C26A A0 00 CLRSCN LDY #$00 ; LOOP TO CLEAR
C26C A9 1B LDA #$1B
C26E 99 00 CC LOOP1 STA #$C000,Y
C271 99 00 CD STA #$D000,Y
Another application for D to A converters involves using two converters, the output of each being connected to the X and Y inputs on an oscilloscope. This configuration can then be used to generate true vector graphics displays; the two D/A converters are switched by using the PA2 line. One of the D/A converters uses only 7 bits with the eighth bit used to control the Z axis or intensity control input on the scope. Alternatively the two D/A converters could be used to control two rotating mirrors for a laser display.

An extension of the waveform generation routine is a music generator. Such a routine is given in Program 15. This is a four voice sophisticated music synthesiser. The program uses four waveform tables, one for each voice. These are shown in Program 16, and their respective waveforms in Fig. 5.13. In addition to the waveform tables it also requires a score table; a sample score table is given in Program 17. It is divided into two sections; the main control table and the music table. It is in two sections for several reasons, the principal one being to allow it to be stored more compactly. The score is compacted by having sections of the score which are identical stored only once and then repeatedly called by the main control table. The control loop also allows the tempo and waveforms of each voice to be changed. This system may seem fairly complex but examination of Program 17 will show that it is fairly straightforward. This program is based on an original idea by Hal Chamberlin in his book *Musical Applications of Microprocessors* published by Hayden Books.

These are the command codes used in the control and music tables:

**Main control table commands**

If byte is FF this specifies that the following byte contains a control code

If followed by 01 then the next byte contains tempo

If followed by 02 then the next four bytes specify the waveform for each voice, each byte being the msb of the start of the specified waveform table

If byte is not FF then each pair of bytes specify the hi, lo address of a pointer into the start of a section of the music table.
The Commodore 64 Kernel and Hardware Revealed

LOC CODE LINE

USRPR =#ID01  :OUTPUT PORT
0000 DDR =#ID03  :DATA DIRECTION
0000 V1PT =#40  :FOUR VOICE WAVEFORM
0000 V2PT =#45  :POINTERS
0000 V3PT =#48  :POINTERS
0000 V4PT =#4B  :POINTERS
0000 INCPT =#4E  :POINTER TO MUSIC
0000 NOTES =#5B  :POINTER TO MUSIC
0000 V11N =#52  :FOUR VOICE INCREMENT
0000 V21N =#54  :POINTERS
0000 V31N =#56  :POINTERS
0000 V41N =#59  :POINTERS
0000 DUR =#5A  :DURATION COUNTER
0000 INCA =#5D  :INITIAL INCPT
0000 TEMPO =#5F  :TEMPO VALUE
0000 * =#0001
0001 0C 08 .WOR END  :NEXT LINE POINTER
0003 0A 00 .WOR 10  :LINE NUMBER 10
0005 9C 32 .BYT #E, '020627, 0 ;SYS#0262
0006 30 32
0008 00 00 END .WOR 0  :END OF BASIC
000E ;ENTRY
000F 78 SEI  :DISABLE IRQ
0011 8D 11 D0 STA #D011
0014 8D 00 CLD  :DISABLE DECIMAL
0015 8D 0F STA #F0  :SET SID VOLUME
0017 8D 10 D4 STA #D10  :TO MAX
001A 91 FF LDA #FF  :SET USER PORT
001C 3D 03 DD STA DDR  :TO OUTPUT
001F A2 00 LDY #$00  :
0021 B5 00 STORE LDA #0000,X  :SAVE OFF ZERO
0023 9D 00 C0 STA #C000,X  :PAGE
0026 88 INX
0027 D0 F8 BNE STORE
0029 A0 00 LDX #$00  :SET UP INCPT
002B 80 52 LDY #$52  :POINT TO VIIN
002D 86 5E STX INCA+1
002F 84 5D STY INCA
0031 A9 0F LDY #$OF  :CONTROL TABLE STARTS
0033 86 F8 STX #FB
0035 84 FC STY #FC
0037 86 41 STX V1PT+1  :ZERO WAVEFORM
0039 86 46 STX V2PT+1  :HIGH BYTES
003B 86 49 STX V3PT+1
003D 86 4C STX V4PT+1
003F 86 4F STX INCPT+1
0041 A9 00 LDY #$00
0043 B1 FB LOOP LDA (#FB),Y  :GET CONTROL CODE
0045 C9 FF CMP #FF  :PLAY CONTROL?
0047 F0 25 BEQ CTRL  :YES
0049 AA TAX  :STORE AS HIGH BYTE
004A C8 INY  :FOR MUSIC
004B 98 TYA
004C 48 PHA
004D B1 FB LDA (#FB),Y  :GET LOW BYTE
004F A8 TAY
0050 20 08 00 JSR 20  :MUSIC
0053 63 PLA
0054 A8 TAY
0055 C8 NEXT INY  :REPEAT UNTIL MUSIC
0056 D0 EB BNE LOOP  :COMPLETE
0058 A2 00 EXIT LDX #$00
005A BD 00 C0 RESTORE LDA #$000,X  :COPY BACK TO
The User Port 169

LOC CODE LINE

085D 95 00 STA $0000,X ; ZERO PAGE
085F EB 00 INX
0860 DO F8 BNE RESTRE
0862 A9 00 LDA #$00 ;NO SID VOLUME
0864 8D 10 D4 STA $D418 ;RESTORE SCREEN
0867 A9 18 LDA #$1B ;RESTORE SCREEN
0869 8D 11 D0 STA $D011
086C 58 CLI ;START IRQ
086D 60 RTS ;FINISHED

086E C8 CTRL INY ;GET CONTROL NUMBER
086F B1 F8 LDA ($FB),Y ;GET TEMPO
0871 30 E5 BMC EXIT ;END OF MUSIC
0873 C9 01 CMP #$01 ;IS VALUE 1?
0875 D0 07 BNE CONTR1 ;NO
0877 C3 INY
0878 B1 F8 LDA ($FB),Y ;GET TEMPO
0879 85 5F STA TEMPO
087C D0 D7 BNE NEXT ;TRY AGAIN
087E C9 02 CONTR1 CMP #$02 ;IS CONTROL 2?
0880 D0 D6 BNE EXIT ;NO EXIT PROG
0882 C3 INY
0883 B1 F8 LDA ($FB),Y ;GET HI BYTE FOR
0885 85 42 STA V1PT+2 ; THE WAVEFORM
0887 C3 INY ; POINTERS OF
0888 B1 F8 LDA ($FB),Y ; FOUR VOICES
0889 85 47 STA V2PT+2
088C C3 INY
088D B1 F8 LDA ($FB),Y
088F 85 4A STA V3PT+2
0891 C3 INY
0892 B1 F8 LDA ($FB),Y
0894 85 4D STA V4PT+2
0896 D0 BD BNE NEXT
0898 86 51 MUSIC STX NOTES+1 ;STORE MUSIC TABLE
089A 84 50 STY NOTES ; POINTERS
089C A0 00 MUSIC1 LDY #$00 ;SET UP TO
089E A5 5D LDA INCA ; TRANSLATE FOUR
08A1 98 E4 STA INCPT ; VOICES INTO INCREMENTS
08A2 A9 7F LDA #$7F ; SET TO READ CONTROL
08A4 8D 00 DC STA $DC00 ; KEY
08A7 AD 01 DC LDA #$DC01 ; GET ANY KEYS
08AA C9 7F CMP #$7F ;STOP KEY?
08AC D0 06 BNE MUSIC9 ;NO
08AE 68 PLA ;CLEAN UP STACK
08AF 68 PLA
08B0 68 PLA
08B1 4C 58 08 JMP EXIT ;EXIT ROUTINE
08B4 81 50 MUSIC9 LDA (NOTES),Y ;GET DURATION
08B6 F0 3E BEQ ENDSNG ;IF ZERO EXIT PHRASE
08B8 C9 01 CMP #$01 ;IF 1 GET NEXT SEGMENT
08BA F0 2B BEQ NXTSEG ; OF PHRASE
08BC 85 5A STA DUR ;IS DURATION
08BE E6 50 MUSIC2 INC NOTES ;INCREMENT MUSIC
08C0 D0 02 BNE MUSIC3 ; POINTER
08C2 E6 51 INC NOTES+1
08C4 B1 50 MUSIC3 LDA (NOTES),Y ;READ IN FOUR
08C6 AA TAX ; VOICES AND STORE
08C7 BD 01 0A LDA FRQTAB,X ; IN VOICE INCREMENT
08CA 91 4E STA (INCPT),Y ; LOCATIONS
08CC E6 4E INC INCPT
08CE BD 00 0A LDA FRQTAB-1,X
08D1 91 4E STA (INCPT),Y
08D3 E6 50 INC NOTES
08D5 D0 02 BNE MUSIC4
The Commodore 64 Kernel and Hardware Revealed

LOC CODE LINE

08D7 E6 51 INC NOTES+1
08D9 E6 4E MUSIC INC INCPT ;REPEAT FOR
08DB A5 4E LDA INCPT ; OTHER VOICES
08DD C9 5A CMP #V4IN+2
08DF D0 E3 BNE MUSIC3
08E1 20 F7 08 JSR PLAY ;PLAY THE NOTES
08E4 4C 9C 08 JMP MUSIC1 ;DO NEXT LINE
08E7 ;
08E7 C8 NXTSEG INY ;GET POINTER TO
08E8 B1 50 LDA <NOTES>,Y ;NEW SEGMENT OF
08EA 49 PHA ;MUSIC
08EB C8 INY
08EC B1 50 LDA <NOTES>,Y
08EE 85 51 STA NOTES+1
08F0 68 PLA
08F1 85 50 STA NOTES
08F3 4C 9C 08 JMP MUSIC1
08F5 ;
08F6 60 ENDSNG RTS ;RETURN TO CONTROL LOOP
08F7 ;
08F7 A0 00 PLAY LDY ##00 ;PLAY THE NOTES
08F9 A6 5F LDX TEMPO
08FB 18 PLAY1 CLC
08FC B1 41 LDA <V1PT+1>,Y ;SUM WAVEFORMS OF
08FE 71 46 ADC <V2PT+1>,Y ;FOUR VOICES FOR
0900 71 49 ADC <V3PT+1>,Y ;OUTPUT
0902 71 4C ADC <V4PT+1>,Y
0904 8D 01 DD STA USRPRT ;OUTPUT VALUE
0907 A5 4B LDA V1PT ;ADD INCREMENTS
0909 65 52 ADC V1IN ;TO THE FOUR WAVE-
090B 85 4B STA V1PT ;FORM TABLE POINTERS
090D A5 41 LDA V1PT+1 ;VOICE 1
090F 85 53 ADC V1IN+1
0911 85 4B STA V1PT+1
0913 A5 45 LDA V2PT ;2
0915 65 54 ADC V2IN
0917 65 4B STA V2PT
0919 A5 46 LDA V2PT+1
091B 65 55 ADC V2IN+1
091D 85 4E STA V2PT+1
091F A5 48 LDA V3PT ;3
0921 65 56 ADC V3IN
0923 85 4B STA V3PT
0925 A5 49 LDA V3PT+1
0927 65 57 ADC V3IN+1
0929 85 49 STA V3PT+1
092B A5 4B LDA V4PT ;4
092D 65 58 ADC V4IN
092F 85 4B STA V4PT
0931 A5 4C LDA V4PT+1
0933 65 59 ADC V4IN+1
0935 85 4C STA V4PT+1
0937 CA 4C DEX
0939 D0 08 BNE TIMWAS ;WASTE TIME
093A C6 5A DEC DUR ;DECREMENT DURATION
093C F0 0C BEQ ENDNOT ;IF DUR=0 THEN DO NEXT LINE
093E A6 5F LDX TEMPO
0940 D0 89 BNE PLAY1
0942 D0 08 TIMWAS BNE *+2 ;WASTE A BIT OF
0944 D0 00 BNE *+2 ;TIME
0946 D0 00 BNE *+2
0948 D0 B1 BNE PLAY1
094A 60 ENDNOT RTS
094B ;
094B FRTRAB =#0A01
094F .END
Symbol table

SYMBOL VALUE
CONTROL 086E CONTR1 087E DDR DD03 DUR 005A
END 080C ENDHOT 094A ENSING 08F6 EXIT 0858
FRGTAB 0001 INCR 005D INCRT 004E LOOP 0043
MUSIC 0899 MUSIC1 085C MUSIC2 084E MUSIC3 08C4
MUSIC4 0899 MUSIC9 0844 NEXT 0855 NOTES 0050
NEXTSEG 08E7 PLAY 08F7 PLAY1 08F8 RESTORE 005A
STORE 0021 TEMPO 005F TIMRAS 0942 USRPRPT DD01
V1IN 0052 VPST 0048 V2IN 0054 V2PT 0045
V3IN 0056 V3PT 0048 V4IN 0058 V4PT 004B

Program 15.

Music data frequency tables

Bass frequency table
Hi-lo ØA7C to ØA93

Wraparound waveform table no 1
Wraparound waveform table no 4

Program 16.

Fig. 5.13. Sample voice waveforms for music synthesis program.
Main control table
FF specifies control code
Followed by 1 = tempo as next byte
Followed by 2 = next 4 bytes are waveform table pointers
If not FF then hi-Io of pointer to following score table

Music table
In groups of 5
Duration and 4 notes, 1 per voice
ΦΦ = return to main control loop
Φ1 = read next 2 bytes as pointer into this table lo-hi
Music table commands
The bytes in this table are stored in groups of 5 bytes; these are a duration value and a note value for each of the four voices.

If the duration byte contains a \( 00 \) then this specifies the end of the score segment and the program returns to the main control table.

If the duration byte contains a \( 01 \) then this specifies that the next two bytes contain a pointer to another section of the music table. This address is stored in lo, hi form.

5.12 Analog to digital conversion

The circuit used to convert an analog signal to a digital value is very simple. It involves the use of a voltage comparator IC and a digital to analog converter. The comparator has two inputs; one is the voltage to be measured and the other is a variable reference voltage. The comparator output will go high when the reference voltage is equal to or greater than the voltage being measured. If the reference voltage is generated by a D to A converter then it is a fairly simple matter to vary the converter output until it matches the input voltage. This point is detected by a change in the comparator output. Fig. 5.12 shows such a circuit.

Analog to digital conversion using the circuit in Fig. 5.12 relies heavily on software to find the correct D to A output value. This could be done simply by ramping up the output voltage from zero (using a simple increment loop) until the desired voltage is reached. This, however, would be very slow and could take up to 255 steps to find the match. A quicker technique, known as successive approximation, requires just eight loops. The successive approximation technique starts by setting the most significant bit (bit 8) to 1 and all other bits to zero. It then tests to see if the voltage resulting from this value is greater or smaller than the voltage to be measured. If it is larger then the msb is left set and if smaller then the msb is cleared. The routine then sets bit 7 to 1 leaving bit 8 in the state defined in the previous loop and all less significant bits set to zero. The same test is then performed to discover whether the resulting voltage value is
10 REM
20 REM PROGRAM TO INPUT 10K VALUES
30 REM FROM AN A TO D CONVERTER AND THEN
40 REM DISPLAY 320 VALUES AT A TIME TO
50 REM A GRAPHICS SCREEN
60 REM
70 REM PROTECT MEMORY
80 REM
90 POKE52, 120:POKE54, 120:POKE56, 120:CLR
100 REM
110 REM READ THE VALUES
120 REM
130 POKE 49263, 1: SY$49248:REM 49263 IS SAMPLING VALUE
140 REM
150 REM GO INTO GRAPHICS MODE
160 REM
170 SY$49627
180 REM
190 REM LOOP TO DISPLAY 32 SCREENS FULL
200 REM
210 POKE 49197, 1: FORI=0TO31:REM 49197 IS SAMPLING VALUE
220 SYS49152, I#320+30720
230 GETA$:IFA$="":THEN230
240 SYS49667:NEXT:SYS49790

Program 18.

greater or less than the input voltage to be measured. Depending on the result
bit 7 is left set or cleared. This procedure is then repeated for all the other bit
positions in the byte, with the result that only eight operations need be
performed to obtain the required value. A successive approximation technique
is shown in the first part of Program 13.

Program 18 is an example of one of the many applications to which an analog
to digital conversion circuit can be applied. This program performs the function
of a simple storage oscilloscope. This storage oscilloscope program is very short
and written in Basic. It does, however, require that the machine code program in
Program 13 is already loaded into memory. The program samples 10240 values
with a maximum sampling rate of approximately 2500 samples per second. This
sampling rate can be varied to less than this by changing the contents of location
49263. The input waveform is then displayed in high resolution as 32 screens of
information.

5.13 Expansion port

The expansion port is a 44 pin edge connector on the rear of the CBM 64. It
gives access to most of the Commodore 64's internal signals. The port is
designed to take two main types of device. The first are simple memory mapped
devices such as ROMs or I/O chips like a 6526 (if you can get one) or a 6522. The
second type of device is more interesting. These are less passive in that they can
read or write direct to memory without going through the processor. The most
common of these devices is the Commodore Z80 card. Using the DMA (direct
memory access) line totally disables the 6510 processor while the card is active.

5.13.1 Pin descriptions

Expansion port
44 pin double sided .1 edge connector socket. Labelled 1–22 (top) and A–Z
(G,I,O and Q skipped)
Power connection (power out to boards)

Pins
1,22,A,Z  Ground 0 V
2,3        +5 volts

Timing signals

Pins   I/O
6     Out  Dot clock 8 MHz approx (varies with TV standard (PAL NTSC ..)
φ2    Out  φ2 Phase two clock

Bus control signals

Pins   I/O
12    Out  BA system buses available from VIC chip
13    Input DMA Direct memory access (gives expansion card control of system buses)
5     Input  R/W  Read/write

Interrupts

Pins   I/O
4     Input  IRQ  Interrupt request
D     Input  NMI  Non maskable interrupt

Memory mapping

Pins   I/O
7     Out  I/O1  Address decoded $DE00-$DEFF
10    Out  I/O2  Address decoded $DF00-$DFFFF
11    Out  ROML  Addr decoded $8000-$A000
1B    Out  ROMH  Addr decoded $E000-$FFFF
8     Input  GAME Expansion ROM at $A000-$CFFFF (no Basic ROM)
9     Input  EXROM Exp ROM at $8000

Reset line

Pins   I/O
C     Both  RES  Reset everything

System buses

Pins   I/O
14-21  Both  Data bus consisting of eight unbuffered lines
         with a maximum load of 1 TTL device.
         Line 14 is D7 and line 21 is D0.

F-Y    Both  Address lines; these sixteen lines are unbuffered
         and have a maximum load of 1 TTL device.
         Line F is A15 and line Y is A0.

5.13.2 ROM cartridge

The expansion port is set up to make ROM cartridges a simple direct connection. An expansion ROM for address $80000 using a 2764 (8K by 8) is not too hard if you can obtain or make a board to fit the expansion port edge connector. Just connect the 13 address lines and 8 data lines. Then connect chip
select to LROM and connect the 64's EXROM to ground. The 7464 pins Vpp, Vcc and PGM go to 5 V and Vss goes to ground.

5.13.3 I/O chips on the expansion port
Wiring up a 6526 or 6522 is similar but clock, interrupt and reset also have to be implemented. It is important to connect this type of chip to I/O1 or I/O2 and not to LROM or HROM.

Figure 5.14 shows 2764 and 6522 pin outs and appropriate expansion port connections.

**MEMORY EXPANSION**

<table>
<thead>
<tr>
<th>PIN #</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>2</td>
<td>+5V</td>
</tr>
<tr>
<td>3</td>
<td>+5V</td>
</tr>
<tr>
<td>4</td>
<td>IRQ</td>
</tr>
<tr>
<td>5</td>
<td>R/W</td>
</tr>
<tr>
<td>6</td>
<td>DOT CLOCK</td>
</tr>
<tr>
<td>7</td>
<td>I/O1</td>
</tr>
<tr>
<td>8</td>
<td>GAME</td>
</tr>
<tr>
<td>9</td>
<td>EXROM</td>
</tr>
<tr>
<td>10</td>
<td>I/O2</td>
</tr>
<tr>
<td>11</td>
<td>ROML</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PIN #</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>BA</td>
</tr>
<tr>
<td>13</td>
<td>DMA</td>
</tr>
<tr>
<td>14</td>
<td>D7</td>
</tr>
<tr>
<td>15</td>
<td>D6</td>
</tr>
<tr>
<td>16</td>
<td>D5</td>
</tr>
<tr>
<td>17</td>
<td>D4</td>
</tr>
<tr>
<td>18</td>
<td>D3</td>
</tr>
<tr>
<td>19</td>
<td>D2</td>
</tr>
<tr>
<td>20</td>
<td>D1</td>
</tr>
<tr>
<td>21</td>
<td>D0</td>
</tr>
<tr>
<td>22</td>
<td>GND</td>
</tr>
</tbody>
</table>

![Memory Expansion Pinout](image)

**Fig. 5.14.** The allocation and function of pins on the memory expansion connector.
Connection of a 2764 EPROM to expansion port lines.

Connection of 6522 via I/O expansion connections.

*Fig. 5.14. (contd.)*
Chapter Six

Interrupts and Their Use

Interrupts are the signals used by peripheral devices, such as the CIA chips, to signal to the processor that they require servicing. This IRQ signal will then cause the processor to halt its current operation temporarily in order to service the interrupt generating device. Having completed this servicing the processor returns to the interrupted program.

6.1 Interrupt requests (IRQ)

The major implementation of IRQs in the Commodore 64's operating system is to scan and receive key presses from the keyboard. This IRQ runs on Timer A of CIA #1. The timer value is set up so that the keyboard is scanned every \( \frac{1}{60} \) of a second. IRQ interrupts can be disabled by setting bit 2 of the processor status register or by the use of the command SEI. To re-enable IRQ, reset bit 2 or use the CLI command. The SEI command is used by the disk operating system to prevent timing errors when accessing the disk.

The only other standard use of IRQs in the operating system of the Commodore 64 is in the tape I/O routines. Rather than just disabling IRQs, the tape system uses IRQs for reading from or writing to the tape. The tape system uses both Timer A and Timer B on CIA #1 for reading and writing. For more information on the tape routines, see Chapter 4.

6.2. Interrupt generating devices

6.2.1 The CIA chips

CIA#1 Register 14 ($DC0D)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Enable/disable (write), occurred (read)</td>
</tr>
<tr>
<td>6</td>
<td>Not used</td>
</tr>
<tr>
<td>5</td>
<td>Not used</td>
</tr>
<tr>
<td>4</td>
<td>FLAG 1 line (cassette read)</td>
</tr>
<tr>
<td>3</td>
<td>Serial data register</td>
</tr>
<tr>
<td>2</td>
<td>TOD clock alarm</td>
</tr>
<tr>
<td>1</td>
<td>Timer B</td>
</tr>
<tr>
<td>0</td>
<td>Timer A</td>
</tr>
</tbody>
</table>
When reading bit 7 is used to determine whether an enabled IRQ on this chip occurred (if more than one device is connected to the IRQ line) i.e. if this bit was not set when the IRQ routine was caused, it must have been either the VIC chip or the expansion port. If bit 7 is set, bits \( B-4 \) will tell what caused the IRQ. It should be noted that when using IRQs, it is advisable to keep a separate record of the IRQs that are enabled, since their respective bits may be set but not necessarily enabled.

When writing, bit 7 is used to tell the CIA whether the lower bits are for disabling or enabling. If bit 7 is set, any other bits set are to enable an IRQ. If bit 7 is not set, any other bits set are to disable an IRQ.

1. **Cassette read FLAG 1 line**
   This line is used by the cassette read routines and creates an IRQ when it is enabled. The tape flags an IRQ on this line when the pulse on the tape goes from high to low. An example of the use of this IRQ is shown in Chapter 4 (fast tape operation).

2. **Serial data register (SDR)**
   The SDR is a serial input/output device of the 6526 CIA chip. When IRQ is enabled on this register, the IRQ will be caused either when the full byte has been read in (input) or when it has been sent out (output). When the IRQ occurs, either a new value to send must be put into the SDR or the byte contained in the SDR will be read and the SDR left to input the next byte. The SDR uses 2 lines on the user port. These lines are SP1 and CNT1, which together are used to send/receive data. When sending, each bit is set on SP and the CNT line is used to clock the bit using Timer A. An example use of the SDR can be found in Chapter 5.

3. **TOD clock alarm**
   When the TOD clock alarm IRQ has been enabled (after setting TOD and the alarm) an IRQ occurs when the value in TOD becomes equal to the value set in the alarm. An example of how to use the TOD clock can be found in Chapter 5.

4. **Timer B**
   Timer B can run in three different modes; as a straight timer, a count down on pulses from the CNT line of the user port, and a count down on Timer A running out. These three methods are outlined in Chapter 5. An IRQ will occur on Timer B in any of the three modes of operation when the value in Timer B clocks past zero.

5. **Timer A**
   Timer A has only one mode of operation; as a straight timer. An IRQ on Timer A will occur when the value in Timer A clocks past zero. Note that with Timers A and B, the timer always decreases until it clocks past zero. Therefore, to time something, the timer should be set to the period and when it runs out the time is up. With CIA IRQs, the IRQ is cleared by reading register 14.

6.2.2 *The VIC chip*
   The VIC chip is also connected to the IRQ line and VIC chip IRQs are controlled by registers 25 and 26 on the VIC chip.
VIC register 25 ($D019)
(Interrupt flag register)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Set on any enabled VIC IRQ occurring</td>
</tr>
<tr>
<td>6-4</td>
<td>Not used</td>
</tr>
<tr>
<td>3</td>
<td>Light pen (I=occurred)</td>
</tr>
<tr>
<td>2</td>
<td>Sprite to sprite collision (I=occurred)</td>
</tr>
<tr>
<td>1</td>
<td>Sprite to background collision (I=occurred)</td>
</tr>
<tr>
<td>0</td>
<td>Raster compare (I=occurred)</td>
</tr>
</tbody>
</table>

VIC register 26 ($D01A)
(Interrupt enable mask)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-4</td>
<td>Not used</td>
</tr>
<tr>
<td>3</td>
<td>Light pen (I=enabled)</td>
</tr>
<tr>
<td>2</td>
<td>Sprite to sprite collision (I=enabled)</td>
</tr>
<tr>
<td>1</td>
<td>Sprite to background collision (I=enabled)</td>
</tr>
<tr>
<td>0</td>
<td>Raster compare (I=enabled)</td>
</tr>
</tbody>
</table>

To enable IRQ, register 26 should be read and the bit to enable set and then written back to register 26. When the IRQ occurs, reading register 25 will tell you which VIC IRQ has occurred. To clear the IRQ, the corresponding bit to clear is written to register 25.

1. **Light pen**
The light pen IRQ occurs when the raster scan reaches the position of the light pen and the light pen values can then be read from registers 19 and 20.

2. **Sprite to sprite collision**
Sprite to sprite collision IRQ occurs when any bit in the sprite to sprite collision register ($D01E$) is set.

3. **Sprite to background collision**
Sprite to background collision IRQ occurs when any bit in the sprite to background collision register ($D01F$) is set.

4. **Raster compare**
Raster compare IRQ occurs when the raster position being displayed becomes equal to the compare value written to registers 17 ($D011$ high bit) and 18 ($D012$).

6.2.3 *The expansion port*
IRQ can be caused by any I/O device connected to the Commodore 64 via the expansion port. There are two 'spare' areas for such I/O devices; they can either be addressed at $DE00$ or $DF00$. See Chapter 5 for an example of adding a 6522 VIA chip to the Commodore 64 via the expansion port.
6.3 Non maskable interrupts (NMI)

NMIs are so named because they cannot be disabled by the SEI command. Normally the NMI routine is not called regularly like the IRQ routine. This is because NMI is only caused by 2 devices:

a) RS232 (user port FLAG sent low)  
b) RESTORE key

There are five other ways of causing an NMI on the 64 that are not implemented in the software. These are Timers A and B, internal shift register, expansion port, and time of day clock on CIA#2. All NMIs except the RESTORE key and the expansion port are controlled by register 14 (SDDØD) on CIA#2. This register is used as a dual purpose write (enable/disable NMI) and read (to determine the source of NMI).

CIA#2 Register 14 (SDDØD)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Enable/disable (write), occurred (read)</td>
</tr>
<tr>
<td>6</td>
<td>Not used</td>
</tr>
<tr>
<td>5</td>
<td>Not used</td>
</tr>
<tr>
<td>4</td>
<td>User port FLAG line RS232 data received</td>
</tr>
<tr>
<td>3</td>
<td>Shift register</td>
</tr>
<tr>
<td>2</td>
<td>TOD clock alarm</td>
</tr>
<tr>
<td>1</td>
<td>Timer B</td>
</tr>
<tr>
<td>Ø</td>
<td>Timer A</td>
</tr>
</tbody>
</table>

When reading bit 7 is used to determine whether an enabled NMI on this chip has occurred (if more than one CIA chip is connected to the NMI line) i.e. if this bit was not set when the NMI routine was caused, the NMI must have been either the RESTORE key or expansion port. If bit 7 is set, bits Ø-4 will tell what caused the NMI. It should be noted that when using NMIs, it is advisable to keep a separate record of the NMIs that are enabled as their respective bits could be set but not enabled.

When writing, bit 7 is used to tell the CIA whether the lower bits are for disabling or enabling. If bit 7 is set, any other bits set are to enable an NMI. If bit 7 is not set, any other bits set are to disable an NMI.

6.4 Devices that cause NMI

1 User port FLAG line

This line is the one used by the RS232 routines and causes an NMI when it is enabled. The method of flagging an NMI on this line is to set the line to +5 V and then to 0 V. This method is outlined in Program 19 which uses 1Ø lines on the user port to transfer a block of memory from one CBM 64 to another (8 data lines and 2 lines to flag the NMI on the other 64). When initialised, the NMI
Interrupts and Their Use 187

C000  **#c000**
C000  A91F  LDA  #c000
C002  3D1903  STA  #c318  !SET NMI TO POINT
C005  A9C0  LDA  #c000
C007  8D1903  STA  #c0319
C00A  A9AE  LDA  #CSAVWED  !TO THE TRANSFER
C00C  3D3203  STA  #c332
C00F  A9C0  LDA  #c000
C011  8D3303  STA  #c033
C014  A990  LDA  #c90  !RECEIVE ROUTINE
C016  8D000D  STA  #c000
C019  A904  LDA  #c04  !SEND ROUTINE
C01B  8D70C0  STA  FLAG
C01E  80  RTS
C01F  !ROUTINE TO RECEIVE A FILE OVER
C01F  !THE USER PORT
C01F  !
C01F  48  NMI  PHA  !PUSH OFF REGISTERS
C020  9A  TXA
C021  49  PHA
C022  98  TYA
C023  48  PHA
C024  A910  LDA  #c10  !WAS NMI CAUSED BY
C026  2CD0DD  BIT  #c00  !THE USER PORT?
C029  D010  BNE  LOADIT  !YES
C02B  20BCFE  JSR  #$FBC  !NO. UPDATE CLOCK
C02E  20E1FF  JSR  #$F1E  !TEST STOP KEY
C031  D037  BNE  EXIT  !NOT DOWN
C033  A97F  LDA  #c7f  !DISABLE USER
C035  8D000D  STA  #c000  !PORT NMI
C038  4C66FE  JMP  #$E66  !DO NORMAL STOP/RESTORE
C03B  !
C03B  A900  LOADIT  LDA  #c00  !SET PORT TO INPUT
C03D  8D00DD  STA  #c000
C040  A0D1DD  LDA  #c1dd  !GET INPUT BYTE
C043  EE20D0  INC  #c000
C046  AE70C0  LDX  FLAG  !SHOW IT IS WORKING
C049  F009  BEQ  STRFLE  !READING FILE OR
C04B  95FA  STA  #$FAX  !LOAD ADDRESS?
C04D  CA  DEX  !DISABLE FLAG
C04E  8E70C0  STX  FLAG  !STORE IT
C051  4C6AC0  JMP  EXIT
C054  !
C054  A000  STRFLE  LDY  #$00  !NOW READING FILE
C056  91FB  STA  (#cFB),y  !STORE THE BYTE
C058  209FC0  JSR  BUMP2  !INCREMENT AND TEST END
C05B  900D  BCC  EXIT  !NOT YET
C05D  A5FD  LDA  #$FD  !SET PROGRAM END
C05F  852D  STA  #$2D  !POINTERS TO END
C061  A5FE  LDA  #$FE  !OF READ FILE
C063  852E  STA  #$2E
C065  A504  LDA  #$04
C067  8D70C0  STA  FLAG
C069  !
C06A  68  EXIT  PLA  !RESTORE REGISTERS
C06B  88  TAY  !AND EXIT NMI
C06C  68  PLA
C06D  A6  TAX
C06E  68  PLA
C06F  40  RTI
C070  !
C070  04  FLAG  BYT 4
C071  !ROUTINE TO SEND A FILE OVER THE
C071  !USER PORT
C071  !
C071  AD70C0  SAVER  LDA  FLAG  !IF RECEIVING,
C074  C004  CMP  #$04  !DON'T SEND
C076  D0F2  BNE  SAVER
C078  A204  LDX  #$04  !POINT TO SAVE
The Commodore 64 Kernal and Hardware Revealed

**Program 19.**

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Instruction</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>C07A</td>
<td>B5AB</td>
<td>LOOP</td>
<td>LDA $AB,X</td>
</tr>
<tr>
<td>C07C</td>
<td>20C2C0</td>
<td>JSR SBYTE</td>
<td>SEND THE BYTE</td>
</tr>
<tr>
<td>C07E</td>
<td>0A</td>
<td>DEX</td>
<td>DO NEXT?</td>
</tr>
<tr>
<td>C080</td>
<td>D0F8</td>
<td>BNE LOOP</td>
<td>YES</td>
</tr>
<tr>
<td>C082</td>
<td>A000</td>
<td>LDV $#90</td>
<td></td>
</tr>
<tr>
<td>C084</td>
<td>B1AC</td>
<td>LOOP1</td>
<td>LDA ($AC),Y</td>
</tr>
<tr>
<td>C086</td>
<td>20C2C0</td>
<td>JSR SBYTE</td>
<td>SEND IT</td>
</tr>
<tr>
<td>C089</td>
<td>2090C0</td>
<td>JSR BUMP</td>
<td>INCUMENT AND TEST END</td>
</tr>
<tr>
<td>C08C</td>
<td>90F6</td>
<td>BCC LOOP1</td>
<td>NOT YET</td>
</tr>
<tr>
<td>C08E</td>
<td>18</td>
<td>CLC</td>
<td>SAVED OK</td>
</tr>
<tr>
<td>C090</td>
<td>60</td>
<td>RTS</td>
<td>DONE</td>
</tr>
<tr>
<td>C0F8</td>
<td>E6AC</td>
<td>BUMP</td>
<td></td>
</tr>
<tr>
<td>C0FA</td>
<td>6002</td>
<td>BNE BUMP1</td>
<td></td>
</tr>
<tr>
<td>C0FC</td>
<td>E6AC</td>
<td>INC $AC</td>
<td>INCMENT LO BYTE</td>
</tr>
<tr>
<td>C0E8</td>
<td>A5AC</td>
<td>BUMP1</td>
<td>ADDRESS TO END</td>
</tr>
<tr>
<td>C0E9</td>
<td>C5AE</td>
<td>CMP $AE</td>
<td></td>
</tr>
<tr>
<td>C0EA</td>
<td>A5AD</td>
<td>LDA $AD</td>
<td></td>
</tr>
<tr>
<td>C0E9</td>
<td>C5AC</td>
<td>CMP $AE</td>
<td></td>
</tr>
<tr>
<td>C0EB</td>
<td>E5AF</td>
<td>SBC $AF</td>
<td></td>
</tr>
<tr>
<td>C0EC</td>
<td>60</td>
<td>RTS</td>
<td></td>
</tr>
<tr>
<td>C0F1</td>
<td>D002</td>
<td>BNE BUMP3</td>
<td>INCENT HI BYTE</td>
</tr>
<tr>
<td>C0F3</td>
<td>E6FC</td>
<td>INC $FC</td>
<td>COMPARE SAVE</td>
</tr>
<tr>
<td>C0F5</td>
<td>A5FC</td>
<td>LDA $FC</td>
<td></td>
</tr>
<tr>
<td>C0F7</td>
<td>C5FD</td>
<td>CMP $FD</td>
<td>ADDRRESS WITH END</td>
</tr>
<tr>
<td>C0F9</td>
<td>A5FC</td>
<td>LDA $FC</td>
<td></td>
</tr>
<tr>
<td>C0FB</td>
<td>E5FE</td>
<td>SBC $FE</td>
<td></td>
</tr>
<tr>
<td>C0FC</td>
<td>60</td>
<td>RTS</td>
<td></td>
</tr>
<tr>
<td>C0FE</td>
<td>A5BA</td>
<td>SAVEWED</td>
<td>WEDGE INTO SAVE VECTOR</td>
</tr>
<tr>
<td>C0FA</td>
<td>907</td>
<td>CMP $#07</td>
<td>TEST DEVICE #</td>
</tr>
<tr>
<td>C0FB</td>
<td>F003</td>
<td>BEQ SAVE1</td>
<td>YES; SEND OVER PORT</td>
</tr>
<tr>
<td>C0FC</td>
<td>4CEDF5</td>
<td>JMP $5ED</td>
<td>NO, NORMAL SAVE</td>
</tr>
<tr>
<td>C0FD</td>
<td>A5C1</td>
<td>SAVE1</td>
<td>SET SAVE START</td>
</tr>
<tr>
<td>C0FE</td>
<td>85AC</td>
<td>STA $AC</td>
<td>ADDRESS FOR USER</td>
</tr>
<tr>
<td>C0FF</td>
<td>A5C2</td>
<td>LDA $AC</td>
<td>PORT SAVE</td>
</tr>
<tr>
<td>C100</td>
<td>85AD</td>
<td>LDA $AC</td>
<td></td>
</tr>
<tr>
<td>C102</td>
<td>4C71C0</td>
<td>JMP SAVVER</td>
<td>SEND FILE</td>
</tr>
<tr>
<td>C104</td>
<td>48</td>
<td>SBYTE</td>
<td>ROUTINE TO SEND 1 BYTE ACROSS</td>
</tr>
<tr>
<td>C106</td>
<td>9FF</td>
<td>PHA</td>
<td>THE USER PORT</td>
</tr>
<tr>
<td>C108</td>
<td>8D03DD</td>
<td>STA $D03</td>
<td>SAVE OFF BYTE</td>
</tr>
<tr>
<td>C10A</td>
<td>68</td>
<td>PLA</td>
<td>SET PORT TO OUTPUT</td>
</tr>
<tr>
<td>C10C</td>
<td>8D01DD</td>
<td>STA $D01</td>
<td>SET PORT TO OUTPUT</td>
</tr>
<tr>
<td>C10E</td>
<td>AD02DD</td>
<td>LDA $D02</td>
<td>SET LINE PA2</td>
</tr>
<tr>
<td>C110</td>
<td>0904</td>
<td>ORA $#04</td>
<td>TO OUTPUT</td>
</tr>
<tr>
<td>C112</td>
<td>8D02DD</td>
<td>STA $D02</td>
<td>SEND PA2  HIGH</td>
</tr>
<tr>
<td>C114</td>
<td>AD00DD</td>
<td>LDA $D00</td>
<td></td>
</tr>
<tr>
<td>C116</td>
<td>0904</td>
<td>ORA $#04</td>
<td></td>
</tr>
<tr>
<td>C118</td>
<td>8D00DD</td>
<td>STA $D00</td>
<td></td>
</tr>
<tr>
<td>C11A</td>
<td>20F3C0</td>
<td>JSR DEL</td>
<td>PAUSE</td>
</tr>
<tr>
<td>C11C</td>
<td>AD00DD</td>
<td>LDA $D00</td>
<td>SEND PA2 LOW</td>
</tr>
<tr>
<td>C11E</td>
<td>29FB</td>
<td>AND $#FB</td>
<td>NMI HAS BEEN CAUSED</td>
</tr>
<tr>
<td>C120</td>
<td>8D00DD</td>
<td>STA $D00</td>
<td>ON RECEIVING MACHINE</td>
</tr>
<tr>
<td>C122</td>
<td>20F3C0</td>
<td>JSR DEL</td>
<td>PAUSE</td>
</tr>
<tr>
<td>C124</td>
<td>A900</td>
<td>LDA $#0</td>
<td>SET USER PORT TO</td>
</tr>
<tr>
<td>C126</td>
<td>8D03DD</td>
<td>STA $D03</td>
<td>INPUT</td>
</tr>
<tr>
<td>C128</td>
<td>EE20DD</td>
<td>INC $D020</td>
<td>SHOW IT IS WORKING</td>
</tr>
<tr>
<td>C12A</td>
<td>60</td>
<td>RTS</td>
<td></td>
</tr>
<tr>
<td>C12C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C130</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C132</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C134</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C136</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C138</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C140</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C142</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C144</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C146</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C148</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C150</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C152</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C154</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C156</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C158</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C160</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C162</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C164</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C166</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C168</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C170</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C172</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C174</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C176</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C178</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C180</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C182</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C184</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C186</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C188</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Program 19.
vector is set to point to the receive routine and the SAVE vector is set to the send routine. When the user of one computer SAVEs a block of memory with device 7, the file is passed through to the other computer by setting a full byte onto the data lines and causing an NMI by setting the PA2 line hi then lo (PA2 is connected to FLAG both ways). The NMI routine then reads the byte from the port and either stores it as a load address or as part of the file.

To send a file, use SAVE"", 7. Files are automatically received.

2. Serial data register
The NMI SDR has exactly the same operation as the IRQ SDR except that instead of lines CNT1 and SP1, lines CNT2 and SP2 are used. SDR use can be seen in Chapter 5.

3. TOD clock alarm
The NMI TOD clock alarm has exactly the same operation as the IRQ TOD clock alarm. An example of how to use the TOD clock can be found in Chapter 5.

4. Timer B
The NMI Timer B has exactly the same operation as the IRQ Timer B.

5. Timer A
The NMI Timer A has the same operation as the IRQ Timer A.

6. RESTORE key
The RESTORE key on the keyboard is connected directly to the NMI line and is not a true NMI. When RESTORE is pressed, the NMI routine is called and if the STOP key is also down, NMI will cause a restart of the computer. This is done by jumping to a routine pointed to by an indirection at $A002: \text{JMP} (A002)$. If a cartridge ROM is in place (with the power-up bytes), JMP ($8002$) is used instead.

7. Expansion port
Expansion port NMI has the same operation as expansion port IRQ except that IRQ occurs if the line is low, whereas NMI occurs when the line goes low.

6.5 The kernal vectors

There are a group of vectors in page three memory that are used for indirect jumps into some of the most useful kernal routines. These have been provided so that the machine code programmer can patch into them to change the operation of the computer. Each vector is a two byte low-high vector to the main machine code kernal routine and by changing its value, you may point it to your own routine.

The vectors are as follows:
<table>
<thead>
<tr>
<th>Address</th>
<th>Default</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0314</td>
<td>$EA31</td>
<td>Vector to the IRQ routine. This vector can be changed to point to your own IRQ routine for things such as screen scrolling etc.</td>
</tr>
<tr>
<td>$0316</td>
<td>$FE66</td>
<td>Vector for BRK instruction is changed by all monitors so that when a BRK is encountered, the computer jumps to the monitor.</td>
</tr>
<tr>
<td>$0318</td>
<td>$FE47</td>
<td>Vector to the NMI routine. Its major use is for the detection of the RESTORE key. Other methods are outlined in Chapter 5.</td>
</tr>
<tr>
<td>$031A</td>
<td>$F34A</td>
<td>Vector to open file routine.</td>
</tr>
<tr>
<td>$031C</td>
<td>$F291</td>
<td>Vector to close file routine.</td>
</tr>
<tr>
<td>$031E</td>
<td>$F2E0</td>
<td>Vector to set input device.</td>
</tr>
<tr>
<td>$0320</td>
<td>$F250</td>
<td>Vector to set output device.</td>
</tr>
<tr>
<td>$0322</td>
<td>$F333</td>
<td>Vector to restore I/O.</td>
</tr>
<tr>
<td>$0324</td>
<td>$F157</td>
<td>Vector to input. This routine is used in all peripheral input. It could be used for function keys etc.</td>
</tr>
<tr>
<td>$0326</td>
<td>$F1CA</td>
<td>Vector to output. This routine controls all output to the same devices as input (except keyboard).</td>
</tr>
<tr>
<td>$0328</td>
<td>$F6ED</td>
<td>Vector to test STOP routine. The most widely used patch is for disabling the STOP key.</td>
</tr>
<tr>
<td>$032A</td>
<td>$F13E</td>
<td>Vector to get. This routine is used to get a single key from the keyboard buffer. The character received is not displayed but is just returned in register .A. The get key has the same operation as input from all devices except the keyboard where input inputs a line until carriage return is pressed.</td>
</tr>
<tr>
<td>$032C</td>
<td>$F32F</td>
<td>Vector to abort I/O.</td>
</tr>
<tr>
<td>$032E</td>
<td>$FE66</td>
<td>Unused vector. This vector can be used by your own routines.</td>
</tr>
<tr>
<td>$0330</td>
<td>$F4A5</td>
<td>Vector to load routine. This vector is jumped to after the load parameters have been set up.</td>
</tr>
<tr>
<td>$0332</td>
<td>$F5ED</td>
<td>Vector to save routine. An example of a patch into this vector can be seen in Chapter 4 and in Program 19 in this chapter.</td>
</tr>
</tbody>
</table>
Index

abort serial I/O files, 59
ACPTR, 49
address bus, 8, 10
allophones, 151
analog interfacing, 155
analog music synthesis, 159
analog to digital converters, 5, 10, 155
anti piracy techniques, 123
ASCII files, 80
auto run, 121

Basic interpreter, 3
BASIN, 52
binary files, 80
BSOUT, 54

cartridge port, 10, 179
cassette buffer, 79
cassette error messages, 84
cassette hardware, 78
cassette operating system routines, 83
cassette operation, 79
clock signals, 8, 10, 13, 130, 137
close all logical files, 59
CLRCH, 59
colour clock, 13
Commodore 64 design concept, 2
computer control for the disabled, 142
data bus, 8, 10
digital to analog conversion, 156
digitising pads, 22

DMA, 10, 13
dot clock, 13
timer handler, 67
expansion port, 179, 185
find any tape header, 92
find correct file on tape, 95
FLAG, 131, 186
function key definition, 19
general function serial routines, 51
get character from current input device, 52, 53
GETIN, 52
handshaking, 131, 133
high speed data transfer, 138, 186
high speed tape operation, 110
I/O, 3, 4, 7, 15, 127, 128, 132
input byte from serial port, 49
interrupt control register, 138
IRQ, 4, 9, 13, 108, 138, 183 ff
joystick, 5, 10, 22, 127
kernal vectors, 189
keyboard, 15, 127,
keyboard buffer, 16
keyboard matrix scanning, 4, 15, 16
keyboard operation modification, 19
keyboard scan simulation program, 17
light pen, 11, 185
listen, 42
load RAM function, 86
LOAD/VERIFY, 62
microprocessor 6510, 2, 7
Index

miscellaneous cassette routines, 96
MPU signal lines, 8, 130
NMI, 9, 13, 84, 186
OPEN, 60
operating system, 3
output character, 54
parallel port expansion, 142, 181
peripheral interface lines, 131
phase 2 clock, 10, 13
PLA, 7, 9, 12
potentiometer joystick, 22
power supply, 7
print ‘saving’, 90
print tape loading messages, 88
printed circuit board, 6, 7
protect cassette from RS232 NMI, 84
RAM, 3, 10, 12, 13
read cassette, 100
read/write, 13
ready, 13
recording method, 81
relay control circuit, 141
return buffer address, 95
ROM cartridge, 180
RS232 close channel, 77
RS232 command register, 73
RS232 connections, 71
RS232 control register, 72
RS232 open channel, 75
RS232 receive from channel, 76
RS232 serial communications, 70
RS232 status register, 73
RS232 transmit to channel, 76
send byte to serial bus, 47
send secondary address after talk, 46
serial bus lines, 32
serial bus timings, 34
serial communications, 32, 70
serial data register, 137, 184
serial system routines, 42
serial system variable declare file, 38
set opened file for input, 54
set opened file for output, 56
set up time out for next dipole, 99
SID 6581, 4, 9, 10
SID signals and lines, 10
sound generation, 4, 10
stop key servicing, 90
switch joystick, 22
system control signals, 13
system logic and timing, 12
TALK, 42
tape error handler, 91
tape IRQ, 108, 123
tape security, 123
time of day (TOD) clock, 4, 9, 135, 184
timers, 4, 9, 84, 134, 184
TKSA, 46
UNLISTEN, 48
UNTALK, 47
user port, 128
user port connections, 127, 128, 129
VIC 6567 Chip, 2, 10, 11, 187
VIC signals and lines, 11
voice synthesis, 149
write cassette, 106
write memory, 97
write tape header, 93
Z80 card, 10
A knowledge of the Commodore 64 kernal software and the hardware with which it interacts is essential for programmers wishing to make full use of the machine's capabilities. The kernal software provides the interface between the user, the BASIC interpreter and the electronics - and a thorough knowledge of its functioning gives the programmer a wealth of ideas and methods for interesting programming techniques.

This book gives the programmer a unique insight into the operation of the Commodore 64 plus a wide variety of very useful hints on subjects as diverse as reconfiguring the keyboard and anti tape-copying security. The book also covers the user port and the addition of external circuitry to it.

The Authors
Nick Hampshire is a well-known author and microcomputer expert who has specialised in Commodore computer equipment. He started the first hobby microcomputer magazine, later absorbed into Practical Computing, of which he was technical editor for several years. He was the co-founder of Popular Computing Weekly and founder and managing editor of Commodore Computing International magazine. He is also the author of over a dozen books on popular computing, including the very successful and widely acclaimed PET Revealed and VIC Revealed.

Richard Franklin and Carl Graham are programmers with Zifra Software Ltd and together with Nick Hampshire have written some of the software included in this book.

Also by Nick Hampshire
THE COMMODORE 64 ROMs REVEALED
0 00 383087 X
ADVANCED COMMODORE 64 BASIC REVEALED
0 00 383088 8
ADVANCED COMMODORE 64 GRAPHICS AND SOUND
0 00 383089 6
THE COMMODORE 64 DISK DRIVE REVEALED
0 00 383091 8

COLLINS
Printed in Great Britain
£10.95 net

ISBN 0-00-383090-X