

NICK HAMPSHIRE
WITH RICHARD FRANKLIN AND CARL GRAHAM

**THE
COMMODORE**

64

**KERNAL AND
HARDWARE
REVEALED**

The Commodore 64 Kernal 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

The Commodore 64 Kernal and Hardware Revealed

Nick Hampshire

with Richard Franklin and Carl Graham



COLLINS

8 Grafton Street, London W1

Collins Professional and Technical Books
William Collins Sons & Co. Ltd
8 Grafton Street, London W1X 3LA

First published in Great Britain by
Collins Professional and Technical Books 1985

Copyright © Nick Hampshire 1985

British Library Cataloguing in Publication Data
Hampshire, Nick

The Commodore 64 kernal and hardware revealed.

I. Commodore 64 (Computer)

I. Title II. Franklin, Richard III. Graham, Carl

001.64'04 QA76.8.C64

ISBN 0-00-383090-X

Typeset by V & M Graphics Ltd, Aylesbury, Bucks
Printed and bound in Great Britain by
Mackays of Chatham, Kent

All rights reserved. No part of this publication may
be reproduced, stored in a retrieval system or transmitted,
in any form, or by any means, electronic, mechanical, photocopying,
recording or otherwise, without the prior permission of the
publishers.

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

<i>Preface</i>	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
<i>Index</i>	193

Preface

Whether you program the CBM 64 in Basic or machine code, an understanding of how the kernal 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 1 of this series, *The Commodore 64 ROMs Revealed*, which gives the entire operating system kernal 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

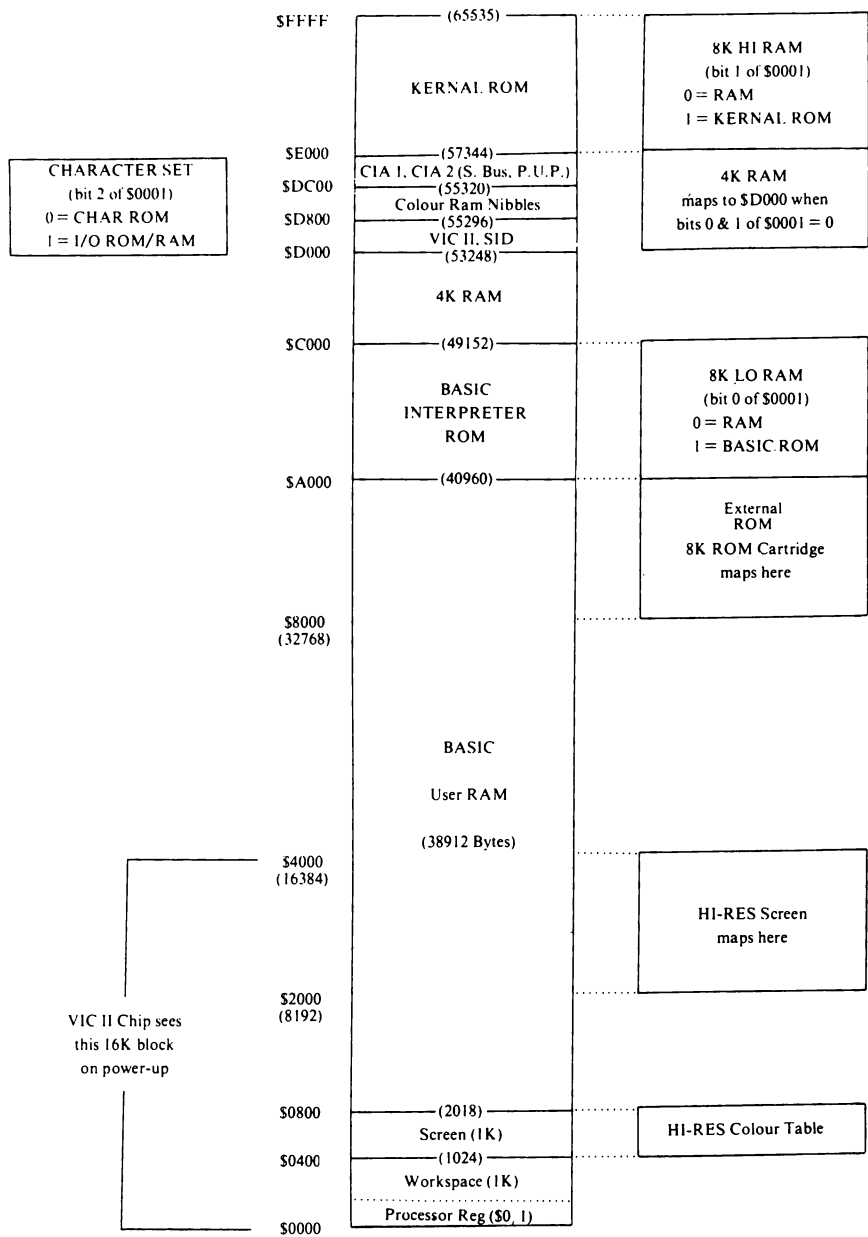


Fig. 1.1. Commodore 64 memory architecture map.

2 The Commodore 64 Kernal and Hardware Revealed

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.

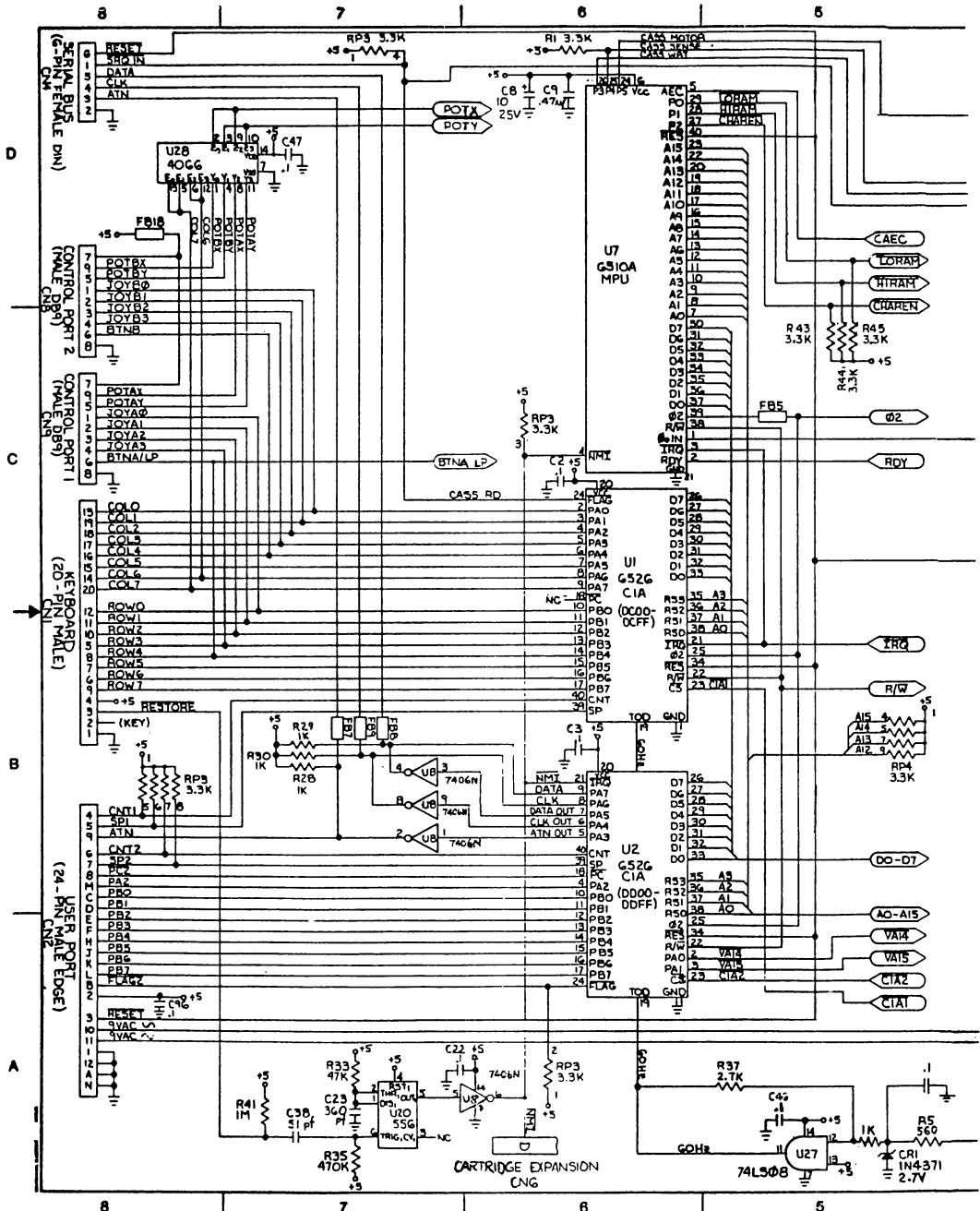


Fig. 1.2. Commodore 64 schematics (Reproduced by courtesy of Commodore Business Machines (UK) Ltd).

The schematic diagram illustrates the internal circuitry of the Commodore 64 system unit. It is organized into several functional blocks:

- Power Input Section (Bottom):** Features a 7-pin male DIN connector (CN7) for power input. It includes a power transformer (T1) with primary windings for 9VAC and 0VAC, and secondary windings for 100V, 250V, and 500V. The transformer is connected to a bridge rectifier (BR1) and a filter capacitor (C97). The output is regulated by a 7805 voltage regulator (U1) to provide a +5V supply. Other components include resistors R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, R16, R17, R18, R19, R20, R21, R22, R23, R24, R25, R26, R27, R28, R29, R30, R31, R32, R33, R34, R35, R36, R37, R38, R39, R40, R41, R42, R43, R44, R45, R46, R47, R48, R49, R50, R51, R52, R53, R54, R55, R56, R57, R58, R59, R60, R61, R62, R63, R64, R65, R66, R67, R68, R69, R70, R71, R72, R73, R74, R75, R76, R77, R78, R79, R80, R81, R82, R83, R84, R85, R86, R87, R88, R89, R90, R91, R92, R93, R94, R95, R96, R97, R98, R99, R100, R101, R102, R103, R104, R105, R106, R107, R108, R109, R110, R111, R112, R113, R114, R115, R116, R117, R118, R119, R120, R121, R122, R123, R124, R125, R126, R127, R128, R129, R130, R131, R132, R133, R134, R135, R136, R137, R138, R139, R140, R141, R142, R143, R144, R145, R146, R147, R148, R149, R150, R151, R152, R153, R154, R155, R156, R157, R158, R159, R160, R161, R162, R163, R164, R165, R166, R167, R168, R169, R170, R171, R172, R173, R174, R175, R176, R177, R178, R179, R180, R181, R182, R183, R184, R185, R186, R187, R188, R189, R190, R191, R192, R193, R194, R195, R196, R197, R198, R199, R200, R201, R202, R203, R204, R205, R206, R207, R208, R209, R210, R211, R212, R213, R214, R215, R216, R217, R218, R219, R220, R221, R222, R223, R224, R225, R226, R227, R228, R229, R230, R231, R232, R233, R234, R235, R236, R237, R238, R239, R240, R241, R242, R243, R244, R245, R246, R247, R248, R249, R250, R251, R252, R253, R254, R255, R256, R257, R258, R259, R260, R261, R262, R263, R264, R265, R266, R267, R268, R269, R270, R271, R272, R273, R274, R275, R276, R277, R278, R279, R280, R281, R282, R283, R284, R285, R286, R287, R288, R289, R290, R291, R292, R293, R294, R295, R296, R297, R298, R299, R300, R301, R302, R303, R304, R305, R306, R307, R308, R309, R310, R311, R312, R313, R314, R315, R316, R317, R318, R319, R320, R321, R322, R323, R324, R325, R326, R327, R328, R329, R330, R331, R332, R333, R334, R335, R336, R337, R338, R339, R340, R341, R342, R343, R344, R345, R346, R347, R348, R349, R350, R351, R352, R353, R354, R355, R356, R357, R358, R359, R360, R361, R362, R363, R364, R365, R366, R367, R368, R369, R370, R371, R372, R373, R374, R375, R376, R377, R378, R379, R380, R381, R382, R383, R384, R385, R386, R387, R388, R389, R390, R391, R392, R393, R394, R395, R396, R397, R398, R399, R400, R401, R402, R403, R404, R405, R406, R407, R408, R409, R410, R411, R412, R413, R414, R415, R416, R417, R418, R419, R420, R421, R422, R423, R424, R425, R426, R427, R428, R429, R430, R431, R432, R433, R434, R435, R436, R437, R438, R439, R440, R441, R442, R443, R444, R445, R446, R447, R448, R449, R450, R451, R452, R453, R454, R455, R456, R457, R458, R459, R460, R461, R462, R463, R464, R465, R466, R467, R468, R469, R470, R471, R472, R473, R474, R475, R476, R477, R478, R479, R480, R481, R482, R483, R484, R485, R486, R487, R488, R489, R490, R491, R492, R493, R494, R495, R496, R497, R498, R499, R500, R501, R502, R503, R504, R505, R506, R507, R508, R509, R510, R511, R512, R513, R514, R515, R516, R517, R518, R519, R520, R521, R522, R523, R524, R525, R526, R527, R528, R529, R530, R531, R532, R533, R534, R535, R536, R537, R538, R539, R540, R541, R542, R543, R544, R545, R546, R547, R548, R549, R550, R551, R552, R553, R554, R555, R556, R557, R558, R559, R560, R561, R562, R563, R564, R565, R566, R567, R568, R569, R570, R571, R572, R573, R574, R575, R576, R577, R578, R579, R580, R581, R582, R583, R584, R585, R586, R587, R588, R589, R590, R591, R592, R593, R594, R595, R596, R597, R598, R599, R600, R601, R602, R603, R604, R605, R606, R607, R608, R609, R610, R611, R612, R613, R614, R615, R616, R617, R618, R619, R620, R621, R622, R623, R624, R625, R626, R627, R628, R629, R630, R631, R632, R633, R634, R635, R636, R637, R638, R639, R640, R641, R642, R643, R644, R645, R646, R647, R648, R649, R650, R651, R652, R653, R654, R655, R656, R657, R658, R659, R660, R661, R662, R663, R664, R665, R666, R667, R668, R669, R670, R671, R672, R673, R674, R675, R676, R677, R678, R679, R680, R681, R682, R683, R684, R685, R686, R687, R688, R689, R690, R691, R692, R693, R694, R695, R696, R697, R698, R699, R700, R701, R702, R703, R704, R705, R706, R707, R708, R709, R710, R711, R712, R713, R714, R715, R716, R717, R718, R719, R720, R721, R722, R723, R724, R725, R726, R727, R728, R729, R730, R731, R732, R733, R734, R735, R736, R737, R738, R739, R740, R741, R742, R743, R744, R745, R746, R747, R748, R749, R750, R751, R752, R753, R754, R755, R756, R757, R758, R759, R760, R761, R762, R763, R764, R765, R766, R767, R768, R769, R770, R771, R772, R773, R774, R775, R776, R777, R778, R779, R780, R781, R782, R7

4 The Commodore 64 Kernal and Hardware Revealed

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

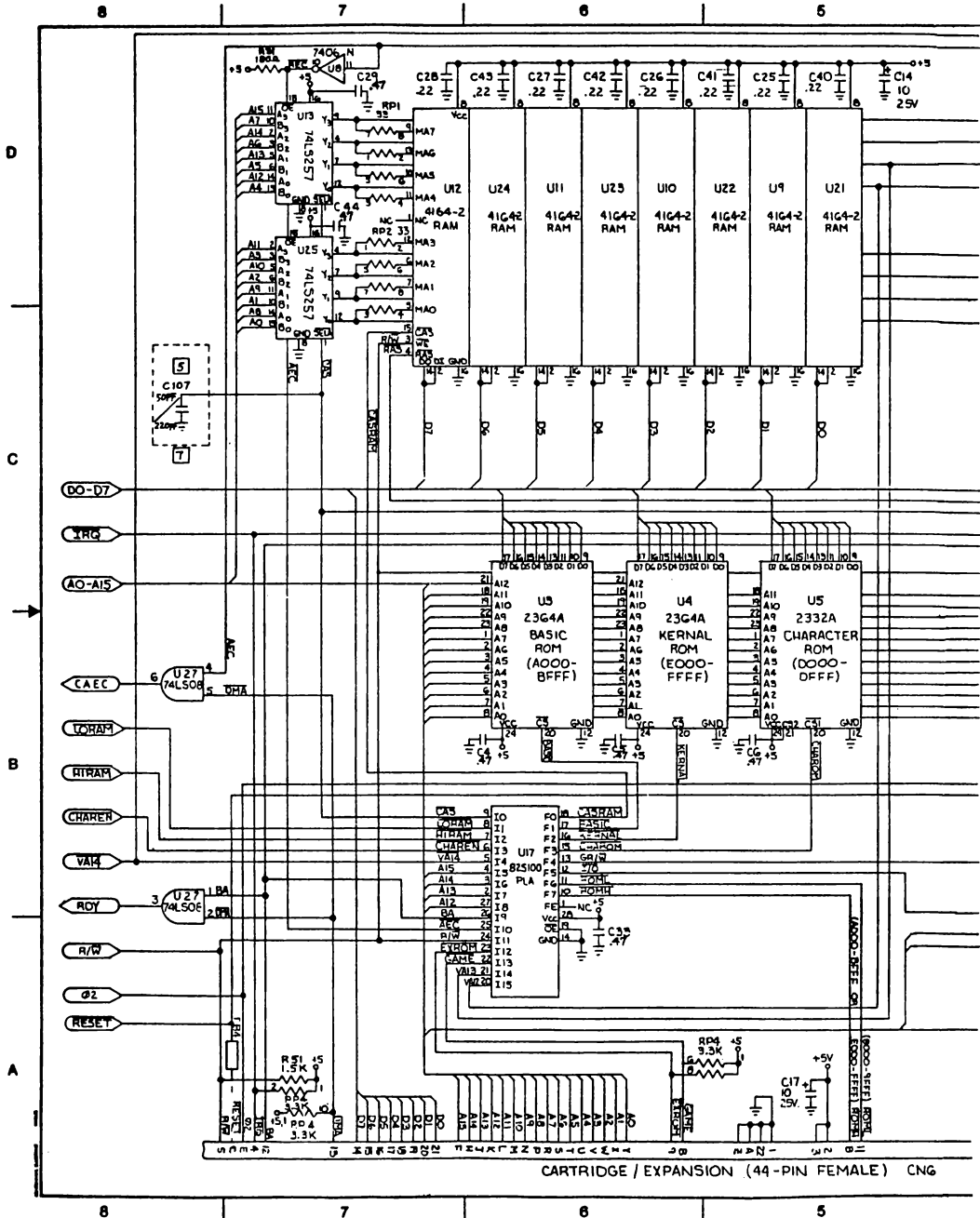


Fig. 1.2. (contd.)

Fig. 1.2. (contd.)

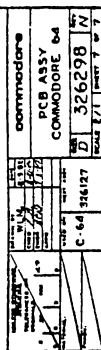


Fig. 1.3. Printed circuit board assembly for the Commodore 64 (Reproduced by courtesy of Commodore Business Machines (UK) Ltd).

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.

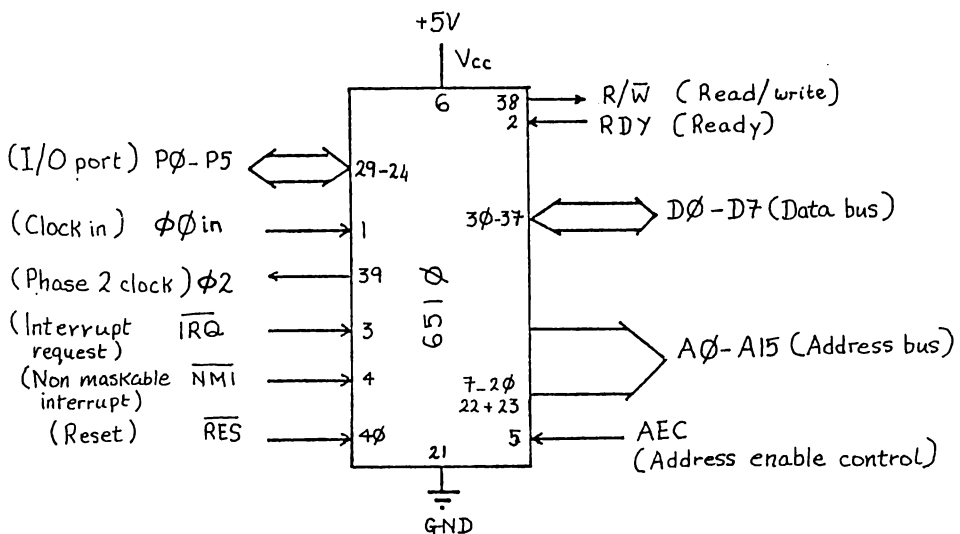


Fig. 1.4. 6510 microprocessor (MPU).

6510 Signals and Lines

Pin 1	$\phi\overline{0}$ in	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/ $\overline{W}$	Read/write. (Output: low flags for processor write.)
Pin 39	ϕ 2in	Phase 2 clock input
Pin 40	$\overline{RES}$	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

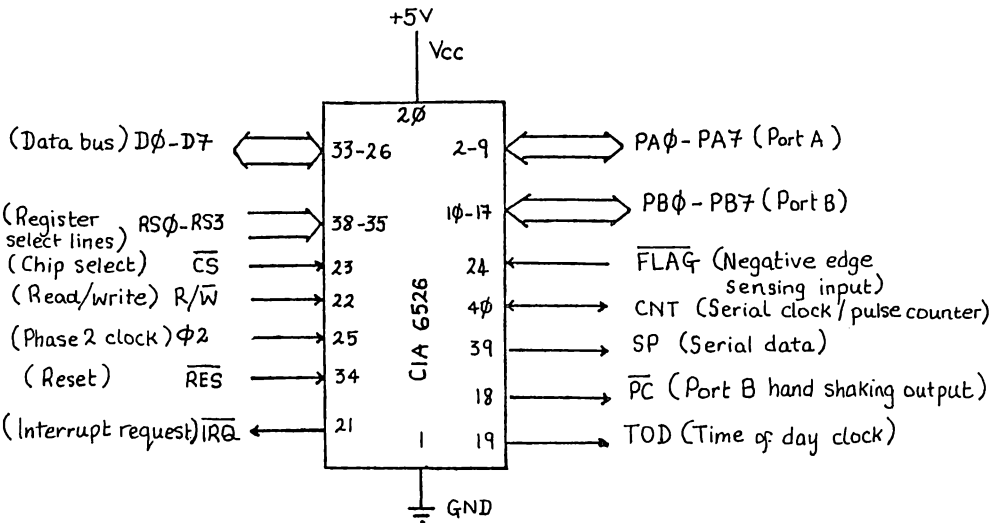


Fig. 1.5. 6526 complex interface adapter (CIA).

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

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

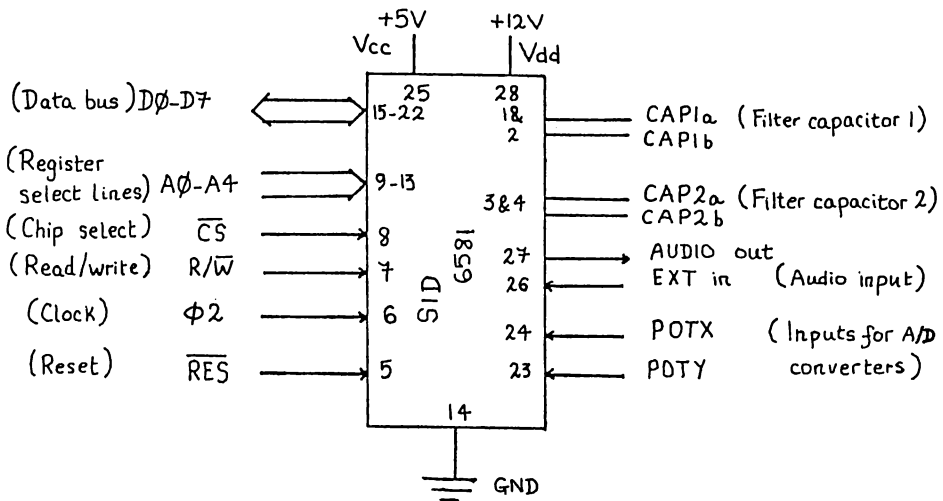


Fig. 1.6. 6581 sound interface device (SID).

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

Pins 1 & 2	CAP1a-b	Filter capacitor
Pins 3 & 4	CAP2a-b	Second filter capacitor
Pin 7	$\overline{R/\overline{W}}$	Read/write
Pin 8	$\overline{CS}$	Chip select. Decoded from address bus by PLA to \$D400
Pins 9-13	A0-A4	Register select
Pin 14	GND	Ground (0 V)
Pins 15-22	D0-D7	Data bus
Pin 23	POT Y	Analog input for A/D converter
Pin 24	POT X	Analog input for second A/D
Pin 25	Vcc	Supply voltage 5 V
Pin 26	EXT in	Audio input
Pin 27	AUDIO	Audio output
Pin 28	Vdd	Supply voltage 12 V

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 (ϕ_2 low ϕ_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 $\overline{R/\overline{W}}$ 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.

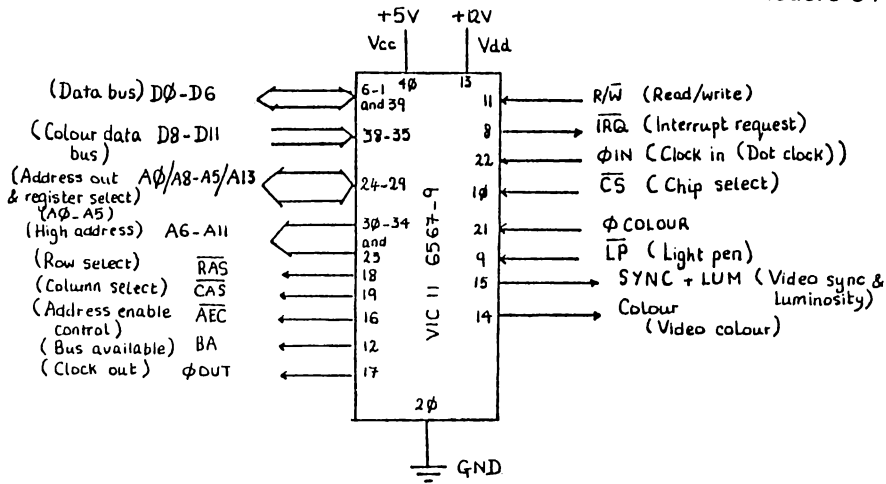


Fig. 1.7. Video interface chip (VIC II).

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	$\overline{\text{IRQ}}$	Interrupt request output
Pin 9	$\overline{\text{LP}}$	Light pen input
Pin 10	$\overline{\text{CS}}$	Chip select
Pin 11	$\text{R}/\overline{\text{W}}$	Read/write
Pin 12	$\overline{\text{BA}}$	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	ϕ	Phase one clock out
Pin 18	$\overline{\text{RAS}}$	Row address select. Dynamic RAM control signal, used for low order of multiplexed address and for refreshing
Pin 19	$\overline{\text{CAS}}$	Column address select. Dynamic RAM control signal for high order address
Pin 20	Vss	Ground (0 V)
Pin 21	ϕ_{colour}	Colour clock in 14-18 MHz
Pin 22	ϕ_{in}	Clock in 8 MHz
Pins 23 & 30-34	A11 & A6-A10	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

12 The Commodore 64 Kernal and Hardware Revealed

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!

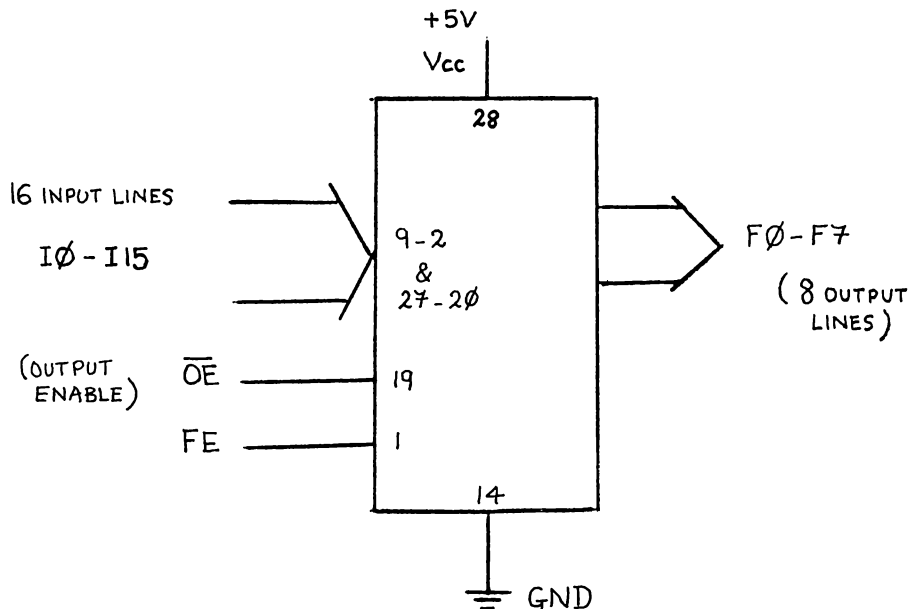


Fig. 1.8. Programmable logic array (PLA).

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

ϕ 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 $\overline{R/W}$

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.

$\overline{\text{Reset}}$ $\overline{\text{RES}}$

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 $\overline{\text{IRQ}}$

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

Non maskable interrupt $\overline{\text{NMI}}$

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 $\overline{\text{CAS}}$ $\overline{\text{RAS}}$ & $\overline{\text{CASRAM}}$

The 4164 dynamic RAM chips have their 16 bit addresses fed to in two lots of 8

14 *The Commodore 64 Kernal and Hardware Revealed*

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:

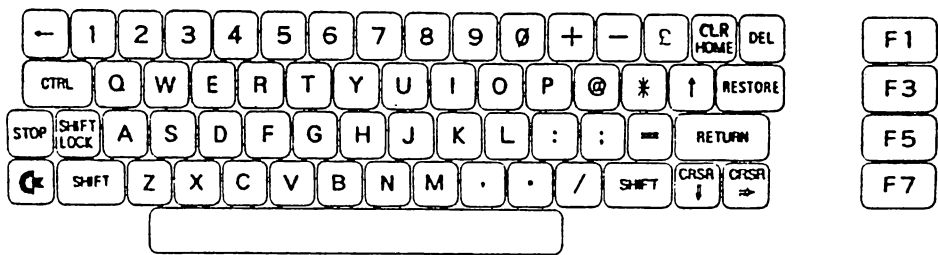


Fig. 2.1. The keyboard layout.

- 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=down, 1=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:

Row # (input)		0	1	2	Column # (output)				
					3	4	5	6	7
0	DEL	3	5	7	9	+	£		1
1	RET	W	R	Y	I	P	*		←
2	→	A	D	G	J	L	;		CTRL
3	F7	4	6	8	0	-	CLR		2
4	F1	Z	C	B	M	.	R.SH		SPACE
5	F3	S	F	H	K	:	=		⏏
6	F5	E	T	U	O	@	↑		Q
7	↓	L.SH	X	V	N	,	/		STOP

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 1. 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 kernal 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 kernal 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 kernal jumpblock at location \$FFCF (vectored at \$0324). A Basic program to emulate the keyboard scanning routine is given in Program 1.

```

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$="XXXXXXXXXXXXXXXXXXXXXXXXXXXX"
1230 DIM K$(71):FORI=0TO71:READK$(I):NEXTI
1240 DIM K(3,64):FORJ=0TO2:FORI=0TO64:K(J,I)=PEEK(60289+J*65+I):NEXTI,J
1250 FORI=0TO64:K(3,I)=PEEK(60536+I):NEXTI
1260 PRINT"##### KEYBOARD SCAN SIMULATION"
1270 PRINT"X OUTPUT: $DC00 INPUT : $DC01"
1280 PRINT"X BIT: 76543210 BIT : 76543210"
1290 PRINT"
1300 PRINT"
1310 PRINT"
1320 PRINT" KEY : |
1330 FORI=1TO4:PRINTP$;:FORJ=1TO1:PRINT"X";:NEXT
1340 PRINT" |"NEXT

```

```

1350 PRINTP$;"XXXXXXXXXXXXXXXXXXXX"
1360 PRINT"XXXXXXXXXXXXCURRENT SHIFT : 0"
1370 PRINT"CURRENT KEY : 64"
1380 PRINT"XXXXXXXXXXXXXXXXXXXX"
1390 GOSUB 1990
1400 POKE 56333,1:FORI=0TO7:P2(I)=2*I:NEXT
1410 KT=0:CS=0:CK=64:VL=8:A=255:GOSUB1700
1420 V=0:POKE56320,V:GOSUB1660:A=PEEK(56321)
1430 IF A=255 THEN 1420
1440 GOSUB 1700
1450 V=254:FORVL=0TO7
1460 POKE 56320,V:GOSUB 1660 DISPLAY OUTNEW
1470 A=PEEK(56321):GOSUB 1700 DISPLAY INPUT
1480 GOSUB 1860 CHECK FOR ANY NEWS DOWN
1490 V=(V*2)AND255)+1:NEXT
1500 IF CS<>3 THEN 1550
1510 IF (PEEK(53248+24) AND 2)=2 THEN 1540
1520 POKE 53248+24,PEEK(53248+24)OR2
1530 GOT01560
1540 POKE 53248+24,PEEK(53248+24)AND253
1550 KT=CS:IFKT>3THENKT=3
1560 CC=K(KT,CK)
1570 IF CC=255 THEN 1650
1580 KB(BP)=CC:BP=BP+1
1590 IF(BP<>10)AND(CC<>13)THEN 1640
1600 FOR I=0TOBP-1:POKE 631+I,KB(I):NEXT
1610 POKE198,BP:POKE 56333,129
1620 PRINT"XXXXXXXXXXXXXXXXXXXXXXXXXXXX"
1630 END
1640 GOSUB 1990
1650 GOT01360
1660 REM
1670 REM DISPLAY OUTPUT VALUE BITWISE
1680 REM
1690 A=V:PRINT"XXXXXXXXXXXXXXXXXXXX";GOTO 1800
1700 REM
1710 REM DISPLAY INPUT VALUE BITWISE
1720 REM
1730 PRINT"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";GOSUB 1800
1740 FORI=7TO0STEP-1:PRINTP$;
1750 FORJ=7TO0STEP-1:PRINT"X";NEXTJ
1760 C=VL*8+I:FORJ=1TO5
1770 C$=MID$(K$(C)+""",J,1)
1780 PRINT"X";IF(A AND P2(I))=0 THEN PRINT"X";
1790 PRINTC$+"X";NEXT:NEXT:PRINT"X";RETURN
1800 REM
1810 REM DISPLAY VALUE IN A BITWISE
1820 REM
1830 FOR I=7 TO 0 STEP -1
1840 C$="0":IF A AND P2(I) THEN C$="1"
1850 PRINT"X"+C$+"X";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"XXXXXXXXXXXXXXXXXXXXXXXXXXXX";CS
1930 PRINT"XXXXXXXXXXXXXXXXXXXXXXXXXXXX";CK;" "
1940 RETURN
1950 IN=K(KT,X+VL*8)
1960 IF IN>4 THEN CK=X+VL*8:GOTO1910
1970 IF IN=3 THEN CK=X+VL*8:GOTO1910
1980 CS=CSORIN:GOTO1910
1990 REM
2000 REM DISPLAY KEYBOARD BUFFER
2010 REM
2020 PRINT"XXXXXXXXXXXXXXXXXXXX"
2030 FOR I=0 TO 9
2040 PRINT"XXXXXXXXXXXXXXXXXXXX";KB(I):NEXT

```

```

2050 PRINT"XXXXXXXXXXXXXXXX";
2060 IF BP=0 THEN 2080
2070 PRINTLEFT$("<XXXXXXXXXXXX",BP-1);"XXXXXXXXXXXXXXXX "
2080 PRINT"XXXXXXXXXXXXXXXXX"
2090 RETURN
2100 REM
2110 REM MNEMONICS FOR KEY PRESS
2120 REM
2130 DATA DEL,RET,C,RT,F7,F1,F3,F5,C,DN
2140 DATA 3,W,A,4,Z,S,E,L,SH
2150 DATA 5,R,D,6,C,F,T,X
2160 DATA 7,Y,G,8,B,H,U,V
2170 DATA 9,I,J,0,M,K,O,N
2180 DATA +,P,L,-,.,":,@,","
2190 DATA \,*,",",,CLR,R,SH,=,↑,/
2200 DATA 1,←,CTRL,2,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      !
033C      ! EACH KEY CAN HAVE A MAXIMUM
033C      ! DEFINITION OF 255 CHARACTERS LONG
033C      ! (MAXIMUM BASIC STRING LENGTH)
033C      !
033C      ! DEFINE A FUNCTION KEY USING:
033C      ! SYS 49163,N,DEFINITION$
033C      !
033C      ! INITIALISE FUNCTION KEYS WITH:
033C      ! SYS 49152.
033C      !
C000      *=$C000
C000 A963      LDA #<FUNCTION
C002 8D2403     STA $0324
C005 A9C0      LDA #>FUNCTION
C007 8D2503     STA $0325
C00A 60        RTS
C00B      !
C00B 20F1B7    DEFINE      JSR $B7F1      !GET KEY#
C00E E009      CPX #$09      !LESS THAN 8?
C010 9005      BCC DEF1      !YES
C012 A20E      ERROR      LDX #$0E      !ILLEGAL QUANTITY
C014 6C0003     JMP (<$0300)    !SEND ERROR
C017      !
C017 E000      DEF1        CPX #$00      !IS IT ZERO?
C019 F0F7      BEQ ERROR      !YES
C01B CA        DEX
C01C 8A        TXA
C01D 48        PHA
C01E 20FDAE     JSR $AEFD
C021 209EAD     JSR $AD9E
C024 20A3B6     JSR $B6A3      !GET STRING
C027 8D51C0     STA STLEN      !DISCARD STRING
C02A 68        PLA
C02B 0A        ASL A
C02C AA        TAX
C02D BD53C0     LDA POINT,X
C030 8524      STA $24
C032 BD54C0     LDA POINT+1,X

```

20 The Commodore 64 Kernal and Hardware Revealed

```

C035 8525          STA $25
C037 A000          LDY #$00
C039 B122 DEF2     LDA ($22),Y
C03B 9124          STA ($24),Y
C03D C8           INY
C03E CC51C0       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 !
C051 00 STLEN     BYT 0
C052 00 FUNCFLG  BYT 0
C053 !
C053 00B800 POINT  WOR $B800,$BC00,$B900,$BD00
C05B 00BA00       WOR $BA00,$BE00,$BB00,$BF00
C063 !
C063 A599 FUNCTION LDA $99          !FROM KEYBOARD?
C065 F003         BEQ FUNC02       !YES
C067 4C57F1       JMP $F157        !DO NORMAL
C06A !
C06A A5D3 FUNC02  LDA $D3          !SAVE CURRENT CURSOR
C06C 85CA         STA $CA          ! COLUMN
C06E A5D6         LDA $D6
C070 85C9         STA $C9          ! AND ROW
C072 98          TYA
C073 48          PHA
C074 8A          TXA
C075 48          PHA
C076 A5D0         LDA $D0          !SCREEN OR KEYBOARD?
C078 F006         BEQ FUNC04       !KEYBOARD
C07A 4C3AE6       JMP $E63A        !DO FOR SCREEN
C07D !
C07D 2016E7 FUNC03 JSR $E716        !DISPLAY CHAR TO SCREEN
C080 AD52C0 FUNC04 LDA FUNCFLG     !DOING FUNCTION KEY?
C083 D066         BNE FUNC09
C085 A5C6         LDA $C6
C087 85CC         STA $CC
C089 8D9202       STA $0292        !ANY CHARS IN BUFFER?
C08C F0F2         BEQ FUNC04       !IF NOT, BLINK CURSOR
C08E 78          SEI              !AUTO SCROLL DOWN
C08F A5CF         LDA $CF          !REPEAT UNTIL CHAR
C091 F00C         BEQ FUNC05       !DISABLE KEYBOARD
C093 A5CE         LDA $CE          !CURSOR BLINK?
C095 AE8702       LDX $0287        !NO
C098 A000         LDY #$00         !RESTORE ORIG CHAR
C09A 84CF         STY $CF          ! AND COLOUR
C09C 2013EA       JSR $EA13
C09F 20B4E5 FUNC05 JSR $E5B4
C0A2 C983         CMP #$83
C0A4 D010         BNE FUNC07
C0A6 A209         LDX #$09
C0A8 78          SEI
C0A9 86C6         STX $C6
C0AB BDE6EC FUNC06 LDA $ECE6,X
C0AE 9D7602       STA $0276,X
C0B1 CA          DEX
C0B2 D0F7         BNE FUNC06
C0B4 F0CA         BEQ FUNC04
C0B6 C90D FUNC07  CMP #$0D
C0B8 D003         BNE FUNC08
C0BA 4C02E6 FUNCEXT JMP $E602
C0BD !
C0BD A6D4 FUNC08  LDX $D4
C0BF D0BC         BNE FUNC03

```


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 "OK";
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 "E ";
120 IF W THEN PRINT "W ";
130 IF S THEN PRINT "S ";
140 IF N THEN PRINT "N ";
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.

```

C000      *=$C000
C000      !*****
C000      !ROUTINE TO READ PADDLE PORTS
C000      ! USES BASIC USR COMMAND
C000      ! TO INITIALISE USR COMMAND
C000      ! POKE 785,0
C000      ! POKE 786,192
C000      !
C000      ! P= USR< PADDLE NUMBER>
C000      ! NOTE PADDLE NUMBER = 0 TO 3
C000      !*****
C000 20BFB1 USR      JSR $B1BF      !FLOAT TO FIXED
C003 A565          LDA $65        !LOW BYTE
C005 2903          AND #3
C007 A8           TAY
C008 A240          LDX ##40
C00A 2902          AND #2        !PORTS 0,1 OR 2,3
C00C F002          BEQ P00R1
C00E A280          LDX ##80
C010 78           P00R1      SEI
C011 8E00DC        STX $DC00      !KEYBOARD SCAN
C014 A280          LDX ##80      !USES $DC00
C016 CA           LOOP      DEX
C017 10FD          BPL LOOP      !DELAY FOR A/D
C019 98           TYA          !CONVERTER
C01A 2901          AND #1
C01C AA           TAX
C01D BC19D4        LDY $D419,X    !READ PORT
C020 58           CLI
C021 A900          LDA #0        !ZERO HIGH BYTE
C023 4C91B3        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 kernal software to control the text screen is split into two sections; print a character to the screen,

24 *The Commodore 64 Kernal and Hardware Revealed*

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*.

Display non control char

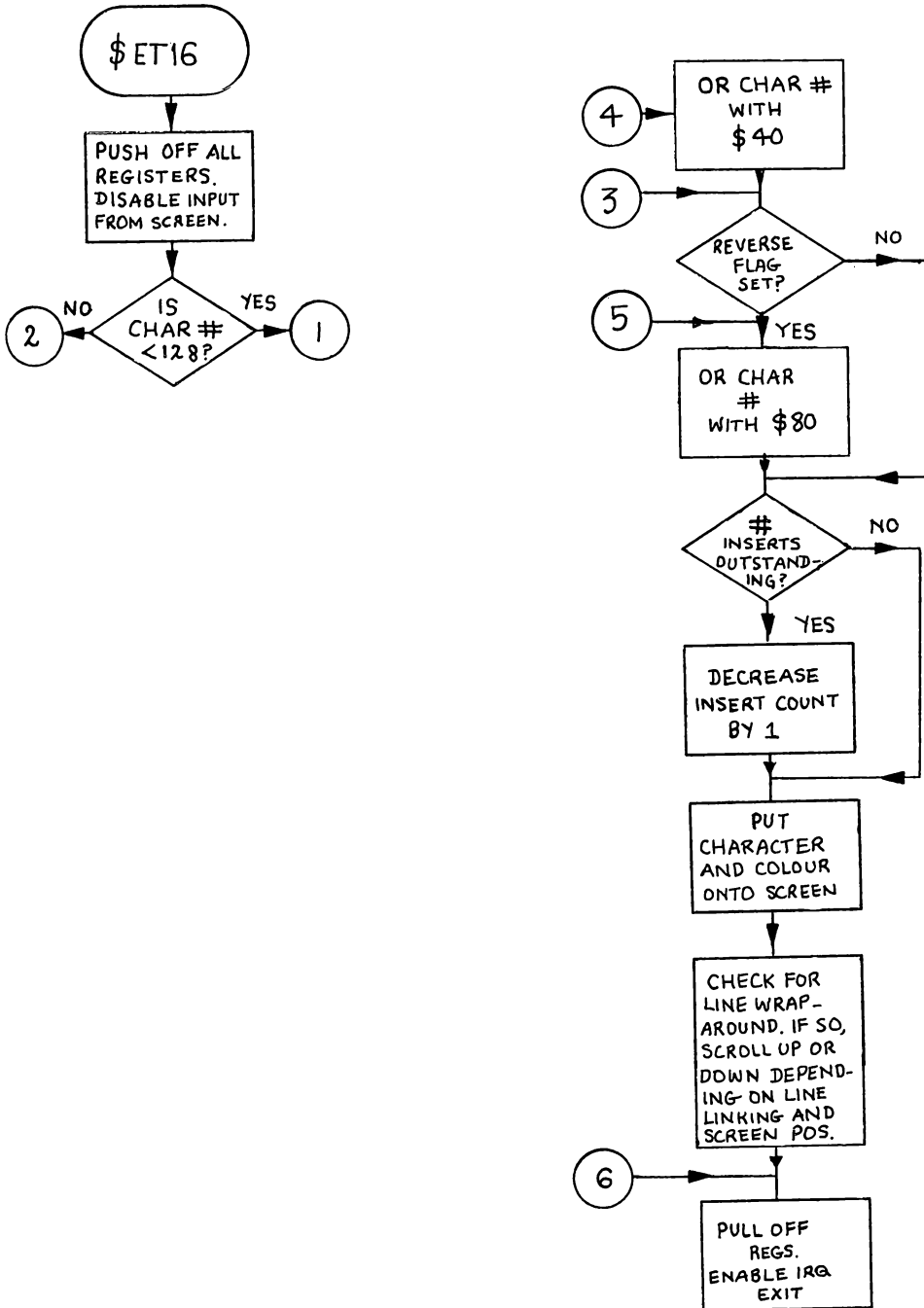


Fig. 2.2. Character output flowchart.

Handle char <128

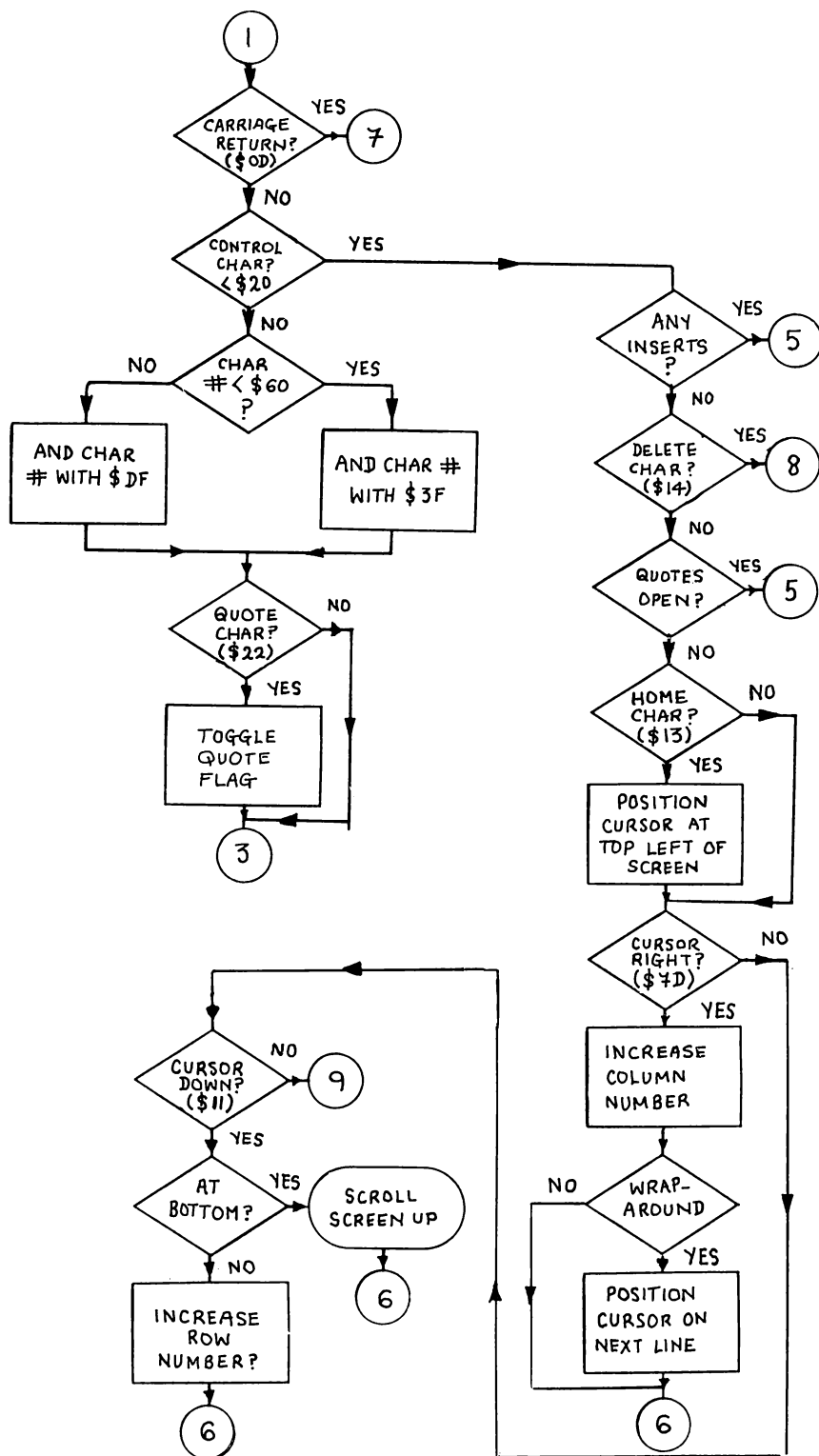
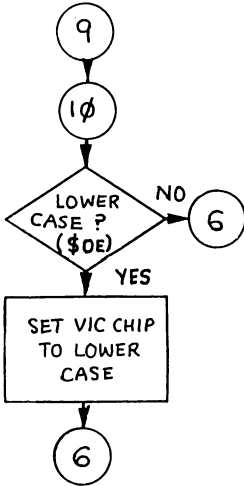


Fig. 2.2. (contd.)

Test for colour
or lower case



Handle colour codes

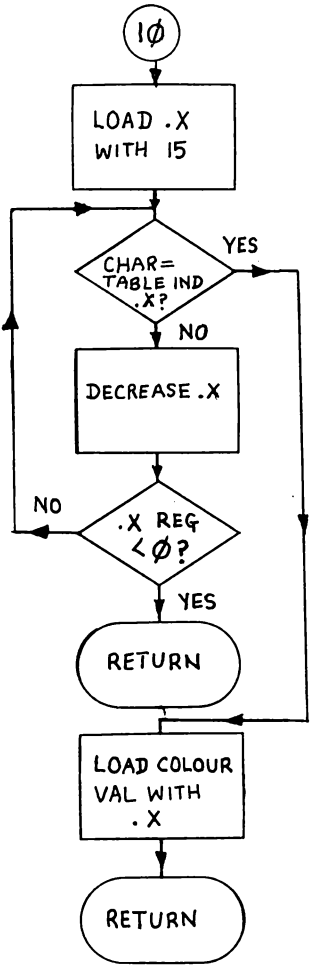
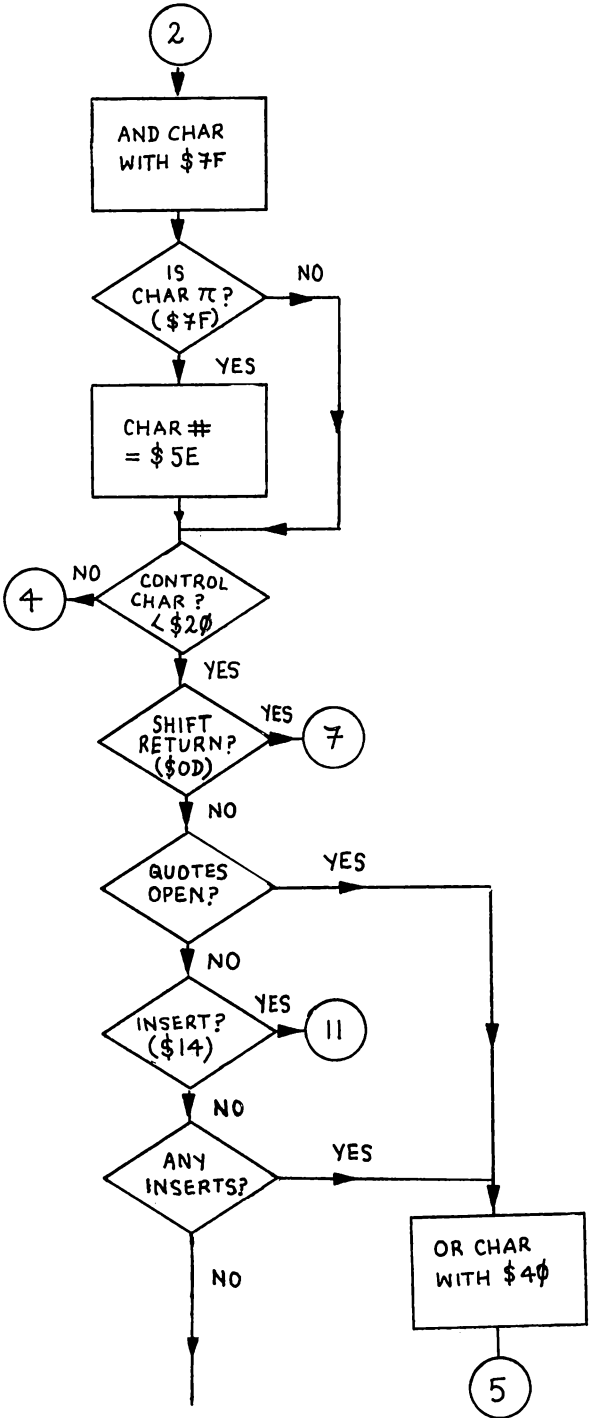


Fig. 2.2. (contd.)

Handle char # > 127



Test for colour or upper case

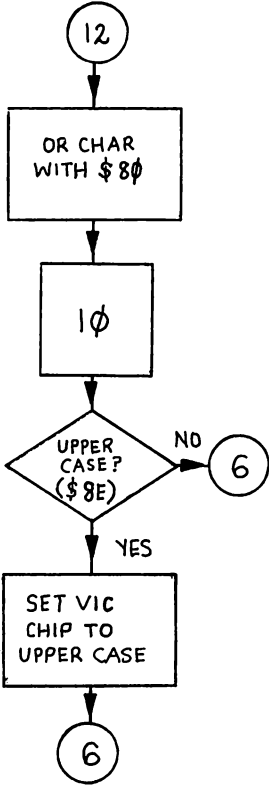


Fig. 2.2. (contd.)

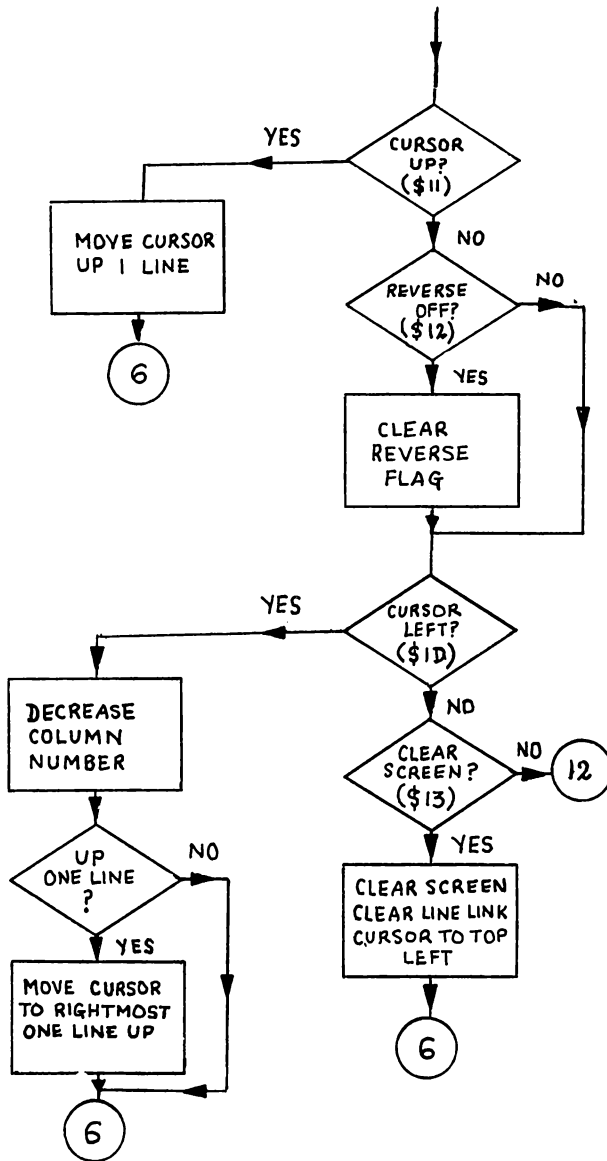
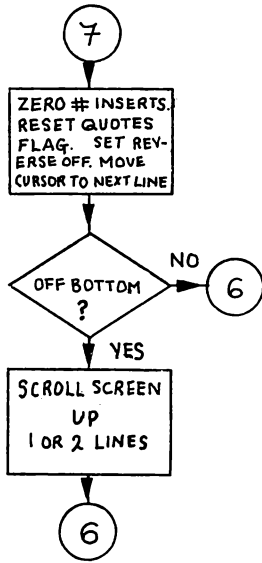
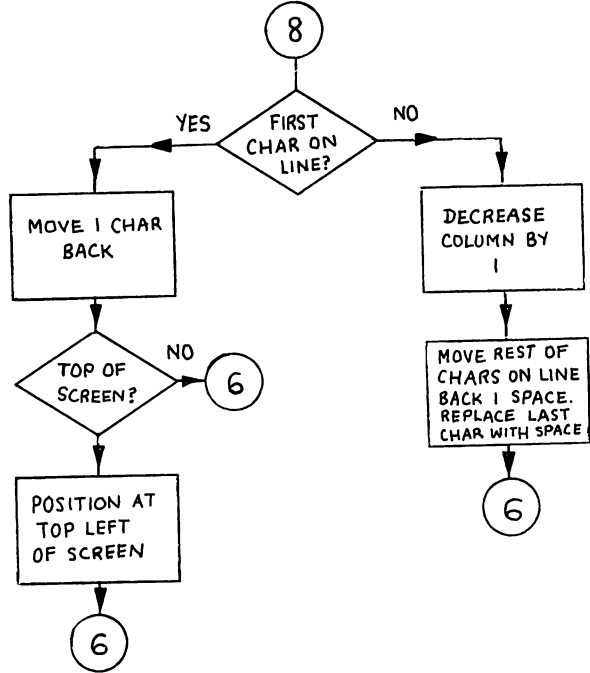


Fig. 2.2. (contd.)

Operate on carriage return



Delete a character



Insert a space

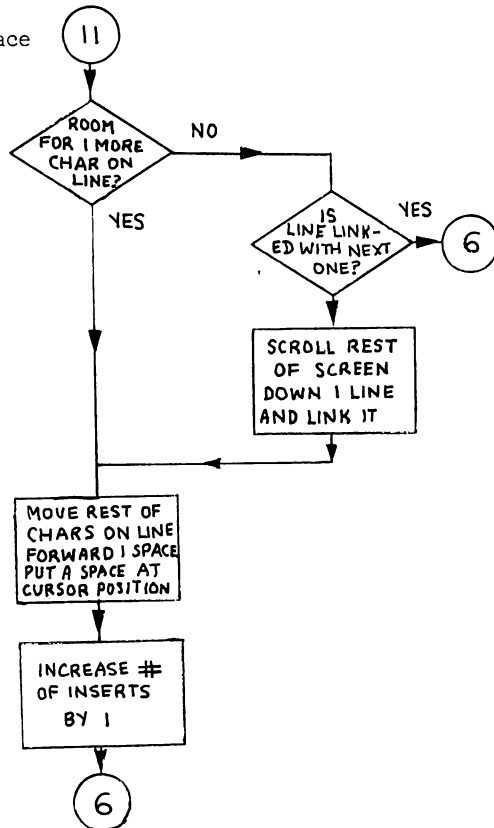


Fig. 2.2. (contd.)

Scroll screen up

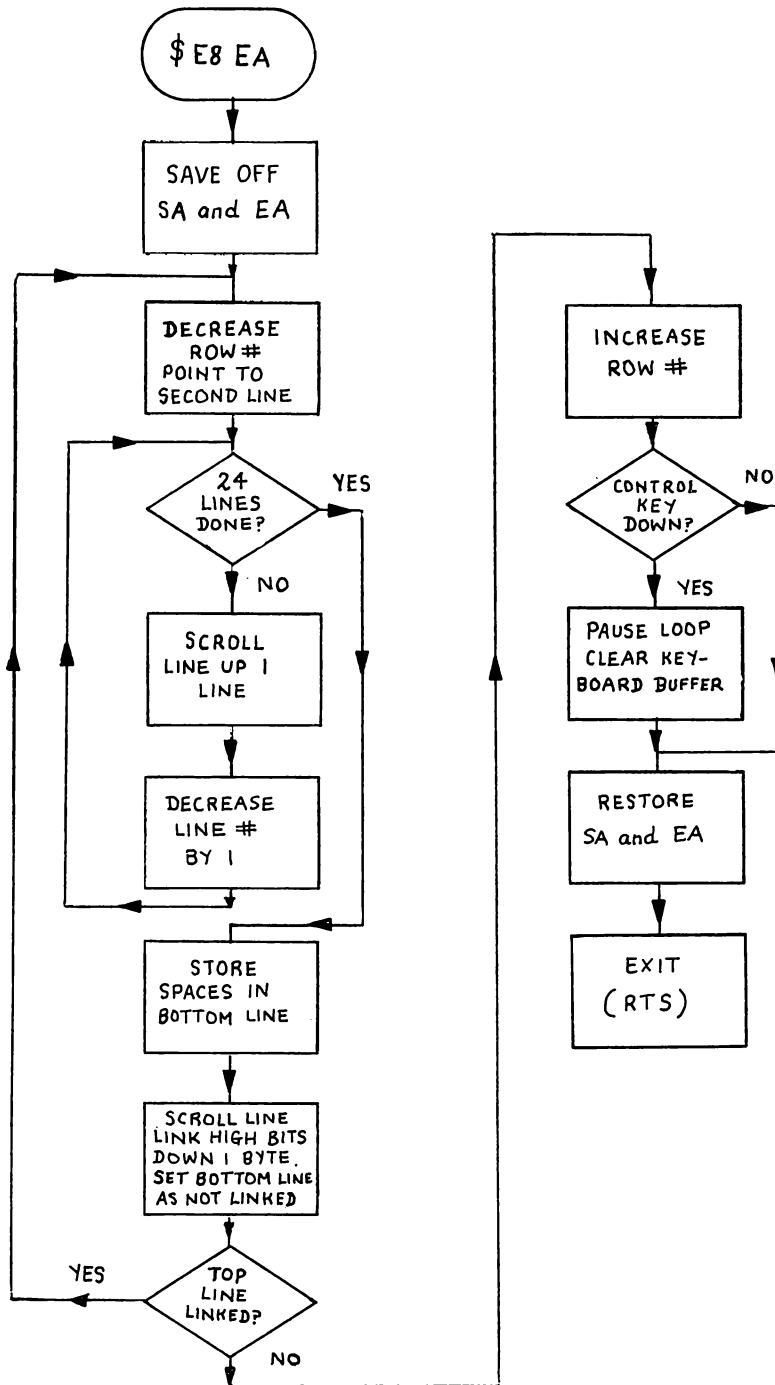


Fig. 2.3. Screen scrolling flowchart.

Chapter Three

Serial Communications

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.

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.

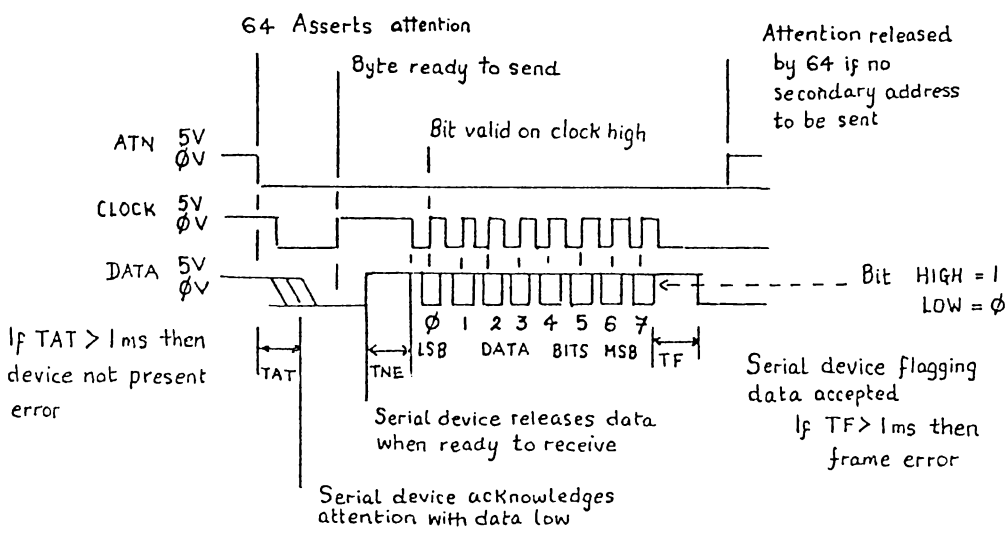


Fig.3.3. Serial bus timing.

2. Normal bytes on serial.

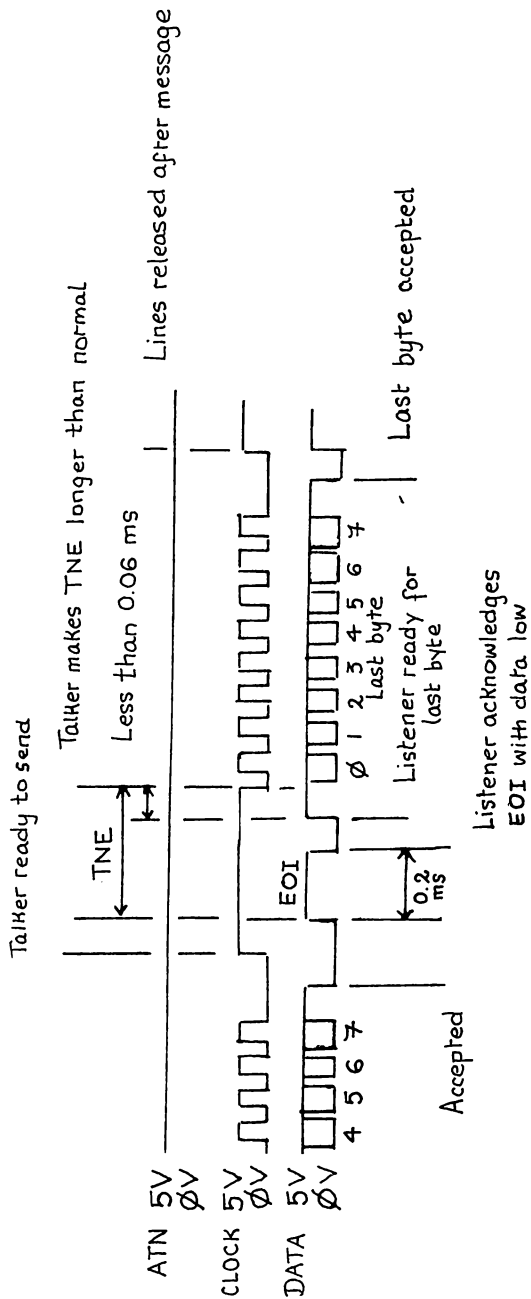


Fig.3.3. (contd.)

3. Last byte of a message (EOI handshake).

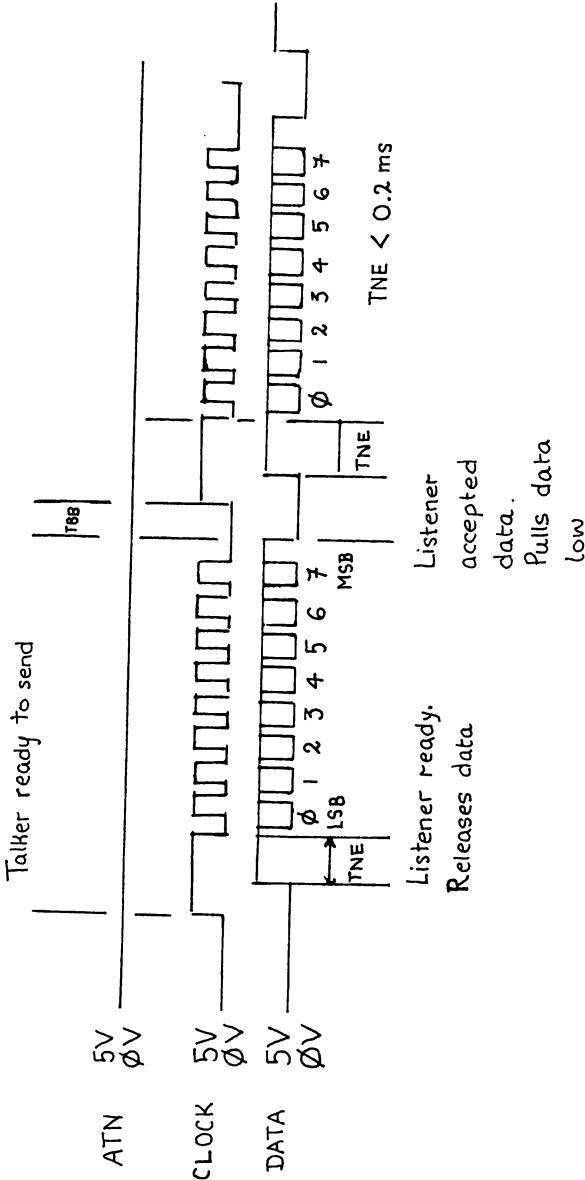


Fig.3.3. (contd.)

4. Listener becomes talker.

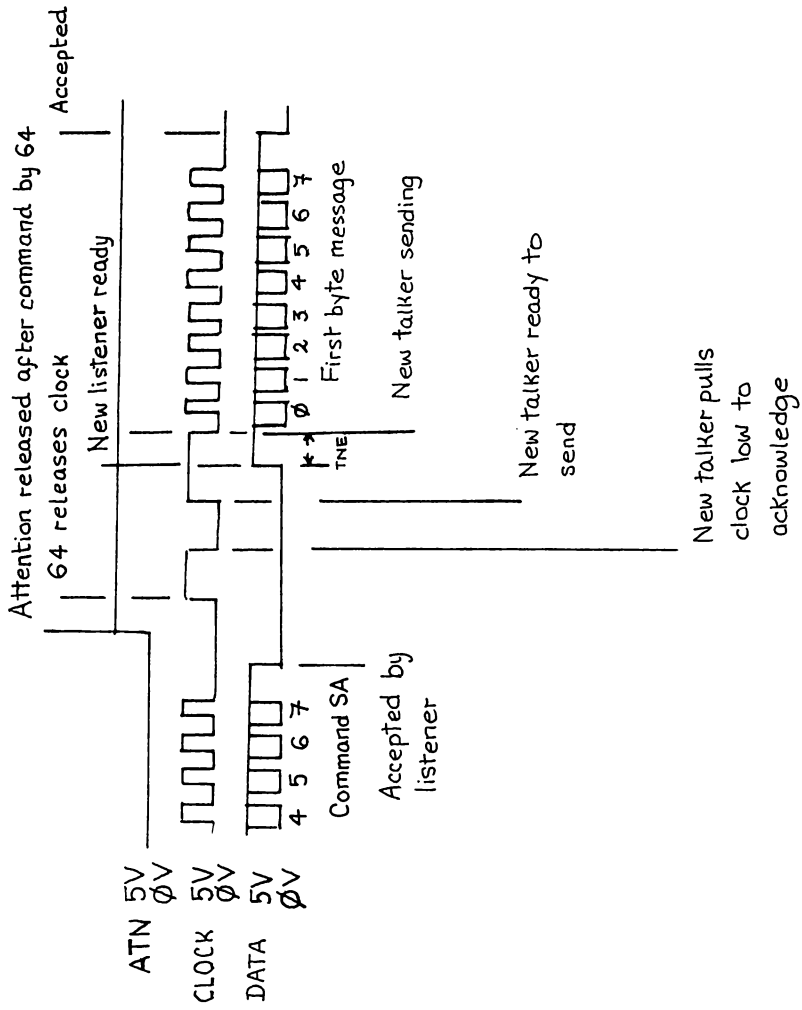


Fig.3.3. (contd.)

38 The Commodore 64 Kernal and Hardware Revealed

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

LOC	CODE	LINE
0000		;*****
0000		; SERIAL SYSTEM
0000		;*****
0000		.LIB KDECLARE
0000		* =\$90
0090		;
0090		;
0090		;KERNAL VARIABLES
0090		;
0090	STATUS **++1	;I/O OPERATION STATUS BYTE
0091	STKEY **++1	;STOP KEY FLAG
0092	SVXT **++1	;TEMPORARY
0093	VERCK **++1	;LOAD OR VERIFY FLAG
0094	C3FO **++1	;IEEE BUFFERED CHAR FLAG
0095	BSDUR **++1	;CHAR BUFFER FOR IEEE
0096	SYNO **++1	;CASSETTE SYNC #
0097	XSAV **++1	;TEMP FOR BASIN
0098	LDTND **++1	;INDEX TO LOGICAL FILE
0099	DFLTN **++1	;DEFAULT INPUT DEVICE #
009A	DFLTO **++1	;DEFAULT OUTPUT DEVICE #
009B	PRTY **++1	;CASSETTE PARITY
009C	DPSW **++1	;CASSETTE DIPOLE SWITCH
009D	MSGFLG **++1	;OS MESSAGE FLAG
009E	FTR1	;CASSETTE ERROR PASS1
009E	T1 **++1	;TEMPORARY 1
009F	TMPC	
009F	FTR2	;CASSETTE ERROR PASS2
009F	T2 **++1	;TEMPORARY 2
00A0	TIME **++3	;24 HOUR CLOCK IN 1/60TH SECONDS
00A3	R2D2	;SERIAL BUS USAGE
00A3	PCNTR **++1	;CASSETTE STUFF
00A4	BSOUR1	;TEMP USED BY SERIAL ROUTINE
00A4	FIRT **++1	
00A5	COUNT	;TEMP USED BY SERIAL ROUTINE
00A5	CNTDN **++1	;CASSETTE SYNC COUNTDOWN
00A6	BUFFT **++1	;CASSETTE BUFFER POINTER
00A7	INBIT	;RS-232 RCVR INPUT BIT STORAGE
00A7	SHCNL **++1	;CASSETTE SHORT COUNT
00A8	BITCI	;RS-232 RCVR BIT COUNT IN
00A8	RER **++1	;CASSETTE READ ERROR
00A9	RINONE	;RS-232 RCVR FLAG FOR START BIT CHECK
00A9	REZ **++1	;CASSETTE READING ZEROS
00AA	RIDATA	;RS-232 RCVR BYTE BUFFER
00AA	RDFLG **++1	;CASSETTE READ MODE
00AB	RIFRTY	;RS-232 RCVR PARITY STORAGE
00AB	SHCNH **++1	;CASSETTE SHORT CNT
00AC	SAL **++1	
00AD	SAH **++1	
00AE	EAL **++1	
00AF	EAH **++1	
00B0	CMFO **++1	
00B1	TEMP **++1	
00B2	TAPE1 **++2	;ADDRESS OF TAPE BUFFER
00B4	BITTS	;RS-232 TRNS BIT COUNT
00B4	SNSW1 **++1	

LOC	CODE	LINE	
00B5		NXTBIT	;RS-232 TRNS NEXT BIT TO BE SENT
00B5		DIFF **+=1	
00B6		RODATA	;RS-232 TRNS BYTE BUFFER
00B6		PRP **+=1	
00B7		FNLEN **+=1	;LENGTH CURRENT FILE N STR
00B8		LA **+=1	;CURRENT FILE LOGICAL ADDR
00B9		SA **+=1	;CURRENT FILE 2ND ADDR
00BA		FA **+=1	;CURRENT FILE PRIMARY ADDR
00BB		FNADR **+=2	;ADDR CURRENT FILE NAME STR
00BD		ROPRTY	;RS-232 TRNS PARITY BUFFER
00BD		OCHAR **+=1	
00BE		FSBLK **+=1	;CASSETTE READ BLOCK COUNT
00BF		MYCH **+=1	
00C0		CAS1 **+=1	;CASSETTE MANUAL/CONTROLLED SWITCH
00C1		TMPO	
00C1		STAL **+=1	
00C2		STAH **+=1	
00C3		MEMUSS	;CASSETTE LOAD TEMPS (2 BYTES)
00C3		TMP2 **+=2	
00C5			
00C5		;VARIABLES FOR SCREEN EDITOR	
00C5			
00C5		LSTX **+=1	;KEY SCAN INDEX
00C6		NDX **+=1	;INDEX TO KEYBOARD Q
00C7		RVS **+=1	;RVS FIELD ON FLAG
00C8		INDX **+=1	
00C9		LSXP **+=1	;X POS AT START
00CA		LSTP **+=1	
00CB		SFDX **+=1	;SHIFT MODE ON PRINT
00CC		BLNSW **+=1	;CURSOR BLINK ENAB
00CD		BLNCT **+=1	;COUNT TO TOGGLE CUR
00CE		GDBLN **+=1	;CHAR BEFORE CURSOR
00CF		BLNON **+=1	;ON/OFF BLINK FLAG
00D0		CRSW **+=1	;INPUT VS GET FLAG
00D1		PNT **+=2	;POINTER TO ROW
00D3		FNTR **+=1	;POINTER TO COLUMN
00D4		QTSW **+=1	;QUOTE SWITCH
00D5		LNMX **+=1	;40/80 MAX POSITION
00D6		TBLX **+=1	
00D7		DATA **+=1	
00D8		INSRT **+=1	;INSERT MODE FLAG
00D9		LDTB1 **+=25	;40/80 LINE FLAGS
00F2		LINTMP **+=1	;TEMPORARY FOR LINE INDEX
00F3		USER **+=2	;SCREEN EDITOR COLOUR IF
00F5		KEYTAB **+=2	;KEYSCAN TABLE INDIRECT
00F7			
00F7		;RS-232 Z-PAGE	
00F7			
00F7		RIBUF **+=2	;RS-232 INPUT BUFFER POINTER
00F9		ROBUF **+=2	;RS-232 OUTPUT BUFFER POINTER
00FB		FREKZP **+=4	;FREE KERNAL ZERO PAGE
00FF		BASZPT **+=1	;LOCATION (\$00FF) USED BY BASIC
0100			
0100		**=\$100	
0100		BAD **+=1	
0101		**=\$200	
0200		BUF **+=89	;BASIC/MONITOR BUFFER
0259			
0259		;TABLES FOR OPEN FILES	
0259			
0259		LAT **+=10	;LOGICAL FILE NUMBERS
0263		FAT **+=10	;PRIMARY DEVICE NUMBERS
026D		SAT **+=10	;SECONDARY ADDRESSES
0277			
0277		;SYSTEM STORAGE	
0277			
0277		KEYD **+=10	;IRQ KEYBOARD BUFFER
0281		MEMSTR **+=2	;START OF MEMORY

40 The Commodore 64 Kernal and Hardware Revealed

LOC	CODE	LINE
0283	MEMSIZ **++2	;TOP OF MEMORY
0285	TIMOUT **++1	;IEEE TIMEOUT FLAG
0286	;	
0286	;SCREEN EDITOR STORAGE	
0286	;	
0286	COLOR **++1	;ACTIVE COLOUR NIBBLE
0287	GDCOL **++1	;ORIGINAL COLOUR BEFORE CURSOR
0288	HIBASE **++1	;BASE LOCATION OF SCREEN (TOP)
0289	XMAX **++1	
028A	RPTFLG **++1	;KEY REPEAT FLAG
028B	KOUNT **++1	
028C	DELAY **++1	
028D	SHFLAG **++1	;SHIFT FLAG BYTE
028E	LSTSHF **++1	;LAST SHIFT PATTERN
028F	KEYLOG **++2	;INDIRECT FOR KEYBOARD TABLE SETUP
0291	MODE **++1	;0-PET MODE, 1-CATTACANNA
0292	AUTODN **++1	;AUTO SCROLL DOWN FLAG (=0 ON,<0 OFF)
0293	;	
0293	;RS-232 STORAGE	
0293	;	
0293	M26CTR **++1	;6526 CONTROL REGISTER
0294	M26CDR **++1	;6526 COMMAND REGISTER
0295	M26AJB **++2	;NON STANDARD (BITTIME/2-100)
0297	RSSTAT **++1	;RS-232 STATUS REGISTER
0298	BITNUM **++1	;NUMBER OF BITS TO SEND (FAST RESPONSE)
0299	BAUDOF **++2	;BAUD RATE FULL BIT TIME
029B	;	
029B	;RECIEVER STORAGE	
029B	;	
029B	RIDBE **++1	;INPUT BUFFER INDEX TO END
029C	RIDBS **++1	;INPUT BUFFER POINTER TO START
029D	;	
029D	;TRANSMITTER STORAGE	
029D	;	
029D	RODBS **++1	;OUTPUT BUFFER INDEX TO START
029E	RODBE **++1	;OUTPUT BUFFER INDEX TO END
029F	;	
029F	IRQTMP **++2	;HOLDS IRQ DURING TAPE OPS
02A1	;	
02A1	;	
02A1	**=\$0300	;PROGRAM INDIRECTS(10)
0300	**=\$0300+20	;KERNAL/OS INDIRECTS (20)
0314	CINV **++2	;IRQ RAM VECTOR
0316	CBINV **++2	;BRK INSTR RAM VECTOR
0318	NMINV **++2	;NMI RAM VECTOR
031A	IOPEN **++2	;INDIRECTS FOR CODE
031C	ICLOSE **++2	;CONFORMS TO KERNAL SPEC 8/19/80
031E	ICKIN **++2	
0320	ICKOUT **++2	
0322	ICLRCH **++2	
0324	IBASIN **++2	
0326	IBSOUT **++2	
0328	ISTOP **++2	
032A	IGETIN **++2	
032C	ICLALL **++2	
032E	USRCMD **++2	
0330	ILOAD **++2	
0332	ISAVE **++2	;SAVESP
0334	;	
0334	**=\$0300+60	
033C	TBUFR **++192	;CASSETTE DATA BUFFER
03FC	;	
03FC	**=\$0400	
0400	CRMSCN **++999	;64 SCREEN
07E7	;	
07E7	**=\$0800	
0800	;RAMLOC	
0800	;	

42 The Commodore 64 Kernal and Hardware Revealed

LOC	CODE	LINE
DD10		LLEN =40 ;SINGLE LINE 40 COLUMNS
DD10		LLEN2 =80 ;DOUBLE LINE 80 COLUMNS
DD10		NLINES =25 ;25 ROWS ON SCREEN
DD10		BLUE =6 ;BLUE SCREEN COLOUR
DD10		LTBLUE =14 ;LT BLUE CHAR COLOUR
DD10		CR =\$D ;CARRIAGE RETURN
DD10		MAXCHR =80
DD10		NWRAP =2
DD10		.END
DD10		.LIB KSER1

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		==\$ED09
ED09		;*****
ED09		;*COMMAND SERIAL BUS TO TALK.
ED09		;*THE ACCUMULATOR MUST BE LOADED WITH THE
ED09		;*DEVICE NUMBER THAT YOU WISH TO TALK.
ED09		;*****
ED09	09 40	LB36 ORA #\$40 ;MAKE ADDR TALK
ED0B	2C	.BYT \$2C ;SKIP NEXT COMMAND

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 SStatus 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

44 The Commodore 64 Kernal and Hardware Revealed

LUC	CODE	LINE
ED0C		;*****
ED0C		;*COMMAND SERIAL BUS TO LISTEN.
ED0C		;*TO USE THIS ROUTINE, THE ACCUMULATOR MUST
ED0C		;*FIRST BE LOADED WITH THE DEVICE NUMBER THAT
ED0C		;*YOU WISH TO LISTEN (RECEIVE DATA).
ED0C		;*****
ED0C	09 20	L966 ORA #\$20 ;MAKE ADDR LISTEN
ED0E	20 A4 F0	JSR \$F0A4 ;PROTECT FROM RS323 NMI
ED11	48	L980 PHA
ED12	24 94	BIT C3F0 ;CHAR IN BUFFER?
ED14	10 0A	BPL L864 ;NO
ED16		;
ED16		;SEND BUFFERED CHAR
ED16		;
ED16	38	SEC ;SET EOI FLAG
ED17	66 A3	ROR R2D2
ED19	20 40 ED	JSR L859 ;SEND LAST CHAR
ED1C	46 94	LSR C3F0 ;BUFFER CLEAR
ED1E	46 A3	LSR R2D2 ;CLEAR EOI FLAG
ED20	68	L864 PLA ;TALK/LISTEN ADDR
ED21	85 95	STA BSOUR
ED23	78	SEI
ED24	20 97 EE	JSR L844
ED27	C9 3F	CMP #\$3F ;CLKHI ONLY ON UNLISTEN
ED29	D0 03	BNE L839
ED2B	20 85 EE	JSR L875
ED2E	AD 00 DD	L839 LDA D2DFA ;ASSERT ATTENTION
ED31	09 08	ORA #\$08
ED33	8D 00 DD	STA D2DFA
ED36		;
ED36	78	L842 SEI
ED37	20 8E EE	JSR L843 ;SET CLOCK LINE LOW
ED3A	20 97 EE	JSR L844
ED3D	20 83 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	B0 64	BCS L856
ED49	20 85 EE	JSR L875 ;CLOCK LINE HI
ED4C	24 A3	BIT R2D2 ;EOI FLAG TEST
ED4E	10 0A	BPL L850
ED50		;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	B0 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 LO
ED62		;
ED62		;SET TO SEND DATA
ED62		;
ED62	A9 08	LDA #\$08 ;COUNT 8 BITS
ED64	85 A3	STA COUNT
ED66	AD 00 DD	L848 LDA D2DFA ;DEBOUNCE BUS
ED69	CD 00 DD	CMF D2DFA
ED6C	D0 FB	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	B0 05	BCS L851
ED75	20 A0 EE	JSR L841
ED78	D0 03	BNE L853
ED7A	20 97 EE	L851 JSR L844
ED7D	20 85 EE	L853 JSR L875 ;CLOCK HI
ED80	EA	NOP
ED81	EA	NOP
ED82	EA	NOP

LOC	CODE	LINE		
ED83	EA		NOF	
ED84	AD 00 DD		LDA D2DFA	
ED87	29 DF		AND #\$0F	;DATA HI
ED89	09 10		ORA #\$10	;CLOCK LO
ED8E	8D 00 DD		STA D2DFA	
ED8E	C6 A5		DEC COUNT	
ED90	D0 D4		BNE L848	
ED92	A9 04		LDA #\$04	;SET TIMER FOR 1 MS
ED94	8D 07 DC		STA D1TBH	
ED97	A9 19		LDA #\$19	
ED99	8D 0F DC		STA D1CRB	
ED9C	AD 0D DC		LDA D1ICR	
ED9F	AD 0D DC	L855	LDA D1ICR	
EDA2	29 02		AND #\$02	
EDA4	D0 0A		BNE L847	
EDA6	20 A9 EE		JSR L854	
EDA9	B0 F4		BCC L855	
EDAB	58		CLI	;ENABLE IRQ
EDAC	60		RTS	
EDAD	A9 80	L856	LDA #\$80	;DEVICE NOT PRESENT
EDAF	2C		.BYT \$2C	
EDB0	A9 03	L847	LDA #\$03	;FRAMING ERROR
EDB2	20 1C FE	L852	JSR \$FE1C	;SEND MESSAGE
EDB5	58		CLI	;ENABLE IRQ
EDB6	18		CLC	
EDB7	90 4A		BCC L1004	;ALWAYS

SECOND

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).

LOC	CODE	LINE		
EDB9			;*****	
EDB9			;*SEND SECONOARY ADDRESS AFTER LISTEN.	
EDB9			;*THIS ROUTINE IS USED TO SEND A SECONDARY	
EDB9			;*ADDRESS AFTER A CALL TO THE LISTEN COMMAND.	
EDB9			;*****	
EDB9	85 95	L871	STA BSOUR	;BUFFER CHAR

46 The Commodore 64 Kernal and Hardware Revealed

```
EDBB 20 36 ED      JSR L842      ;SEND IT
EDBE                      ;
EDBE                      ;RELEASE ATTENTION
EDBE                      ;
EDBE AD 00 DD      L983      LDA D2DFA
EDC1 29 F7                      AND #$F7
EDC3 8D 00 DD      STA D2DFA      ;RELEASE
EDC6 60                      RTS
```

TKSA

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 SStatus 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

LOC	CODE	LINE
EDC7		;*****
EDC7		;SEND TALK SA.
EDC7		;*THIS ROUTINE IS USED TO SEND A SECONDARY
EDC7		;*ADDRESS TO A DEVICE THAT HAS ALREADY BEEN
EDC7		;*COMMANDED TO TALK.
EDC7		;*****
EDC7	85 95	L860 STA BSOUR ;BUFFER CHAR
EDC9	20 36 ED	JSR L842 ;SEND SA
EDCC		;
EDCC		;SHIFT OVER TO LISTENER
EDCC		;
EDCC	78	L970 SEI ;DISABLE IRQ
EDCD	20 A0 EE	JSR L841 ;DATA LINE LO
EDD0	20 BE ED	JSR L983
EDD3	20 85 EE	JSR L875 ;CLOCK LINE HI
EDD6	20 A9 EE	L968 JSR L854 ;WAIT FOR CLOCK LO
EDD9	30 FB	BMI L968
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 SStatus 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.

LOC	CODE	LINE
EDDD		;*****
EDDD		;BUFFERED OUTPUT TO SERIAL BUS
EDDD		;*****
EDDD	24 94	L861 BIT C3P0 ;BUFFERED CHAR?
EDDF	30 05	BMI L949 ;YES, SEND LAST
EDE1	38	SEC ;NO
EDE2	66 94	ROR C3P0 ;SET BUFFERED CHAR FLAG
EDE4	D0 05	BNE L862 ;ALWAYS
EDE6		;
EDE6	48	L949 PHA ;SAVE CURRENT CHAR
EDE7	20 40 ED	JSR L859 ;SEND LAST CHAR
EDEA	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 SStatus var. \$90) – attention not acknowledged by data low within 1 ms

48 The Commodore 64 Kernal and Hardware Revealed

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	L963 SEI
EDF0	20 8E EE	JSR L943
EDF3	AD 00 DD	LDA D2DFA ;PULL ATN
EDF6	09 08	ORA #\$08
EDF8	8D 00 DD	STA D2DFA
EDFB	A9 5F	LDA #\$5F ;UNTALK
EDFD	2C	.BYT \$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 SStatus 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
EDFE		;*****
EDFE		;SEND UNLISTEN.
EDFE		;THIS ROUTINE SENDS AN 'UNLISTEN' TO
EDFE		;THE SERIAL BUS. IT WILL TELL ALL DEVICES
EDFE		;IN LISTEN MODE TO STOP LISTENING.
EDFE		;*****
EDFE	A9 3F	L1006 LDA #\$3F ;UNLISTEN COMMAND
EE00	20 11 ED	JSR L980 ;SEND
EE03		;
EE03		;RELEASE ALL LINES
EE03		;
EE03	20 BE ED	L1004 JSR L983 ;RELEASE ATN
EE06		;

```

EE06          ;DELAY THEN RELEASE CLOCK AND DATA
EE06          J
EE06 8A      L858  TXA          ;DELAY APPROX 60 MICRO SECS
EE07 A2 0A   L858  LDX #0A
EE09 CA      L876  DEX
EE0A D0 FD   L876  BNE L876
EE0C AA      TAX
EE0D 20 85 EE JSR L875
EE10 4C 97 EE JMP L844

```

ACPTR

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			;*****	
EE13			;INPUT A BYTE FROM SERIAL BUS	
EE13			;*****	
EE13	78	L865	SEI	;DISABLE IRQ
EE14	A9 00		LDA #\$00	;SET EOI/ERROR FLAG
EE16	85 A5		STA COUNT	
EE18	20 85 EE		JSR L875	;RELEASE CLOCK LINE
EE18	20 A9 EE	L943	JSR L854	;WAIT FOR CLOCK HI
EE1E	10 FB		BFL L943	
EE20	A9 01	L866	LDA #\$01	;SET TIMER B FOR 256 US
EE22	80 07 DC		STA D1TBH	
EE25	A9 19		LDA #\$19	
EE27	80 0F DC		STA D1CRB	
EE2A	20 97 EE		JSR L844	
EE2D	AD 0D DC		LDA D1ICR	
EE30	AD 0D DC	L872	LDA D1ICR	
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
EE3A	30 F4		BMI L872	;NOT YET
EE3C	10 18		BPL L870	;YES
EE3E			;	
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			;TIMER RAN OUT, DO AN EOI	
EE47			;	
EE47	20 A0 EE	L867	JSR L841	;DATA LINE LO
EE4A	20 85 EE		JSR L875	;DELAY, SET DATA HI
EE4D	A9 40		LDA #\$40	
EE4F	20 1C FE		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			;BYTE TRANSFER	
EE56			;	
EE56	A9 08	L870	LDA #\$08	;SET UP COUNTER
EE58	85 A5		STA COUNT	
EE5A	AD 00 DD	L869	LDA D2DPA	;WAIT FOR CLOCK HI
EE5D	CD 00 DD		CMP D2DPA	;DEBOUNCE
EE60	D0 F8		BNE L869	
EE62	0A		ASL A	
EE63	10 F5		BPL L869	
EE65	66 A4		ROR BSOUR1	;ROTATE DATA IN
EE67	AD 00 DD	L873	LDA D2DPA	;WAIT FOR CLOCK LO
EE6A	CD 00 DD		CMP D2DPA	;DEBOUNCE
EE6D	D0 F8		BNE L873	
EE6F	0A		ASL A	
EE70	30 F5		BMI L873	
EE72	C6 A5		DEC COUNT	
EE74	D0 E4		BNE L869	;MORE BITS
EE76	20 A0 EE		JSR L841	;DATA LO

EE79	24 90		BIT STATUS	;CHECK FOR EOI
EE7B	50 03		BVC L874	;NONE
EE7D	20 06 EE		JSR L85B	;DELAY AND DATA HI
EE80	AS A4	L874	LDA B50UR1	
EE82	58		CLI	;ENABLE IRQ
EE83	18		CLC	;GOOD EXIT
EE84	60		RTS	

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.

LOC	CODE	LINE
EE85		;SET CLOCK LINE HI (INVERTED)
EE85		;SET CLOCK LINE HI (INVERTED)
EE85		;SET CLOCK LINE HI (INVERTED)
EE85	AD 00 DD	L875 LDA D2DFA
EE88	29 EF	AND #\$EF
EE8A	8D 00 DD	STA D2DFA
EE8D	60	RTS
EE8E		;SET CLOCK LINE LO (INVERTED)
EE8E		;SET CLOCK LINE LO (INVERTED)
EE8E		;SET CLOCK LINE LO (INVERTED)
EE8E	AD 00 DD	L843 LDA D2DFA
EE91	09 10	ORA #\$10
EE93	8D 00 DD	STA D2DFA
EE96	60	RTS
EE97		;SET DATA LINE HI (INVERTED)
EE97		;SET DATA LINE HI (INVERTED)
EE97	AD 00 DD	L844 LDA D2DFA
EE9A	29 DF	AND #\$DF
EE9C	8D 00 DD	STA D2DFA
EE9F	60	RTS
EEA0		;SET DATA LINE LO (INVERTED)
EEA0		;SET DATA LINE LO (INVERTED)
EEA0		;SET DATA LINE LO (INVERTED)
EEA0	AD 00 DD	L841 LDA D2DFA
EEA3	09 20	ORA #\$20
EEA5	8D 00 DD	STA D2DFA
EEA8	60	RTS
EEA9		;DEBOUNCE THE PIA
EEA9		;DEBOUNCE THE PIA
EEA9		;DEBOUNCE THE PIA
EEA9	AD 00 DD	L854 LDA D2DFA
EEAC	CD 00 DD	CMP D2DFA
EEAF	D0 FB	BNE L854
EEB1	0A	ASL A
EEB2	60	RTS
EEB3		;DELAY 1 MS
EEB3		;DELAY 1 MS
EEB3		;DELAY 1 MS
EEB3	8A	L846 TXA

52 The Commodore 64 Kernal and Hardware Revealed

```
EEB4 A2 B8          LDX #B8
EEB6 CA            L845 DEX
EEB7 D0 FD          BNE L845
EEB9 AA            TAX
EEBA 60            RTS
EEBB               .END
EEBB               .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: None

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

LOC	CODE	LINE
EEBB		==F13E
F13E		;
F13E		*****
F13E		;* GETIN -- GET CHARACTER FROM CHANNEL.
F13E		;* CHANNEL IS DETERMINED BY DFLTIN.
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		;* ADVANCE TO BASIN. THE CHARACTER IS
F13E		;* RETURNED IN .A. IF ZERO, NULL CHAR.
F13E		*****
F13E		;
F13E	A5 99	NGETIN LDA DFLTIN ;CHECK DEVICE
F140	D0 08	BNE L924 ;NOT KEYBOARD
F142	A5 C6	LDA NOX ;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 XSAV ;SAVE .Y, USED IN RS-232
F150	20 86 F0	JSR \$F086
F153	A4 97	LDY XSAV ;RESTORE .Y
F155	18	L944 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 $\neq 0$ (EOF), the character 13 (carriage return) is returned with CARRY clear. Otherwise one byte is read using the ACPTR routine.

LOC	CODE	LINE
F157		;
F157		;*****
F157		;* BASIN-- INPUT CHARACTER FROM CHANNEL.
F157		;* BASIN DIFFERS FROM GETIN ON KEYBOARD
F157		;* AND RS-232 ONLY. THE SCREEN EDITOR
F157		;* MAKES READY AN ENTIRE LINE WHICH IS
F157		;* PASSED CHARACTER BY CHARACTER UP
F157		;* TO THE CARRIAGE RETURN. THE CHARACTER
F157		;* IS RETURNED IN .A. ZERO FOR NULL CHAR
F157		;* OTHER DEVICES ARE:
F157		;* 0 --- KEYBOARD
F157		;* 1 --- CASSETTE
F157		;* 2 --- RS-232
F157		;* 3 --- SCREEN
F157		;* 4-31 --- SERIAL BUS
F157		;*****
F157		;
F157	A5 99	NBASIN LDA DFLTN ;CHECK DEVICE
F159	D0 0B	BNE L927 ;NOT KEYBOARD
F15B		;
F15B		;INPUT FROM KEYBOARD
F15B		;
F15B	A5 D3	LDA FNTR ;SAVE CURRENT:
F15D	85 CA	STA LSTP ;CURSOR COLUMN,
F15F	A5 D6	LDA TELX
F161	85 C9	STA LSXP ;LINE NUMBER
F163	4C 32 E6	JMP \$E632 ;BLINK CURSOR UNTIL RETURN
F166		;
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 C8	STA INDX ;ON THIS LINE
F170	4C 32 E6	JMP \$E632 ;PICK UP CHARACTERS
F173		;
F173	B0 38	L928 BCS L939 ;DEVICES>3
F175	C9 02	CMP #\$02 ;RS-232?
F177	F0 3F	BEQ \$F1B8 ;YES
F179		;
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 0D	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
F188		==F1CA
F1CA		;
F1CA		*****
F1CA		; BSOUT -- OUTPUT CHAR STORED IN .A TO
F1CA		; CHANNEL DETERMINED BY VARIABLE DFLT0:
F1CA		; 0 --- INVALID
F1CA		; 1 --- CASSETTE
F1CA		; 2 --- RS-232
F1CA		; 3 --- SCREEN
F1CA		; 4-31 --- SERIAL BUS
F1CA		*****
F1CA		;
F1CA	48	NBSOUT PHA ;PRESERVE .A
F1CB	A5 9A	LDA DFLT0 ;CHECK DEVICE
F1CD	C9 03	CMF #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 \$F1DB ;DEVICE 1 OR 2
F1D7		;
F1D7		;PRINT TO SERIAL BUS
F1D7		;
F1D7	68	PLA
F1D8	4C DD ED	JMP L861
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 F3	NCHKIN JSR L1000 ;FILE OPENED?
F211	F0 03	BEQ L950 ;YES
F213	4C 01 F7	JMP L1009 ;NO, FILE NOT OPEN
F216	20 1F F3	L950 JSR L1002 ;GET FILE INFO
F219	A5 BA	LDA FA
F21B	F0 16	BEQ L963 ;KEYBOARD
F21D		;
F21D		;COULD BE SCREEN, RS-232, OR SERIAL
F21D		;
F21D	C9 03	CMP #03 ;SCREEN?
F21F	F0 12	BEQ L963 ;YES, DONE
F221	B0 14	BCS L961 ;SERIAL
F223	C9 02	CMF #02 ;RS-232?
F225	00 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	CPX #60 ;INPUT?
F22E	F0 03	BEQ L963 ;YES
F230	4C 0A F7	JMP L971 ;NO, NOT INPUT FILE
F233	85 99	L963 STA DFLTIN ;SET INPUT
F235	18	CLC ;GOOD RETURN
F236	60	RTS
F237		:

56 The Commodore 64 Kernal and Hardware Revealed

LOC	CODE	LINE
F237		;A SERIAL DEVICE MUST TALK
F237		;
F237	AA	L961 TAX ;SAVE DEVICE #
F238	20 09 ED	JSR L836 ;TALK
F23B	A5 B9	LDA SA ;SECOND?
F23D	10 06	BPL L962 ;YES, SEND IT
F23F	20 CC ED	JSR L970 ;NO, LET GO
F242	4C 48 F2	JMP L967
F245	20 C7 ED	L962 JSR L860 ;SEND TALK SA
F248	8A	L967 TXA
F249	24 90	BIT STATUS ;DID IT LISTEN?
F24B	10 E6	BPL L963 ;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		*****

```

LOC   CODE      LINE
F250
F250  20 0F F3   ; NCKOUT JSR L1000      ;FILE IN TABLE;
F253  F0 03      BEQ L969        ;YES
F255  4C 01 F7      JMP L1009      ;NO, FILE NOT OPEN
F258  20 1F F3   L969 JSR L1002      ;GET TABLE INFO
F25B  A5 BA      LDA FA
F25D  D0 03      BNE L979        ;NOT KEYBOARD
F25F  4C 0D F7   L972 JMP L965      ;KEYBOARD, NOT OUTPUT FILE
F262
F262      ;
F262      ;COULD BE SCREEN, SERIAL,
F262      ; CASSETTE, OR RS-232
F262      ;
F262  C9 03   L979  CMP #03        ;SCREEN?
F264  F0 0F      BEQ L977        ;YES, DONE
F266  B0 11      BCS L975        ;NO, SERIAL
F268  C9 02      CMP #02        ;RS-232?
F26A  D0 03      BNE L973        ;NO, MUST BE CASSETTE
F26C  4C E1 EF      JMP $EFE1      ;SET UP FOR RS-232
F26F
F26F      ;
F26F      ;CHECK FOR CASSETTE FILE TYPE
F26F      ;
F26F  A6 B9   L973  LDX SA
F271  E0 60      CPX #60        ;INPUT FILE?
F273  F0 EA      BEQ L972        ;YES, ERROR
F275  85 9A   L977  STA DFL10      ;SET OUTPUT
F277  18      CLC                ;GOOD RETURN
F278  60      RTS
F279
F279      ;
F279      ;SERIAL DEVICES
F279      ;
F279  AA      L975  TAX            ;SAVE DEVICE #
F27A  20 0C ED      JSR L966      ;LISTEN
F27D  A5 B9      LDA SA          ;AND SECOND?
F27F  10 05      BPL L976      ;YES
F281  20 BE ED      JSR L983      ;NO, RELEASE LINES
F284  D0 03      BNE L981      ;ALWAYS
F286  20 B9 ED   L976 JSR L871      ;SEND LISTEN SA
F289  8A      L981  TXA
F28A  24 90      BIT STATUS      ;DID IT LISTEN?
F28C  10 E7      BPL L977      ;YES, FINISH
F28E  4C 07 F7      JMP L1026      ;NO, DEVICE NOT PRESENT
F291      .END
F291      .LIB KSER4

```

CLOSE

Entry point: \$FFC3

Function: Close a logical file.

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

Output parameters: CARRY clear.

Registers used: .A, .Y, .X

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.

LUC	CODE	LINE
F291		;
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 L982 ;FOUND
F296	18	CLC ;ELSE RETURN
F297	60	RTS
F298	20 1F F3	L982 JSR L1002 ;GET FILE DATA
F298	8A	TXA ;SAVE TABLE INDEX
F29C	48	FHA
F29D	A5 8A	LDA FA ;CHECK DEVICE #
F29F	F0 50	BEQ L987 ;KEYBOARD, DONE
F2A1	C9 03	CMF #03 ;SCREEN?
F2A3	F0 4C	BEQ L987 ;YES, DONE
F2A5	B0 47	BCS L997 ;SERIAL
F2A7	C9 02	CMF #02 ;RS-232?
F2A9	D0 1D	BNE #F2C8 ;NO, MUST BE TAPE
F2AB		;
F2AB		;\$F2EE
F2EE		;CLOSE A SERIAL FILE
F2EE		;
F2EE	20 42 F6	L997 JSR L1001
F2F1		;
F2F1		;REMOVE FILE ENTRIES FROM TABLES
F2F1		;
F2F1	68	L987 PLA ;GET TABLE INDEX
F2F2	AA	L986 TAX
F2F3	C6 98	DEC LDTND
F2F5	E4 98	CPX LDTND ;IS IT AT END?
F2F7	F0 14	BEQ L989 ;YES, DONE
F2F9	A4 98	LDY LDTND ;NO, SHIFT LAST ENTRY
F2FB	B9 59 02	LDA LAT,Y ;INTO DELETED ENTRY'S
F2FE	9D 59 02	STA LAT,X ;POSITION
F301	B9 63 02	LDA FAT,Y
F304	9D 63 02	STA FAT,X
F307	B9 6D 02	LDA SAT,Y
F30A	9D 6D 02	STA SAT,X
F30D	18	L989 CLC ;GOOD EXIT
F30E	60	RTS
F30F		;FIND FILE ENTRY
F30F		;
F30F	A9 00	L1000 LDA #00
F311	85 90	STA STATUS
F313	8A	TXA
F314	A6 98	L957 LDX LDTND
F316	CA	L984 DEX
F317	30 15	BMI L960
F319	DD 59 02	CMF LAT,X
F31C	D0 F8	BNE L984
F31E	60	RTS
F31F		;
F31F		;FETCH TABLE ENTRIES
F31F		;
F31F	BD 59 02	L1002 LDA LAT,X
F322	85 B8	STA LA

LOC	CODE	LINE
F324	BD 63 02	LDA FAT,X
F327	85 BA	STA FA
F329	BD 6D 02	LDA SAT,X
F32C	85 B9	STA SA
F32E	60	L960 RTS
F32F		.END
F32F		.LIB KSER5

CLALL

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.

LOC	CODE	LINE
F32F		;
F32F		;*****
F32F		;* CLALL -- CLOSE ALL LOGICAL FILES.
F32F		;* DELETES ALL TABLE ENTRIES AND
F32F		;* RESTORES DEFAULT I/O CHANNELS AND
F32F		;* CLEARS SERIAL PORT DEVICES
F32F		;*****
F32F		;
F32F	A9 00	NCLALL LDA #\$00
F331	85 98	STA LDND ;FORGET ALL FILES

CLRCH

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 LDX #03
F335	E4 9A	CPX DFLT0 ;OUTPUT CHANNEL SERIAL?
F337	B0 03	BCS L1001 ;NO
F339	20 FE ED	JSR L1006 ;YES, UNLISTEN
F33C	E4 99	CPX DFLT0 ;INPUT CHANNEL SERIAL?
F33E	B0 03	BCS L1003 ;NO
F340	20 EF ED	JSR L863 ;YES, UNTALK
F343	86 9A	STX DFLT0 ;OUTPUT CHANNEL=3
F345	A9 00	LDA #000
F347	85 99	STA DFLT0 ;INPUT CHANNEL=0
F349	60	RTS
F34A		.END
F34A		.LIB KSER6

OPEN

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

Registers used: .A, .X, .Y

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.

LOC	CODE	LINE
F34A		;
F34A		;*****
F34A		;* OPEN -- OPEN A FILE.
F34A		;* CREATES AN ENTRY IN THE FILE
F34A		;* FILE TABLES CONSISTING OF LOGICAL
F34A		;* FILE NUMBER, DEVICE NUMBER, AND SEC
F34A		;* ADDRESS NUMBER.
F34A		;* ROUTINES SETLFS & SETNAM SHOULD
F34A		;* BE USED FIRST.
F34A		;*****
F34A		;
F34A	A6 B8	NOPEN LDX LA ;CHECK FILE #
F34C	D0 03	BNE L1005 ;NOT KEYBOARD
F34E	4C 0A F7	JMP L971 ;NOT INPUT FILE
F351	20 0F F3	L1005 JSR L1000 ;ALREADY OPEN?
F354	D0 03	BNE L1007 ;NO
F356	4C FE F6	JMP L1011 ;YES, FILE OPEN
F359	A6 98	L1007 LDX LDIND ;END OF TABLE?
F35B	E0 0A	CPX #0A
F35D	90 03	BCC L1008 ;NO
F35F	4C FB F6	JMP L1097 ;YES, TOO MANY FILES
F362	E6 98	L1008 INC LDTND ;NEW FILE
F364	A5 B8	LDA LA
F366	9D 59 02	STA LAT,X ;STORE FILE #
F369	A5 B9	LDA SA
F36B	09 60	ORA #60 ;MAKE SA SERIAL
F36D	85 B9	STA SA
F36F	9D 6D 02	STA SAT,X ;STORE SA
F372	A5 BA	LDA FA
F374	9D 63 02	STA FAT,X ;STORE DEVICE #
F377		;
F377		;PERFORM DEVICE SPECIFIC OPEN TASKS
F377		;
F377	F0 5A	BEQ L1030 ;KEYBOARD, DONE
F379	C9 03	CMF #03 ;SCREEN?
F37B	F0 56	BEQ L1030 ;YES, DONE
F37D	90 05	BCC \$F384 ;CASSETTE OR RS-232
F37F	20 D5 F3	JSR L1021 ;OPEN SERIAL
F382	90 4F	BCC L1030 ;ALWAYS, DONE
F384		;-----
F384		*=\$F3D3
F3D3	18	L1030 CLC
F3D4	60	L1012 RTS ;EXIT
F3D5		;
F3D5		;OPEN SERIAL
F3D5		;
F3D5	A5 B9	L1021 LDA SA
F3D7	30 FA	BMI L1030 ;NO SA, DONE
F3D9	A4 B7	LDY FNLEN
F3DB	F0 F6	BEQ L1030 ;NO FILENAME, DONE

62 The Commodore 64 Kernal and Hardware Revealed

LOC	CODE	LINE
F3D0	A9 00	LDA #000
F3D1	85 90	STA STATUS
F3E1	A5 BA	LDA FA
F3E3	20 0C ED	JSR L966 ;DEVICE LA TO LISTEN
F3E6	A5 B9	LDA SA
F3E8	09 F0	ORA #F0
F3EA	20 B9 ED	JSR L871
F3ED	A5 90	LDA STATUS ;DEVICE THERE?
F3EF	10 05	BPL L1014 ;YES
F3F1	68	PLA ;NO
F3F2	68	PLA
F3F3	4C 07 F7	JMP L1026 ;DEVICE NOT PRESENT
F3F6	A5 B7	L1014 LDA FNLEN
F3F8	F0 0C	BEQ L1033 ;NO NAME, DONE
F3FA		;
F3FA		;SEND FILE NAME
F3FA		;
F3FA	A0 00	LDY #000
F3FC	B1 88	L1031 LDA (FNADR),Y
F3FE	20 DD ED	JSR L861
F401	C8	INY
F402	C4 B7	CPY FNLEN
F404	D0 F6	BNE L1031
F406	4C 54 F6	L1033 JMP L999 ;UNLISTEN AND RETURN
F409		;
F409		.END
F409		.LIB KSER7

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 (≠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

Registers used: .A, .X, .Y

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.

LUC	CODE	LINE
F409		==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	STX MEMUSS+1 ;HI ALT START
F4A2	6C 30 03	JMP (ILOAD)
F4A5	85 93	NLOAD STA VERCK ;STORE VERIFY FLAG
F4A7	A9 00	LDA #00
F4A9	85 90	STA STATUS
F4AB	A5 BA	LDA FA ;CHECK DEVICE #
F4AD	D0 03	BNE L1046
F4AF	4C 13 F7	L1241 JMP L1049 ;KEYBOARD, BAD DEVICE
F4B2	C9 03	L1046 CMP #03 ;SCREEN?
F4B4	F0 F9	BEQ L1241 ;YES
F4B6	90 7B	BCC \$F533 ;TAPE
F4B8		;
F4B8		;LOAD FROM SERIAL BUS DEVICES
F4B8		;
F4B8	A4 B7	LDY FNLEN ;MUST HAVE FILENAME
F4BA	D0 03	BNE L1045 ;O.K.
F4BC	4C 10 F7	JMP L974 ;MISSING FILENAME
F4BF	A6 B9	L1045 LDX SA
F4C1	20 AF F5	JSR L1062 ;'SEARCHING'
F4C4	A9 60	LDA #60 ;SPECIAL LOAD COMMAND
F4C6	85 B9	STA SA
F4C8	20 05 F3	JSR L1021 ;OPEN FILE

64 The Commodore 64 Kernal and Hardware Revealed

LOC	CODE	LINE	
F4CB	A5 BA		LDA FA
F4CD	20 09 ED		JSR L836 ;TALK, ESTABLISH CHANNEL
F4D0	A5 B9		LDA SA
F4D2	20 C7 ED		JSR L860 ;TELL IT TO LOAD
F4D5	20 13 EE		JSR L865 ;GET FIRST BYTE
F4D8	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
F4E8	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 #FFD ;MASK OFF TIMEOUT
F4F5	25 90		AND STATUS
F4F7	85 90		STA STATUS
F4F9	20 E1 FF		JSR \$FFE1 ;STOP KEY?
F4FC	D0 03		BNE L1055 ;NO
F4FE	4C 33 F6		JMP 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 #00
F512	D1 AE		CMP (EAL),Y ;VERIFY IT
F514	F0 08		BEQ L1056 ;O.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	E6 AE	L1056	INC EAL ;INCREMENT STORE ADDR
F520	D0 02		BNE L1057
F522	E6 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
F5A9	18	L1067	CLC ;GOOD EXIT
F5AA			;
F5AA			;SET UP END ADDRESS
F5AA			;
F5AA	A6 AE		LDX EAL
F5AC	A4 AF		LDY EAH
F5AE	60	L1059	RTS
F5AF			;
F5AF			;PRINT 'SEARCHING [FOR NAME]'
F5AF			;
F5AF	A5 9D	L1062	LDA MSGFLG ;PRINT IT?
F5B1	10 1E		BPL L1071 ;NO
F5B3	A0 0C		LDY #0C ;'SEARCHING'
F5B5	20 2F F1		JSR \$F12F
F5B8	A5 B7		LDA FNLEN

LOC	CODE	LINE		
F5BA	F0 15		BEQ L1071	
F5BC	A0 17		LDY #17	; 'FOR'
F5BE	20 2F F1		JSR \$F12F	
F5C1			;PRINT FILE NAME	
F5C1				
F5C1	A4 B7	L1022	LDY FNLEN	;NAME?
F5C3	F0 0C		BEQ L1071	;NO, DONE
F5C5	A0 00		LDY #00	
F5C7	B1 BB	L1091	LDA (FNADR),Y	
F5C9	20 D2 FF		JSR \$FFD2	
F5CC	C8		INY	
F5CD	C4 B7		CPY FNLEN	
F5CF	D0 F6		BNE L1091	
F501	60	L1071	RTS	
F5D2			;PRINT LOADING/VERIFYING	
F5D2				
F5D2	A0 49	L1070	LDY #49	;ASSUME 'LOADING'
F5D4	A5 93		LDA VERCK	;CHECK FLAG
F5D6	F0 02		BEQ L1052	;YES, LOADING
F5D8	A0 59		LDY #59	; 'VERIFYING'
F5DA	4C 2B F1	L1052	JMP \$F12B	
F5DD		.END		
F5DD			.LIB KSER0	

SAVE

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

Registers used: .A, .X, .Y

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

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	
F5DD			;
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	86 AE	L1072	STX EAL
F5DF	84 AF		STY EAH
F5E1	AA		TAX ;SET UP START
F5E2	B3 00		LDA \$00,X
F5E4	85 C1		STA STAL
F5E6	B5 01		LDA \$01,X
F5E8	85 C2		STA STAH
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		BEQ L1242 ;SCREEN, BAD DEVICE
F5F8	90 5F		BCC \$F659 ;NO, TAPE
F5FA	A9 61		LDA #\$61 ;YES
F5FC	85 B9		STA SA
F5FE	A4 B7		LDY FNLEN
F600	D0 03		BNE L1074
F602	4C 10 F7		JMP L974 ;MISSING FILE NAME
F605	20 D5 F3	L1074	JSR L1021 ;OPEN
F608	20 8F F6		JSR L1087 ;'SAVING'
F60B	A5 BA		LDA FA
F60D	20 0C ED		JSR L966 ;LISTEN
F610	A5 B9		LDA SA
F612	20 B9 ED		JSR L871 ;LISTEN SA
F615	A0 00		LDY #\$00
F617	20 BE FB		JSR \$FB8E
F61A	A5 AC		LDA SAL
F61C	20 DD ED		JSR L861
F61F	A5 AD		LDA SAH
F621	20 DD ED		JSR L861
F624	20 D1 FC	L1077	JSR \$FCD1 ;COMPARE START TO END
F627	B0 16		BCS L1082 ;HAVE REACHED END
F629	B1 AC		LDA (SAL),Y
F62B	20 DD ED		JSR L861
F62E	20 E1 FF		JSR \$FFE1 ;STOP KEY?
F631	D0 07		BNE L1054 ;NO
F633	20 42 F6	L1084	JSR L1081 ;YES, CLOSE
F636	A9 00		LDA #\$00
F638	38		SEC
F639	60		RTS
F63A			;
F63A	20 DB FC	L1054	JSR \$FCDB ;INCREMENT CURRENT ADDR
F63D	D0 E5		BNE L1077

```

LOC    CODE    LINE
F63F  20 FE ED  L1082 JSR L1006      ;UNLISTEN
F642  24 B9      L1081 BIT SA
F644  30 11      BMI L1034
F646  A5 BA      LDA FA
F648  20 0C ED      JSR L966      ;LISTEN
F64B  A5 B9      LDA SA
F64D  29 EF      AND #EF
F64F  07 E0      ORA #E0
F651  20 B9 ED      JSR L871      ;LISTEN SA
F654  20 FE ED  L999 JSR L1006      ;UNLISTEN
F657  18          L1034 CLC        ;GOOD EXIT
F658  60          RTS
F659          ;-----
F659          *=$F6BE
F68E  60          L1090 RTS
F68F          ;
F68F          ;PRINT 'SAVING [FILE NAME]'
F68F          ;
F68F  A5 9D      L1087 LDA MSGFLG    ;PRINT IT?
F691  10 FB      BPL L1090      ;NO
F693  A0 51      LDY #51        ;'SAVING'
F695  20 2F F1      JSR $F12F
F698  4C C1 F5      JMP L1022    ;SEND FILENAME
F69B          .END
F69B          .LIB KSER9

```

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

Registers used: .A, .X, .Y

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.

```

LOC   CODE          LINE

F6FB      *=$F6FB
F6FB      ;*****
F6FB      ;* ERROR HANDLER.
F6FB      ;* PRINTS KERNAL ERROR MESSAGE IF BIT 6
F6FB      ;* OF MSGFLG IS SET. RETURNS WITH ERROR
F6FB      ;* # IN .A AND CARRY SET.
F6FB      ;*****
F6FB      ;
F6FB A9 01          L1097 LDA #$01          ;TOO MANY FILES
F6FD 2C              .BYT $2C
F6FE A9 02          L1011 LDA #$02          ;FILE OPEN
F700 2C              .BYT $2C
F701 A9 03          L1009 LDA #$03          ;FILE NOT OPEN
F703 2C              .BYT $2C
F704 A9 04          L959 LDA #$04           ;FILE NOT FOUND
F706 2C              .BYT $2C
F707 A9 05          L1026 LDA #$05          ;DEVICE NOT PRESENT
F709 2C              .BYT $2C
F70A A9 06          L971 LDA #$06          ;NOT INPUT FILE
F70C 2C              .BYT $2C
F70D A9 07          L965 LDA #$07          ;NOT OUTPUT FILE
F70F 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 20 CC FF       JSR $FFCC             ;RESTORE I/O
F719 A0 00           LDY #$00
F71B 24 9D           BIT MSGFLG          ;PRINT ERROR?
F71D 50 0A           BVC L1018           ;NO
F71F 20 2F F1       JSR $F12F           ;PRINT 'I/O ERROR #'
F722 68              PLA
F723 48              PHA
F724 09 30           ORA #$30            ;MAKE ERROR # ASCII
F726 20 D2 FF       JSR $FFD2            ;PRINT IT
F729 68              L1018 PLA
F72A 38              SEC
F72B 60              RTS
F72C              ;-----
F72C              .END
F72C              .END

```

Symbol table

SYMBOL	VALUE						
AUTODN	0292	BAD	0100	BASZPT	00FF	BAUDOF	0299
BDF	0002	BDFH	0004	BITCI	00A8	BITNUM	0298
BITIS	00B4	BLF	0001	BLNCT	00CD	BLNON	00CF
BLNSW	00CC	BLUE	0006	BSOUR	0095	BSOUR1	00A4
BUF	0200	BUFFT	00A6	BUSFZ	00C0	C3FO	0094
CAS1	00C0	CBINV	0316	CBMCOL	D800	CBMSCN	0400
CINV	0314	CKERR	0020	CMFO	00B0	CNTDN	00A5
COLM	DC00	COLOR	0286	COUNT	00A5	CR	000D
CRSW	00D0	D1CRA	DC0E	D1CRB	DC0F	D1DDRA	DC02
D1DDRB	DC03	D1DFA	DC00	D1DPB	DC01	D1ICR	DC0D
D1I0DB	DC0C	D1TAH	DC05	D1TAL	DC04	D1TBH	DC07
D1TBL	DC06	D1TOD1	DC08	D1TOD2	DC09	D1TOD3	DC0A
D1TOD4	DC0B	D2CRA	DD0E	D2CRB	DD0F	D2DDRA	DD02
D2DDRB	DD03	D2DFA	DD00	D2DFB	DD01	D2ICR	DD0D
D2I0DB	DD0C	D2TAH	DD05	D2TAL	DD04	D2TBH	DD07
D2TBL	DD06	D2TOD1	DD08	D2TOD2	DD09	D2TOD3	DD0A
D2TOD4	DD0B	DATA	00D7	DELAY	028C	DFLIN	0099
DFLTO	009A	DIFF	00B5	DFSW	009C	EAH	00AF
EAL	00AE	EOT	0005	FA	00BA	FAT	0263
FIRT	00A4	FNADR	00BB	FNLEN	00B7	FREKZF	00FB
FSBLK	00BE	GDBLN	00CE	GDCOL	0287	HIBASE	0288

SYMBOL VALUE

IRASIN	0324	IRSOUT	0326	ICHKIN	031E	ICKOUT	0320
ICLALL	032C	ICLOSE	031C	ICLRCH	0322	IGETIN	032A
ILUAD	0330	INBIT	00A7	INDX	00C8	INSRT	00D8
IOPEN	031A	IRQTMP	029F	ISAVE	0332	ISTOP	0328
KEYD	0277	KEYLOG	028F	KEYTAB	00F5	KOUNT	028B
L1000	F30F	L1001	F33C	L1002	F31F	L1003	F343
L1004	EE03	L1005	F351	L1006	EDFE	L1007	F359
L1008	F362	L1009	F701	L1011	F6FE	L1012	F3D4
L1014	F3F6	L1018	F729	L1021	F3D5	L1022	F5C1
L1026	F707	L1030	F3D3	L1031	F3FC	L1033	F406
L1034	F657	L1045	F48F	L1046	F4B2	L1048	F4F0
L1049	F713	L1051	F4F3	L1052	F5DA	L1053	F51C
L1054	F63A	L1055	F501	L1056	F51E	L1057	F524
L1058	F530	L1059	F5AE	L1062	F5AF	L1067	F5A9
L1070	F5D2	L1071	F5D1	L1072	F5DD	L1074	F605
L1075	F5F4	L1077	F624	L1081	F642	L1082	F63F
L1084	F633	L1087	F68F	L1090	F68E	L1091	F5C7
L1097	F6FB	L1241	F4AF	L1242	F5F1	L836	ED09
L839	ED2E	L840	ED50	L841	EEA0	L842	ED36
L843	EE8E	L844	EE97	L845	EEB6	L846	EEB3
L847	EDB0	L848	ED66	L849	ED55	L850	ED5A
L851	ED7A	L852	EDB2	L853	ED7D	L854	EEA9
L855	ED9F	L856	EDAD	L858	EE06	L859	ED40
L860	EDC7	L861	EDDD	L862	EDEB	L863	EDEF
L864	ED20	L865	EE13	L866	EE20	L867	EE47
L868	EE3E	L869	EE5A	L870	EE36	L871	EDB9
L872	EE30	L873	EE67	L874	EE80	L875	EE85
L876	EE09	L924	F14A	L926	F14E	L927	F166
L928	F173	L932	F1B1	L933	F1D5	L939	F1AD
L941	F1B5	L943	EE1B	L944	F155	L945	F1B4
L946	F1B3	L949	EDE6	L950	F216	L957	F314
L958	F22A	L959	F704	L960	F32E	L961	F237
L962	F245	L963	F233	L965	F70D	L966	ED0C
L967	F248	L968	EDD6	L969	F258	L970	EDCC
L971	F70A	L972	F25F	L973	F26F	L974	F710
L975	F279	L976	F286	L977	F275	L979	F262
L980	ED11	L981	F289	L982	F298	L983	EDBE
L984	F316	L986	F2F2	L987	F2F1	L989	F30D
L990	F49E	L997	F2EE	L999	F654	LA	00B8
LAT	0259	LBERR	0008	LDIR1	00D9	LDTHD	0098
LINTMP	00F2	LLEN	0028	LLEN2	0050	LNMX	00D5
LSTP	00CA	LSTSHF	028E	LSTX	00C5	LSPF	00C9
LTLUE	000E	M26AJB	0295	M26CDR	0294	M26CTR	0293
MAXCHR	0050	MEMSIZ	0283	MEMSTR	0281	MEMUSS	00C3
MODE	0291	MSGFLG	009D	MYCH	00BF	NBASIN	F157
NBSOUT	F1CA	NCHKIN	F20E	NCKOUT	F250	NCLALL	F32F
NCLOSE	F291	NCLRCH	F333	NDX	00C6	NGETIN	F13E
NLINES	0019	NLOAD	F4A5	NMINV	0318	NOPEN	F34A
NSAVE	F5ED	NWRAP	0002	NXTBIT	00B5	UCHAR	00BD
PCNTR	00A3	PLF	0003	PNT	00D1	PNTR	00D3
PRP	00B6	PRTY	009B	PTR1	009E	PTR2	009F
QTSW	00D4	R2D2	00A3	RDFLG	00AA	RER	00AB
REZ	00A9	RIBUF	00F7	RIDATA	00AA	RIDBE	029B
KIDBS	029C	RINONE	00A9	RIFRTY	00AB	ROBUF	00F9
RODATA	00B6	RODBE	029E	RODBS	029D	ROPRTY	00BD
ROWS	DC01	RFTFLG	028A	RSSTAT	0297	RVS	00C7
SA	00B9	SAH	00AD	SAL	00AC	SAT	026D
SBERR	0004	SFDX	00CB	SHCNH	00AB	SHCNL	00A7
SHFLAG	028D	SIDREG	D400	SNSW1	00B4	SPERR	0010
STAH	00C2	STAL	00C1	STATUS	0090	STKEY	0091
SVXT	0092	SYNO	0096	11	009E	T2	009F
TAF1	00B2	TBLX	00D6	TBUFR	033C	TEMP	00E1
TIME	00A0	TIMOUT	0285	TMP2	00C3	TMPC	009F
TMPO	00C1	USER	00F3	USRCMD	032E	VERCK	0093
VICREG	D000	XMAX	0289	XSAV	0097		

3.4 RS232 serial communications

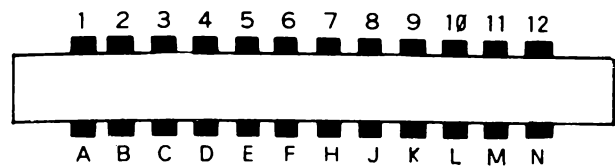
The CBM 64 is able to communicate with peripheral devices, known as an RS232 I/O port. The name RS232 simply refers to an industry standard form of serial communication for computing devices. A serial I/O port can consist of as few as three lines, an output or transmit line, an input or receive line 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:

CIA line #	RS232 pin #	CIA pin #	Abv	EIA	In/ Out	Modes	Function
GND	1	A	GND	AA	---	1,2	Protective ground
FLAG	3	B	SIN	BB	In	1,2	Received data
PB0	3	C	SIN	BB	In	1,2	Connected to FLAG
PB1	4	D	RTS	CA	Out	2	Request to send
PB2	20	E	DTR	CD	Out	2	Data terminal ready
PB3	18	F	RI	CE	In	3	Ring indicator
PB4	8	H	DCD	CF	In	2	Received line signal
PB5	Not assigned						
PB6	5	K	CTS	CB	In	2	Clear to send
PB7	6	L	DSR	CC	In	2	Data set ready
PA2	2	M	SOUT	BA	Out	1,2	Transmitted data
GND	7	N	GND	AB	---	2,3	Signal ground

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



PIN	TYPE	RS232 FUNCTION
A		
B	FLAG	
C	PB0	— Sin
D	PB1	— RTS
E	PB2	— DTR
F	PB3	— RI
H	PB4	— DCD
J		
K	PB6	— CTS
L	PB7	— DSR
M	PA2	— Sout
N	GND	— GND

PIN		
	1	Protective Ground
	2	Transmitted Data
	3	Received Data
	4	Request To Send
	5	Clear To Send
	6	Data Set Ready
	7	Signal Ground
	8	Carrier Detect
	9	(not used)
	10	"
	11	"
	12	"
	13	"
	14	"
	15	"
	16	"
	17	"
	18	"
	19	"
	20	Data Terminal Ready
	21	(not used)
	22	"
	23	"
	24	"
	25	"

Q14	10	AA
Q15	20	BA
Q16	30	BB
Q17	40	CA
Q18	50	CB
Q19	60	CC
Q20	70	AB
Q21	80	CF
Q22	90	
Q23	100	
Q24	110	
Q25	120	
	130	

Fig. 3.4. CBM 64 RS232 connector, pin allocations and E/A line coding.

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

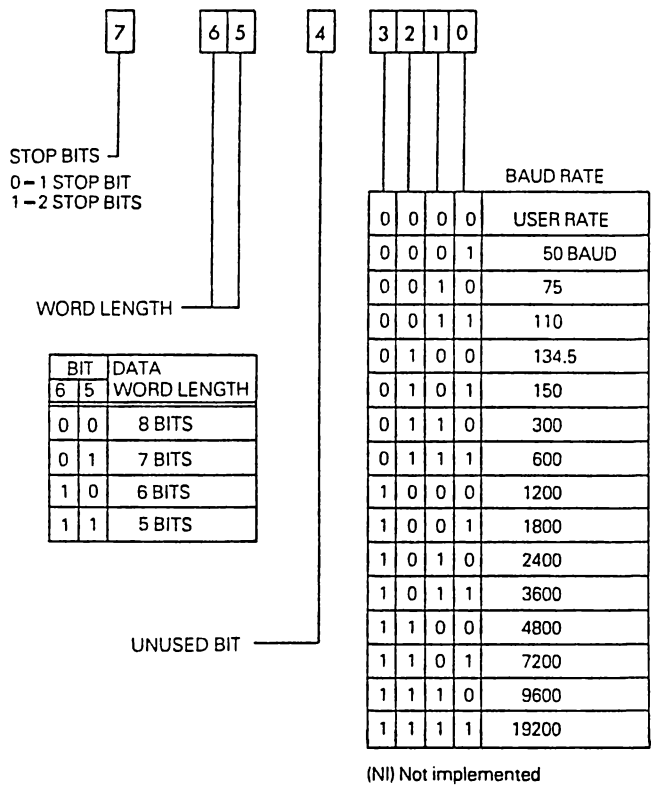


Fig. 3.5. Function of bits in the CBM 64 RS232 control register.

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 30 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 \$0294 Decimal 660

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

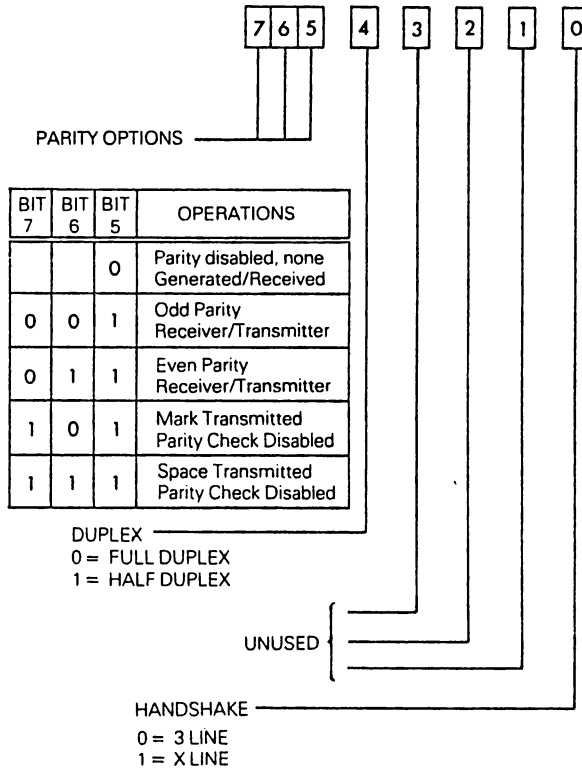


Fig. 3.6. Function of bits in the CBM 64 RS232 command register.

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 \$0297 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:

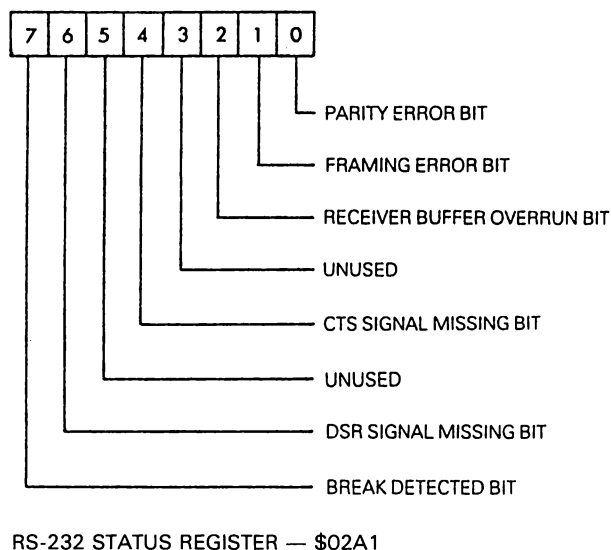


Fig. 3.7. Function of bits in the CBM 64 RS232 status register.

\$A7	– receiver bit storage
\$A8	– receiver bit count
\$A9	– receiver flag start bit check
\$AA	– receiver byte buffer
\$AB	– receiver parity storage
\$B4	– transmitter bit count
\$B5	– transmitter next bit
\$B6	– transmitter byte buffer
\$BD	– transmitter parity storage
\$F7–\$F8	– input buffer pointer
\$F9–\$FA	– output buffer pointer
\$0293	– RS232 control register
\$0294	– RS232 command register
\$0297	– RS232 status register
\$0298	– number of bits to send/receive
\$0299–	
\$029A	– baud rate
\$029B	– input buffer index to end
\$029C	– input buffer index to start
\$029D	– output buffer index to start
\$029E	– output buffer index to end

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

\$EF13	- set up to send next byte
\$EF2E	- set errors
\$EF4A	- calculate number of bits to be sent
\$EF59	- NMI routine to collect data into bytes
\$EF63	- calculate parity
\$EF69	- shift data bit in
\$EF6E	- have stop bit so store in buffer
\$EF7E	- enable to receive a byte
\$EF90	- receiver start bit check
\$EF97	- put data in buffer (at parity time)
\$EFB3	- parity checking
\$EFBC	- check calculated parity
\$EFC7	- errors reported
\$EFE1	- output a file over RS232
\$EFE9	- check for DSR and RTS
\$EFF2	- check for active input
\$EFF9	- wait for CTS to be off
\$EFFE	- turn on RTS
\$F006	- wait for CTS to go on
\$F014	- buffer handler to output a character
\$F028	- set up if necessary to output
\$F04D	- input a file over RS232
\$F059	- check for DSR and not RTS
\$F062	- wait for active output to be done
\$F068	- turn off RTS
\$F070	- wait for DCD to go high
\$F077	- enable FLAG for RS232 input
\$F07D	- if not 3 line half then see if we need to turn on FLAG
\$F086	- input a character buffer handler
\$F09B	- receiver always runs
\$F0A4	- protect serial/cassette from RS232 NMIs

3.5 Using the RS232 port

3.5.1 Opening an RS232 channel

Basic syntax: OPEN lf,2,0,“(control register) (command register)”

The syntax coding is as follows:

lf – normal logical file ID (1–255). If lf>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: \$FFC0

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 #lf,(string variable)

lf – 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 lf 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 lf

lf – 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 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

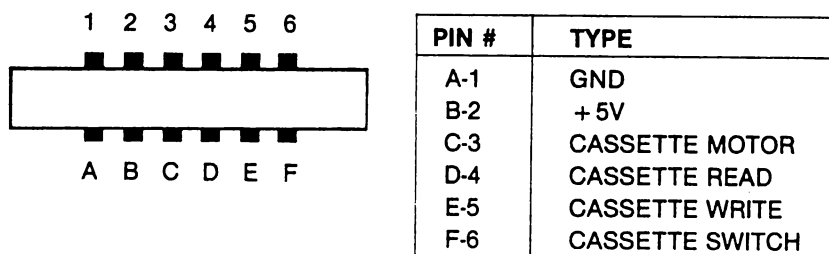


Fig. 4.1. The allocation and function of pins on the cassette connector.

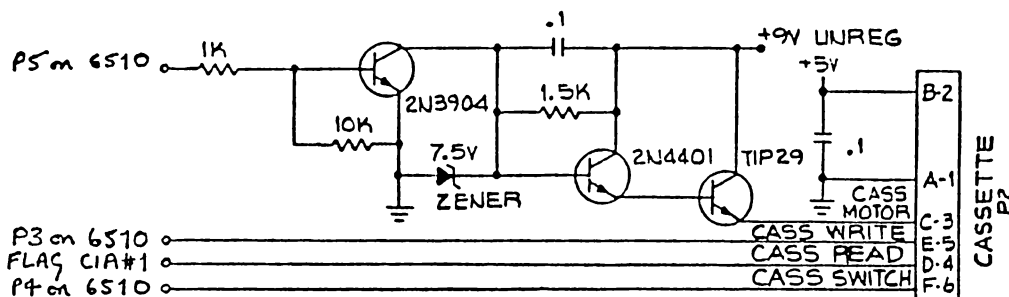


Fig. 4.2. The cassette circuit and its connection to the 6522 chips.

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 1 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.

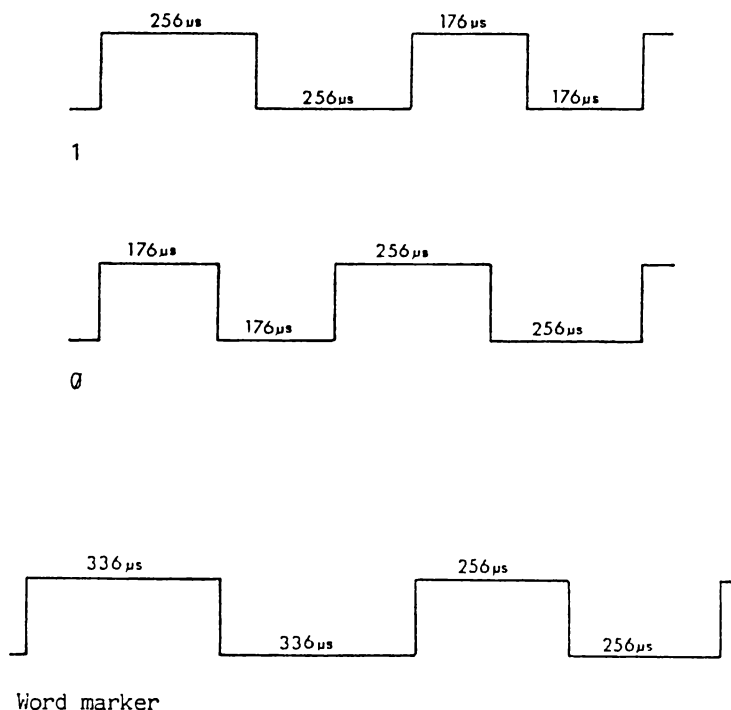


Fig. 4.3. Output waveforms to the cassette recorder.

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:

LOC	CODE	LINE			
DD10			== \$F0A4		
F0A4			;		
F0A4			;PROTECT SERIAL/CASSETTE FROM		
F0A4			;RS-232 NMI'S		
F0A4			;		
F0A4	48	L921	PHA	;SAVE A	
F0A5	AD A1 02		LDA \$02A1	;RS232 ENABLES ?	
F0A8	F0 11		BEQ L923	;NO	
F0AA	AD A1 02	L838	LDA \$02A1		
F0AD	29 03		AND #\$03		
F0AF	D0 F9		BNE L838		
F0B1	A9 10		LDA #\$10	;DISABLE FLAG	
F0B3	8D 0D DD		STA D2ICK		
F0B6	A9 00		LDA #\$00		
F0B8	8D A1 02		STA \$02A1		
F0BB	68	L923	PLA	;ALL DONE	
F0BC	60		RTS		
F0BD			;		

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:

LOC	CODE	LINE
F0BD		;ERROR MESSAGES
F0BD		;
F0BD	0D	MS1 .BYT \$D,'I/O ERROR ', \$A3
F0BE	49 2F	
F0CB	A3	
F0C9	0D	MS5 .BYT \$D,'SEARCHING', \$A0
F0CA	53 45	
F0D3	A0	
F0D4	46 4F 52	MS6 .BYT 'FOR', \$A0
F0D7	A0	
F0D8	0D	MS7 .BYT \$D,'PRESS PLAY ON TAP', \$C5
F0D9	50 52	
F0EA	C5	
F0EB	50 52	MS8 .BYT 'PRESS RECORD & PLAY ON TAP', \$C5
F105	C5	
F106	0D	MS10 .BYT \$D,'LOADIN', \$C7
F107	4C 4F	
F10D	C7	
F10E	0D	MS11 .BYT \$D,'SAVING', \$A0
F10F	53 41	
F115	A0	
F116	0D	MS21 .BYT \$D,'VERIFYIN', \$C7
F117	56 45	
F11F	C7	
F120	0D	MS17 .BYT \$D,'FOUND', \$A0
F121	46 4F	
F126	A0	
F127	0D	MS18 .BYT \$D,'OK', \$8D
F128	4F 4B	
F12A	8D	
F12B		;
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	89 BD F0	L1073 LDA MS1,Y
F132	08	PHP
F133	29 7F	AND #\$7F
F135	20 02 FF	JSR \$FFD2
F138	C8	INY
F139	28	PLP
F13A	10 F3	BPL L1073
F13C	18	L925 CLC
F13D	60	RTS
F13E		.END
F13E		.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, ≠ 0 then verify

Output parameters:

.X – return high LOAD address hi

.Y – return high LOAD address lo

Routine source code:

```

LOC    CODE        LINE

F13E                      **=$F49E
F49E                      ;
F49E                      ;*****
F49E                      ;* LOAD RAM FUNCTION.
F49E                      ;*   LOADS FROM CASSETTE OR SERIAL BUS
F49E                      ;*   DEVICES >=4 TO 31 AS DETERMINED BY
F49E                      ;*   CONTENTS OF VARIABLE FA. VERIFY FLAG
F49E                      ;*   IN .A.
F49E                      ;*   ALT LOAD IF SA=0, NORMAL SA=1
F49E                      ;*   .X, .Y LOAD ADDRESS IF SA=0
F49E                      ;*   .A=0 PERFORMS LOAD, <>0 IS VERIFY.
F49E                      ;*   HIGH LOAD RETURN IN .X,.Y
F49E                      ;*   USE SETLFS & SETNAM BEFORE THIS ROUTINE
F49E                      ;*****
F49E                      ;
F49E  86 C3          L990  STX MEMUSS      ;LO ALT START
F4A0  84 C4          STY MEMUSS+1        ;HI ALT START
F4A2  6C 30 03      JMP (ILOAD)
F4A3  83 93          NLOAD STA VERECK      ;STORE VERIFY FLAG
F4A7  A9 00          LDA #000
F4A9  85 90          STA STATUS
F4AB  A5 BA          LDA FA              ;CHECK DEVICE #
F4AD  D0 03          BNE L1046
F4AF  4C 13 F7      L1241 JMP L1049        ;KEYBOARD, BAD DEVICE
F4B2  C9 03          L1046 CNP #03        ;SCREEN?
F4B4  F0 F9          BEQ L1241           ;YES
F4B6  90 7B          BCC L1050           ;TAPE
F4B8                      ;-----
F4B8                      ;
F4B8                      **=$F530
F530  4C 04 F7      L1058 JMP L959        ;FILE NOT FOUND
F533                      ;
F533                      ;***** LOAD FROM TAPE
F533                      ;

```

LOC	CODE	LINE		
F533	4A	L1050	LSR A	;TAPE?
F534	B0 03		BCS L1047	;YES
F536	4C 13 F7		JMP L1049	;NO, BAD DEVICE
F539	20 D0 F7	L1047	JSR L1104	;SET TAPE POINTERS
F53C	B0 03		BCS L1060	
F53E	4C 13 F7		JMP L1049	;DEALLOCATED
F541	20 17 F8	L1060	JSR L938	; 'PRESS PLAY ON TAPE'
F544	B0 68		BCS L1059	;STOP KEY?
F546	20 AF F5		JSR L1062	; 'SEARCHING'
F549	A5 B7	L1061	LDA FNLEN	;NAME?
F54B	F0 09		BEQ L1066	;NO, LOAD FIRST PROG
F54D	20 EA F7		JSR L1108	;YES, FIND A FILE
F550	90 0B		BCC L1063	;FOUND
F552	F0 5A		BEQ L1059	;STOP KEY
F554	B0 DA		BCS L1058	;NO, END OF TAPE
F556	20 2C F7	L1066	JSR L1098	;FIND ANY HEADER
F559	F0 33		BEQ L1059	;STOP KEY
F55B	B0 D3		BCS L1058	;NO HEADER
F55D	A5 90	L1063	LDA STATUS	
F55F	29 10		AND %SPERR	;MUST HAVE GOT HEADER RIGHT
F561	38		SEC	
F562	D0 4A		BNE L1059	;IS BAD
F564	E0 01		CPX %BLF	;MOVEABLE?
F566	F0 11		BEQ L1068	;YES
F568	E0 03		CPX %PLF	;PROGRAM
F56A	D0 DD		BNE L1061	;NO, TRY FOR NEXT
F56C	A0 01	L1064	LDY %01	;FIXED LOAD
F56E	B1 B2		LDA (TAPE1),Y	;ADDRESS IN BUFFER
F570	83 C3		STA MEMUSS	;IS LOAD ADDRESS
F572	C8		INY	
F573	B1 B2		LDA (TAPE1),Y	
F575	85 C4		STA MEMUSS+1	
F577	B0 04		BCS L1065	
F579	A5 B9	L1068	LDA SA	;MONITOR LOAD?
F57B	D0 EF		BNE L1064	;YES, FIXED TYPE
F57D	A0 03	L1065	LDY %03	;TAPEA-TAPESTA
F57F	B1 B2		LDA (TAPE1),Y	
F581	A0 01		LDY %01	
F583	F1 B2		SBC (TAPE1),Y	
F585	AA		TAX	;LO TO .X
F586	A0 04		LDY %04	
F588	B1 B2		LDA (TAPE1),Y	
F58A	A0 02		LDY %02	
F58C	F1 B2		SBC (TAPE1),Y	
F58E	A8		TAY	;HI TO .Y
F58F	18		CLC	;EA=STA+(TAPEA-TAPESTA)
F590	8A		TXA	
F591	65 C3		ADC MEMUSS	
F593	85 AE		STA EAL	
F595	98		TYA	
F596	65 C4		ADC MEMUSS+1	
F598	85 AF		STA EAH	
F59A	A5 C3		LDA MEMUSS	;SET UP START ADDRESS
F59C	85 C1		STA STAL	
F59E	A5 C4		LDA MEMUSS+1	
F5A0	85 C2		STA STAH	
F5A2	20 D2 F5		JSR L1070	; 'LOADING'
F5A5	20 4A F8		JSR L940	;LOAD TAPE BLOCK
F5A8	24		.BYT \$24	;SKIP NEXT COMMAND
F5A9	18	L1067	CLC	;GOOD EXIT
F5AA				
F5AA				;SET UP END ADDRESS
F5AA				
F5AA	A6 AE		LDX EAL	
F5AC	A4 AF		LDY EAH	
F5AE	60	L1059	RTS	
F5AF				

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    A5 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
F5B8    A5 B7              LDA FNLEN
F5BA    F9 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 BB              L1091    LDA (FNADR),Y
F5C9    20 02 FF           JSR $FFD2
F5CC    C8                 INY
F5CD    C4 B7              CPY FNLEN
F5CF    D0 F6              BNE L1091
F5D1    60                 L1071    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 KTAPE2

```

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		;*****
F5DD		;* SAVE MEMORY FUNCTION.
F5DD		;* SAVES TO CASSEITE 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	86 AE	L1072 STX EAL ;STORE END ADDRESS
F5DF	84 AF	STY EAH
F5E1	AA	TAX ;SET UP START
F5E2	85 00	LDA \$00,X
F5E4	85 C1	STA STAL
F5E6	85 01	LDA \$01,X
F5E8	85 C2	STA STAH
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	BEQ L1242 ;SCREEN, BAD DEVICE
F5F8	90 5F	BCC L1085 ;NO, TAPE
F5FA		;------
F5FA		*=\$F659
F659		;
F659		;***** TAPE SAVE
F659		;
F659	4A	L1085 LSR A ;RS-232?
F65A	80 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	80 25	BCS L1090 ;STOP KEY
F669	20 8F F6	JSR L1087 ;'SAVING'

90 The Commodore 64 Kernal and Hardware Revealed

LOC	CODE	LINE		
F66C	A2 03		LDX #PLF	;DECIDE TYPE TO SAVE
F66E	A5 B9		LDA SA	;1-PLF, 0-BLF
F670	29 01		AND #\$01	
F672	D0 02		BNE L1086	
F674	A2 01		LDX #BLF	
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	B0 0D		BCS L1090	;STOP KEY
F681	A5 B9		LDA SA	
F683	29 02		AND #\$02	;WRITE END OF TAPE?
F685	F0 06		BEQ L1088	;NO
F687	A9 05		LDA #\$05	
F689	20 6A F7		JSR L1099	;WRITE END TABLE BLOCKS
F68C	24		LDY \$24	;SKIP COMMAND
F68D	18	L1088	CLC	
F68E	60	L1090	RTS	

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		
F68F				;
F68F				;PRINT 'SAVING [FILENAME]'
F68F				;
F68F	A5 9D	L1087	LDA MSGFLG	;PRINT IT?
F691	10 FB		BPL L1090	;NO
F693	A0 51		LDY #MS11-MS1	; 'SAVING'
F695	20 2F F1		JSR L1073	
F698	4C C1 F5		JMP L1022	;SEND FILENAME
F69B		.END		
F69B			.LIB KTAPE3	

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		;
F6ED	A5 91	NSTOP LDA STKEY ;VALUE OF LAST ROW
F6EF	C9 7F	CMP #\$7F ;STOP KEY POSITION
F6F1	D0 07	BNE L1243 ;NOT DOWN
F6F3	08	PHP
F6F4	20 CC FF	JSR \$FFCC ;CLEAR CHANNELS
F6F7	85 C6	STA NDX ;CLEAR KEY QUEUE
F6F9	28	PLP
F6FA	60	L1243 RTS

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.

92 The Commodore 64 Kernal and Hardware Revealed

Input parameters: None

Output parameters: None

Routine source code:

LOC	CODE	LINE
F6FB		;*****
F6FB		;* ERROR HANDLER.
F6FB		;* PRINTS KERNAL ERROR MESSAGE IF BIT 6
F6FB		;* OF MSGFLG IS SET. RETURNS WITH ERROR
F6FB		;* # IN .A AND CARRY SET.
F6FB		;*****
F6FB		;
F6FB	A9 01	L1097 LDA #\$01 ;TOO MANY FILES
F6FD	2C	.BYT \$2C
F6FE	A9 02	L1011 LDA #\$02 ;FILE OPEN
F700	2C	.BYT \$2C
F701	A9 03	L1009 LDA #\$03 ;FILE NOT OPEN
F703	2C	.BYT \$2C
F704	A9 04	L959 LDA #\$04 ;FILE NOT FOUND
F706	2C	.BYT \$2C
F707	A9 05	L1026 LDA #\$05 ;DEVICE NOT PRESENT
F709	2C	.BYT \$2C
F70A	A9 06	L971 LDA #\$06 ;NOT INPUT FILE
F70C	2C	.BYT \$2C
F70D	A9 07	L965 LDA #\$07 ;NOT OUTPUT FILE
F70F	2C	.BYT \$2C
F710	A9 08	L974 LDA #\$08 ;MISSING FILENAME
F712	2C	.BYT \$2C
F713	A9 09	L1049 LDA #\$09 ;BAD DEVICE #
F715	48	PHA ;ERROR # ON STACK
F716	20 CC FF	JSR \$FFCC ;RESTORE I/O
F719	A0 00	LDY #MS1-MS1
F71B	24 9D	BIT MSGFLG ;PRINT ERROR?
F71D	30 0A	BVC L1018 ;NO
F71F	20 2F F1	JSR L1073 ;PRINT 'I/O ERROR #'
F722	68	PLA
F723	48	PHA
F724	09 30	ORA #\$30 ;MAKE ERROR # ASCII
F726	20 D2 FF	JSR \$FFD2 ;PRINT IT
F729	68	L1018 PLA
F72A	38	SEC
F72B	60	RTS
F72C		.END
F72C		.LIB KTAPE4

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:

LOC	CODE	LINE
F72C		;*****
F72C		;*****
F72C		; * FIND ANY TAPE HEADER.
F72C		; * READS TAPE DEVICE UNTIL ONE OF THE
F72C		; * FOLLOWING BLOCK TYPES IS FOUND: BOFH--
F72C		; * BASIC DATA FILE HEADER, BLF--BASIC
F72C		; * LOAD FILE. FOR SUCCESS, CARRY IS
F72C		; * CLEAR ON RETURN. FOR FAILURE, CARRY
F72C		; * IS SET ON RETURN. IN ADDITION, .A IS
F72C		; * 0 IF STOP KEY WAS PRESSED.
F72C		;*****
F72C		;*****
F72C	A5 93	L1098 LDA VERCK ;SAVE OLD VERIFY
F72E	48	PHA
F72F	20 41 F8	JSR L1029 ;READ TAPE BLOCK
F732	68	PLA
F733	85 93	STA VERCK ;RESTORE VERIFY
F735	B0 32	BCS L1101 ;READ TERMINATED
F737	A0 00	LDY #00
F739	B1 B2	LDA (TAPE1),Y ;GET HEADER TYPE
F73B	C9 05	CMP #E0T ;END OF TAPE?
F73D	F0 2A	BEQ L1101 ;YES
F73F	C9 01	CMP #BLF ;BASIC LOAD FILE?
F741	F0 08	BEQ L1027 ;YES
F743	C9 03	CMP #PLF ;FIXED LOAD FILE?
F745	F0 04	BEQ L1027 ;YES
F747	C9 04	CMP #BDFH ;BASIC DATA FILE?
F749	D0 E1	BNE L1098 ;NO, TRY AGAIN
F74B	AA	L1027 TAX ;FILE TYPE IN .X
F74C	24 9D	BIT #MSGFLG ;PRINT MESSAGE?
F74E	10 17	BPL L1102 ;NO
F750	A0 63	LDY #MS17-MS1 ;'FOUND'
F752	20 2F F1	JSR L1073
F755	A0 05	LDY #05
F757	B1 B2	L1100 LDA (TAPE1),Y ;OUTPUT COMPLETE
F759	20 02 FF	JSR \$FFD2 ;FILENAME
F75C	C8	INY
F75D	C0 15	CPY #15
F75F	D0 F6	BNE L1100
F761	A5 A1	LDA TIME+1 ;WAIT FOR 8.5 SECONDS
F763	20 E0 E4	JSR \$E4E0 ; OR FOR THE CBM KEY
F766	EA	NOP
F767	18	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

94 The Commodore 64 Kernal and Hardware Revealed

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		;
F76A		;*****
F76A		;* WRITE TAPE HEADER
F76A		;* ERROR IF TAPE BUFFER DE-ALLOCATED
F76A		;* CARRY CLEAR IF O.K.
F76A		;*****
F76A		;
F76A	85 9E	L1099 STA T1
F76C	20 D0 F7	JSR L1104 ;GET BUFFER ADDRESS
F76F	90 5E	BCC L1106 ;NOT ALLOCATED
F771	A5 C2	LDA STAH ;PRESERVE START AND END
F773	48	PHA ; ADDRESSES
F774	A5 C1	LDA STAL
F776	48	PHA
F777	A3 AF	LDA EAH
F779	48	PHA
F77A	A5 AE	LDA EAL
F77C	48	PHA
F77D	A0 BF	LDY #BUFSZ-1 ;BLANK TAPE BUFFER
F77F	A9 20	LDA #\$20 ;SPACE CHARS
F781	91 B2	L998 STA (TAPE1),Y
F783	88	DEY
F784	D0 FB	BNE L998
F786	A5 9E	LDA T1 ;BLOCK TYPE IN HEADER
F788	91 B2	STA (TAPE1),Y
F78A	C8	INY
F78B	A5 C1	LDA STAL ;START ADDRESS IN HEADER
F78D	91 B2	STA (TAPE1),Y
F78F	C8	INY
F790	A5 C2	LDA STAH
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 EAH
F79C	91 B2	STA (TAPE1),Y
F79E	C8	INY ;FILENAME IN HEADER
F79F	84 9F	STY T2
F7A1	A0 00	LDY #\$00
F7A3	84 9E	STY T1
F7A5	A4 9E	L1105 LDY T1
F7A7	C4 B7	CPY FNLEN
F7A9	F0 0C	BEQ L1107
F7AB	B1 BB	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	L1107 JSR L995 ;SET UP START & END
F7BA	A9 69	LDA #\$69 ;ADDR OF HEADER & SET
F7BC	83 AB	STA SHCNH ; TIME FOR LEADER

LOC	CODE	LINE		
F7BE	20 6B FB		JSR L1089	;WRITE FILE TO TAPE
F7C1	A8		TAY	;SAVE ERROR CODE IN .Y
F7C2	68		PLA	;RESTORE START & END
F7C3	85 AE		STA EAL	; ADDRESSES
F7C5	68		PLA	
F7C6	85 AF		STA EAH	
F7C8	68		PLA	
F7C9	85 C1		STA STAL	
F7CB	68		PLA	
F7CC	85 C2		STA STAH	
F7CE	98		TYA	;RESTOKE ERROR CODE
F7CF	60	L1106	RTS	

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:

LOC	CODE	LINE		
F7D0				
F7D0			;RETURN BUFFER ADDRESS	
F7D0				
F7D0	A6 B2	L1104	LDX TAPE1	
F7D2	A4 B3		LDY TAPE1+1	
F7D4	C0 02		CPY #\$02	;ALLOCATED?
F7D6	60		RTS	;CARRY CLEAR, DE-ALLOCATED
F7D7	20 D0 F7	L995	JSR L1104	;GET PTR TO CASSETTE
F7DA	8A		TXA	
F7DB	85 C1		STA STAL	;SAVE START LOW
F7DD	18		CLC	
F7DE	69 C0		ADC #BUFSZ	;COMPUTE POINTER TO END
F7E0	85 AE		STA EAL	;SAVE END LOW
F7E2	98		TYA	
F7E3	85 C2		STA STAH	;SAVE START HI
F7E5	69 00		ADC #\$00	;COMPUTE POINTER TO END
F7E7	85 AF		STA EAH	;SAVE END HIGH
F7E9	60		RTS	

Find correct file on tape

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		;*****
F7EA		;* FIND CORRECT FILE ON TAPE
F7EA		;* IF FNLEN = 0 THEN USE
F7EA		;* FIRST HEADER FOUND
F7EA		;*****
F7EA		;
F7EA	20 2C F7	L1108 JSR L1098 ;FIND ANY HEADER
F7ED	B0 1D	BCS 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	18	L1112 CLC ;SUCCESS
F80C	60	L1111 RTS
F80D		.END
F80D		.LIB KTAPE.S

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

\$F84A – read LOAD block entry

Routine source code:

LOC	CODE	LINE
F80D		;INCREASE POINTER IN TAPE BUFFER
F80D		;
F80D	20 D0 F7	L1110 JSR L1104 ;GET BUFFER ADDRESS
F810	E6 A6	INC BUFPT
F812	A4 A6	LDY BUFPT
F814	C0 C0	CPY #BUFSZ ;CHECK END BUFFER
F816	60	RTS
F817		;
F817		;WAIT FOR PLAY SWITCH
F817		;
F817	20 2E F8	L938 JSR L1116
F81A	F0 1A	BEQ L1113
F81C	A0 1B	LDY #MS7-MS1 ;'PRESS PLAY...'
F81E	20 2F F1	L1020 JSR L1073
F821	20 D0 F8	L1117 JSR L1125 ;TEST STOP KEY
F824	20 2E F8	JSR L1116 ;TEST CASSETTE SWITCHES
F827	D0 F8	BNE L1117
F829	A0 6A	LDY #MS18-MS1 ;'OK'
F82B	4C 2F F1	JMP L1073
F82E		;
F82E		;TEST CASSETTE SWITCH
F82E		;
F82E	A9 10	L1116 LDA #\$10 ;CHECK PORT
F830	24 01	BIT \$01 ;CLOSED?
F832	D0 02	BNE L1113 ;NO
F834	24 01	BIT \$01 ;DEBOUNCE
F836	18	L1113 CLC ;GOOD EXIT
F837	60	RTS
F838		;
F838		;CHECK FOR RECORD & PLAY
F838		;
F838	20 2E F8	L1114 JSR L1116
F83B	F6 F9	BEQ L1113
F83D	A0 2E	LDY #MS8-MS1 ;'PRESS RECORD...'
F83F	D0 DD	BNE L1020
F841		;
F841		;READ HEADER BLOCK
F841		;
F841	A9 00	L1029 LDA #\$00
F843	85 90	STA STATUS
F845	85 93	STA VERCK
F847	20 D7 F7	JSR L995
F84A		;
F84A		;READ LOAD BLOCK ENTRY
F84A		;
F84A	20 17 F8	L940 JSR L938 ;'PRESS PLAY...'
F84D	B0 1F	BCS L1109 ;STOP KEY
F84F	78	SEI
F850	A9 00	LDA #\$00 ;CLEAR FLAGS
F852	85 AA	STA RDFLG
F854	85 B4	STA SNSW1
F856	85 B0	STA CMPO
F858	85 9E	STA PIR1
F85A	85 9F	STA PTR2
F85C	85 9C	STA DPSW
F85E	A9 90	LDA #\$90 ;ENABLE FOR TAPE IRQ
F860	A2 0E	LDX #\$0E ;POINT IRQ VECTOR TO READ
F862	D0 11	BNE L1118 ;ALWAYS

Write memory

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 STAL,STAH
F867		;AND EAL,EAH AS A BLOCK
F867		;*****
F867	A9 14	L952 LDA #\$14 ;BETWEEN BLOCK SHORTS
F869	85 AB	STA SHCNH
F86B	20 38 FB	L1089 JSR L1114 ;'PRESS RECORD...'
F86E	B0 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 LDY #\$7F ;KILL UNWANTED IRQ
F877	8C 0D DC	STY D1ICR
F87A	8D 0D DC	STA D1ICR ;ENABLE WANTED
F87D	AD 0E DC	LDA D1CRA
F880	09 19	ORA #\$19
F882	8D 0F DC	STA D1CRB
F885	29 91	AND #\$91
F887	8D A2 02	STA \$02A2
F88A	20 A4 F0	JSR L921 ;WAIT FOR RS232
F88D	AD 11 D0	LDA VICREG+17 ;BLANK SCREEN
F890	29 EF	AND #\$EF
F892	8D 11 D0	STA VICREG+17
F895	AD 14 03	LDA CINV ;MOVE IRQ TO IRQ TEMP
F898	8D 9F 02	STA IRQTMP ;FOR CASSETTE OPS
F89B	AD 15 03	LDA CINV+1
F89E	8D A0 02	STA IRQTMP+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

LOC	CODE	LINE		
F8AB	A5 01		LDA \$01	;TURN CASSETTE MOTOR ON
F8AD	29 1F		AND #\$1F	
F8AF	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	
F8B9	D0 F0		BNE L1124	
F8BA	CA		DEX	
F8BB	D0 F8		BNE L1119	
F8BD				
F8BD			; ENABLE TAPE IRQ ROUTINES TO	
F8BD			;START WRITE OPERATION	
F8BD	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 L1115	;YES, RETURN
F8C7	20 D0 F8		JSR L1125	;NO CHECK STOP
F8CA	20 BC F6		JSR \$F6BC	;UPDATE TIME
F8CD	4C BE F8		JMP L1123	;STAY IN LOOP
F8D0	20 E1 FF	L1125	JSR \$FFE1	;TOP KEY DOWN?
F8D3	18		CLC	;ASSUME NOT
F8D4	D0 08		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 #\$00	;DISABLE IRQTMP
F8DE	8D 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				
F8E2			; SET UP TIMEOUT WATCH FOR NEXT DIPOLE	
F8E2				
F8E2	86 B1	L1126	STX TEMP	;TIMEOUT CONSTANT
F8E4	A5 B0		LDA CMPO	;CMPO*5
F8E6	0A		ASL A	
F8E7	0A		ASL A	
F8E8	18		CLC	
F8E9	65 B0		ADC CMPO	
F8EB	18		CLC	

100 The Commodore 64 Kernal and Hardware Revealed

LOC	CODE	LINE		
F8EC	65 B1		ADC TEMP	;ADJUST LONG BYTE COUNT
F8EE	85 B1		STA TEMP	
F8F0	A9 00		LDA #\$00	
F8F2	24 B0		BIT CMPO	;CHECK CMPO..
F8F4	30 01		BMI L1146	; MINUS, NO ADJUST
F8F6	2A		ROL A	; PLUS, ADJUST POS
F8F7	06 B1	L1146	ASL TEMP	;MULTIPLY CORRECTED
F8F9	2A		ROL A	; VALUE BY 4
F8FA	06 B1		ASL TEMP	
F8FC	2A		ROL A	
F8FD	AA		TAX	
F8FE	AD 06 DC	L1128	LDA D1TBL	;WATCH OUT FOR ROLLOVER
F901	C9 16		CMP #\$16	;TIME FOR ROUTINE?
F903	90 F9		BCC L1128	;TOO CLOSE SO WAIT
F905	65 B1		ADC TEMP	;CALCULATE AND
F907	8D 04 DC		STA D1TAL	; STROKE ADJUSTED TIME COUNT
F90A	8A		TXA	
F90B	6D 07 DC		ADC D1T8H	;ADJUST FOR H1 TIME COUNT
F90E	8D 05 DC		STA D1TAH	
F911	AD A2 02		LDA \$02A2	
F914	8D 0E DC		STA D1CRA	
F917	8D A4 02		STA \$02A4	
F91A	AD 0D DC		LDA D1ICR	
F91D	29 10		AND #\$10	
F91F	F0 09		BEQ L1129	
F921	A9 F9		LDA #\$F9	
F923	48		PHA	
F924	A9 2A		LDA #\$2A	
F926	48		PHA	
F927	4C 43 FF		JMP \$FF43	
F92A	58	L1129	CLI	
F92B	60		RTS	
F92C		.END		
F92C			.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:

LOC	CODE	LINE
F92C		;
F92C		;*****
F92C		;* CASSETTE READ SUBROUTINES
F92C		;*****

LOC	CODE	LINE		
F92C			;	
F92C			;TAPE READ IRQ ROUTINE	
F92C			;	
F92C	AE 07 DC	L1130	LDX D1TBH	;GET TIME SINCE LAST IRQ
F92F	A0 FF		LDY #\$FF	;COMPUTE COUNTER DIFF
F931	98		TYA	
F932	ED 06 DC		SBC D1TBL	
F935	EC 07 DC		CPX D1TBH	;TIMER HIGH KOLLOVER?
F938	D0 F2		BNE L1130	;YES, RECOMPUTE
F93A	86 B1		STX TEMP	
F93C	AA		TAX	
F93D	8C 06 DC		STY D1TBL	;RE-LOAD TIMER B
F940	8C 07 DC		STY D1TBH	
F943	A9 19		LDA #\$19	
F945	8D 0F DC		STA D1CRB	
F948	AD 0D DC		LDA D11CR	
F94B	8D A3 02		STA \$02A3	
F94E	98		TYA	
F94F	E5 B1		SBC TEMP	;CALCULATE HIGH
F951	86 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 B0		LDA CMPO	;CALC MIN PULSE VALUE
F95B	18		CLC	
F95C	69 3C		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 03		BEQ L1132	;NO, CONTINUE
F966	4C 60 FA		JMP L1154	;YES, FINISH BYTE
F969			;	
F969	A6 A3	L1132	LDX PCNTR	;9 BITS READ?
F96B	30 1B		BMI L1134	;YES, GOTO ENDING
F96D	A2 00		LDX #\$00	;SET BIT VAL TO ZERO
F96F	69 30		ADC #\$30	;ADD UP TO HALF WAY BETWEEN
F971	65 B0		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	;i?
F97E	B0 17		BCS L1137	;YES
F980	69 2C		ADC #\$2C	;MOVE TO LONGLONG
F982	65 B0		ADC CMPO	
F984	C5 B1		CMP TEMP	;LONGLONG?
F986	90 03		BCC 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 A9	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	
F99C	E5 B1		SBC TEMP	;SUBTRACT INPUT VAL
F99E	65 92		ADC SVXT	; ADD DIFF TO TEMP STORE
F9A0	85 92		STA SVXT	; USED TO ADJUST SOFT SERVO
F9A2	A5 A4		LDA FIRT	;FLIP DIPOLE FLAG
F9A4	49 01		EOR #\$01	

LOC	CODE	LINE	
F9A6	85 A4		STA F1RT
F9A8	F0 2B		BEQ L1143 ;SECOND HALF OF DIPOLE
F9AA	86 D7		STX DATA ;FIRST HALF SO STORE VAL
F9AC			;
F9AC	A5 B4	L1141	LDA SNSW1 ;NO BYTE START?
F9AE	F0 22		BEQ L1150 ;YES, RETURN
F9B0	AD A3 02		LDA \$02A3 ;TIMER A IRQ'D?
F9B3	29 01		AND #\$01
F9B5	D0 05		BNE L1133 ;YES
F9B7	AD A4 02		LDA \$02A4
F9BA	D0 16		BNE L1150 ;NO, EXIT
F9BC	A9 00	L1133	LDA #\$00 ;SET DIPOLE FLAG FOR FIRST HALF
F9BE	85 A4		STA F1RT
F9C0	8D A4 02		STA \$02A4
F9C3	A5 A3		LDA PCNTR ;WHERE IN BYTE
F9C5	10 30		BPL L1148 ; STILL DOING DATA
F9C7	30 BF		BMI L1134 ; PROCESS PARITY
F9C9			;
F9C9	A2 A6	L1144	LDX #\$A6 ;SETUP FOR LONGLONG
F9CB	20 E2 F8		JSR L1126
F9CE	A5 9B		LDA PRTY ;EVEN PARITY?
F9D0	D0 B9		BNE L1136 ;NO, SET ERROR
F9D2	4C BC FE	L1150	JMP \$FEBC ;RESTORE REGS AND RTI
F9D5			;
F9D5	A5 92	L1143	LDA SVXT ;ADJUST SOFT SERV0?
F9D7	F0 07		BEQ L1149 ;NO
F9D9	30 03		BMI L1142 ;YES, MORE BASE TIME
F9DB	C6 B0		DEC CMPO ;YES, LESS BASE TIME
F9DD	2C		.BYT \$2C ;SKIP NEXT
F9DE	E6 B0	L1142	INC CMPO
F9E0	A9 00	L1149	LDA #\$00 ;CLEAR DIFF FLAG
F9E2	85 92		STA SVXT
F9E4	E4 D7		CPX DATA ;CONSEC. LIKE VALS IN DIPOLE?
F9E6	D0 0F		BNE L1148 ;NO, PROCESS INFO
F9E8	8A		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		CMF #\$10 ; 16?
F9F1	90 B9		BCC L1141 ;NO, CONTINUE
F9F3	85 96		STA SYNO ;YES, FLAG SYNO
F9F5	B0 B5		BCS L1141 ;ALWAYS
F9F7			;
F9F7	8A	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 PCNTR ;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 MYCH ; INTO BYTE STOK
FA08	A2 DA		LDX #\$DA ;SETUP FOR NEXT DIPOLE
FA0A	20 E2 F8		JSR L1126
FA0D	4C BC FE		JMP \$FEBC ;RESTORE REGS AND RTI
FA10			;
FA10			;
FA10			;
FA10	A5 96	L1145	LDA SYNO ;GOT BLOCK SYNC?
FA12	F0 04		BEQ L1140 ;NO
FA14	A5 B4		LDA SNSW1 ;HAD REAL BYTE?
FA16	F0 07		BEQ L1151 ;NO
FA18	A5 A3	L1140	LDA PCNTR ;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

104 The Commodore 64 Kernal and Hardware Revealed

LOC	CODE	LINE		
FAB1	20 1C FE		JSR \$FE1C	;YES, LONG BLOCK ERROR
FAB4	D0 04		BNE L1158	;ALWAYS
FAB6	A9 00	L1156	LDA #\$00	
FAB8	85 AA		STA RDFLG	;NEW MODE, WAIT FOR SYNC
FAB8	4C BC FE	L1158	JMP \$FEB0	;EXIT, DONE
FABD	70 31	L1159	BVS L1163	;LOADING
FABF	D0 18		BNE L1162	;SYNCING
FA91	A5 B5		LDA DIFF	;HAVE BLOCK SYNC?
FA93	D0 F5		BNE L1158	;YES, EXIT
FA95	A5 B6		LDA PRP	;FIRST BYTE IN ERROR?
FA97	D0 F1		BNE L1158	;YES, EXIT
FA99	A5 A7		LDA SHCNL	;MOVE FSBLK TO CARRY
FA9B	4A		LSR A	
FA9C	A5 B0		LDA OCHAR	; SHOULD BE A HEADER COUNT CHAR
FA9E	30 03		BMI L1157	;NEG, FIRST BLOCK DATA
FAA0	90 18		BCC L1161	;EXPECTING FIRST BLOCK DATA
FAA2	18		CLC	
FAA3	B0 15	L1157	BCS L1161	;EXPECTING 2ND BLOCK
FAA5	29 0F		AND #\$0F	;MASK OFF HIGH STROKE HEADER
FAA7	85 AA		STA RDFLG	; COUNT IN MODE FLAG
FAA9	C6 AA	L1162	DEC RDFLG	;WAIT FOR KEAL DATA
FAAB	D0 DD		BNE L1158	; REAL
FAAD	A9 40		LDA #\$40	;NEXT UP IS REAL DATA
FAAF	85 AA		STA RDFLG	; SET DATA MODE
FAB1	20 8E FB		JSR L1174	;SETUP ADDR POINTERS
FAB4	A9 00		LDA #\$00	
FAB6	85 AB		STA SHCNH	
FAB8	F0 D0		BEQ L1158	;ALWAYS, EXIT
FABA	A9 80	L1161	LDA #\$80	;IGNORE BYTES MODE
FABC	85 AA		STA RDFLG	
FABE	D0 CA		BNE L1158	;ALWAYS
FAC0	A5 B5	L1163	LDA DIFF	;END OF BLOCK?
FAC2	F0 0A		BEQ L1160	;YES
FAC4	A9 04		LDA \$SBERR	;SHORT BLOCK ERROR
FAC6	20 1C FE		JSR \$FE1C	
FAC7	A9 00		LDA #\$00	;FORCE RDFLG FOR ERROR
FACB	4C 4A FB		JMP L1167	
FACE	20 D1 FC	L1160	JSR L1193	;END OF STORE AREA?
FAD1	90 03		BCC L1164	;NOT YET
FAD3	4C 4B FB		JMP L1172	;YES
FAD6	A6 A7	L1164	LDX SHCNL	;WHICH PASS?
FAD8	CA		DEX	
FAD9	F0 2D		BEQ L1169	;SECOND
FAD8	A5 93		LDA VERCK	;LOAD OR VERIFY?
FADD	F0 0C		BEQ L1166	;LOADING
FADF	A0 00		LDY #\$00	;VERIFYING
FAE1	A5 BD		LDA OCHAR	
FAE3	D1 AC		CMP (SAL),Y	;COMPARE
FAE5	F0 04		BEQ L1166	;GOOD, CONTINUE
FAE7	A9 01		LDA #\$01	;BAD, FLAG
FAE9	B5 B6		STA PRP	; AS ERROR
FAEB				
FAEB				
FAEB				
FAEB	A5 B6	L1166	LDA PRP	;CHK FOR ERRORS
FAED	F0 4B		BEQ L1171	;NONE
FAEF	A2 3D		LDX #\$3D	;MAX OF 30
FAF1	E4 9E		CPX PTR1	;REACHED MAX?
FAF3	90 3E		BCC L1173	;YES, FLAG 2ND PASS
FAF5	A6 9E		LDX PTR1	;INDEX INTO BAD
FAF7	A5 AD		LDA SAH	; AND STORE BAD LOC
FAF9	9D 01 01		STA BAD+1,X	; IN BAD TABLE
FAFC	A5 AC		LDA SAL	
FAFE	9D 00 01		STA BAD,X	
FB01	E8		INX	;ADVANCE TO NEXT
FB02	E8		INX	
FB03	86 9E		STX PTR1	
FB05	4C 3A FB		JMP L1171	;STORE CHAR

LOC	CODE	LINE		
FB08			;	
FB08			;CHECK BAD TABLE FOR RE-TRY	
FB08			;	
FB08	A6 9F	L1169	LDX PTR2	;DONE ALL IN TABLE?
FB0A	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 0B		BEQ L1168	;LOADING
FB24	A5 8D		LDA OCHAR	;VERIFYING
FB26	A0 00		LDY #00	
FB28	D1 AC		CMP (SAL),Y	
FB2A	F0 17		BEQ L1170	;O.K.
FB2C	C8		INY	;Y=1
FB2D	84 B6		STY PRP	;FLAG IT AS AN ERROR
FB2F	A5 B6	L1168	LDA PRP	;SECOND PASS ERROR?
FB31	F0 07		BEQ L1171	;NO
FB33	A9 10	L1173	LDA #SPERR	
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	A8		TAY	
FB3F	A5 8D		LDA OCHAR	
FB41	91 AC		STA (SAL),Y	;STORE CHARACTER
FB43	20 DB FC	L1170	JSR L1080	;NEXT ADDRESS
FB46	D0 43		BNE L1177	;ALWAYS
FB48			;	
FB48	A9 80	L1172	LDA #00	;SET SKIP NEXT DATA
FB4A	85 AA	L1167	STA RDFLG	
FB4C	78		SEI	
FB4D	A2 01		LDX #01	
FB4F	8E 0D DC		STX D11CR	
FB52	AE 0D DC		LDX D11CR	
FB55	A6 BE		LDX FSBLK	;DEC FSBLK FOR NEXT PASS
FB57	CA		DEX	
FB58	30 02		BMI L1165	;DONE, FSBLK=0
FB5A	86 BE		STX FSBLK	;ELSE, NEXT
FB5C	C6 A7	L1165	DEC SHCNH	;DEC PASS CALC
FB5E	F0 08		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	F0 23		BEQ L1177	;ALWAYS, EXIT
FB68			;	
FB68	20 93 FC	L1175	JSR L1192	;READ IT ALL, EXIT
FB6B	20 8E FB		JSR L1174	;RESTORE SAL & SAH
FB6E	A0 00		LDY #00	;SHCNH=0
FB70	84 AB		STY SHCNH	;USED TO CALC PARITY BYTE
FB72			;	
FB72			;COMPUTE PARITY BYTE	
FB72			;	
FB72	B1 AC	L1176	LDA (SAL),Y	;CALC BLOCK BCC
FB74	45 AB		EOR SHCNH	
FB76	85 AB		STA SHCNH	
FB78	20 DB FC		JSR L1080	;BUMP ADDRESS
FB7B	20 D1 FC		JSR L1193	;A1 END?
FB7E	90 F2		BCC L1176	;NOT YET
FB80	A5 AB		LDA SHCNH	;BCC CHAR MATCH?
FB82	45 BD		EOR OCHAR	

Entry point: \$FBA6

Routine source code:

```

LOC      CODE      LINE
FBA6      ;
FBA6      ;*****
FBA6      ;* CASSETTE WRITE SUBROUTINES.
FBA6      ;*      FSBK IS BLOCK COUNTER FOR RECORD
FBA6      ;*      = 0 SECOND DATA
FBA6      ;*      = 1 FIRST DATA
FBA6      ;*      = 2 FIRST HEADER
FBA6      ;*****
FBA6      ;
FBA6      ;TOGGLE WRITE BIT ACCORDING TO LSB
FBA6      ;IN OCHAR
FBA6      ;
FBA6      A5 B0      L1122 LDA OCHAR          ;BIT TO WRITE INTO CARRY
FBA8      4A          LSR A
FBA9      A9 60      LDA #$60                ;ASSUME CARRY CLEAR (SHORT)
FBA8      90 02      BCC L1184                ;CORRECT
FBAD      A9 B0      L1185 LDA #$B0            ;SET LONG
FBAF      A2 90      L1184 LDX #$90            ;SET AND STORE TIME
FBB1      8D 06 DC    L1178 STA D17BL         ;LO BYTE
FBB4      8E 07 DC    STX D17BH               ;HI BYTE
FBB7      AD 0D DC    LDA D11CR               ;CLEAR IRQ
FBB8      A9 19      LDA #$19
FBB8      8D 0F DC    STA D1CRB               ;FORCE LOAD & START TIMER
FBBF      A5 01      LDA $01                  ;TOGGLE WRITE BIT
FBC1      49 08      EOR #$08
FBC3      85 01      STA $01

```

LOC	CODE	LINE		
FBC5	29 08		AND #408	;LEAVE JUST WRITE BIT
FBC7	60		RTS	
FBC8	38	L1181	SEC	;FLAG PRP FOR END OF BLOCK
FBC9	66 B6		ROR PRP	
FBCB	30 3C		BMI L1183	;ALWAYS
FBCD				
FBCD			;TAPE WRITE IRQ ENTRY	
FBCD				
FBCD				
FBCD	A5 A8	WRTN	LDA REK	;CHECK FOR ONE LONG
FBCF	D0 12		BNE L1191	
FBD1	A9 10		LDA #410	;WRITE LONG BIT
FBD3	A2 01		LDX #401	
FBD5	20 B1 FB		JSR L1178	
FBD8	D0 2F		BNE L1183	
FBD8	E6 A8		INC REK	
FBD8	A5 B6		LDA PRP	;END OF BLOCK?
FBD8	10 29		BPL L1183	;NO, CONTINUE
FBE0	4C 57 FC		JMP L1194	;YES, FINISH OFF
FBE3				
FBE3	A5 A9	L1191	LDA REZ	;CHECK FOR A ONE BIT
FBE3	D0 09		BNE L1180	
FBE7	20 AD FB		JSR L1185	
FBEA	D0 1D		BNE L1183	
FBEA	E6 A9		INC REZ	
FBEA	D0 19		BNE L1183	
FBEA				
FBEA	20 A6 FB	L1180	JSR L1122	;WRITE
FBEA	D0 14		BNE L1183	;ON BIT LOW, EXIT
FBEA	A5 A4		LDA FIRI	;FIRST OF DIPOLE?
FBEA	49 01		EOR #401	
FBEA	85 A4		STA FIRI	
FBEA	F0 0F		BEQ L1179	;DIPOLE DONE
FBEA	A5 BD		LDA OCHAR	;FLIPS BIT FOR COMPLEMENTARY
FBEA	49 01		EOR #401	
FBEA	85 BD		STA OCHAR	
FBEA	29 01		AND #401	;TOGGLE PARITY
FBEA	45 9B		EOR PRTY-	
FBEA	85 9B		STA PRTY	
FBEA	4C BC FE	L1183	JMP \$FEBC	;RESTORE REGS AND RTI
FBEA				
FBEA	46 BD	L1179	LSR OCHAR	;NEXT BIT
FBEA	C6 A3		DEC PCNTR	;DEC COUNTER FOR # BITS
FBEA	A3 A3		LDA PCNTR	;8 BITS SENT?
FBEA	F0 3A		BEQ L1190	;YES, DO PARITY
FBEA	10 F3		BPL L1183	;NO, SEND REST
FBEA				
FBEA	20 97 FB	L1186	JSR L1079	;CLEAN UP COUNTERS
FBEA	58		CLI	;ALLOW INTERRUPTS TO NEST
FBEA	A5 A5		LDA CNTDN	;WRITING HEADER COUNTER?
FBEA	F0 12		BEQ L1189	;NO
FBEA	A2 00		LDX #400	;WRITE HEADER COUNTERS
FBEA	86 D7		STX DATA	;CLEAR BCC
FBEA	C6 A5		DEC CNTDN	
FBEA	A6 BE		LDX FSBLK	;FIRST BLOCK HEADER?
FBEA	E0 02		CPX #402	
FBEA	D0 02		BNE L1196	;NO
FBEA	09 80		ORA #480	;YES, MARK 1ST BLOCK HEADER
FBEA	85 BD	L1196	STA OCHAR	;WRITE CHARS IN HEADER
FBEA	D0 D9		BNE L1183	
FBEA	20 D1 FC	L1189	JSR L1193	;ADDR=END?
FBEA	90 0A		BCC L1188	;NOT YET
FBEA	D0 71		BNE L1181	;MARK END
FBEA	E6 AD		INC SAH	
FBEA	A5 D7		LDA DATA	;WRITE BCC
FBEA	85 BD		STA OCHAR	
FBEA	B0 CA		BCS L1183	;ALWAYS

LOC	CODE	LINE		
FC3F			;	
FC3F	A0 00	L1188	LDY #\$00	;NEXT CHAR
FC41	B1 AC		LDA (SAL),Y	
FC43	85 B0		STA OCHAR	;STORE IN OUTPUT CHAR
FC45	45 D7		EOR DATA	;UPDATE BCC
FC47	85 D7		STA DATA	
FC49	20 DB FC		JSR L1080	;BUMP ADDRESS
FC4C	D0 BB		BNE L1183	;ALWAYS
FC4E			;	
FC4E	A5 9B	L1190	LDA PRTY	;PARITY INTO OCHAR
FC50	49 01		EOR #\$01	
FC52	85 BD		STA OCHAR	; FOR NEXT BIT
FC54	4C BC FE	L1187	JMP \$FEBC	;RESTORE REGS AND RTI
FC57			;	
FC57	C6 BE	L1194	DEC FSBLK	;END?
FC59	D0 03		BNE L1182	;BLOCK ONLY
FC5B	20 CA FC		JSR L1121	;WRITE SO TURN OFF MOTOR
FC5E	A9 50	L1182	LDA #\$50	;PUT 80 CASSETTE
FC60	85 A7		STA SHCNL	; SYNCs AT END
FC62	A2 08		LDX #\$08	
FC64	78		SEI	
FC65	20 BD FC		JSR L1195	;SET VECTOR TO WRITE ZEROS
FC68	D0 EA		BNE L1187	;ALWAYS

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:

LOC	CODE	LINE		
FC6A			;	
FC6A			; TAPE IRQ ENTRY FOR	
FC6A			;	
FC6A	A9 78	WRTZ	LDA #\$78	;WRITING LEADING ZEROS
FC6C	20 AF FB		JSR L1184	; FOR SYNC
FC6F	D0 E3		BNE L1187	
FC71	C6 A7		DEC SHCNL	;DONE WITH LOW SYNC?
FC73	D0 DF		BNE L1187	;NO
FC75	20 97 FB		JSR L1079	;YES, CLEAN UP COUNTERS
FC78	C6 AB		DEC SHCNH	;DONE WITH SYNC?
FC7A	10 D8		BPL L1187	;NO
FC7C	A2 0A		LDX #\$0A	;YES, VECTOR FOR DATA
FC7E	20 BD FC		JSR L1195	
FC81	58		CLI	
FC82	E6 AB		INC SHCNH	;ZERO SHCNH
FC84	A5 BE		LDA FSBLK	;DONE?
FC86	F0 30		BEQ L1198	;YES, SYSTEM RESTORE

```

LOC      CODE      LINE
FC88      20 8E FB      JSR L1174
FC88      A2 07      LDZ #509      ;SETUP FOR HEADER COUNT
FC8D      86 A5      STX CNTDN
FC8F      86 86      STX PRP
FC91      D0 83      BNE L1186      ;ALWAYS
FC93      ;
FC93      08      L1192 PHP      ;CLEAN UP IRQ AND
FC94      73      SEL      ; RESTORE PIA'S
FC95      AD 11 D0      LDA VICREG+17 ;RESTORE SCREEN
FC98      09 10      ORA #10
FC9A      8D 11 D0      STA VICREG+17
FC9D      20 CA FC      JSR L1121      ;TURN OFF MOTOR
FCA0      A9 7F      LDA #7F      ;CLEAR INTERRUPTS
FCA2      8D 0D DC      STA DIICR
FCA5      20 DD FD      JSR $FDD0      ;RESTORE KEYBOARD IRQ
FCAB      AD A0 02      LDA IRQIMP+1 ;RESTORE KEYBOARD INTERRUPT VECTOR
FCAB      F0 09      BEQ L1127      ;NO IRQ
FCAD      8D 15 03      STA CINV+1
FCB0      AD 9F 02      LDA IRQTMP
FCB3      8D 14 03      STA CINV
FCB6      28      L1127 PLP
FCB7      60      RTS
FCB8      ;
FCB8      20 93 FC      L1198 JSR L1192      ;RESTORE SYSTEM IRQ
FCB8      F0 97      BEQ L1187      ;CAME 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 KEYSKAN
FCBD      ;*           = 14 READ DATA FROM TAPE
FCBD      ;*****
FCBD      ;
FCBD      8D 93 FD      L1195 LDA $FD93,X      ;MOVE IRQ VECTORS
FCC0      8D 14 03      STA CINV      ; TO VECTOR TABLE
FCC3      8D 94 FD      LDA $FD94,X
FCC6      8D 15 03      STA CINV+1
FCC9      60      RTS
FCCA      ;
FCCA      A5 01      L1121 LDA $01      ;TURN OFF CASSETTE MOTOR
FCCC      09 20      ORA #20
FCCE      85 01      STA $01
FCD0      60      RTS
FCD1      ;*****
FCD1      ;* COMPARE START AND END OF LOAD/SAVE
FCD1      ;* ADDRESSES. SUBROUTINE CALLED BY
FCD1      ;* TAPE READ, SAVE, TAPE WRITE
FCD1      ;*****
FCD1      ;
FCD1      38      L1193 SEC
FCD2      A5 AC      LDA SAL
FCD4      E5 AE      SBC EAL
FCD6      A5 AD      LDA SAH
FCD8      E5 AF      SBC EAH
FCDA      60      RTS
FCDB      ;
FCDB      ;BUMP ADDRESS POINTER SAL
FCDB      ;
FCDB      E4 AC      L1080 INC SAL
FCD0      D0 02      BNE L1083
FCDF      E6 AD      INC SAH
FCE1      60      L1083 RTS
FCE2      ;
FCE2      ;-----
FCE2      ;
FCE2      ;*=$FD9B
FD9B      ;
FD9B      ; TAPE IRQ VECTORS

```

LOC	CODE	LINE
FD9E		;
FD9E	6A FC	.WOR WRTZ ;WRITE ZEROS TAPE
FD9D	CD FB	.WOR WRIN ;WRITE NORMAL TAPE
FD9F	31 EA	.WOR \$EA31 ;NORMAL IRQ
FDA1	2C F9	.WOR L1130 ;READ TAPE
FDA3		;
FDA3		.END
FDA3		.END

4.7 High speed tape operation

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 ZIFRAlload.

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.

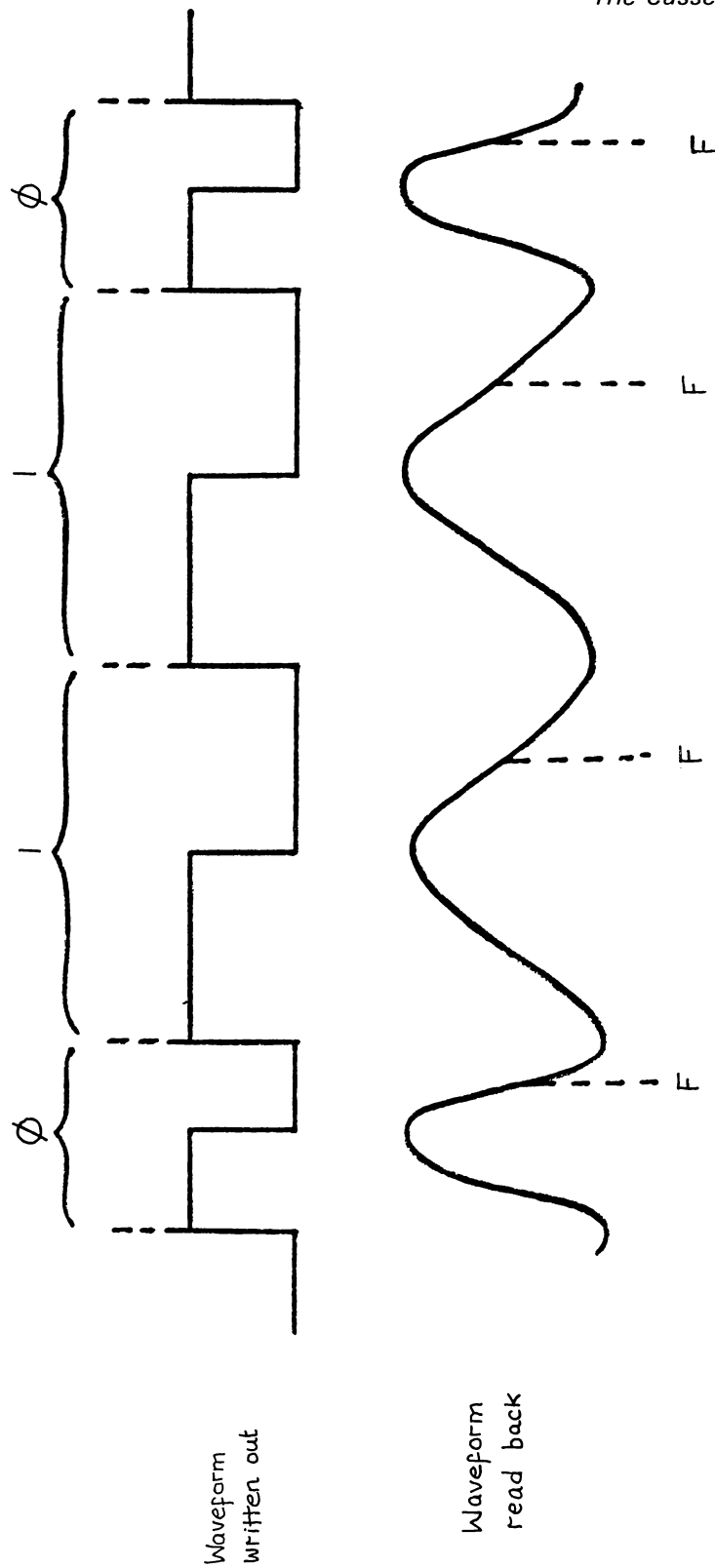


Fig. 4.4. High speed tape waveform.

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.

```

033C      ! FAST TAPE SAVE FOR THE 64.
033C      !*****
033C      !
033C      !THIS ROUTINE WILL SAVE A PROGRAM
033C      !TO TAPE SO THAT WHEN LOADED BACK
033C      !IT WILL LOAD FASTER THAN THE
033C      !1541 DISK DRIVE.
033C      !
033C      !AN OPTION FOR AUTO-RUN IS
033C      !INCLUDED.
033C      !
C000      *=$C000
C000 A90B      LDA #<SAVVEC      !CHANGE SAVE VECTOR
C002 8D3203     STA $0332        ! TO GO TO NEW
C005 A9C0      LDA #>SAVVEC     ! SAVE ROUTINE
C007 8D3303     STA $0333
C00A 60        RTS
C00B      !
C00B 48        SAVVEC          PHA      !SAVE OFF .A
C00C A5BA      LDA $BA          !GET DEVICE #
C00E C907      CMP #$07        !NUMBER ??
C010 F004      BEQ TSAVE       !YES
C012 68        PLA
C013 4CEDF5     JMP $F5ED       !DO NORMAL SAVE
C016      !
C016 A5B9      TSAVE          LDA $B9      !GET SEC. ADDR.
C018 8D29C2     STA RUNFLG      !FLAG FOR AUTO-RUN
C01B A00F      LDY #$0F
C01D A920      LDA #$20
C01F 998CC1     LOOP1         STA FLNAME,Y !BLANK FILENAME
C022 88        DEY
C023 10FA      BPL LOOP1
C025 A4B7      LDY $B7          !GET FILENAME LENGTH
C027 C011      CPY #$11        !GREATER THAN 16?
C029 9002      BCC LOOP2      !NO
C02B A010      LDY #$10        !ONLY 1ST 16 CHARS
C02D 88        DEY
C02E 3008      BMI TSAVE1
C030 B1BB      LDA (<$B),Y     !GET FILENAME
C032 998CC1     STA FLNAME,Y   !STORE IT
C035 4C2DC0     JMP LOOP2      !DO NEXT CHAR
C038      !
C038 A047      TSAVE1         LDY #$47
C03A B944C1     TSAVE2         LDA LOADER,Y !GET LOADER BYTE
C03D 99BC02     STA $02BC,Y    !STORE IT TO SAVE
C040 88        DEY
C041 10F7      BPL TSAVE2     !FOR ALL BYTES
C043 A901      LDA #$01
C045 AA        TAX
C046 A8        TAY
C047 20BAFF     JSR $FFBA      !SET FILE DETAILS
C04A A99E      LDA #$9E        !LENGTH OF FILENAME
C04C A28C      LDX #<FLNAME    !FILENAME LO
C04E A0C1      LDY #>FLNAME    !FILENAME HI
C050 20BDF5     JSR $FFBD      !SET NAME DETAILS
C053 A900      LDA #$00        !STOP NAME FROM BEING
C055 859D      STA $9D         !PRINTED ON SAVE
C057 A902      LDA #$02

```

```

C059 85FC          STA #FC          !SAVE START HI
C05B A9BC          LDA #$BC
C05D 85FB          STA #FB          !SAVE START LO
C05F A9FB          LDA #$FB          !POINT TO START
C061 A204          LDX #$04          !SAVE END LO
C063 A003          LDY #$03          !SAVE END HI
C065 20D8FF        JSR $FFD8        !SAVE LOADER FILE
C068 A983          LDA #$83
C06A 8D0203        STA $0302        !RESET WARM START
C06D A9A4          LDA #$A4          ! VECTOR
C06F 8D0303        STA $0303
C072 A901          LDA #$01
C074 859D          STA $9D
C076 A000          LDY #$00
C078 A200          LDX #$00
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
C099 852B          STA $2B          !ZERO LO BYTE
C09B B12B          LDA ($2B),Y      !GET A BYTE
C09D 20FAC0        JSR WRTBYT        !WRITE IT
C0A0 C8            INY
C0A1 D002          BNE TSAVE3
C0A3 E62C          INC $2C
C0A5 C42D          CPY $2D          !CHECK END OF SAVE
C0A7 A52C          LDA $2C
C0A9 E52E          SBC $2E
C0AB 90EE          BCC TSAVLOOP
C0AD A5FB          LDA $FB
C0AF 20FAC0        JSR WRTBYT        !NOT YET
C0B2 200EC1        JSR WRTBIT        !GET CHECKSUM
C0B5 A91B          LDA #$1B        !WRITE IT
C0B7 8D11D0        STA $D011        !CLOSE OFF LAST BIT
C0BA A937          LDA #$37
C0BC 8501          STA $01          !UNBLANK SCREEN
C0BE 58            CLI
C0BF 68            PLA
C0C0 852C          STA $2C          !STOP TAPE
C0C2 68            PLA
C0C3 852B          STA $2B          !RESTART IRQ
C0C5 2084FF        JSR $FF84        !GET START HI
C0C8 4C74A4        JMP $A474        !STORE IT
C0CB              !
C0CB A906          LDA #$06        !GET START LO
C0CD 8501          STA $01          !STORE IT
C0CF A90B          LDA #$0B        !GET START LO
C0D1 8D11D0        STA $D011        !STORE IT
C0D4 CA            DEX              !BLANK SCREEN
C0D5 D0FD          BNE HEADR1      !PAUSE FOR TAPE
C0D7 88            DEY              ! TO GET TO FULL
C0D8 D0FA          BNE HEADR1      ! SPEED
C0DA 78            SEI              !DISABLE IRQ
C0DB A9A0          LDA #$A0        !INITIAL TIMER
C0DD 8D04DD        STA $DD04        ! VALUE FOR DELAY
C0E0 A900          LDA #$00

```

C0E2	8D05DD		STA \$DD05	
C0E5	A919		LDA #\$19	
C0E7	8D0EDD		STA \$DD0E	!START TIMER
C0EA	A040		LDY #\$40	
C0EC	A940	HEADR2	LDA #\$40	!01000000 FOR
C0EE	20FAC0		JSR WRTBYT	!ALIGNMENT
C0F1	58		DEY	
C0F2	D0F8		BNE HEADR2	!WRITE 64 OF THEM
C0F4	A95A		LDA #\$5A	!CHECK ALIGNMENT
C0F6	20FAC0		JSR WRTBYT	!WRITE IT
C0F9	60		RTS	
C0FA		!		
C0FA	85BD	WRTBYT	STA \$BD	!STORE BYTE
C0FC	45FB		EOR \$FB	!CHECKSUM
C0FE	85FB		STA \$FB	
C100	A908		LDA #\$08	!LOOP FOR 8 BITS
C102	85A3		STA \$A3	
C104	26BD	WBYTE1	ROL \$BD	!BIT INTO CARRY
C106	20EC1		JSR WRTBIT	!WRITE THE BIT
C109	C6A3		DEC \$A3	
C10B	D0F7		BNE WBYTE1	!DO NEXT BIT
C10D	60		RTS	
C10E		!		
C10E	A240	WRTBIT	LDX #\$40	!ASSUME ZERO BIT
C110	9002		BCC WBIT1	!CORRECT ASSUMPTION
C112	A290		LDX #\$90	!SET FOR ONE BIT
C114	8E04DD	WBIT1	STX \$DD04	!SET TIMER
C117	A900		LDA #\$00	
C119	8D05DD		STA \$DD05	
C11C	A901		LDA #\$01	!WAIT FOR TIMER
C11E	2C0DDD	WBIT2	BIT \$DD0D	
C121	F0FB		BEQ WBIT2	
C123	A501		LDA \$01	!TOGGLE WRITE BIT
C125	4908		EOR #\$08	! IN 6510 REGISTER
C127	8501		STA \$01	
C129	EE20D0		INC \$D020	!SHOW IT IS WORKING
C12C	A919		LDA #\$19	
C12E	8D0EDD		STA \$DD0E	!START TIMER
C131	A901		LDA #\$01	!WAIT FOR TIMER
C133	2C0DDD	WBIT3	BIT \$DD0D	
C136	F0FB		BEQ WBIT3	
C138	A501		LDA \$01	!TOGGLE WRITE BIT
C13A	4908		EOR #\$08	! IN 6510 REGISTER
C13C	8501		STA \$01	
C13E	A919		LDA #\$19	
C140	8D0EDD		STA \$DD0E	!START TIMER
C143	60		RTS	
C144		!		
C144		!THE LOADER STARTS HERE		
C144		!		
C144	AD20D0	LOADER	LDA \$D020	
C147	85FE		STA \$FE	!SAVE BORDER COLOUR
C149	A9A4		LDA #\$A4	!RESET WARM START
C14B	8D0303		STA \$0303	! VECTOR
C14E	A983		LDA #\$83	
C150	8D0203		STA \$0302	
C153	205103		JSR \$0351	!FAST LOAD THE FILE
C156	A5FE		LDA \$FE	
C158	8D20D0		STA \$D020	!RESTORE BORDER COLOUR
C15B	A937		LDA #\$37	
C15D	8501		STA \$01	!STOP THE TAPE
C15F	58		CLI	!RESTORE IRQ
C160	A91B		LDA #\$1B	
C162	8D11D0		STA \$D011	!UNBLANK SCREEN
C165	2084FF		JSR \$FF84	!RESET I/O
C168	A5FC		LDA \$FC	!COMPARE CALCULATED TO
C16A	C5FB		CMP \$FB	! LOADED CHECKSUM
C16C	D015		BNE LODERR	!DIFFERENT..ERROR
C16E	2063A6		JSR \$A663	!CLR
C171	ADDE03		LDA \$03DE	!GET AUTO-RUN FLAG

C174	F00A	BEQ EXIT	!NO RUN
C176	208EA6	JSR \$A68E	!SET CHARGET POINTER
C179	A900	LDA #\$00	
C17B	859D	STA \$9D	!FLAG RUN MODE
C17D	4CAEA7	JMP \$A7AE	!RUN
C180	!		
C180	6C0203	EXIT	JMP (\$0302)
C183	!		!WARM START
C183	A21D	LODERR	LDX #\$1D
C185	4C37A4		JMP \$A437
C188	!		!'?LOAD ERROR'
C188	8BE3		!SEND ERROR
C18A	BC02		
C188	8BE3	WOR \$E38B	!ERROR LINK
C18A	BC02	WOR \$02BC	!WARM START LINK
C18C	!		
C18C	202020	FLNAME	TXT "
C19C	!		"
C19C	!		
C19C	!		!16 SPACES
C19C	!		!*
C19C	!		!\$0351
C19C	!		!
C19C	208703	JSR \$0387	!READ HEADER
C19F	20BA03	JSR \$03BA	!READ A BYTE
C1A2	A8	TAY	!LOAD LO
C1A3	A900	LDA #\$00	
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	852D	STA \$2D	!END LO
C1B1	20BA03	JSR \$03BA	!READ A BYTE
C1B4	852E	STA \$2E	!END HI
C1B6	20BA03	JSR \$03BA	!READ A BYTE
C1B9	91C1	STA (\$C1),Y	!STORE IT
C1BB	45FC	EOR \$FC	!CALCULATE CHECKSUM
C1BD	85FC	STA \$FC	
C1BF	C8	INY	
C1C0	D002	BNE TLOAD2	
C1C2	E6C2	INC \$C2	
C1C4	C42D	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	!		!*\$0367
C1D2	!		!
C1D2	A907	LDA #\$07	
C1D4	8501	STA \$01	!START TAPE
C1D6	A90B	LDA #\$0B	
C1D8	8D11D0	STA \$D011	!BLANK SCREEN
C1DB	CA	DEX	!PAUSE FOR TAPE TO
C1DC	D0FD	BNE RHEAD1	! REACH FULL SPEED
C1DE	88	DEY	
C1DF	D0FA	BNE RHEAD1	
C1E1	78	SEI	!DISABLE IRQ
C1E2	84FC	STY \$FC	!ZERO CHECKSUM
C1E4	8C05DD	STY \$DD05	!SET TIMER HI
C1E7	A9F8	LDA #\$F8	
C1E9	8D04DD	STA \$DD04	!SET TIMER LO
C1EC	A200	LDX #\$00	
C1EE	20C803	RHEAD2	JSR \$03C8
C1F1	26BD	ROL \$BD	!READ A BIT
C1F3	A5BD	LDA \$BD	!INTO BYTE
C1F5	C940	CMF #\$40	
C1F7	D0F5	BNE RHEAD2	!ALIGNED?
C1F9	20BA03	RHEAD3	JSR \$03BA
C1FC	C940	CMF #\$40	!NOT YET
C1FE	F0F9	BEQ RHEAD3	!READ A BYTE
			!IS IT 64?
			!YES

```

C200 C95A          CMP #$5A          !ALIGNMENT CHECK?
C202 D0EA          BNE RHEAD2        !NO
C204 60           RTS
C205             !
C205             !*=$03BA
C205             !
C205 A901          LDA #$01
C207 85BD          STA $BD
C209 20C803 GBYTE1 JSR $03C8        !READ A BIT
C20C 26BD          ROL $BD          !INTO BYTE
C20E 90F9          BCC GBYTE1       !COMPLETE BYTE
C210 A5BD          LDA $BD
C212 60           RTS
C213             !
C213             !*=$03C8
C213             !
C213 A910          LDA #$10        !WAIT FOR BIT
C215 2C0DDC GBIT1  BIT $DC0D
C218 F0FB          BEQ GBIT1
C21A AD0DD          LDA $DD0D
C21D 48           PHA
C21E A919          LDA #$19
C220 8D0EDD        STA $DD0E        !START TIMER
C223 68           PLA
C224 EE20D0        INC $D020        !SHOW IT IS WORKING
C227 4A           LSR A            !MOVE BIT INTO CARRY
C228 60           RTS
C229             !
C229             !*=$03DE
C229             !
C229 00           RUNFLG          BYT 0

```

Program 5.

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.

```

033C             ! FAST TAPE SAVE FOR THE 64.
033C             !*****
033C             !
033C             !THIS ROUTINE WILL SAVE A PROGRAM
033C             !TO TAPE SO THAT WHEN LOADED BACK
033C             !IT WILL LOAD WITH THE SCREEN ON.
033C             !
033C             !*=$C000
033C             !
033C A90B          LDA #<SAVEVEC      !CHANGE SAVE VECTOR
0302 8D3203        STA $0332        ! TO GO TO NEW
0305 A9C0          LDA #>SAVEVEC    ! SAVE ROUTINE
0307 8D3303        STA $0333
030A 60           RTS
030B             !
030B 48           SAVVEC          PHA          !SAVE OFF .A
030C A5BA          LDA $BA          !GET DEVICE #
030E C907          CMP #$07        !NUMBER ??

```

C010 F004	BEQ TSAVE	!YES
C012 68	PLA	
C013 4CEDF5	JMP \$F5ED	!DO NORMAL SAVE
C016	!	
C016 A00E TSAVE	LDY #\$0E	
C018 A920	LDA #\$20	
C01A 9999C1 LOOP1	STA FLNAME+1,Y	!BLANK FILENAME
C01D 88	DEY	
C01E D0FA	BNE LOOP1	
C020 A4B7	LDY \$B7	!GET FILENAME LENGTH
C022 C00F	CPY #\$0F	!GREATER THAN 14?
C024 9002	BCC LOOP2	!NO
C026 A00E	LDY #\$0E	!ONLY 1ST 14 CHARS
C028 88	DEY	
C029 3008	BMI TSAVE1	
C02B B1B8	LDA (\$B8),Y	!GET FILENAME
C02D 999AC1	STA FLNAME+2,Y	!STORE IT
C030 4C28C0	JMP LOOP2	!DO NEXT CHAR
C033	!	
C033 A058 TSAVE1	LDY #\$58	
C035 B93FC1 TSAVE2	LDA LOADER,Y	!GET LOADER BYTE
C038 99AB02	STA \$02AB,Y	!STORE IT TO SAVE
C03B 88	DEY	
C03C 10F7	BPL TSAVE2	!FOR ALL BYTES
C03E A901	LDA #\$01	
C040 AA	TAX	
C041 A8	TAY	
C042 20BAFF	JSR \$FFBA	!SET FILE DETAILS
C045 A9BB	LDA #\$BB	!LENGTH OF FILENAME
C047 A298	LDX #<FLNAME	!FILENAME LO
C049 A0C1	LDY #>FLNAME	!FILENAME HI
C04B 20BDDF	JSR \$FFBD	!SET NAME DETAILS
C04E A900	LDA #\$00	
C050 859D	STA \$9D	
C052 A902	LDA #\$02	
C054 85FC	STA \$FC	!SAVE START HI
C056 A9AB	LDA #\$AB	
C058 85FB	STA \$FB	!SAVE START LO
C05A A9FB	LDA #\$FB	!POINT TO START
C05C A204	LDX #\$04	!SAVE END LO
C05E A003	LDY #\$03	!SAVE END HI
C060 20D8FF	JSR \$FFD8	!SAVE LOADER FILE
C063 A983	LDA #\$83	
C065 8D0203	STA \$0302	!RESET WARM START
C068 A9A4	LDA #\$A4	! VECTOR
C06A 8D0303	STA \$0303	
C06D A9FF	LDA #\$FF	
C06F 859D	STA \$9D	
C071 A200	LDX #\$00	
C073 A000	LDY #\$00	
C075	!	
C075	!THE FAST SAVE ROUTINE	
C075	!STARTS HERE.	
C075	!	
C075 20C6C0 SAVEIT	JSR WRTHDR	!WRITE ALIGNMENT BYTES
C078 A52B	LDA \$2B	!GET START LO
C07A 48	PHA	
C07B 20F5C0	JSR WRTBYT	!WRITE IT
C07E A52C	LDA \$2C	!GET START HI
C080 48	PHA	
C081 20F5C0	JSR WRTBYT	!WRITE IT
C084 A52D	LDA \$2D	!GET END LO
C086 20F5C0	JSR WRTBYT	!WRITE IT
C089 A52E	LDA \$2E	!GET END HI
C08B 20F5C0	JSR WRTBYT	!WRITE IT
C08E 84FB	STY \$FB	!ZERO CHECKSUM
C090 A42B	LDY \$2B	!GET PAGE OFFSET
C092 A900	LDA #\$00	
C094 852B	STA \$2B	!ZERO LO BYTE
C096 B12B TSAVELOOP	LDA (\$2B),Y	!GET A BYTE

C098	20F5C0		JSR WRTBYT	!WRITE IT
C09B	C8		INY	
C09C	D002		BNE TSAVE3	
C09E	E62C		INC \$2C	
C0A0	C42D	TSAVE3	CPY \$2D	!CHECK END OF SAVE
C0A2	A52C		LDA \$2C	
C0A4	E52E		SBC \$2E	
C0A6	90EE		BCC TSAVELOOP	!NOT YET
C0A8	A5FB		LDA \$FB	!GET CHECKSUM
C0AA	20F5C0		JSR WRTBYT	!WRITE IT
C0AD	2009C1		JSR WRTBIT	!CLOSE OFF LAST BIT
C0B0	A91B		LDA #\$1B	
C0B2	8D11D0		STA \$D011	!UNBLANK SCREEN
C0B5	A937		LDA #\$37	
C0B7	8501		STA \$01	!STOP TAPE
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	!STORE IT
C0C0	2084FF		JSR \$FF84	!RESET I/O
C0C3	4C74A4		JMP \$A474	!EXIT TO 'READY.'
C0C6		!		
C0C6	A906	WRTHDR	LDA #\$06	!BASIC ROM OUT &
C0C8	8501		STA \$01	! START TAPE
C0CA	A90B		LDA #\$0B	
C0CC	8D11D0		STA \$D011	!BLANK SCREEN
C0CF	CA	HEADR1	DEX	!PAUSE 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 \$DD04	! VALUE FOR DELAY
C0DB	A900		LDA #\$00	
C0DD	8D05DD		STA \$DD05	
C0E0	A919		LDA #\$19	
C0E2	8D0EDD		STA \$DD0E	!START TIMER
C0E5	A040		LDY #\$40	
C0E7	A940	HEADR2	LDA #\$40	!01000000 FOR
C0E9	20F5C0		JSR WRTBYT	!ALIGNMENT
C0EC	88		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	2009C1		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 \$DD04	!SET TIMER
C112	A900		LDA #\$00	
C114	8D05DD		STA \$DD05	
C117	A901		LDA #\$01	!WAIT FOR TIMER
C119	2C0DDDD	WBIT2	BIT \$DD0D	
C11C	F0FB		BEQ WBIT2	
C11E	A501		LDA \$01	!TOGGLE WRITE BIT
C120	4908		EOR #\$08	! IN 6510 REGISTER
C122	8501		STA \$01	

C124	EE20D0	INC \$D020	!SHOW IT IS WORKING
C127	A919	LDA #\$19	
C129	8D0EDD	STA \$DD0E	!START TIMER
C12C	A901	LDA #\$01	!WAIT FOR TIMER
C12E	2C0DDD WBITS	BIT \$DD0D	
C131	F0FB	BEQ WBITS	
C133	A501	LDA \$01	!TOGGLE WRITE BIT
C135	4908	EOR #\$08	! IN 6510 REGISTER
C137	8501	STA \$01	
C139	A919	LDA #\$19	
C13B	8D0EDD	STA \$DD0E	!START TIMER
C13E	60	RTS	
C13F	!		
C13F	!THE LOADER STARTS HERE		
C13F	!		
C13F	A005 LOADER	LDY #\$05	
C141	A920	LDA #\$20	!BLANK OUT 'READY.'
C143	995004 BLLOOP	STA \$0450.Y	
C146	88	DEY	
C147	10FA	BPL BLLOOP	
C149	78	SEI	!DISABLE IRQ
C14A	A905	LDA #\$05	
C14C	8501	STA \$01	!START TAPE
C14E	A91F	LDA #\$1F	!DISABLE KEYBOARD
C150	8D0DDD	STA \$DD0D	
C153	8D0DDC	STA \$DC0D	
C156	AD0DDD	LDA \$DD0D	
C159	AD0DDC	LDA \$DC0D	
C15C	A968	LDA #\$68	!SET TIMER
C15E	8D04DC	STA \$DC04	
C161	A903	LDA #\$03	
C163	8D05DC	STA \$DC05	
C166	A990	LDA #\$90	!ENABLE TAPE IRQ
C168	8D0DDC	STA \$DC0D	
C16B	A951	LDA #\$51	!SET IRQ VECTOR
C16D	8DFEFF	STA \$FFFE	! TO POINT TO
C170	A903	LDA #\$03	! LOAD ROUTINE
C172	8DFFFF	STA \$FFFF	
C175	A977	LDA #\$77	!SET NMI VECTOR
C177	8DFAFF	STA \$FFFA	! TO POINT TO
C17A	A903	LDA #\$03	! AN RTI
C17C	8DFBFF	STA \$FFFB	
C17F	A900	LDA #\$00	!CLEAR LOADED FLAG
C181	8502	STA \$02	
C183	58	CLI	
C184	4CDD03	JMP \$03DD	!WAIT FOR END OF LOAD
C187	!		
C187	!*= \$02F3		
C187	!		
C187	A983	LDA #\$83	!RESET WARM START
C189	8D0203	STA \$0302	! VECTOR
C18C	A9A4	LDA #\$A4	
C18E	8D0303	STA \$0303	
C191	4C84FF	JMP \$FF84	!RESET I/O
C194	8BE3AB	WOR \$E38B, \$02AB	
C198	!		
C198	932A20 FLNAME	TXT "C"	
C1A8	!14 SPACES		
C1A8	48	PHA	!IRQ ENTRY POINT
C1A9	98	TYA	
C1AA	48	PHA	
C1AB	AD05DC	LDA \$DC05	!GET TIMER HI BYTE
C1AE	A019	LDY #\$19	!RESTART TIMER
C1B0	8C0EDC	STY \$DC0E	
C1B3	4902	EOR #\$02	!FLIP BIT 1
C1B5	4A	LSR A	! AND SHIFT TO
C1B6	4A	LSR A	! CARRY
C1B7	26A9	ROL \$A9	!MOVE BIT INTO
C1B9	A5A9	LDA \$A9	! BYTE READ
C1BB	9002	BCC BITGOT	!WHEN BYTE READ

120 The Commodore 64 Kernal and Hardware Revealed

C1BD	B00D		BCS EXIT	!NOT COMPLETE BYTE
C1BF	C940	BITGOT	CMP #\$40	!ALIGNMENT?
C1C1	D009		BNE EXIT	!NO
C1C3	A916		LDA #\$16	!SET NEW ADDRESS
C1C5	8D6503		STA \$0365	
C1C8	A9FE	EX1	LDA #\$FE	!GET READY FOR
C1CA	85A9		STA \$A9	! A NEW BYTE
C1CC	AD0DDC	EXIT	LDA \$DC0D	!CLEAR IRQ
C1CF	68		PLA	
C1D0	A8		TAY	
C1D1	68		PLA	
C1D2	40		RTI	!EXIT IRQ
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 \$0365	! ALIGNMENT ROUTINE
C1E0	D0E6		BNE EX1	
C1E2	A930	BITGT2	LDA #\$30	!SET NEW ADDRESS
C1E4	8D6503		STA \$0365	! TO READ IN LOAD
C1E7	A900		LDA #\$00	! ADDRESSES. CLEAR
C1E9	85C1		STA \$C1	! CHECKSUM
C1EB	F0DB		BEQ EX1	
C1ED		!		
C1ED	85FB		STA \$FB	!STORE LOAD ADDRESS
C1EF	EE9703		INC \$0397	!INCREASE STORE
C1F2	AD9703		LDA \$0397	! UNTIL 4 BYTES
C1F5	C9FF		CMP #\$FF	
C1F7	D0CF		BNE EX1	!NOT YET
C1F9	A943		LDA #\$43	!STORE NEW ADDRESS
C1FB	8D6503		STA \$0365	!FOR READING FILE
C1FE	D0C8		BNE EX1	
C200		!		
C200	A000		LDY #\$00	
C202	91FB		STA (\$FB),Y	!STORE A BYTE
C204	45C1		EOR \$C1	!CALCULATE CHECKSUM
C206	85C1		STA \$C1	
C208	EE00D8		INC \$D800	!SHOW IT IS WORKING
C20B	E6FB		INC \$FB	!INCREASE ADDRESS
C20D	D002		BNE BITGT5	
C20F	E6FC		INC \$FC	
C211	A5FB	BITGT5	LDA \$FB	!END OF LOAD?
C213	C5FD		CMP \$FD	
C215	A5FC		LDA \$FC	
C217	E5FE		SBC \$FE	
C219	90AD		BCC EX1	!NOT YET
C21B	A965		LDA #\$65	!NEW ADDRESS FOR
C21D	8D6503		STA \$0365	!CHECKSUM
C220	D0A6		BNE EX1	
C222		!		
C222	85C2		STA \$C2	!STORE CHECKSUM
C224	A9FF		LDA #\$FF	!FLAG END OF LOAD
C226	8502		STA \$02	
C228	A902		LDA #\$02	!RESET BRANCH TO
C22A	8D6503		STA \$0365	! ALIGNMENT
C22D	A9FB		LDA \$FB	
C22F	8D9703		STA \$0397	
C232	D094		BNE EX1	
C234		!		
C234		!*= \$03DD		
C234		!		
C234	A502	PAUSE	LDA \$02	!WAIT FOR FILE
C236	F0FC		BEQ PAUSE	! TO BACKGROUND LOAD
C238	A900		LDA #\$00	
C23A	8502		STA \$02	
C23C	A907		LDA #\$07	
C23E	8501		STA \$01	!RESET KERNAL ROM
C240	20F302		JSR \$02F3	

C243 2063A6	JSR \$A663	!CLR
C246 A5C1	LDA \$C1	!COMPARE CHECKSUMS
C248 C5C2	CMP \$C2	
C24A D003	BNE LODERR	!DIFFERENT..ERROR
C24C 4C74A4	JMP \$A474	!GO TO 'READY.'
C24F		
C24F A21D LODERR	LDX #\$1D	!'?LOAD ERROR'
C251 4C37A4	JMP \$A437	!SEND ERROR

Program 6.

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:

Method 1	:1 minute	
Disk	:1 minute	10 seconds
Method 2	:1 minute	25 seconds
Normal tape	:8 minutes	40 seconds

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 I/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 charget 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:

	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

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.

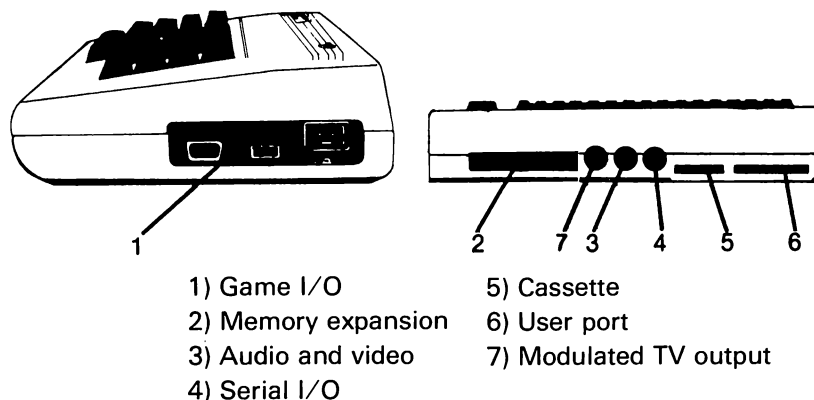
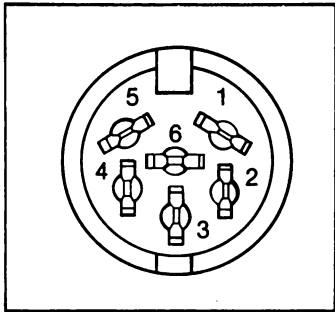


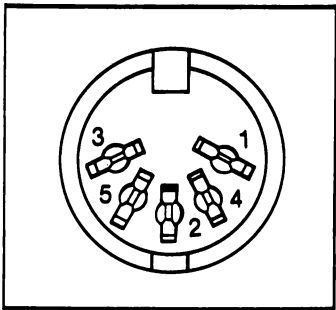
Fig. 5.1. The position of the different CBM 64 I/O outputs.

SERIAL I/O



PIN #	TYPE
1	SERIAL SRQ IN
2	GND
3	SERIAL ATN IN/OUT
4	SERIAL CLK IN/OUT
5	SERIAL DATA IN/OUT
6	NC

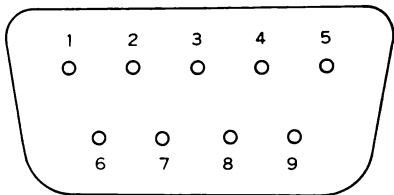
AUDIO/VIDEO



PIN #	TYPE
1	LUMINANCE
2	GND
3	AUDIO OUT
4	COMP VIDEO
5	AUDIO IN

GAME I/O

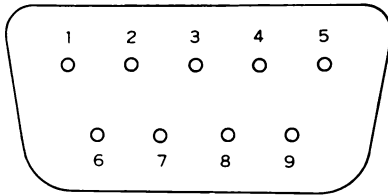
Port #1



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

Fig. 5.1. (contd.)

Port #2



PIN #	TYPE	NOTE
1	JOY0	MAX. 100mA
2	JOY1	
3	JOY2	
4	JOY3	
5	POT Y	
6	BUTTON B	
7	+ 5V	
8	GND	
9	POT X	

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 \$DC00 to \$DDFF (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 I/O 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

PIN #	TYPE	NOTE	PIN #	TYPE	NOTE
1	GND	100mA MAX.	A	GND	
2	+5V		B	FLAG	
3	RESET		C	PB0	
4	CNT1		D	PB1	
5	SP1		E	PB2	
6	CNT2		F	PB3	
7	SP2		H	PB4	
8	PC2		J	PB5	
9	SERIAL ATN IN		K	PB6	
10	+9V AC	100mA MAX.	L	PB7	
11	+9V AC		M	PA2	
12	GND		N	GND	

Fig. 5.2. The allocation and function of pins on the user port connector.

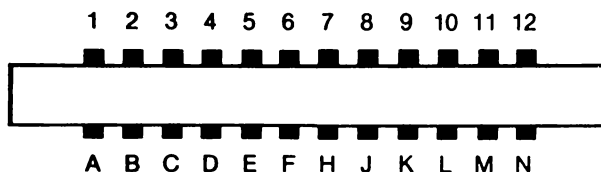


Fig. 5.3. User port edge connector line definition.

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/Ø, 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 $\phi 2$ clock line. The individual registers are addressed by the register select lines connected to the bottom address lines AØ–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 ($\phi 2$) – data transfers between the 6526 and the processor take place only when the $\phi 2$ clock is high. This clock also acts as a time base for the internal 6526 timers and shift register. On the CBM 64 the $\phi 2$ clock is derived from the 651Ø microprocessor chip which in turn is derived from the $\phi 0$ clock produced by the VIC chip. The $\phi 2$ 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 (RSØ, RS1, RS2, RS3) – the four register select lines are connected to the processor address bus lines AØ – 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 $\phi 2$ 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 ($\overline{\text{FLAG}}$, PC) – the $\overline{\text{FLAG}}$ 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.

I/O line number	Data direction	DDR contents if line = in	DDR contents for example
0	in	1	0
1	in	2	0
2	in	4	0
3	in	8	0
4	out	16	16
5	out	32	32
6	out	64	64
7	out	128	128
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.

I/O line number	Data direction of line	High or low	Value of line in I/O reg
0	in	–	–
1	in	–	–
2	in	–	–
3	in	–	–
4	out	high	16
5	out	low	0
6	out	low	0
7	out	high	128
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 = PEEK (56577) 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 $\overline{\text{FLAG}}$ line is an input which, on receiving a negative transition, will set the $\overline{\text{FLAG}}$ bit in the interrupt control register. The $\overline{\text{FLAG}}$ 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:

CIA#1 IRQ interrupt

- Timer A – System 60 Hz interrupt
 - Real time clock updating
 - Keyboard scanning
 - Cassette read/write timing
 - User programmable functions
- Timer B – Cassette read/write timing
 - Serial port timing
 - User programmable functions
- TOD – User programmable functions

Note that Timer A is used for both updating the kernal 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

- Timer A – RS232 port I/O timing
 - User programmable functions
- Timer B – RS232 port I/O timing
 - User programmable functions
- TOD – User programmable

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 \$DC04	decimal 56324	
	CIA#2 - Hex \$DD04	decimal 56580	
Register 6	- Timer A high order byte		
	CIA#1 - Hex \$DC05	decimal 56325	
	CIA#2 - Hex \$DD05	decimal 56581	
Register 7	- Timer B low order byte		
	CIA#1 - Hex \$DC06	decimal 56326	
	CIA#2 - Hex \$DD06	decimal 56582	
Register 8	- Timer B high order byte		
	CIA#1 - Hex \$DC07	decimal 56327	
	CIA#2 - Hex \$DD07	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 0.669 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 interrupts 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 POKE56583,0:POKE56582,0
20 POKE56581,255:POKE56580,255
30 POKE56590,16+1:POKE56591,64+16+8+1
35 T1=TI
40 IF(PEEK(56589)AND2)<2THEN40
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:

Reg #	CIA#1 location	CIA#2 location	Function
8	\$DC08 - 56328	\$DD08 - 56584	TOD 1/10 secs
9	\$DC09 - 56329	\$DD09 - 56585	TOD secs
10	\$DC0A - 56330	\$DD0A - 56586	TOD mins
11	\$DC0B - 56331	\$DD0B - 56587	TOD hours

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:

Register	Byte	7	6	5	4	3	2	1	0
8 - TOD 1/10 sec		0	0	0	0	T8	T4	T2	T1
9 - TOD seconds		0	shi4	shi2	shi1	slo8	slo4	slo2	slo1
10 - TOD minutes		0	mhi4	mhi2	mhi1	mlo8	mlo4	mlo2	mlo1
11 - TOD hours	PM flag		0	0	hhi1	hlo8	hlo4	hlo2	hlo1

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.

```
0 CIA=56576: REM CIA#2. FOR CIA#1, CIA=56320
1 POKECIA+14,PEEK(CIA+14)OR128
2 MO=0:GOSUB 1000 SET TOD TIME
4 MO=1:GOSUB 1000 SET TOD ALARM
5 PRINT"  :  :  M"
10 H=PEEK(CIA+11): REM HOURS/AM OR PM
20 M=PEEK(CIA+10): REM MINUTES
30 S=PEEK(CIA+9): REM SECONDS
40 S10=PEEK(CIA+8): REM 1/10 SECONDS
50 POKE1024,((HAND112)/16)OR48
60 POKE1025,((HAND15)OR48
70 POKE1027,((MAND240)/16)OR48
80 POKE1028,((MAND15)OR48
90 POKE1030,((SAND240)/16)OR48
```

```

100 POKE1031,(SAND15)OR48
110 POKE1033,S10OR48
120 B8=16:IF<HAND128><128 THEN B8=1
130 POKE1035,B8
140 IF PEEK<CIA+13>AND4=4THENPRINT"TIMER ALARM":END
150 GOTO 10
997 REM
998 REM SET CLOCK OR TIME
999 REM
1000 IF MO=0 THEN PRINT"SET TIME":POKECIA+15,PEEK<CIA+15>AND127:GOTO 1020
1010 PRINT"SET ALARM":POKECIA+15,PEEK<CIA+15>OR128
1020 INPUT"HOURS";H:H=INT(H)
1030 IF H<1 OR H>12 THEN PRINT"":GOTO 1020
1040 PRINT"AM OR PM ?";
1050 GETA$:IFA$<>"A"AND A$<>"P"THEN1050
1060 PRINTA$
1061 IF MO=1 THEN 1070
1065 IF H=12 AND A$="A" THEN A$="P":GOTO1070
1066 IF H=12 AND A$="P" THEN A$="A"
1070 B=128:IFA$="A"THENB=0
1080 POKECIA+11,B+INT<H/10>*16+H-INT<H/10>*10
1090 INPUT"MINUTES";M:M=INT(M)
1100 IF M<0 OR M>59 THEN PRINT"":GOTO1090
1110 POKECIA+10,INT<M/10>*16+M-INT<M/10>*10
1120 INPUT"SECONDS";S:S=INT(S)
1130 IF S<0 OR S>59 THEN PRINT"":GOTO1120
1140 POKECIA+9,INT<S/10>*16+S-INT<S/10>*10
1145 IF MO=1 THEN 1170
1150 PRINT"PRESS ANY KEY TO START TIMER"
1160 GETA$:IFA$=""THEN1160
1170 POKECIA+8,0:RETURN

```

Program 8.

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 $\phi 2$ 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 POKE56580,2:POKE56581,0
20 POKE56590,64+16+1
25 FORI=1TOLEN(A$)
26 J=ASC(MID$(A$,I,1))
30 POKE56588,J:PRINTCHR$(J);
50 IF (PEEK(56589)AND8)<8 THEN 50
60 NEXT:POKE56588,0
65 IF (PEEK(56589)AND8)<8 THEN 65
70 POKE 56590,PEEK(56590) AND (128+63)
80 POKE56588,0
90 IF (PEEK(56589)AND8)<8 THEN 90
100 I=PEEK(56588):IFI<>0THENPRINTCHR$(I);:GOTO90
110 GOTO5

10 POKE56579,1
20 POKE56326,255:POKE56327,255
30 POKE56335,32+16+1
40 PRINT"Q"
50 PRINT"Q"65535-(PEEK(56326)+PEEK(56327)*256)
60 FORI=0TO100:NEXT
70 GETA$:IFA$=""THEN70
80 POKE56577,1:POKE56577,0
90 GOTO 50

10 POKE56590,PEEK(56590)AND(128+63)
20 POKE56579,1
30 GETA$:IFA$=""THEN30
40 POKE56577,0:POKE56577,1
50 IF(PEEK(56589)AND8)<8THEN30
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:

ICR bit #	Interrupt source
0	Underflow from Timer A
1	Underflow from Timer B
2	TOD clock alarm
3	Serial data register full/empty
4	Flag line input

Bits 5 and 6 are not used, while bit 7 is used to set or clear the ICR mask register.

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:

Control register A

Bit	State	Function
0	0	start Timer A
	1	stop Timer A; in one shot mode this bit is reset on underflow
1	0	line PB6 functions normally as an I/O line
	1	output from Timer A appears on line PB6
2	0	pulse output on PB6 (only if bit 1 set)
	1	toggle output on PB6 (only if bit 1 set)
3	0	Timer A in continuous running mode
	1	Timer A in one shot mode

Control register A

Bit	State	Function
4	0	no effect
	1	forces the Timer A counter to be loaded from the Timer A latch
5	0	Timer A counts $\phi 2$ clock pulses
	1	Timer A counts positive transitions on the CNT line
6	0	SP line in input mode using external shift pulses on CNT line
	1	SP line in output mode. CNT sources shift pulses
7	0	TOD clock timing pulse at 60 Hz (use on US machines)
	1	TOD clock timing pulse at 50 Hz (use on UK machines)

Control Register B

Bit	State	Function
0	0	stop Timer B
	1	start Timer B; in one shot mode this bit is reset on underflow
1	0	normal I/O operation on the PB7 line
	1	output from Timer B appears on line PB7
2	0	pulse output on PB7 (only if bit 1 of CRB set)
	1	toggled output on PB7 (only if bit 1 of CRB set)
3	0	Timer B in continuous running mode
	1	Timer B in one shot mode
3	0	no effect
	1	forces the Timer B counter to be loaded from the Timer B latch
5,6		these two bits select one of four input modes for Timer B:
Bit 6	Bit 5	Mode
0	0	Timer B counts $\phi 2$ clock pulses
0	1	Timer B counts positive CNT transitions
1	0	Timer B counts Timer A underflow pulses
1	1	Timer B counts Timer A underflow pulses while the CNT line is high
7	0	writing to TOD registers will set TOD clock
	1	writing to TOD registers will set TOD alarm

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.

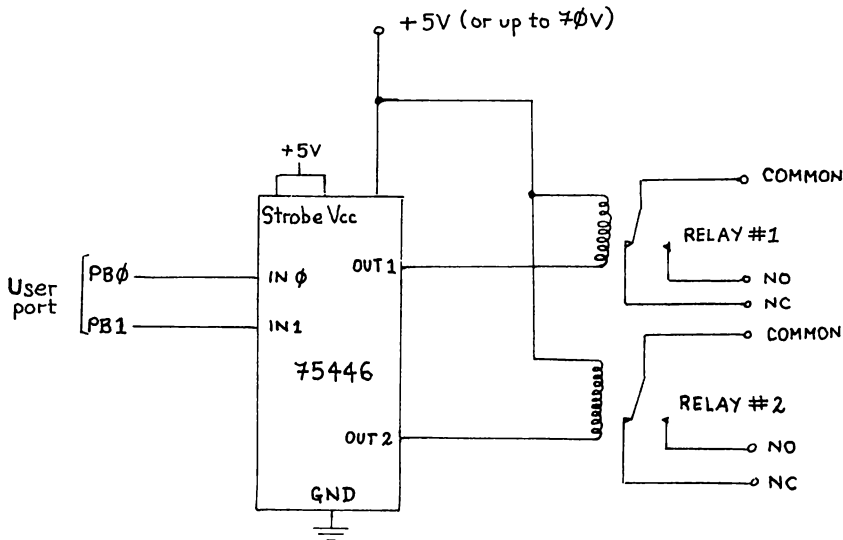


Fig. 5.4. Dual relay control circuit.

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.


```

220 DATAN,0,R,P,J,#,OF
230 DATAC,D,H,M,X,@,IS
240 DATAF,B,W,Y,K,/,TO
250 DATA0,2,4,6,8,,IN
260 DATA1,3,5,7,9,!,THAT
270 DATA,+, -, *, /, ^, $
280 DATA", " , ? , = , % , ( , ) , <
290 DATA&, ^, " ; , " : , [ , ] , >
400 DATA" SP ",1,7,=," <> ", "GOTO ", " ON "
410 DATA0,2,6,"", ">,"INPUT"," VAL "
420 DATA3,4,5,#,<,"POKE ","CHR$ "
430 DATA8,9,,," " <= ", "PEEK ","STR$ "
440 DATA),(<,$,^," ">= ", "LEN ", "LEFT$ "
450 DATA+,-,*,/, " ", "ASC ", "RIGHT$ "
460 DATA"LIST ", "DATA ", "IF ", "GOSUB", " DIM ", "TRON ", "MID$ "
470 DATA"RUN ", "CONT ", "THEN ", "RTRN ", "STOP ", "TROFF", " LOG "
480 DATA"READ ", "INT ", "ABS ", "SGN ", "RND ", "SQR ", "EXP "
490 DATA"PRINT", " FOR ", "NEXT ", "SIN ", "COS ", "TAN ", " " "
500 DATA" ",1,7,=," <> ,GOTO,ON
510 DATA0,2,6,"", ">,"INPUT,VAL<
520 DATA3,4,5,#,<,"POKE<,"CHR$<
530 DATA8,9,,," " <=,"PEEK<,"STR$<
540 DATA),(<,$,^," ">=,"LEN<,"LEFT$<
550 DATA+,-,*,/, " ", "ASC<,"RIGHT$<
560 DATALIST,DATA,IF,GOSUB,DIM,TRACEON,MID$<
570 DATARUN,CONT,THEN,RETURN,STOP,TRACEOFF,LOG<
580 DATAREAD,INT<,"ABS<,"SGN<,"RND<,"SQR<,"EXP<
590 DATAPRINT,FOR,NEXT,SIN<,"COS<,"TAN<," "
1000 RESTORE
1010 DIMDF$(2,6),DA$(1,9,6),TA$(1,9,6),DV$(20),TV$(20)
1020 FORI=0TO2:FORJ=0TO6
1030 READDF$(I,J)
1040 NEXTJ:NEXTI
1050 FORI=0TO1
1060 FORJ=0TO9
1070 FORK=0TO6
1080 READDA$(I,J,K)
1090 IFLEN(DA$(I,J,K))=1THENDDA$(I,J,K)=" "+DA$(I,J,K)+" "
1100 NEXTK:NEXTJ
1110 FORJ=0TO9
1120 FORK=0TO6
1130 READTA$(I,J,K)
1140 NEXTK:NEXTJ
1150 NEXTI
1170 TA$(0,5,5)=CHR$(34)
1190 TA$(1,3,3)=CHR$(34)
1200 FORI=0TO20
1210 DV$(I)=" ":TV$(I)=" "
1220 NEXTI
1700 UP=56577:UX=1
1710 VN=0:M=0:A$="":UC=1:ED=0:HP=0:TV=0:NV=0
1720 PD=70:PQ=50:FM=0
1730 GOSUB 20000
2000 PRINT"*****ADVICE";
2010 PRINT"*****HOME";
2020 PRINT"*****";
2030 FORI=0TO9:PRINT" ";
2040 FORJ=0TO6:PRINTDA$(M,I,J);
2050 NEXTJ:NEXTI
2055 PRINT
2060 PRINT"*****";"*****";"*VARIABLES";
2070 PRINT"*****";
2080 FORI=0TO2:PRINT" ";
2090 FORJ=0TO6:PRINTDV$(I*7+J);
2100 NEXTJ:NEXTI
2110 PRINT"*****FUNCTIONS";
2120 PRINT"*****";
2130 FORI=0TO2:PRINT" ";
2140 FORJ=0TO6:PRINTDF$(I,J);
2150 NEXTJ:NEXTI

```

```

2160 PRINT"☐";A$;
3000 PRINT"☐":PRINT
3005 IFHP=0GOTO3040
3010 PRINT"☐☐☐☐☐ADVICE ☐SELECT ANY BOX FOR HELP☐";
3020 HP=HP+1
3030 GOTO3200
3040 IFLEN(A$)+LEN(N$)+LEN(Y$)+LEN(Z$)<160GOTO3070
3050 PRINT"☐☐☐☐☐WARNING☐PRINT STRING — NOW!☐";GOTO3200
3070 FORI=1TOLEN(N$)
3080 NB$=MID$(N$,I,1)
3090 IFUC>0GOTO3130
3110 IFNB$<"A"ORNB$>"Z"GOTO3130
3120 NB$=CHR$(ASC(NB$)+128)
3130 A$=A$+NB$
3141 IFUC=2THENGOSUB20100
3142 IFUC=2THENUC=0
3143 NEXTI
3145 N$=""
3146 IFUC=3THENUC=2
3150 PRINT"☐";A$;
3160 IFED=0GOTO3180
3170 PRINT"☐";Y$;"☐";Z$;
3180 PRINTFL$;FL$=""
3200 PRINT"☐":PRINT:PRINT"☐☐☐☐☐HOME☐";
3205 IF(PEEK(UP)ANDUX)=0THEN3205
3210 IF(PEEK(UP)ANDUX)GOTO3210
3212 IFHP>0THEN3220
3213 IFED>0THEN3220
3214 IFNV>0THEN3220
3215 PRINT"☐☐☐☐☐ADVICE
3220 PRINT"☐☐☐☐☐HOME";
3230 PRINT"☐☐☐☐☐";
3240 FORI=0TO15
3250 PRINT"☐ > ☐";
3260 FORJ=1TOPD:IF(PEEK(UP)ANDUX)GOTO3400
3265 NEXTJ
3270 PRINT"☐☐☐☐☐ ☐☐☐☐☐";
3280 PRINT"☐";:IFI=9ORI=12THENPRINT"☐";
3290 NEXTI
3310 GOTO3200
3400 IF(PEEK(UP)ANDUX)GOTO3400
3420 PRINT"☐☐☐☐☐ ";
3430 IFI>9GOTO3600
3440 FORJ=0TO6
3450 PRINT"☐";DA$(M,I,J);"☐";
3460 FORK=1TOPQ:IF(PEEK(UP)ANDUX)GOTO3490
3465 NEXTK
3470 PRINT"☐☐☐☐☐";DA$(M,I,J);
3480 NEXTJ
3485 PRINT"☐☐☐☐☐";DA$(M,I,J-1);:GOTO3200
3490 PRINT"☐☐☐☐☐";DA$(M,I,J);
3500 IFHP=2GOTO40000
3505 IF NV>0 GOTO7720
3510 N$=TA$(M,I,J):GOTO3000
3600 IFI>12GOTO3800
3610 II=I-10
3620 FORJ=0TO6:PRINT"☐";DV$(II*7+J);"☐";
3630 FORK=1TOPQ:IF(PEEK(UP)ANDUX)GOTO3670
3635 NEXTK
3650 PRINT"☐☐☐☐☐";DV$(II*7+J);
3660 NEXTJ
3665 PRINT"☐☐☐☐☐";DV$(II*7+J-1);:GOTO3200
3670 PRINT"☐☐☐☐☐";DV$(II*7+J);
3680 IFHP=2GOTO42000
3685 IF NV>0 GOTO7740
3690 N$=TV$(II*7+J):GOTO3000
3800 II=I-13
3810 FORJ=0TO6:PRINT"☐";DV$(II,J);"☐";
3830 FORK=1TOPQ:IF(PEEK(UP)ANDUX)GOTO3860
3835 NEXTK

```

```

3840 PRINT"#####";DF$(II,J);
3850 NEXTJ
3855 PRINT"#####";DF$(II,J-1);:GOTO3200
3860 PRINT"#####";DF$(II,J);
3870 SS=(I*7+J)+1
3875 IF NY>0 GOTO7720
3880 IFHP=2GOTO44000
4000 IFSS>7GOTO4020
4010 ONSSGOTO4100,4300,4500,4700,4900,5100,5300
4020 IFSS>14GOTO4040
4030 ONSS-7GOTO5500,5700,5900,6100,6300,6500,6700
4040 ONSS-14GOTO6900,7100,7300,7500,7700,7900,8100
4100 REM SET TO LOWER CASE
4110 UC=0:GOSUB20100:GOTO3200
4300 REM BEGIN A SENTENCE
4310 UC=3:GOSUB20000:A$=A$+" ";:GOTO3000
4500 REM DELETE CHARACTER
4505 IFLEN(A$)=0GOTO3200
4510 IFLEN(A$)>1GOTO4530
4520 A$="":FL$="":GOTO3000
4530 A$=LEFT$(A$,LEN(A$)-1):FL$="":GOTO3000
4700 REM DELETE WORD
4701 IFLEN(A$)=0GOTO3000
4702 IFLEN(A$)=1GOTO4745
4705 FORI=LEN(A$)TO2STEP-1
4710 NB$=MID$(A$,I,1)
4720 IFNB$="":GOTO4750
4730 A$=LEFT$(A$,I-1):FL$=FL$+" "
4740 NEXTI
4745 IFLEN(A$)=1ANDLEFT$(A$,1)<>" "THENGOSUB4800
4750 GOTO3000
4800 A$="":FL$=FL$+" ":RETURN
4900 REM NOT DEFINED
4910 GOTO3200
5100 REM NOT DEFINED
5110 GOTO3200
5300 REM CR-SEND CR/LF TO PRINTER
5310 OPENPR,PR
5320 PRINT#PR
5330 CLOSE PR
5340 GOTO3000
5500 REM SET TO UPPER CASE
5510 UC=1:GOSUB20000:GOTO3200
5700 REM PARAGRAPH
5710 OPENPR,PR
5720 GOSUB 21000:GOSUB 20900
5725 GOSUB 20000
5730 PRINT#PR:PRINT#PR:PRINT#PR
5740 A$=" "
5750 UC=3
5760 CLOSE PR
5770 GOTO3000
5900 REM PRINT CONTENTS OF A$
5910 OPEN PR,PR
5920 GOSUB21000:GOSUB20900
5930 A$=" "
5940 CLOSEPR
5950 GOTO3000
6100 REM SEND LINE TO MICRO
6110 OPEN MC,MC
6120 PRINT#MC,A$
6125 GOSUB20900
6130 A$=" "
6140 CLOSEMC
6150 GOTO3000
6300 REM ADD A MAKE A$ THE NEXT VARIABLE
6310 IF VNC21 THEN 6340
6320 PRINT"NO ROOM*ADVICE - NO ROOM FOR NEW VARIABLE";
6330 GOTO3000
6335 IF A$=""THEN3000

```

```

6340 TV$(VN)=A$
6350 DV$(VN)=LEFT$(TV$(VN)+      ",5)
6360 DV$(VN)=LEFT$(DV$(VN),4)
6365 DV$(VN)=DV$(VN)+CHR$(32)
6370 GOSUB22000
6380 VN=VN+1
6390 GOTO3000
6500 REM SEND ASCII 3 TO MICRO (BREAK-IN)
6510 OPEN MC,MC
6520 PRINT#MC,CHR$(3);CHR$(3);
6530 CLOSE MC
6540 GOTO3000
6700 REM UNDEFINED
6710 GOTO 3200
6900 REM TEXT MODE
6910 M=0:GOTO2000
7100 REM PROGRAM MODE
7110 M=1:UC=1:GOSUB20000:GOTO2000
7300 REM USER MODE
7310 GOTO30000
7500 REM HELP SUB-SYSTEM
7510 HP=1:GOTO3000
7700 REM DELETE VARIABLE
7701 IF VN=0 GOTO7710
7703 PRINT"*****ADVICE - NO VARIABLES TO REMOVE";
7705 GOTO3000
7710 PRINT"*****ADVICE - SELECT VARIABLE TO BE REMOVED";
7713 NV=1
7715 GOTO3000
7720 REM UNKNOWN VARIABLE
7725 PRINT"*****ADVICE - NO VARIABLE SELECTED!      ";
7730 NV=0
7735 GOTO3000
7740 REM DELETE SELECTED VARIABLE
7745 PRINT"*****ADVICE - O.K.                        ";
7750 FOR K=11*7+J TO 19
7755 DV$(K)=DV$(K+1)
7760 TV$(K)=TV$(K+1)
7762 NEXT K
7765 DV$(20)="      "
7770 TV$(20)=""
7780 GOSUB 22000
7785 NV=0
7790 VN=VN-1
7795 GOTO3000
7900 REM UNDEFINED
7910 GOTO3200
8100 REM DELETE BUFFER
8110 GOSUB20900:A$="":GOTO3000
10000 PRINT"*****ADVICE NO HELP IN THIS VERSION";
10005 HP=0
10010 GOTO3000
20000 PRINT"*****LC* ";
20010 PRINT"***** UC* ";
20020 DF$(1,0)=" UC* ":DF$(0,0)=" LC  "
20030 RETURN
20100 PRINT"*****LC* ";
20110 PRINT"***** UC  ";
20120 DF$(1,0)=" UC  ":DF$(0,0)=" LC* "
20130 RETURN
20900 PRINT" ";
20910 FORL=1TO10
20920 PRINT"      ";
20930 NEXT
20940 RETURN
21000 PRINT#PR,A$
21010 RETURN
22000 PRINT"*****";
22010 FORI=0TO2:PRINT" ";
22020 FORJ=0TO6:PRINTDV$(I*7+J);

```

```

22030 NEXT J:NEXT I
22040 RETURN
30000 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"***";
30180 IF(PEEK(UP)AND UX)=0 GOTO 30180
30190 IF(PEEK(UP)AND UX) GOTO 30190
30200 PRINT"II II";
30210 FOR I=1 TO 14
30220 PRINT"***";
30230 FOR J=1 TO PD
30240 IF(PEEK(UP)AND UX) GOTO 30300
30250 NEXT J
30260 PRINT"II II";
30270 NEXT I
30280 PRINT"II ";
30290 GOTO 30170
30300 PRINT"II ";
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 30600
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 GOTO30170
30700 REM BBC2
30710 POKE UP,(PEEK(UP)OR3)AND254
30720 GOTO30170
30800 REM ITV
30810 POKE UP,(PEEK(UP)OR3)AND253
30820 GOTO30170
30900 REM RADIO ON
30910 POKE UP,PEEK(UP)AND247
30920 GOTO30170
31000 REM RADIO OFF
31010 POKE UP,PEEK(UP)OR8
31020 GOTO30170
31100 REM LAMP ON
31110 POKE UP,PEEK(UP)AND239
31120 GOTO30170
31200 REM LAMP OFF
31210 POKE UP,PEEK(UP)OR16
31220 GOTO30170
31300 REM DOWN FASTER
31310 PD=PD-10:IF PD<30 THEN PD=30
31320 GOTO30170
31400 REM DOWN SLOWER
31410 PD=PD+10
31420 GOTO30170
31500 REM ACROSS FASTER

```

```

31510 PQ=PQ-10:IF PQ<30 THEN PQ=30
31520 GOTO30170
31600 REM ACROSS SLOWER
31610 PQ=PQ+10
31620 GOTO30170
31700 REM RETURN
31710 GOTO2000
40000 REM HELP WITH LETTERS
40010 GOSUB 20900
40020 PRINT"BY SELECTING '";DA$(M,I,J);
40030 IF LEN(TA$(M,I,J))=0THEN PRINT" NOTHING"
40040 IF LEN(TA$(M,I,J))=1THEN PRINT" THE CHARACTER '";TA$(M,I,J);""
40050 IF LEN(TA$(M,I,J))>1THEN PRINT" THE STRING '";TA$(M,I,J);""
40060 PRINT"IS ADDED TO THE BUFFER."
40070 GOTO49000
42000 REM HELP WITH VARIABLES
42010 GOSUB 20900:PRINT"";
42020 IF LEN(TV$(II*7+J))>0 THEN 42060
42030 PRINT"THIS VARIABLE ('";II*7+J;") IS EMPTY;"
42040 PRINT"SEE +VAR AND -VAR."
42050 GOTO49000
42060 PRINT"BY SELECTING VARIABLE '";DV$(II*7+J);" THEN"
42070 PRINT"J";TV$(II*7+J);" IS ADDED"
42080 PRINT"TO THE BUFFER."
42090 GOTO49000
44000 REM HELP WITH FUNCTIONS
44005 GOSUB 20900:PRINT"";
44010 IF SS>7 GOTO 44030
44020 ON SS GOTO45000,45100,45200,45300,45400,45500,45600
44030 IF SS>14 GOTO 44050
44040 ON SS-7 GOTO 45700,45800,45900,46000,46100,46200,46300
44050 ON SS-14 GOTO 46400,46500,46600,46700,46800,46900,47000
45000 PRINT"" LC ■ FOLLOWING LETTERS WILL BE LOWER"
45010 PRINT"CASE. ALSO SEE ' UC '."
45020 GOTO49000
45100 PRINT"SENT ■ END OF SENTENCE, ADDS A FULL"
45110 PRINT"STOP AND THREE SPACES TO THE BUFFER"
45120 PRINT"AND MAKES THE NEXT LETTER A CAPITAL."
45130 GOTO49000
45200 PRINT"DEL ■ DELETES THE MOST RECENT"
45210 PRINT"CHARACTER IN BUFFER."
45220 GOTO49000
45300 PRINT"DELW ■ DELETES THE MOST RECENT WORD"
45310 PRINT"IN THE BUFFER, NOT INCLUDING SPACES."
45320 GOTO49000
45400 PRINT"NOT DEFINED - CODE AT 4900"
45410 GOTO49000
45500 PRINT"NOT DEFINED - CODE AT 5100"
45510 GOTO49000
45600 PRINT"NL ■ NEWLINE ON PRINTER, DOES NOT"
45610 PRINT"AFFECT BUFFER."
45620 GOTO49000
45700 PRINT"UC ■ FOLLOWING LETTERS WILL BE"
45710 PRINT"UPPER CASE - ALSO SEE ' LC '."
45720 GOTO49000
45800 PRINT"PARA ■ END OF PARAGRAPH, PRINTS"
45810 PRINT"BUFFER, THREE NEW LINES, AND SETS"
45820 PRINT"BUFFER TO THREE SPACES."
45830 GOTO49000
45900 PRINT"SEND ■ PRINTS CONTENTS OF BUFFER."
45910 PRINT"ERASES BUFFER AND TAKES A NEWLINE."
45920 GOTO49000
46000 PRINT"ESC ■ SENDS BUFFER TO 2ND. MICRO,"
46010 PRINT"AND ESCAPE CHARACTER."
46020 GOTO49000
46100 PRINT"+VAR ■ SAVES THE CONTENTS OF THE BUFFER"
46110 PRINT"IN THE NEXT FREE VARIABLE LOCATION."
46120 PRINT"SHOWS FIRST FOUR CHARACTERS."
46140 GOTO49000
46200 PRINT"↑C ■ SENDS CONTROL C (ASCII(3)) TO"

```

```

46210 PRINT"THE 2ND. MICRO, ACTS AS A BREAK IN."
46220 GOTO49000
46300 PRINT"NOT DEFINED - CODE AT 6700"
46310 GOTO49000
46400 PRINT"TEXT ■ GOTO TEXT MODE, DISPLAYS ASCII"
46410 PRINT"CHARACTER SET AND A SELECTION OF"
46420 PRINT"FREQUENTLY USED WORDS."
46430 GOTO49000
46500 PRINT"PROG ■ PROGRAMMING MODE, DISPLAYS CHAR-"
46510 PRINT"ACTERS AND WORDS USED IN BASIC. SETS"
46520 PRINT"UC, VARIABLES MAY BE DEFINED IN TEXT."
46540 GOTO49000
46600 PRINT"USER ■ USE TO CONTROL EXTERNAL"
46610 PRINT"EQUIPMENT (T.V., RADIO ETC.) AND"
46620 PRINT"ALTER CURSOR SCAN SPEED."
46630 GOTO49000
46700 PRINT"HELP ■ A HELP SYSTEM. USE AT FIRST FOR"
46710 PRINT"GENERAL INFORMATION, THEN TO CHECK"
46720 PRINT"CONTENTS OF VARIABLES - HELP/VARIABLE."
46730 GOTO49000
46800 PRINT"VAR ■ REMOVES SELECTED VARIABLE AND"
46810 PRINT"SHIFTS REMAINING ONES TO FILL SPACE."
46820 GOTO49000
46900 PRINT"NOT DEFINED - CODE AT 7900"
46910 GOTO49000
47000 PRINT"DEL ■ CLEARS BUFFER - NO OTHER EFFECT."
49000 PRINT"NOADVICE - ON/OFF TO RETURN          ";
49010 IF(PEEK(UP)AND UX)=0 GOTO 49010
49020 IF(PEEK(UP)AND UX) GOTO49020
49030 GOSUB 20900
49040 PRINT"NOADVICE - O.K.          ";
49050 HP=0
49060 GOTO 3000

```

Program 10.

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

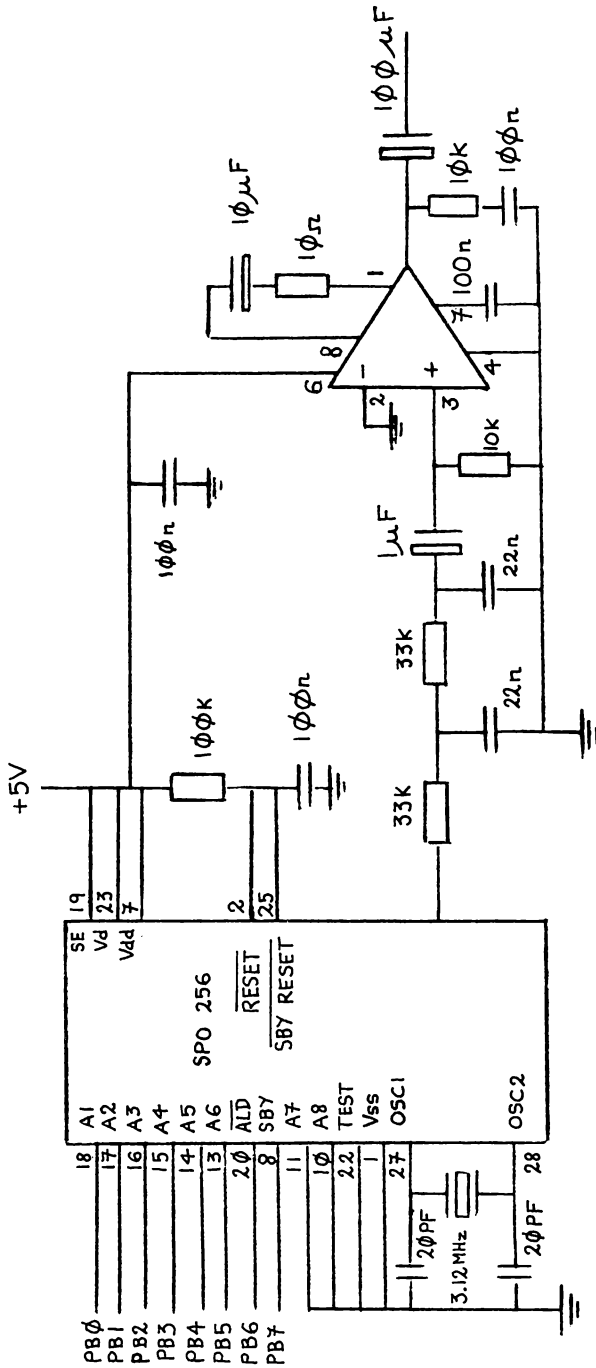


Fig. 5.6. Voice synthesiser circuit.

Table 5.1. Allophone address (Reproduced by courtesy of General Instruments).

Octal Address	Sample Allophone	Word	Duration	Octal Address	Sample Allophone	Word	Duration
000	PA1	PAUSE	10ms	040	/AW/	Out	250ms
001	PA2	PAUSE	30ms	041	/DD2/	Do	80ms
002	PA3	PAUSE	50ms	042	/GG3	Wig	120ms
003	PA4	PAUSE	100ms	043	/VV/	Vest	130ms
004	PA5	PAUSE	200ms	044	/GG1/	Guest	80ms
005	/OY/	Boy	290ms	045	/SH/	Ship	120ms
006	/AY/	Sky	170ms	046	/ZH/	Azure	130ms
007	/EH/	End	50ms	047	/RR2/	Brain	80ms
010	/KK3/	Comb	80ms	050	/FF/	Food	110ms
011	/PP/	Pow	150ms	051	/KK2/	Sky	140ms
012	/JH/	Dodge	100ms	052	/KK1/	Can't	120ms
013	/NN1/	Thin	170ms	053	/ZZ/	Zoo	150ms
014	/IH/	Sit	50ms	054	/NG	Anchor	200ms
015	/TT2/	To	100ms	055	/LL/	Lake	80ms
016	/RR1/	Rural	130ms	056	/WW/	Wool	140ms
017	/AX/	Succeed	50ms	057	/XR/	Repair	250ms
020	/MM/	Milk	180ms	060	/WH/	Whig	150ms
021	/TT1/	Part	80ms	061	/YY1/	Yes	90ms
022	/DH1/	They	140ms	062	/CH/	Church	150ms
023	/IY/	See	170ms	063	/ER1/	Fir	110ms
024	/EY/	Beige	200ms	064	/ER2/	Fir	210ms
025	/DD1/	Could	50ms	065	/OW/	Beau	170ms
026	/UW1/	To	60ms	066	/DH2/	They	180ms
027	/AO/	Aught	70ms	067	/SS/	Vest	60ms
030	/AA/	Hot	60ms	070	/NN2/	No	140ms
031	/YY2/	Yes	130ms	071	/HH2/	Hoe	130ms
032	/AE/	Hat	80ms	072	/OR/	Store	240ms
033	/HH1/	He	90ms	073	/AR/	Alarm	200ms
034	/BB1/	Business	40ms	074	/YR/	Clear	250ms
035	/TH/	Thin	130ms	075	/GG2/	Got	80ms
036	/UH/	Book	70ms	076	/EL/	Saddle	140ms
037	/UW2/	Food	170ms	077	/BB2/	Business	60ms

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.

```

1 REM *****
2 REM * SPEECH DATA TO USER PORT *
3 REM *****
4 REM
5 POKE54272+24,15
10 DD=56576
20 POKEDD+3,127
35 READD:IFD=-1THENEND
40 POKEDD+1,D OR64
50 POKEDD+1,D AND63
51 IFPEEK(DD+1)<128THEN51
60 GOTO 35
70 DATA42,59,45,3
80 DATA 3,39,12,50,59,21,3
85 DATA 3,26,11,33,3
90 DATA 3,11,12,41,3
91 DATA 3,46,52,41,3
93 DATA 40,58,3
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).

	Same sound represented by different letters	Different sounds represented by the same letter(s)
Vowels	meat feet Pete people penny	vein foreign deism deicer geisha
Consonants	ship tension precious nation	although ghastly cough

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

		Labial ¹	Labio-Dental ²	Inter-Dental ³	Alveo-lar ⁴	Palatal ⁵	Velar ⁶	Glottal ⁷
Stops:	Voiceless Voiced	PP BB			TT DD		KK GG	
Fricatives:	Voiceless Voiced	WH	FF VV	TH DH	SS ZZ	SH ZH*		HH
Affricates:	Voiceless Voiced					CH JH		
Nasals:	Voiced	MM			NN		NG*	
Resonants:	Voiced	WW			RR, LL	YY		
*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).

	FRONT	CENTRAL	BACK
HIGH	YR IY IH*		UW# UH*#
MID	EY EH* XR	ER AX*	OW# OY#
LOW	AE*	AW# AY AR AA*	AO*# OR#

* SHORT VOWELS # ROUNDED VOWELS

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

DD2-AO-TT2-ER1	"daughter"
KK3-AX-LL-AY-DD1	"collide"
SS-SS-IH-SS-TT2-ER1	"sister"
KK1-LL-AW-NN1	"clown"
KK3-UH-KK1-IY	"cookie"
LL-EH-TT2-ER	"letter"
LL-IH-TT2-EL	"little"
AX-NG-KK3-EL	"uncle"
KK1-AX-MM-PP1-YY1-UW1-TT2-ER	"computer"
EH-KK1-SS-TT2-EH-EH-NN1-TT2	"extent"
TT2-UW2	"two"
AX-LL-AR-MM	"alarm"
SS-KK3-OR	"score"
FF-ER2	"fir"

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

Silence	
PA1 (10ms)	before BB, DD, GG, and JH
PA2 (30ms)	before BB, DD, GG, and JH
PA3 (50ms)	before PP, TT, KK, and CH, and between words
PA4 (100ms)	between clauses and sentences
PA5 (200ms)	between clauses and sentences
Short Vowels	
*/IH/	sitting, stranded
*/EH/	extent, gentlemen
*/AE/	extract, acting
*/UH/	cookie, full
*/AO/	talking, song
*/AX/	lapel, instruct
*/AA/	pottery, cotton
Long Vowels	
/IY/	treat, people, penny
/EY/	great, statement, tray
/AY/	kite, sky, mighty
/OY/	noise, toy, voice
/UW1/	after clusters with YY: computer
/UW2	in monosyllabic words: two, food
/OW/	zone, close, snow
/AW/	sound, mouse, down

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 and used singly in final position
* /TH/	
* /SS/	
/SH/	shirt, leash, nation
/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, AY, 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.

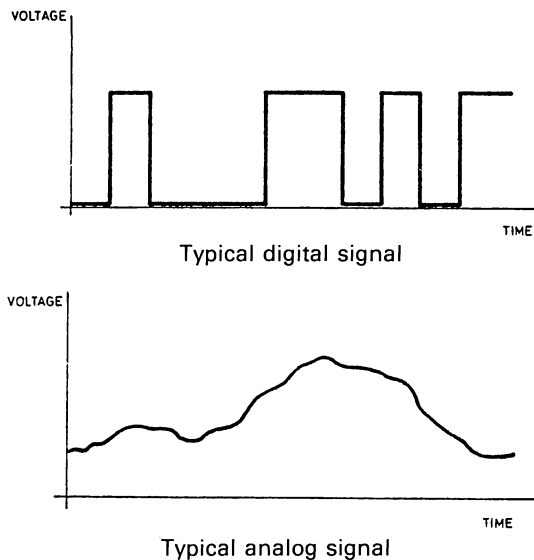


Fig. 5.7. Comparison of digital and analog waveforms (Reproduced by courtesy of *Practical Electronics*).

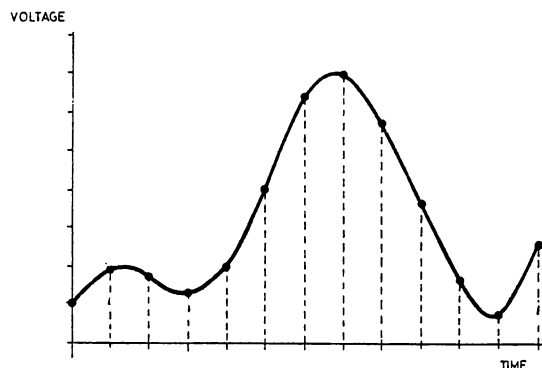


Fig. 5.8. Digital sampling of an analog waveform (Reproduced by courtesy of *Practical Electronics*).

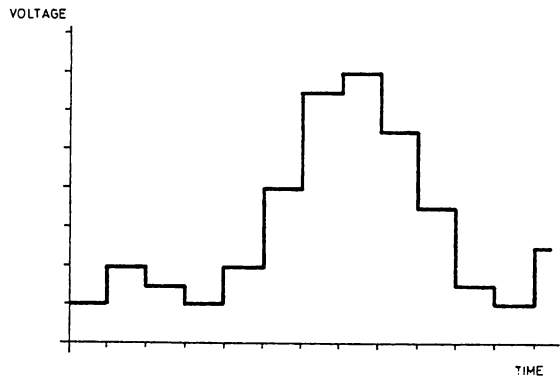


Fig. 5.9. Digital synthesis of an analog waveform (Reproduced by courtesy of *Practical Electronics*).

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

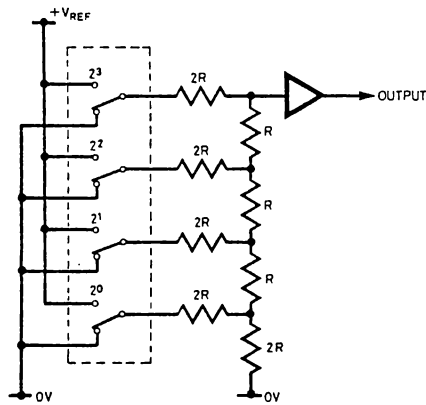


Fig. 5.10. Digital to analog converter using R-2R resistor network (Reproduced by courtesy of *Practical Electronics*).

PIN CONNECTIONS (TOP VIEW)

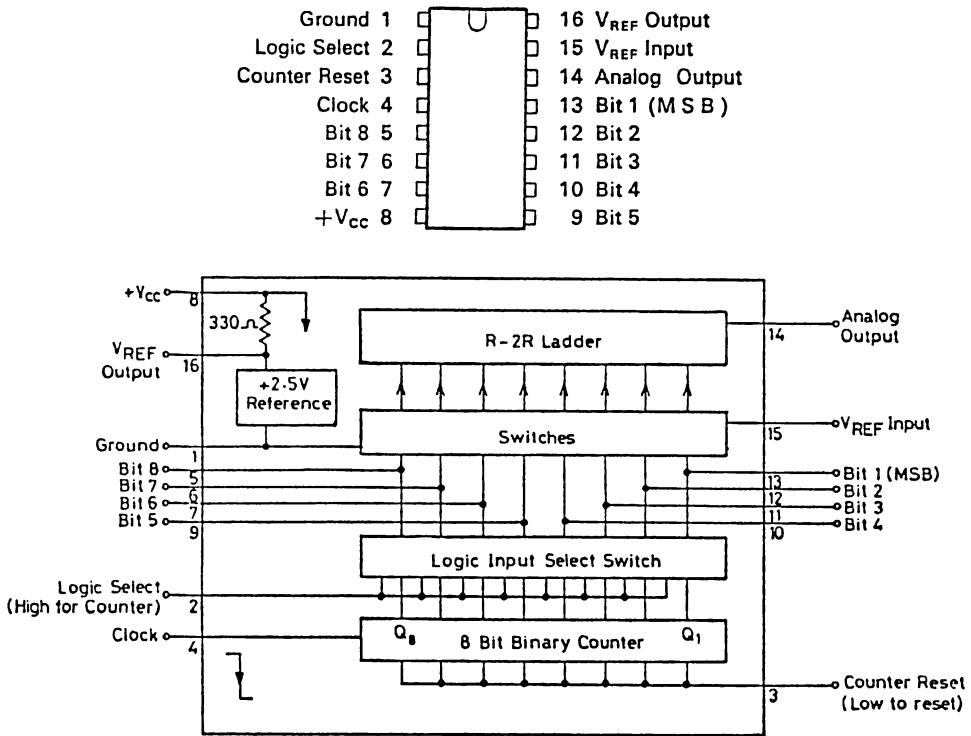


Fig. 5.11. Pin connections and block diagram for the A/D and D/A converter (Reproduced by courtesy of RS Components Limited).

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

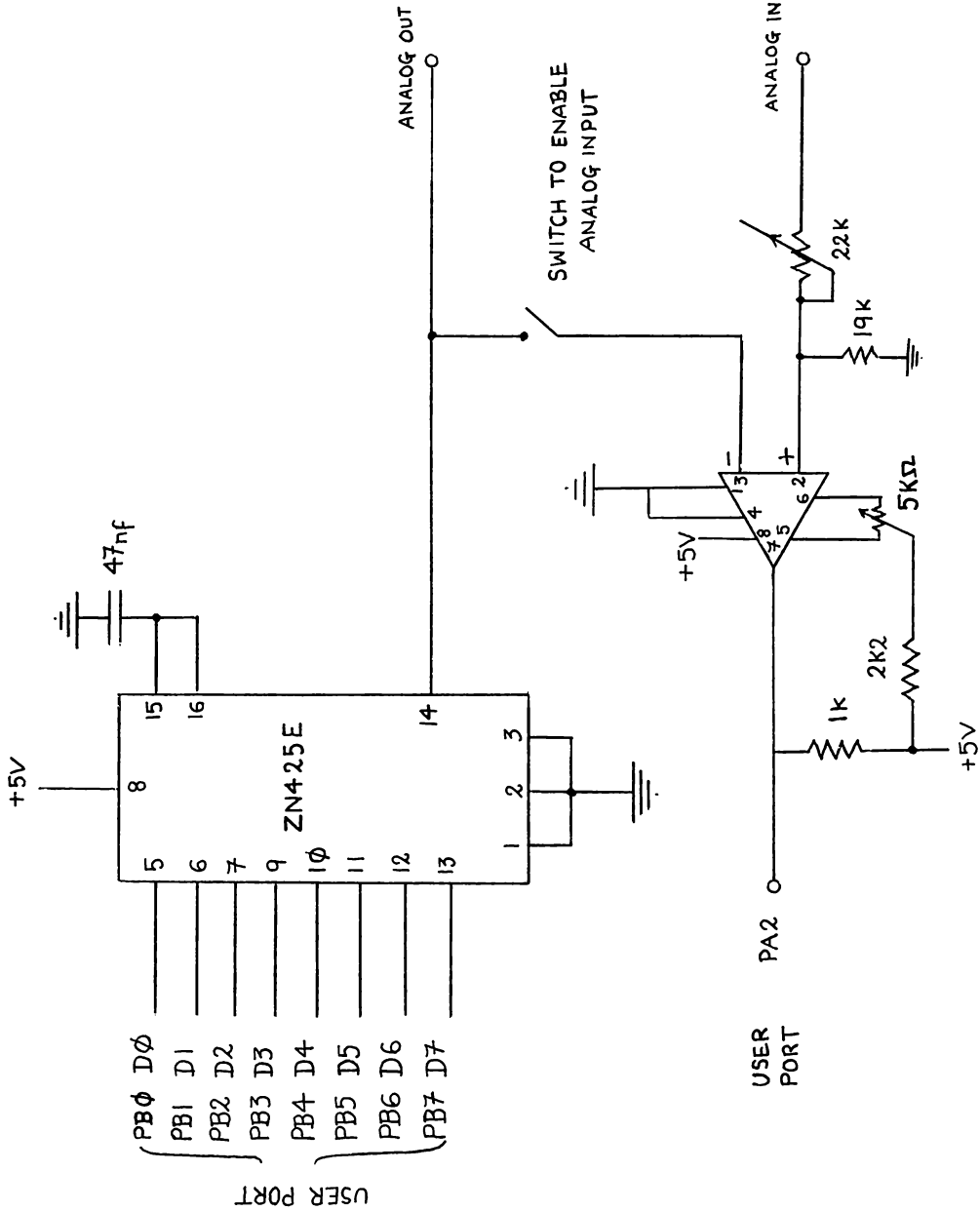


Fig. 5.12. Analog to digital and digital to analog conversion circuit.

```

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   FUNCTN JSR $AEFD      ;GET VALUE FOR
C294      20 8A AD      JSR $AD8A      ; TIMER A
C297      20 F7 B7      JSR $B7F7
C29A      A9 D1        LDA #CNMI      ;SET NMI VECTOR
C29C      8D 18 03      STA $0318
C29F      8D FA FF      STA $FFFA
C2A2      A9 C2        LDA #>NMI
C2A4      8D 19 03      STA $0319
C2A7      8D FB FF      STA $FFFB
C2AA      A5 14        LDA $14
C2AC      8D 04 DD      STA $DD04      ;STORE TIMER VALUE
C2AF      A5 15        LDA $15
C2B1      8D 05 DD      STA $DD05
C2B4      A9 FF        LDA #$FF
C2B6      8D 03 DD      STA $DD03      ;USER PORT TO OUTPUT
C2B9      A9 0F        LDA #$0F
C2BB      8D 18 D4      STA $D400+24   ;MAX SID VOLUME
C2BE      A9 00        LDA #$00
C2C0      8D 01 C3      STA FLAG      ;POINTER TO TABLE
C2C3      AD 01 DD      LDA $DD0D      ;CLEAR NMIS
C2C6      A9 81        LDA #$81      ;SET TIMER A NMI
C2C8      8D 0D DD      STA $DD0D
C2CB      A9 11        LDA #$11      ;START TIMER A
C2CD      8D 0E DD      STA $DD0E
C2D0      60          RTS
C2D1      ;
C2D1      48          NMI   PHA          ;STORE REGISTERS
C2D2      8A          TXA
C2D3      48          PHA
C2D4      98          TYA
C2D5      48          PHA
C2D6      A9 01        LDA #$01      ;CAUSED BY TIMER A?
C2D8      2C 0D DD      BIT $DD0D
C2DB      D0 11        BNE SEND      ;YES
C2DD      A9 7F        LDA #$7F      ;CLEAR ENABLED NMIS
C2DF      8D 0D DD      STA $DD0D
C2E2      20 84 FF      JSR $FF84      ;RESET I/O
C2E5      20 8A FF      JSR $FF8A      ;RESET KERNAL
C2E8      68          NMEXIT PLA      ;PULL REGISTERS
C2E9      A8          TAY
C2EA      68          PLA
C2EB      AA          TAX
C2EC      68          PLA
C2ED      40          RTI
C2EE      ;
C2EE      AE 01 C3      SEND  LDX FLAG      ;GET POINTER
C2F1      8D 13 C3      LDA TABLE,X    ;GET BYTE
C2F4      8D 01 DD      STA $DD01      ;SEND TO USER PORT
C2F7      8A          TXA
C2F8      18          CLC
C2F9      69 01        STORE  ADC #$01      ;ADD SAMPLING VALUE
C2FB      8D 01 C3      STA FLAG
C2FE      4C E8 C2      JMP NMEXIT      ;EXIT NMI
C301      ;
C301      00          FLAG  .BYT 0

```

```

LOC   CODE          LINE
C302                ;
C302  20 8A FF      DISAB JSR $FF8A
C305  20 84 FF              JSR $FF84
C308  A9 FE              LDA #$FE
C30A  8D FB FF              STA $FFFB
C30D  A9 43              LDA #$43
C30F  8D FA FF              STA $FFFA
C312  60                RTS
C313                ;
C313                TABLE **+=256
C413                .END

```

Symbol table

SYMBOL	VALUE						
ALLOOP1	C0A3	ATOD	C09C	CLRMEM	C203	CLRSCN	C26A
DISAB	C302	DISPLY	C000	DIV8	C125	DONE	C1F0
DOT	C0D1	ERROR	C013	EXIT	C0CD	FLAG	C301
FUNCTN	C291	GRAPH	C1E1	GXY	C198	KERIN	C1D1
KEROUT	C1C9	LOOP	C206	LOOP1	C26E	LOOP2	C0C3
MUL320	C147	MUL8	C12E	NMEXIT	C2E8	NMI	C2D1
NORM	C27E	OK	C018	PLOT1	C18A	PLOTIT	C181
READ10	C060	RLOOP1	C06E	RLOOP2	C070	SEND	C2EE
SETUP	C092	STORE	C2F9	TABLE	C313	TOP2X	C179
TX	C1C7	XER	C0E1	%OK	C0E3	YND70K	C123
ZLOOP	C022	ZLOOP1	C03A	ZLOOP2	C042	ZPLOT	C04F

Program 12.

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.

```

10 PRINT"CHSE:"
20 PRINT"M      1 - JOYSTICK"
30 PRINT"M      2 - KEYBOARD"
40 PRINT"M      3 - FUNCTION"
50 GETA$:IFA$=""THEN50
60 A=VAL(A$):IFA<10RAD>3THEN50
90 T=49939:SYS49627
100 DIMT(255):FORI=0TO255:T(I)=127:POKET+I,127:SYS49537,I,T(I)-27:NEXT
110 ONAGOSUB1000,2000,3000
120 GETA$:IFA$=""THEN120

```

```

130 SYS49790
140 END
1000 REM
1010 REM INPUT WAVEFORM USING JOYSTICK
1020 REM
1030 X=0:Y=T(X)-27
1040 A=PEEK(56320):E=0:W=0:S=0:N=0
1050 IF (AAND16)=0 THEN RETURN
1060 IF (AAND8)=0 THEN E=1:GOTO1110
1070 IF (AAND4)=0 THEN W=-1:GOTO1110
1080 IF (AAND2)=0 THEN S=-1:GOTO1110
1090 IF (AAND1)=0 THEN N=1:GOTO1110
1100 GOTO 1040
1110 GOSUB 11100:GOTO1040
2000 REM
2010 REM INPUT WAVEFORM USING KEYBOARD
2020 REM
2030 X=0:Y=T(X)-27
2040 GETA$:E=0:W=0:S=0:N=0
2050 IF A$=CHR$(13) THEN RETURN
2060 IF A$="I" THEN E=1:GOTO 2110
2070 IF A$="W" THEN W=-1:GOTO 2110
2080 IF A$="S" THEN S=-1:GOTO 2110
2090 IF A$="N" THEN N=1:GOTO 2110
2100 GOTO 2040
2110 GOSUB 11100:GOTO2040
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)^2)
3050 FOR I=0 TO 255
3060 V=FNF(I):SYS49537,I,T(I)-27
3070 POKET+I,V:T(I)=V:SYS49537,I,T(I)-27
3080 NEXT I:RETURN
11100 IF E OR W THEN 11120
11110 IF (Y+S+N)<0 THEN RETURN
11111 IF (Y+S+N)>199 THEN RETURN
11112 SYS49537,X,Y:Y=Y+S+N
11115 T(X)=(Y+27):POKET+X,(Y+27):SYS49537,X,Y:RETURN
11120 X=(X+E+W)AND255:Y1=T(X)-27:SYS49537,X,Y1:SYS49537,X,Y
11130 T(X)=Y+27:POKET+X,Y+27:RETURN

```

Program 13.

LOC	CODE	LINE
0000		*=C000
C000		.LIB DIS.SIMPLE
C000	20 FD AE	DISPLY JSR \$AEFD
C003	20 8A AD	JSR \$AD8A ;GET MEMORY ADDRESS
C006	20 F7 B7	JSR \$B7F7 ; OF DISPLAY
C009	A5 15	LDA \$15
C00B	C9 78	CMP #\$78 ;LESS THAN BOTTOM?
C00D	90 04	BCC ERROR ;YES
C00F	C9 A0	CMP #\$A0 ;IN RANGE?
C011	90 05	BCC OK ;YES
C013	A9 01	ERROR LDA #\$01 ;FLAG ERROR
C015	85 02	STA \$02 ; AND EXIT
C017	60	RTS
C018	A9 00	OK LDA #\$00 ;FLAG O.K.
C01A	85 02	STA \$02
C01C	A9 00	LDA #\$00 ;SET X COUNTER
C01E	85 FD	STA \$FD
C020	85 FE	STA \$FE
C022	A0 00	ZLOOP LDY #\$00
C024	B1 14	LDA (\$14),Y ;GET A VALUE
C026	20 4F C0	JSR ZPLOT ;PLOT IT
C029	A5 14	LDA \$14 ;INCREASE TABLE POINTER
C02B	18	CLC ;BY SAMPLING RATE
C02C	69 01	ADC #\$01

LUC	CODE	LINE	
C02E	85 14		STA \$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
C03C	E6 FD		INC \$FD ;INCREASE X
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 D4		BNE ZLOOP ;NO
C04E	60		RTS ;YES, EXIT
C04F			;
C04F	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	85 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	
C060		.LIB	READ.SIMPLE
C060	78	READ10	SEI ;DISABLE IRQ
C061	A9 0B		LDA #\$0B ;BLANK SCREEN
C063	8D 11 D0		STA \$D011
C066	A9 00		LDA #\$00 ;SET POINTER TO
C068	85 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	A0 00		LDY #\$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	85 FD		STA \$FD
C081	A5 FE		LDA \$FE
C083	69 00		ADC #\$00
C085	85 FE		STA \$FE
C087	C9 A0		CMP #\$A0 ;END OF TABLE?
C089	D0 E3		BNE RLOOP1 ;NO
C08B	A9 1B		LDA #\$1B ;RESTORE SCREEN
C08D	8D 11 D0		STA \$D011
C090	58		CLI ;ENABLE IRQ
C091	60		RTS
C092		;	
C092	A9 FF	SETUP	LDA #\$FF ;SET USER PORT
C094	8D 03 DD		STA \$DD03 ; TO OUTPUT
C097	A9 FB		LDA #\$FB ;SET ONE INPUT LINE
C099	8D 02 DD		STA \$DD02 ; (PA2)
C09C	A9 80	ATOD	LDA #\$80 ;SET A TO D
C09E	8D 01 DD		STA \$DD01 ;START VALUE
C0A1	85 61		STA \$61
C0A3	AD 01 DD	ALoop1	LDA \$DD01 ;START MAIN LOOP
C0A6	05 61		ORA \$61
C0A8	8D 01 DD		STA \$DD01
C0AB	EA		NOP
C0AC	EA		NOP
C0AD	AD 00 DD		LDA \$DD00 ;INPUT FROM

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 \$DD01 ; TRY HALF VALUE
C0BA	8D 01 DD		STA \$DD01
C0BD	46 61		LSR \$61 ;DECREASE SEARCH STEP
C0BF	F0 0C		BEQ EXIT ;COMPLETE
C0C1	10 E0		BPL ALOOP1 ;TRY AGAIN
C0C3	46 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
C0CB	D0 D6		BNE ALOOP1 ;TRY AGAIN
C0CD	AD 01 DD	EXIT	LDA \$DD01 ;VALUE RETURNED IN
C0DD	60		RTS ; .A
C0D1			.END
C0D1			.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	90 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
C0E9	C9 C8		CMP #\$C8
C0EB	B0 F4		BCS XER
C0ED	A9 C7		LDA #199
C0EF	38		SEC
C0F0	E5 5B		SBC \$5B
C0F2	85 5B		STA \$5B
C0F4	A5 59		LDA \$59 ;CALCULATE THE BIT TO
C0F6	29 07		AND #\$07 ; BE PLOTTED AS
C0F8	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		LDA TOP2X,X
C103	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
C108	4A		LSR A
C109	4A		LSR A
C10A	0A		ASL A
C10B	AA		TAX
C10C	BD 47 C1		LDA MUL320,X
C10F	85 57		STA \$57
C111	BD 48 C1		LDA MUL320+1,X

164 The Commodore 64 Kernal and Hardware Revealed

LOC	CODE	LINE
C114	85 58	STA \$58
C116		;
C116		;ADD Y AND 7 TO \$57
C116		;
C116	A5 58	LDA \$58
C118	29 07	AND #\$07
C11A	18	CLC
C11B	65 57	ADC \$57
C11D	85 57	STA \$57
C11F	90 02	BCC YND70K
C121	E6 58	INC \$58
C123		;
C123		;CALCULATE INT(X/8)
C123		;
C123	A0 03	YND70K LDY #\$03
C125	46 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	LDY #\$03
C12E	06 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	85 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	85 58	STA \$58
C145	18	CLC
C146	60	RTS
C147	00 00	MUL320 .WOR 0,320,640,960,1280
C149	40 01	
C14B	80 02	
C14D	C0 03	
C14F	00 05	
C151	40 06	.WOR 1600,1920,2240,2560,2880
C153	80 07	
C155	C0 08	
C157	00 0A	
C159	40 0B	
C15B	80 0C	.WOR 3200,3520,3840,4160,4480
C15D	C0 0D	
C15F	00 0F	
C161	40 10	
C163	80 11	
C165	C0 12	.WOR 4800,5120,5440,5760,6080
C167	00 14	
C169	40 15	
C16B	80 16	
C16D	C0 17	
C16F	00 19	.WOR 6400,6720,7040,7360,7680
C171	40 1A	
C173	80 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      ;
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 #$00
C18F    B1 57      LDA ($S7),Y          ; OTHERWISE PLOT POINT
C191    45 5E      EOR $5E
C193    91 57      STA ($S7),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    8E C7 C1      STX TX
C1A8    8C C8 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 C8 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    78      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    85 01      STA $01
C1D8    58      CLI
C1D9    68      PLA
C1DA    60      RTS
C1DB      .END

```

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 D0	STA \$D011 ; SELECT BIT MAP MODE
C1E6	A9 3D	LDA #\$3D
C1E8	8D 18 D0	STA \$D018 ; CHOOSE HIRES SCREEN
C1EB	A9 C8	LDA #\$C8
C1ED	8D 16 D0	STA \$D016 ; ELSE SET HIRES MODE
C1F0	AD 02 DD	DONE LDA \$DD02 ; SELECT BANK 2 FOR HIRES
C1F3	09 03	ORA #\$03 ; SCREEN
C1F5	8D 02 DD	STA \$DD02
C1F8	AD 00 DD	LDA \$DD00
C1FB	29 FC	AND #\$FC
C1FD	09 00	ORA #\$00
C1FF	8D 00 DD	STA \$DD00
C202	60	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 \$EB00,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	BNE LOOP
C269	60	RTS
C26A	A0 00	CLRSCN LDY #\$00 ; LOOP TO CLEAR
C26C	A9 1B	LDA #\$1B
C26E	99 00 CC	LOOP1 STA \$CC00,Y
C271	99 00 CD	STA \$CD00,Y

```

LOC    CODE    LINE
C274  99 00 CE          STA $CE00,Y
C277  99 00 CF          STA $CF00,Y
C27A  88             DEY
C27B  D0 F1          BNE LOOP1
C27D  60             RTS
C27E                .END
C27E                .LIB  NORM.SIMPLE
C27E                ;
C27E                ; ROUTINE TO RETURN TO NORMAL SCREEN
C27E                ;
C27E  AD 02 DD  NORM LDA $DD02
C281  29 FC          AND #$FC
C283  8D 02 DD          STA $DD02          ; BACK TO BANK 0
C286  A9 1B          LDA #$1B
C288  8D 11 D0          STA $D011          ; BIT MAP MODE OFF
C28B  A9 15          LDA #$15
C28D  8D 18 D0          STA $D018          ; NORMAL SCREEN
C290  60             RTS
C291                .END

```

Program 14.

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.

LOC	CODE	LINE	
0000		USRPRT	=\$DD01 ;OUTPUT PORT
0000		DDR	=\$DD03 ;DATA DIRECTION
0000		V1PT	=\$40 ;FOUR VOICE WAVEFORM
0000		V2PT	=\$45 ; POINTERS
0000		V3PT	=\$48
0000		V4PT	=\$4B
0000		INCPT	=\$4E
0000		NOTES	=\$50 ;POINTER TO MUSIC
0000		V1IN	=\$52 ;FOUR VOICE INCREMENT
0000		V2IN	=\$54 ; POINTERS
0000		V3IN	=\$56
0000		V4IN	=\$58
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	9E	.BYT	'\$9E','02062',0 ;SYS02062
0006	30 32		
000B	00		
000C	00 00	END	.WOR 0 ;END OF BASIC
000E		;	
000E		;	ENTRY
000E		;	
000E	78	SEI	;
000F	A9 0B	LDA	#\$0B ;DISABLE IRQ
0011	8D 11 D0	STA	\$D011 ;DISABLE SCREEN DMA
0014	D8	CLD	;
0015	A9 0F	LDA	#\$0F ;DISABLE DECIMAL
0017	8D 18 D4	STA	\$D418 ;SET SID VOLUME
001A	A9 FF	LDA	#\$FF ; TO MAX
001C	8D 03 DD	STA	DDR ;SET USER PORT
001F	A2 00	LDX	#\$00 ;TO OUTPUT
0021	B5 00	STORE	LDA \$0000,X ;SAVE OFF ZERO
0023	9D 00 C0	STA	\$C000,X ;PAGE
0026	E8	INX	
0027	D0 F8	BNE	STORE
0029	A2 00	LDX	#\$00
002B	A0 52	LDY	#\$52 ;SET UP INCA TO
002D	86 5E	STX	INCA+1 ; POINT TO V1IN
002F	84 5D	STY	INCA
0031	A0 0F	LDY	#\$0F ;CONTROL TABLE STARTS
0033	86 FB	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	A0 00	LDY	#\$00
0043	B1 FB	LOOP	LDA (<\$FB>),Y ;GET CONTROL CODE
0045	C9 FF	CMP	#\$FF ;PLAY CONTROL?
0047	F0 25	BEQ	CNTRL ;YES
0049	AA	TAX	;
004A	C8	INY	;
004B	98	TYA	;
004C	48	PHA	;
004D	B1 FB	LDA	(<\$FB>),Y ;GET LOW BYTE
004F	A8	TAY	
0050	20 98 08	JSR	MUSIC ;PLAY THE PIECE
0053	68	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	RESTRE	LDA \$C000,X ;COPY BACK TO

LOC	CODE	LINE		
085D	95 00		STA \$0000,X	; ZERO PAGE
085F	E8		INX	
0860	D0 F8		BNE RESTRE	
0862	A9 00		LDA #\$00	; NO SID VOLUME
0864	8D 18 D4		STA \$D418	
0867	A9 1B		LDA #\$1B	; RESTORE SCREEN
0869	8D 11 D0		STA \$D011	
086C	58		CLI	; START IRQ
086D	60		RTS	; FINISHED
086E				
086E	C8		; CNTRL INY	; GET CONTROL NUMBER
086F	B1 FB		LDA (\$FB),Y	
0871	30 E5		BMI EXIT	; END OF MUSIC
0873	C9 01		CMP #\$01	; IS VALUE 1?
0875	D0 07		BNE CONTR1	; NO
0877	C8		INX	
0878	B1 FB		LDA (\$FB),Y	; GET TEMPO
087A	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	C8		INX	
0883	B1 FB		LDA (\$FB),Y	; GET HI BYTE FOR
0885	85 42		STA V1PT+2	; THE WAVEFORM
0887	C8		INX	; POINTERS OF THE
0888	B1 FB		LDA (\$FB),Y	; FOUR VOICES
088A	85 47		STA V2PT+2	
088C	C8		INX	
088D	B1 FB		LDA (\$FB),Y	
088F	85 4A		STA V3PT+2	
0891	C8		INX	
0892	B1 FB		LDA (\$FB),Y	
0894	85 4D		STA V4PT+2	
0896	D0 BD		BNE NEXT	
0898				
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
08A0	85 4E		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				
08B4	B1 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	

Symbol table

SYMBOL VALUE							
CNTR0L	086E	CONTR1	087E	DDR	DD03	DUR	005A
END	080C	ENDNOT	094A	ENDSNG	08F6	EXIT	0858
FRQTAB	0A01	INCA	005D	INCPT	004E	LOOP	0843
MUSIC	0898	MUSIC1	089C	MUSIC2	08BE	MUSIC3	08C4
MUSIC4	08D9	MUSIC9	08B4	NEXT	0855	NOTES	0050
NXTSEG	08E7	PLAY	08F7	PLAY1	08FB	RESTRE	085A
STORE	0821	TEMPO	005F	TIMWAS	0942	USRPRT	DD01
V1IN	0052	V1PT	0040	V2IN	0054	V2PT	0045
V3IN	0056	V3PT	0048	V4IN	0058	V4PT	004B

Program 15.

Music data frequency tables

Bass frequency table

Hi-lo 0A7C to 0A93

```

:0A00 00 00 01 E9 02 06 02 25'.....%
:0A08 02 45 02 68 02 8C 02 B3'.E.....+
:0A10 02 DC 03 08 03 36 03 67'.#.....6..
:0A18 03 9A 03 D1 04 0B 04 49'....0...I
:0A20 04 8A 04 CF 05 19 05 66'...Γ....
:0A28 05 B8 06 0F 06 6C 06 CD'.-.....\
:0A30 07 35 07 A3 08 17 08 92'.5.-....
:0A38 09 15 09 9F 0A 31 0A CC'.....1.L
:0A40 0B 71 0C 1F 0C D7 0D 98'.....0..
:0A48 0E 6A 0F 45 10 2E 11 24'...E...$
:0A50 12 29 13 3E 14 62 15 99'...).>....
:0A58 16 E2 18 3E 19 AF 1B 36'...>...6
:0A60 1C D4 1E 8B 20 5C 22 48'.1.. \ "H
:0A68 24 52 26 7B 28 C5 2B 31'$R&.(~+1
:0A70 2D C3 30 7C 33 5E 36 6C'--0.316.
:0A78 39 A8 3D 15 00 F4 01 03'9#.....
:0A80 01 12 01 23 01 34 01 46'...#.4.F
:0A88 01 5A 01 6D 01 84 01 9B'.Z.....
:0A90 01 B3 01 CD 00 00 00 00'.+.\....

```

Wraparound waveform table no 1

```

:0B00 33 34 35 36 36 37 38 39'34566789
:0B08 39 3A 3A 3B 3B 3B 3C 3C'9:1111<<
:0B10 3C 3C 3C 3C 3C 3C 3C 3C'<<<<<<<<
:0B18 3C 3C 3C 3B 3B 3B 3B 3B'<<<1111
:0B20 3A 3A 3A 3A 3A 3A 39 39'1111:99
:0B28 39 39 39 39 39 39 39 39'999999999
:0B30 3A 3A 3A 3A 3A 3B 3B 3B'11111111
:0B38 3B 3C 3C 3C 3D 3D 3D 3D'<<<====
:0B40 3E 3E 3E 3E 3F 3F 3F 3F'>>>>????
:0B48 3F 3F 3F 3F 3F 3F 3F 3F'????????
:0B50 3E 3E 3E 3D 3D 3C 3C 3B'>>>==<<
:0B58 3B 3A 39 38 38 37 36 35'1:988765
:0B60 34 33 32 31 30 2F 2E 2D'43210/,-
:0B68 2C 2B 2A 29 28 27 26 25'./+*)(('&%
:0B70 24 23 22 21 21 20 1F 1F'$##"!! ..
:0B78 1E 1E 1D 1D 1D 1D 1C 1C'.....
:0B80 1C 1C 1D 1D 1D 1D 1D 1E'.....
:0B88 1E 1F 1F 20 20 21 21 22'... !!"
:0B90 23 23 24 24 25 26 26 27'###$%&&
:0B98 28 28 29 29 29 2A 2A 2B'(( ))**+
:0BA0 2B 2B 2B 2B 2B 2B 2B 2A'++++++*
:0BA8 2A 2A 29 29 28 27 27 26'(**)></'&
:0BB0 25 24 23 22 21 20 1F 1D'/%$#"! ..
:0BB8 1C 1B 19 18 17 15 14 13'.....
:0BC0 11 10 0F 0D 0C 0B 09 08'.....
:0BC8 07 06 05 04 03 03 02 01'.....
:0BD0 01 00 00 00 00 00 00 00'.....
:0BD8 00 00 01 01 01 02 03 04'.....
:0BE0 05 06 07 08 09 0B 0C 0D'.....
:0BE8 0F 10 12 13 15 16 18 1A'.....
:0BF0 1B 1D 1F 20 22 23 25 27'... "#%
:0BF8 28 2A 2B 2C 2E 2F 30 31'<(*+./01

```

Wraparound waveform table no 2

```

:0C00 22 22 22 22 22 22 22 22 22' " " " " " " " " " "
:0C08 22 21 20 20 1F 1F 1E 1D' " " " " " " " " " "
:0C10 1D 1C 1C 1C 1C 1B 1B 1C' " " " " " " " " " "
:0C18 1B 1C 1C 1C 1D 1D 1E 1F' " " " " " " " " " "
:0C20 1F 20 20 21 21 22 25 25' " " " " " " " " " "
:0C28 26 26 26 27 28 28 29 2A' &&&' (<)*
:0C30 2A 2A 2C 2C 2E 2F 30 32' * , , / 02
:0C38 32 34 34 37 38 39 3A 3A' 244789:
:0C40 3B 3C 3C 3D 3D 3D 3C 3C' <<==<<
:0C48 3B 3A 39 38 35 34 32 30' : 985420
:0C50 2E 2B 29 27 25 21 20 1E' , + ) ' % ! .
:0C58 1D 1C 1B 19 19 18 18 18' " " " " " " " " " "
:0C60 18 19 19 1B 1C 1D 1E 1F' " " " " " " " " " "
:0C68 20 21 22 22 25 25 26 26' " " " " " " " " " "
:0C70 26 26 25 25 25 22 21 21' &&&' " " " " " " " "
:0C78 20 1F 1E 1D 1C 1C 1B 19' " " " " " " " " " "
:0C80 19 19 1B 1B 18 18 18 19' " " " " " " " " " "
:0C88 19 19 1B 1C 1C 1D 1E 1E' " " " " " " " " " "
:0C90 1E 1F 1F 20 20 20 20 20' " " " " " " " " " "
:0C98 20 1F 1F 1F 1E 1E 1E 1D' " " " " " " " " " "
:0CA0 1C 1C 1B 1B 19 18 18 17' " " " " " " " " " "
:0CA8 17 17 16 15 15 14 14 13' " " " " " " " " " "
:0CB0 13 12 10 0F 0E 0D 0C 0B' " " " " " " " " " "
:0CB8 0A 09 07 06 05 04 03 02' " " " " " " " " " "
:0CC0 01 01 00 00 00 00 00 00' " " " " " " " " " "
:0CC8 01 02 03 05 06 09 0B 0D' " " " " " " " " " "
:0CD0 0E 10 13 15 18 19 1C 1D' " " " " " " " " " "
:0CD8 1E 20 20 21 22 25 22 22' " " " " " " " " " "
:0CE0 22 22 21 20 1F 1E 1E 1C' " " " " " " " " " "
:0CE8 1C 1B 19 18 17 17 17 17' " " " " " " " " " "
:0CF0 16 16 17 18 18 18 19 1B' " " " " " " " " " "
:0CF8 1B 1D 1D 1E 1F 20 21 21' " " " " " " " " " "

```

Wraparound waveform table no 3

```

:0D00 20 20 20 21 21 21 21 21' " " " " " " " " " "
:0D08 21 20 20 1F 1F 1E 1D 1C' " " " " " " " " " "
:0D10 1B 1A 1A 19 18 18 17 17' " " " " " " " " " "
:0D18 17 17 18 19 1A 1B 1C 1E' " " " " " " " " " "
:0D20 1F 21 23 25 27 29 2A 2C' , ! # % ' ) * ,
:0D28 2D 2E 2F 2F 2F 2F 2E 2D' - . / / / / . -
:0D30 2C 2A 28 26 23 20 1E 1B' , * ( & # . .
:0D38 18 15 13 10 0E 0C 0B 0A' " " " " " " " " " "
:0D40 0A 0A 0A 0B 0C 0E 11 13' " " " " " " " " " "
:0D48 16 1A 1D 21 24 2B 2B 2F' " " " " " " " " " "
:0D50 32 34 37 38 3A 3A 3A 3A' 2478: : :
:0D58 39 37 35 32 2F 2C 28 24' 9752/ , ( $
:0D60 1F 1B 17 13 0F 0C 09 06' " " " " " " " " " "
:0D68 04 03 02 02 02 03 05 07' " " " " " " " " " "
:0D70 0A 0D 11 15 1A 1E 23 27' " " " " " " " " " "
:0D78 2B 2F 33 36 39 3C 3D 3E' + / 369<=>
:0D80 3E 3E 3D 3C 39 36 33 2F' > ) = < 963/
:0D88 2B 27 23 1E 1A 15 11 0D' + ' # . . . .
:0D90 0A 07 05 03 02 02 02 03' " " " " " " " " " "
:0D98 04 06 09 0C 0F 13 17 1B' " " " " " " " " " "
:0DA0 1F 24 28 2C 2F 32 35 37' , $ ( , / 257
:0DA8 39 3A 3A 3A 3A 38 37 34' 9: : : 874
:0DB0 32 2F 2B 28 24 21 1D 1A' 2/ + ( $ ! . .
:0DB8 16 13 11 0E 0C 0B 0A 0A' " " " " " " " " " "
:0DC0 0A 0A 0B 0C 0E 10 13 15' " " " " " " " " " "
:0DC8 18 1B 1E 20 23 26 28 2A' " " " " " " " " " "
:0DD0 2C 2D 2E 2F 2F 2F 2F 2E' , - . / / / / .
:0DD8 2D 2C 2A 29 27 25 23 21' - , * ) ' % # !
:0DE0 1F 1E 1C 1B 1A 19 18 17' " " " " " " " " " "
:0DE8 17 17 17 18 18 19 1A 1A' " " " " " " " " " "
:0DF0 1B 1C 1D 1E 1F 1F 20 20' " " " " " " " " " "
:0DF8 21 21 21 21 21 21 20 20' " " " " " " " " " "

```

Wraparound waveform table no 4

```

:0E00 20 20 20 20 20 20 20 20'
:0E08 21 21 21 22 22 22 23 23'!!!""###
:0E10 24 25 25 26 26 27 28 28'$%&&'<<
:0E18 29 2A 2B 2B 2C 2D 2E 2E'>*+,-..
:0E20 2F 30 31 32 32 33 34 34'/0122344
:0E28 35 36 36 37 37 38 38 38'56677888
:0E30 38 38 38 38 38 38 38 38'88888888
:0E38 37 36 35 34 33 32 31 2F'7654321/
:0E40 2E 2D 2B 29 27 26 24 22'.,-+)'&$"
:0E48 20 1E 1C 1A 18 16 15 13'.....
:0E50 12 10 0F 0E 0D 0C 0C 0B'.....
:0E58 0B 0B 0C 0C 0D 0E 0F 11'.....
:0E60 13 14 16 18 1B 1D 1F 21'.....!
:0E68 24 26 28 2A 2C 2D 2F 30'$&(*,-/0
:0E70 30 31 31 31 30 2F 2E 2D'01110/-
:0E78 2B 29 27 24 22 20 1D 1B'+)'$" ..
:0E80 19 17 15 13 12 11 11 11'.....
:0E88 11 12 13 14 16 18 1A 1C'.....
:0E90 1E 21 23 25 27 29 2A 2B'.!#%<)*+
:0E98 2C 2C 2C 2B 2A 29 27 25',,,+*)'%
:0EA0 23 21 1E 1C 1A 18 17 16'#!.....
:0EA8 15 15 15 16 17 18 1A 1C'.....
:0EB0 1E 20 22 24 25 27 28 28'. "%%<<<
:0EB8 28 28 27 26 25 23 21 20'<<<&%#!
:0EC0 1E 1C 1B 1A 19 18 18 19'.....
:0EC8 1A 1B 1C 1D 1F 21 22 23'.....!"#
:0ED0 24 25 25 25 25 24 23 22'$%&%%$##"
:0ED8 20 1F 1E 1D 1C 1B 1B 1B'.....
:0EE0 1C 1C 1D 1E 1F 20 21 22'.....!"
:0EE8 22 22 22 22 22 21 21 20'""""!!
:0EF0 1F 1F 1E 1E 1E 1E 1E 1E'.....
:0EF8 1F 1F 1F 20 20 20 20 20'....

```

Program 16.

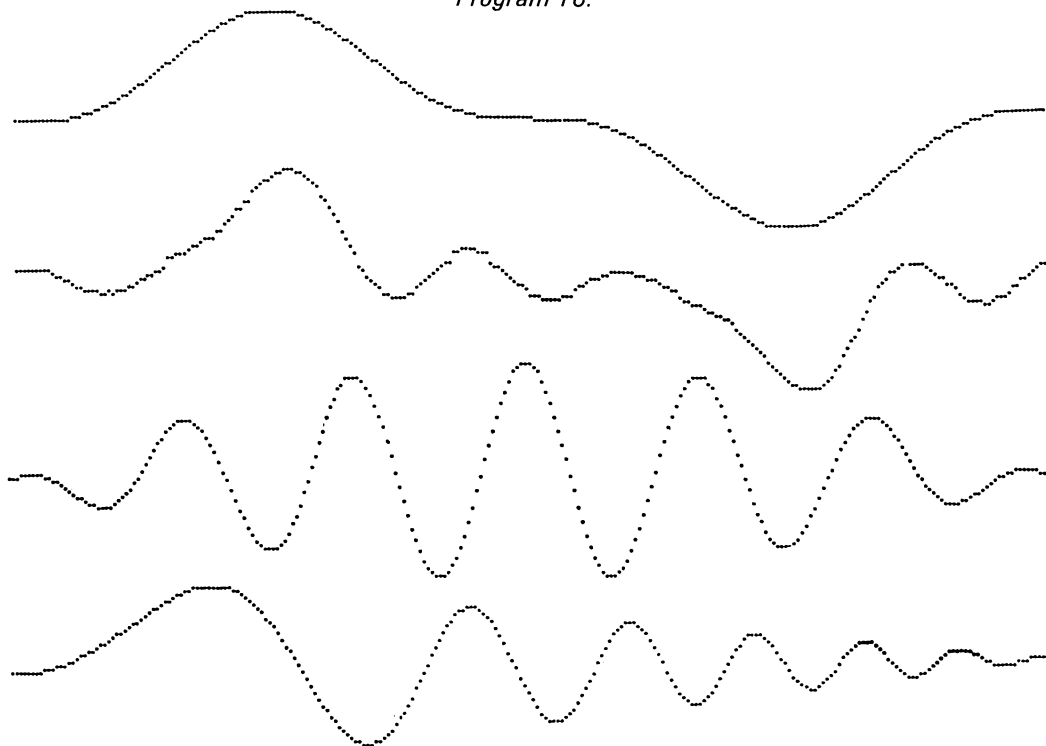


Fig. 5.13. Sample voice waveforms for music synthesis program.

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-lo of pointer to following score table

```

:0F00 FF 01 30 FF 02 0E 0E 0E'...0.....
:0F08 0E 10 00 10 1A 10 00 10'.....%
:0F10 1A 10 00 10 00 10 00 10'.....
:0F18 1A FF 02 0B 0C 0D 0E 10'.....
:0F20 20 10 8F 11 80 11 95 11'.....
:0F28 AA 10 8F 11 E7 11 95 11' L.....
:0F30 FC 12 4D 13 1B 12 4D 13'..M...M.
:0F38 94 10 8F 11 E7 11 95 14'.....
:0F40 AD 11 BE 10 8F 11 80 11' L'.....
:0F48 95 14 AD 12 10 14 C2 10'..L...L.
:0F50 8F 11 80 11 95 16 71 16'.....
:0F58 DB 10 9E 11 80 11 95 16'+.....
:0F60 71 16 FF FF FF 00 00 00'.....

```

Music table

In groups of 5

Duration and 4 notes, 1 per voice

ØØ = return to main control loop

Ø1 = read next 2 bytes as pointer into this table lo-hi

```

:1000 09 80 80 80 06 09 80 80'.....
:1008 84 0A 09 80 80 88 0E 25'.....%
:1010 80 80 8A 10 20 00 00 00'....
:1018 00 00 60 00 00 00 00 00'.....
:1020 34 48 30 22 10 0C 00 00'4H0"....
:1028 00 00 14 4A 32 2C 14 0C'...J2,..
:1030 00 00 00 00 34 4E 36 30'....4N60
:1038 18 0C 00 00 00 00 14 52'.....R
:1040 3A 32 1A 0C 00 00 00 00'2.....
:1048 34 54 3C 36 1E 0C 00 00'4T<6....
:1050 00 00 14 52 3A 32 1A 0C'...R:2..
:1058 00 00 00 00 34 4E 36 30'....4N60
:1060 18 0C 00 00 00 00 14 4A'.....J
:1068 32 2C 14 0C 00 00 00 00'2'.....
:1070 34 48 30 28 10 3C 00 00'4H0<,<..
:1078 00 00 40 58 54 4E 48 20'..EXTNH
:1080 00 00 00 00 40 00 00 10'....@...
:1088 8A 20 00 00 00 00 00 20'.....
:1090 40 28 00 00 20 44 2C 00'@(<..D)..
:1098 00 20 48 30 00 00 40 4A'..H0...@J
:10A0 00 32 02 20 4A 40 3A 02'2..J@:..
:10A8 40 40 00 00 0A 20 40 3A'@@...@:
:10B0 32 10 20 3A 00 22 1A 20'2..:..".
:10B8 3A 00 22 00 20 3A 32 28'2'..":2<
:10C0 00 40 36 2C 26 08 20 32'..@6.&..2
:10C8 2C 26 08 40 30 24 00 06'&..@0$.
:10D0 20 44 00 00 00 40 40 3A' D...@@:
:10D8 2E 04 20 40 3C 30 06 20'..@<0.
:10E0 40 00 10 8A 20 40 28 00'@...@(<
:10E8 00 20 44 2C 00 00 20 48'..D...H
:10F0 30 00 00 20 4A 32 00 00'0...J2..
:10F8 20 4C 34 00 00 40 4E 36' L4...@N6
:1100 00 10 20 4E 3C 40 10 40'..N<@.@
:1108 48 00 00 1E 20 48 3C 40'H...H<@
:1110 1E 40 40 28 00 18 20 40'..@@(<..@
:1118 3C 36 18 40 3C 30 28 10'<6.@<0<
:1120 20 36 30 28 10 40 32 00' 60<.@2.

```

```

:1128 00 1A 20 32 28 22 1A 40'.. 2<".@
:1130 32 00 00 14 20 32 28 22'2... 2<"
:1138 14 20 32 00 00 10 20 40'.. 2... @
:1140 28 00 10 20 44 2C 00 10'<... D...
:1148 20 48 30 00 0A 20 4A 32' H0.. J2
:1150 00 0A 20 4E 36 00 0A 40'.. N6..@
:1158 52 00 00 02 20 52 4A 40'R... RJ@
:1160 02 40 4A 00 00 0A 20 4A'..@J... J
:1168 40 3A 10 40 40 00 00 16'@..@@...
:1170 20 40 3A 32 16 40 3A 32' @:2.@:2
:1178 28 16 20 32 32 28 16 00'<.. 22<..
:1180 40 3C 32 24 14 20 40 00'@<2$. @
:1188 28 14 40 44 00 2C 0C 20'<..@D...
:1190 48 00 30 0A 00 40 4A 00'H.0..@J.
:1198 32 06 20 4A 44 3E 06 40'2.. JD>.@
:11A0 48 00 30 06 20 44 3E 2C'H.0.. D>
:11A8 06 00 40 40 28 10 8A 20'..@@<..
:11B0 40 36 30 28 40 40 28 00'@60<@<..
:11B8 06 20 48 36 00 00 40 4E'.. H6..@N
:11C0 3E 14 8E 20 4E 3E 00 00'>.. N>..
:11C8 40 4A 3E 36 06 20 44 3E'@J>6.. D>
:11D0 36 00 60 4E 40 36 18 60'6..N@6..
:11D8 4E 3E 32 14 40 4E 3C 30'ND>2.@N<0
:11E0 10 20 00 00 00 00 00 40'.. .....@
:11E8 3C 00 24 14 20 3C 32 2C'<..$. <2>
:11F0 14 40 44 00 2C 0C 20 44'..@D... D
:11F8 32 2C 0A 00 40 40 00 28'2...@@.<
:1200 10 20 40 3C 30 28 40 40'.. @<0<@@
:1208 00 28 06 20 48 00 30 00'<.. H.0.
:1210 40 4E 3C 36 10 20 4E 3C'@N<6.. NC
:1218 36 00 40 4A 3C 32 10 20'6..@J<2.
:1220 48 3C 30 00 20 4A 40 3A'H0.. J@:
:1228 1A 20 4A 40 3A 14 20 4A'.. J@:.. J
:1230 40 3A 10 20 4A 40 3A 0C'@:.. J@:..
:1238 20 4A 40 3A 0A 20 4A 40' J@:.. J@
:1240 3A 06 40 4A 40 3A 02 20'>..@J@:..
:1248 00 00 00 00 00 34 32 00'.....42.
:1250 00 1A 0C 00 00 00 00 20'.....
:1258 32 00 00 1A 40 44 2C 24'2...@D$. $
:1260 0C 20 44 2C 34 1A 40 42'.. D$.@B
:1268 2A 22 02 20 44 2C 24 1A'*".. D$. $
:1270 40 4E 2E 1E 0C 20 4E 3E'@N... ND>
:1278 36 1E 40 4A 2C 1E 02 20'6..@J...
:1280 44 2C 20 0E 40 46 2E 22'D..@F.. "
:1288 10 20 44 2C 20 00 40 46'.. D..@F
:1290 2E 22 02 20 46 2E 22 00'..". F.. "
:1298 20 46 2E 22 10 20 46 2E' F..". F.
:12A0 24 10 20 46 2E 28 00 20'$.. F.<..
:12A8 46 2E 22 02 20 46 2E 1E'F..". F..
:12B0 02 20 46 2E 1A 00 40 46'.. F...@F
:12B8 28 22 1A 20 44 2C 26 02'<". D.&..
:12C0 60 46 28 1A 02 40 48 2A'..F<..@H*
:12C8 22 1A 20 46 2E 28 02 60'..". F.<..
:12D0 48 2A 1A 02 40 4A 2C 24'H*..@J$. $
:12D8 0C 20 4A 32 00 1A 40 44'.. J2..@D
:12E0 2C 00 14 20 3C 00 24 0C'>.. <$. $
:12E8 40 32 00 1A 02 20 00 00'@2....
:12F0 00 00 40 40 3A 32 02 20'..@@:2.
:12F8 00 00 00 00 40 44 24 1A'.....@D$.
:1300 0C 20 44 24 2C 1A 40 42'.. D$.@B
:1308 2A 22 02 20 44 2C 24 1A'*".. D$. $
:1310 40 4E 2E 1A 0C 20 4E 2E'@N... N.
:1318 36 1A 00 40 4A 2C 1A 02'6..@J...
:1320 20 44 24 2C 1A 40 40 28' D$.@@<
:1328 22 1A 20 40 28 22 1A 40'..". @<".@
:1330 4A 32 00 10 20 4A 32 22'J2.. J2"
:1338 10 20 4A 00 00 0A 20 4A'.. J... J
:1340 00 00 0C 20 4A 28 22 10'.... J<".
:1348 40 4A 00 00 02 20 4A 28'@J... J<
:1350 22 02 40 40 28 24 1E 20'..@@<$.

```

```

:1358 40 28 24 1E 40 4E 36 00'@($,@N6.
:1360 10 20 4E 28 24 10 20 4E'. N($, N
:1368 28 00 18 20 4E 28 00 1A'<... N<..
:1370 20 4E 28 24 1E 20 48 30' N($, H0
:1378 24 10 20 44 2C 24 14 20'$, D,$.
:1380 40 28 24 18 60 4A 28 22'@($,..J("&
:1388 1A 60 4E 30 24 10 60 52'...N0$..R
:1390 3A 28 02 00 40 4A 2C 20'<...@J,
:1398 08 20 44 24 20 08 40 46'. D$ ,@F
:13A0 24 1E 06 20 46 2E 24 00'$.. F,$.
:13A8 40 44 2C 24 16 20 46 2E'@D,$. F.
:13B0 24 00 40 52 3A 2A 18 20'$,@R:*.
:13B8 52 3A 2A 00 40 4E 36 2A'R:*,@N6*
:13C0 0A 20 48 30 2A 0A 40 4A'. H0*,@J
:13C8 32 00 0C 20 4A 32 2C 24'2... J2,$
:13D0 40 44 00 2C 02 20 4A 32'@D... J2
:13D8 2C 00 40 46 2E 22 10 20'...@F,".
:13E0 4A 32 22 00 40 4E 36 22'J2".@N6"
:13E8 02 20 52 3A 22 00 20 54'. R:". T
:13F0 3C 32 2C 20 54 32 2C 1E'<2, T2,.
:13F8 20 54 32 2C 1A 20 54 32' T2,. T2
:1400 2C 16 20 54 32 2C 14 20'... T2,.
:1408 54 32 2C 10 20 00 00 00' T2)...
:1410 0C 40 00 00 00 00 14 54'.@.....T
:1418 4A 44 00 0C 00 00 00 00'JD.....
:1420 14 54 4A 44 00 0C 00 00'TJD....
:1428 00 00 14 54 4A 44 00 0C'...TJD..
:1430 00 00 00 00 20 54 4A 44'.... TJD
:1438 00 20 54 4E 44 24 20 54'. TND$ T
:1440 4A 44 22 34 54 4A 44 20'JD"4TJD
:1448 0C 00 00 20 08 14 54 4A'.... ..TJ
:1450 44 20 0C 00 00 20 08 20'D ....
:1458 56 4A 44 1E 20 56 4A 44'VJD. VJD
:1460 1A 20 56 4A 44 16 34 56'. VJD.4V
:1468 4A 44 14 0C 00 00 00 14'JD.....
:1470 14 56 4A 44 14 0C 00 00'VJD....
:1478 00 14 20 58 4E 48 10 40'... XNH.@
:1480 00 00 00 00 14 00 00 28'.....<
:1488 10 0C 00 00 00 00 14 00'.....
:1490 00 28 10 0C 00 00 00 00'<.....
:1498 14 00 00 28 10 0C 00 00'...<....
:14A0 00 00 20 00 00 28 10 40'... ..<.@
:14A8 00 00 00 00 00 40 40 28'.....@<
:14B0 10 8A 20 40 36 30 28 40'... @60<@
:14B8 48 00 30 06 20 48 40 30'H.0. H00
:14C0 00 00 20 38 34 2E 08 20'... 84..
:14C8 38 34 2E 0C 20 38 34 2E'84.. 84.
:14D0 10 40 00 00 00 12 20 38'.@.... 8
:14D8 32 2A 12 40 00 00 00 08'2*..@....
:14E0 20 38 32 2A 08 40 00 00' 82*..@..
:14E8 00 02 20 38 32 2A 02 40'... 82*..@
:14F0 01 00 00 30 20 38 32 2A'.... 82*
:14F8 8C 40 00 00 00 8A 20 38'.@.... 8
:1500 34 28 0C 40 00 00 00 08'4<..@....
:1508 20 3C 34 28 08 20 38 34'<4<.. 84
:1510 28 08 14 00 00 00 08 0C'<.....
:1518 00 00 00 00 14 00 00 00'.....
:1520 0C 0C 00 00 00 00 14 00'.....
:1528 00 00 10 0C 00 00 00 00'.....
:1530 14 00 00 00 12 0C 00 00'.....
:1538 00 00 14 00 00 00 14 0C'.....
:1540 00 00 00 00 40 00 00 00'.....@...
:1548 16 20 38 34 2E 16 40 00'. 84...@.
:1550 00 00 10 20 38 34 2E 10'... 84..
:1558 40 00 00 00 08 20 38 34'@.... 84
:1560 2E 08 40 00 00 04 20'...@....
:1568 38 34 2E 30 40 00 00 00'84...@...
:1570 8C 20 38 32 2A 8C 14 3C'. 82*...<
:1578 32 2A 8C 0C 00 00 00 8C'2*.....
:1580 14 00 32 2A 8C 0C 00 00'...2*....

```

```

:1588 00 8C 14 3C 32 2A 8C 0C'...<2*..
:1590 00 00 00 8C 20 38 32 2A'.... 82*
:1598 8C 14 38 32 2A 08 0C 38'...82*..8
:15A0 32 2A 00 14 38 32 2A 0C'2*..82*..
:15A8 0C 38 32 2A 00 14 00 00'..82*....
:15B0 00 10 0C 00 00 00 14'.....
:15B8 00 00 00 12 0C 00 00 00'.....
:15C0 00 14 00 00 00 16 0C 00'.....
:15C8 00 00 00 40 00 00 00 1A'....@....
:15D0 20 38 32 2A 1A 40 00 00' 82*..@..
:15D8 00 12 20 38 32 2A 12 40'.. 82*..@
:15E0 00 00 00 08 20 38 32 2A'.... 82*
:15E8 08 40 00 00 00 02 20 38'..@.... 8
:15F0 32 2A 8C 40 00 00 00 04'2*..@....
:15F8 20 3C 34 2A 04 40 00 00' <4*..@..
:1600 00 0C 20 3C 34 2A 0C 40'.. <4*..@
:1608 00 00 00 12 20 3C 34 2A'.... <4*
:1610 12 40 00 00 00 10 20 3C'..@.... <
:1618 34 2A 0C 40 00 00 00 08'4*..@....
:1620 20 38 34 2E 08 40 00 00' 84..@..
:1628 00 08 20 38 34 2E 10 40'.. 84..@
:1630 00 00 00 16 20 36 2E 2A'.... 6.*
:1638 16 40 00 00 00 12 20 36'..@.... 6
:1640 2E 2A 0C 60 40 38 2E 16'..*...@8..
:1648 14 46 40 38 08 0C 00 00'..F@8...
:1650 00 00 14 46 40 38 08 0C'...F@8..
:1658 00 00 00 00 14 46 40 38'.....F@8
:1660 08 0C 00 00 00 00 40 48'.....@H
:1668 40 3C 10 20 00 00 00 00'@<....
:1670 00 40 40 00 00 10 20 00'..@<....
:1678 00 00 00 40 4A 00 00 1A'...@J...
:1680 20 00 00 00 00 40 40 00' .....@<..
:1688 00 10 20 00 00 00 40'.. .....@
:1690 4E 00 00 1E 20 00 00 00'N....
:1698 00 40 40 00 00 10 20 00'..@<....
:16A0 00 00 00 40 52 00 00 22'...@R..."
:16A8 20 00 00 00 00 00 52 00' .....-R.
:16B0 00 22 40 40 28 10 8A 20'.."@<...
:16B8 40 36 30 28 40 48 00 30'@60<@H.0
:16C0 06 20 48 40 36 30 40 4E'.. H@60@N
:16C8 3C 36 10 20 4E 3C 00 00'<6..N<..
:16D0 40 4A 3C 32 10 20 48 40'@J<2..@H
:16D8 3C 30 00 60 4A 40 3A 1A'<0...J@:.
:16E0 60 44 3E 32 20 40 48 40'..D>2 @H@
:16E8 3C 1E 20 00 00 00 20'<.....
:16F0 40 3C 30 10 20 44 3C 30'@<0..D<0
:16F8 10 20 48 3C 30 10 00 14'.. H<0...
:1700 4A 40 1A 02 0C 40 4A 00'J@...@J.
:1708 00 14 40 4A 2C 14 0C 40'..@J...@
:1710 4A 00 00 14 40 4A 28 10'J...@J<..
:1718 0C 40 4A 00 00 14 40 4A'..@J...@J
:1720 24 0C 0C 40 4A 00 20 14'$..@J..
:1728 40 4A 22 0A 0C 40 4A 00'@J"...@J.
:1730 00 14 40 4A 1E 06 0C 40'..@J...@
:1738 4A 00 00 14 4E 46 1A 02'J...NF..
:1740 0C 46 4E 00 00 14 46 4E'..FN...FN
:1748 2C 14 0C 46 4E 00 00 14'..FN...
:1750 46 4E 28 10 0C 46 4E 00'FN<...FN.
:1758 00 14 46 4E 24 0C 0C 46'..FN$.F
:1760 4E 00 00 14 46 4E 22 0A'N...FN".
:1768 0C 46 4E 00 00 14 46 4E'..FN...FN
:1770 1E 06 0C 46 4E 00 00 14'..FN...
:1778 4A 52 1A 02 0C 4A 52 00'JR...JR.
:1780 00 14 4A 52 2C 14 0C 4A'..JR...J
:1788 52 00 00 14 4A 52 28 10'R...JR<..
:1790 0C 4A 52 00 00 14 4A 52'JR...JR
:1798 24 0C 0C 4A 52 00 00 14'$..JR...
:17A0 4A 52 22 0A 0C 4A 52 00'JR"...JR.
:17A8 00 14 4A 52 1E 06 0C 4A'..JR...J
:17B0 52 00 00 14 4E 54 1A 02'R...NT..

```

```

.:17B8 0C 4E 54 00 00 14 4E 54'..NT...NT
.:17C0 2C 14 0C 4E 54 00 00 14'...NT...
.:17C8 4E 54 28 10 0C 4E 54 00'NT<..NT.
.:17D0 00 14 4E 54 24 0C 0C 4E'..NT$.N
.:17D8 54 00 00 14 4E 54 22 0A'T...NT"'.
.:17E0 0C 4E 54 00 00 14 4E 54'..NT...NT
.:17E8 1E 06 0C 4E 54 00 00 20'...NT..
.:17F0 52 58 1A 02 20 3A 00 00'RX.. :..
.:17F8 0A 20 40 00 00 10 20 4A'..@... J
.:1800 00 00 1A 20 40 00 00 10'....@...
.:1808 20 4A 00 00 1A 20 52 00'J... R.
.:1810 00 22 20 4A 00 00 1A 20'.." J...
.:1818 52 00 00 22 20 58 00 00'R.." X..
.:1820 28 20 52 00 00 22 20 58'< R.." X
.:1828 00 00 28 80 62 58 52 32'..<..XR2
.:1830 20 00 00 00 00 20 00 00'.....
.:1838 10 8A 40 00 00 00 02 80'..@.....
.:1840 00 00 00 00 00 00 00 00'.....
.
```

*Program 17.***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:SYS49248:REM 49263 IS SAMPLING VALUE
140 REM
150 REM GO INTO GRAPHICS MODE
160 REM
170 SYS49627
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 ..))

 $\phi 2$ Out $\phi 2$ 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 $R/\overline{W}$ Read/write**Interrupts***Pins* *I/O*4 Input $\overline{IRQ}$ Interrupt requestD Input $\overline{NMI}$ Non maskable interrupt**Memory mapping***Pins* *I/O*7 Out $\overline{I/O1}$ Address decoded \$DE00-\$DEFF10 Out $\overline{I/O2}$ Address decoded \$DF00-\$DFFF11 Out $\overline{ROML}$ Addr decoded \$8000-\$A000B Out $\overline{ROMH}$ Addr decoded \$E000-\$FFFF8 Input $\overline{GAME}$ Expansion ROM at \$A000-\$C000 (no Basic ROM)9 Input $\overline{EXROM}$ Exp ROM at \$8000**Reset line***Pins* *I/O*C Both $\overline{RES}$ 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 \$8000 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

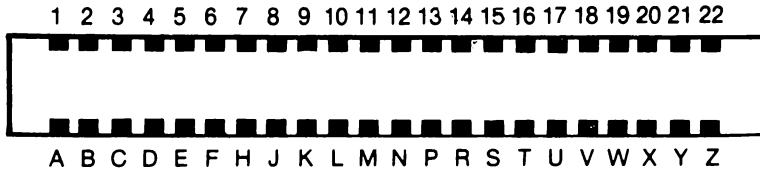
select to LROM and connect the 64's EXROM to ground. The 7464 pins Vpp, Vcc and PGM go to 5 V and Vss goes to ground.

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



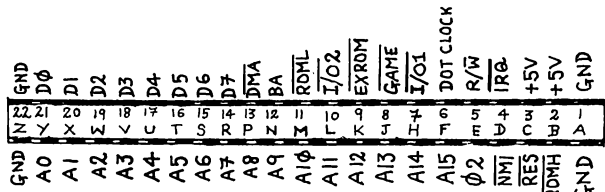
PIN #	TYPE
1	GND
2	+5V
3	+5V
4	$\overline{\text{IRQ}}$
5	$\text{R}/\overline{\text{W}}$
6	DOT CLOCK
7	$\text{I}/\overline{\phi 1}$
8	$\overline{\text{GAME}}$
9	$\overline{\text{EXROM}}$
10	$\text{I}/\overline{\phi 2}$
11	ROML

PIN #	TYPE
12	BA
13	$\overline{\text{DMA}}$
14	D7
15	D6
16	D5
17	D4
18	D3
19	D2
20	D1
21	D0
22	GND

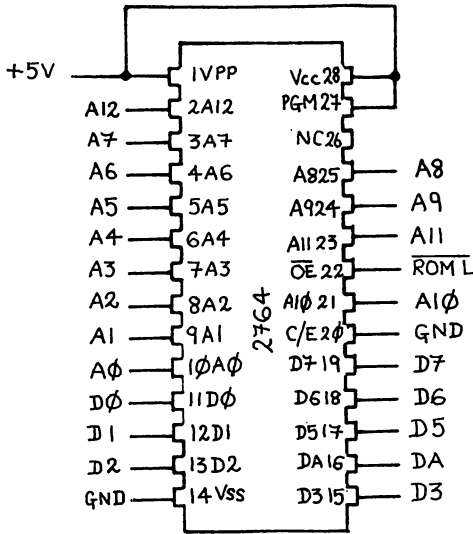
PIN #	TYPE
A	GND
B	$\overline{\text{ROMH}}$
C	$\overline{\text{RESET}}$
D	$\overline{\text{NMI}}$
E	$\phi 2$
F	A15
H	A14
J	A13
K	A12
L	A11
M	A10

PIN #	TYPE
N	A9
P	A8
R	A7
S	A6
T	A5
U	A4
V	A3
W	A2
X	A1
Y	A0
Z	GND

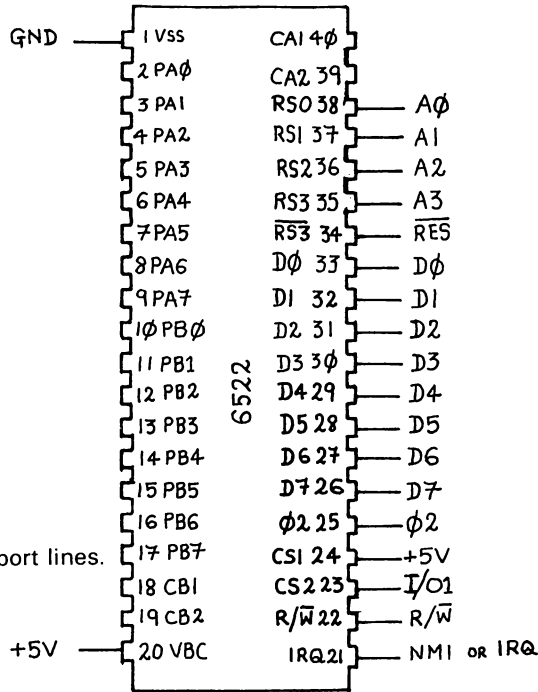
Fig. 5.14. The allocation and function of pins on the memory expansion connector.



Expansion port connector lines.



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 1/60th 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)

Bit	7	Enable/disable (write), occurred (read)
	6	Not used
	5	Not used
	4	FLAG 1 line (cassette read)
	3	Serial data register
	2	TOD clock alarm
	1	Timer B
	0	Timer A

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 0-4 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 $\overline{\text{FLAG}}$ 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)

Bit	7	Set on any enabled VIC IRQ occurring
	6-4	Not used
	3	Light pen (1=occurred)
	2	Sprite to sprite collision (1=occurred)
	1	Sprite to background collision (1=occurred)
	0	Raster compare (1=occurred)

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

Bit	7-4	Not used
	3	Light pen (1=enabled)
	2	Sprite to sprite collision (1=enabled)
	1	Sprite to background collision (1=enabled)
	0	Raster compare (1=enabled)

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 (30 – \$D01E) is set.

3. Sprite to background collision

Sprite to background collision IRQ occurs when any bit in the sprite to background collision register (31 – \$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 $\overline{\text{FLAG}}$ 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 (\$DD0D) 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 (\$DD0D)

Bit	7	Enable/disable (write), occurred (read)
	6	Not used
	5	Not used
	4	User port $\overline{\text{FLAG}}$ line RS232 data received)
	3	Shift register
	2	TOD clock alarm
	1	Timer B
	0	Timer A

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 0–4 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 $\overline{\text{FLAG}}$ 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 10 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

```

C000      *=$C000
C000 A91F      LDA #<NMI          !SET NMI TO POINT
C002 8D1803    STA $0318          ! TO THE TRANSFER
C005 A9C0      LDA #>NMI          ! RECEIVE ROUTINE
C007 8D1903    STA $0319
C00A A9AE      LDA #<SAVWED       !SET SAVE TO POINT
C00C 8D3203    STA $0332          ! TO THE TRANSFER
C00F A9C0      LDA #>SAVWED       ! SEND ROUTINE
C011 8D3303    STA $0333
C014 A990      LDA #$90           !ENABLE USER
C016 8D0DD0    STA $DD0D          ! PORT NMI
C019 A904      LDA #$04           !RESET RECEIVE FLAG
C01B 8D70C0    STA FLAG
C01E 60        RTS
C01F          !
C01F          !ROUTINE TO RECEIVE A FILE OVER
C01F          !THE USER PORT
C01F          !
C01F 48        NMI              PHA          !PUSH OFF REGISTERS
C020 8A        TXA
C021 48        PHA
C022 98        TYA
C023 48        PHA
C024 A910      LDA #$10           !WAS NMI CAUSED BY
C026 2C0DD0    BIT $DD0D          ! THE USER PORT?
C029 D010      BNE LOADIT         !YES
C02B 20BCF6    JSR $F6BC          !NO, UPDATE CLOCK
C02E 20E1FF    JSR $FFE1          !TEST STOP KEY
C031 D037      BNE EXIT           !NOT DOWN
C033 A97F      LDA #$7F          !DISABLE USER
C035 8D0DD0    STA $DD0D          ! PORT NMI
C038 4C66FE    JMP $FE66          !DO NORMAL STOP/RESTORE
C03B          !
C03B A900      LOADIT            LDA #$00          !SET PORT TO INPUT
C03D 8D03DD    STA $DD03
C040 AD01DD    LDA $DD01          !GET INPUT BYTE
C043 EE20D0    INC $D020          !SHOW IT IS WORKING
C046 AE70C0    LDX FLAG          !READING FILE OR
C049 F009      BEQ STRFLE        ! LOAD ADDRESS?
C04B 95FA      STA $FA,X         !LOAD ADDRESS
C04D CA        DEX               !DECREASE FLAG
C04E 8E70C0    STX FLAG          !STORE IT
C051 4C6AC0    JMP EXIT         !EXIT NMI
C054          !
C054 A000      STRFLE            LDY #$00          !NOW READING FILE
C056 91FB      STA ($FB),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 952D      STA $2D           ! POINTERS TO END
C061 A5FE      LDA $FE           ! OF READ FILE
C063 852E      STA $2E
C065 A904      LDA #$04          !RESET LOADING FLAG
C067 8D70C0    STA FLAG
C06A          !
C06A 68        EXIT            PLA          !RESTORE REGISTERS
C06B AS        TRY              ! AND EXIT NMI
C06C 68        PLA
C06D AA        TAX
C06E 68        PLA
C06F 40        RTI
C070          !
C070 04        FLAG            BYT 4
C071          !
C071          !ROUTINE TO SEND A FILE OVER THE
C071          ! USER PORT
C071          !
C071 AD70C0    SAVER            LDA FLAG          !IF RECEIVING,
C074 C904      CMP #$04          ! DON'T SEND
C076 D0F9      BNE SAVER
C078 A204      LDX #$04          !POINT TO SAVE

```

```

C07A B5AB LOOP LDA $AB,X !GET ADDRESS BYTE
C07C 20C2C0 JSR SBYTE !SEND THE BYTE
C07F CA DEX !DO NEXT?
C080 D0F8 BNE LOOP !YES
C082 A000 LDY #$00
C084 B1AC LOOP1 LDA (<$AC),Y !GET A FILE BYTE
C086 20C2C0 JSR SBYTE !SEND IT
C089 2090C0 JSR BUMP !INCREMENT AND TEST END
C08C 90F6 BCC LOOP1 !NOT YET
C08E 18 CLC !SAVED OK
C08F 60 RTS !DONE
C090 !
C090 E6AC BUMP INC $AC !INCREMENT LO BYTE
C092 D002 BNE BUMP1
C094 E6AD INC $AD !INCREMENT HI BYTE
C096 A5AC BUMP1 LDA $AC !COMPARE SAVE
C098 C5AE CMP $AE ! ADDRESS TO END
C09A A5AD LDA $AD ! ADDRESS
C09C E5AF SBC $AF
C09E 60 RTS
C09F !
C09F E6FB BUMP2 INC $FB !INCREMENT LO BYTE
C0A1 D002 BNE BUMP3
C0A3 E6FC INC $FC !INCREMENT HI BYTE
C0A5 A5FB BUMP3 LDA $FB !COMPARE LOAD
C0A7 C5FD CMP $FD ! ADDRESS WITH END
C0A9 A5FC LDA $FC ! ADDRESS
C0AB E5FE SBC $FE
C0AD 60 RTS
C0AE !
C0AE !WEDGE INTO SAVE VECTOR
C0AE !
C0AE A5BA SAVED LDA $BA !TEST DEVICE #
C0B0 C907 CMP #$07 !DEVICE ??
C0B2 F003 BEQ SAVE1 !YES, SEND OVER PORT
C0B4 4CEDF5 JMP $F5ED !NO, NORMAL SAVE
C0B7 A5C1 SAVE1 LDA $C1 !SET SAVE START
C0B9 85AC STA $AC ! ADDRESS FOR USER
C0BB A5C2 LDA $C2 ! PORT SAVE
C0BD 85AD STA $AD
C0BF 4C71C0 JMP SAYER !SEND FILE
C0C2 !
C0C2 !ROUTINE TO SEND 1 BYTE ACROSS
C0C2 ! THE USER PORT
C0C2 !
C0C2 48 SBYTE PHA !SAVE OFF BYTE
C0C3 A9FF LDA #$FF !SET PORT TO OUTPUT
C0C5 8D03DD STA $DD03
C0C8 68 PLA !GET BYTE
C0C9 8D01DD STA $DD01 !SEND TO PORT
C0CC AD02DD LDA $DD02 !SET LINE PA2
C0CF 0904 ORA #$04 ! TO OUTPUT
C0D1 8D02DD STA $DD02
C0D4 AD00DD LDA $DD00 !SEND PA2 HIGH
C0D7 0904 ORA #$04
C0D9 8D00DD STA $DD00
C0DC 20F3C0 JSR DEL !PAUSE
C0DF AD00DD LDA $DD00 !SEND PA2 LOW
C0E2 29FB AND #$FB !NMI HAS BEEN CAUSED
C0E4 8D00DD STA $DD00 ! ON RECEIVING MACHINE
C0E7 20F3C0 JSR DEL !PAUSE
C0EA A900 LDA #$0 !SET USER PORT TO
C0EC 8D03DD STA $DD03 ! INPUT
C0EF EE20D0 INC $D020 !SHOW IT IS WORKING
C0F2 60 RTS
C0F3 !
C0F3 A910 DEL LDA #$10 !PAUSE FOR DATA
C0F5 E901 DE SBC #$01 !TO BE READ
C0F7 D0FC BNE DE
C0F9 60 RTS

```

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 $\overline{\text{FLAG}}$ 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: 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:

<i>Address</i>	<i>Default</i>	<i>Use</i>
\$0314	SEA31	Vector to the IRQ routine. This vector can be changed to point to your own IRQ routine for things such as screen scrolling etc.
\$0316	\$FE66	Vector for BRK instruction is changed by all monitors so that when a BRK is encountered, the computer jumps to the monitor.
\$0318	\$FE47	Vector to the NMI routine. Its major use is for the detection of the RESTORE key. Other methods are outlined in Chapter 5.
\$031A	\$F34A	Vector to open file routine.
\$031C	\$F291	Vector to close file routine.
\$031E	\$F20E	Vector to set input device.
\$0320	\$F250	Vector to set output device.
\$0322	\$F333	Vector to restore I/O.
\$0324	\$F157	Vector to input. This routine is used in all peripheral input. It could be used for function keys etc.
\$0326	\$F1CA	Vector to output. This routine controls all output to the same devices as input (except keyboard).
\$0328	\$F6ED	Vector to test STOP routine. The most widely used patch is for disabling the STOP key.
\$032A	\$F13E	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.
\$032C	\$F32F	Vector to abort I/O.
\$032E	\$FE66	Unused vector. This vector can be used by your own routines.
\$0330	\$F4A5	Vector to load routine. This vector is jumped to after the load parameters have been set up.
\$0332	\$F5ED	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.

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
- character output, 25
- CHKIN, 54
- CHKOUT, 56
- CIA 6526, 4, 8, 15, 33, 70, 78, 79, 127 ff
- CIA signals and lines, 9
- CIOUT, 47
- CLALL, 59
- clock signals, 8, 10, 13, 130, 137
- CLOSE, 57
- 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

- error 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

- 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
- save memory function, 89
- screen, 23
- screen display software, 23
- screen scrolling, 31
- secondary address, 45
- 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

ISBN 0-00-383090-X

COLLINS
Printed in Great Britain

£10.95 net



9 780003 830903



This was brought to you

from the archives of

<http://retro-commodore.eu>