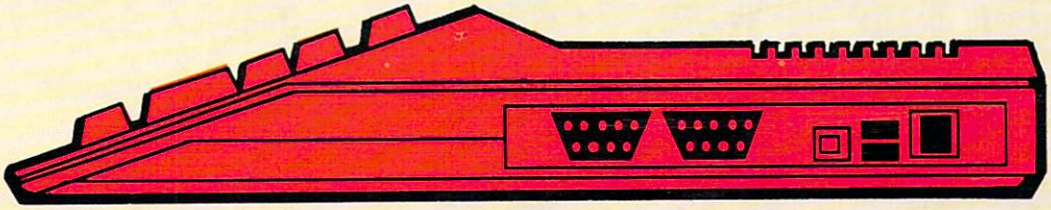
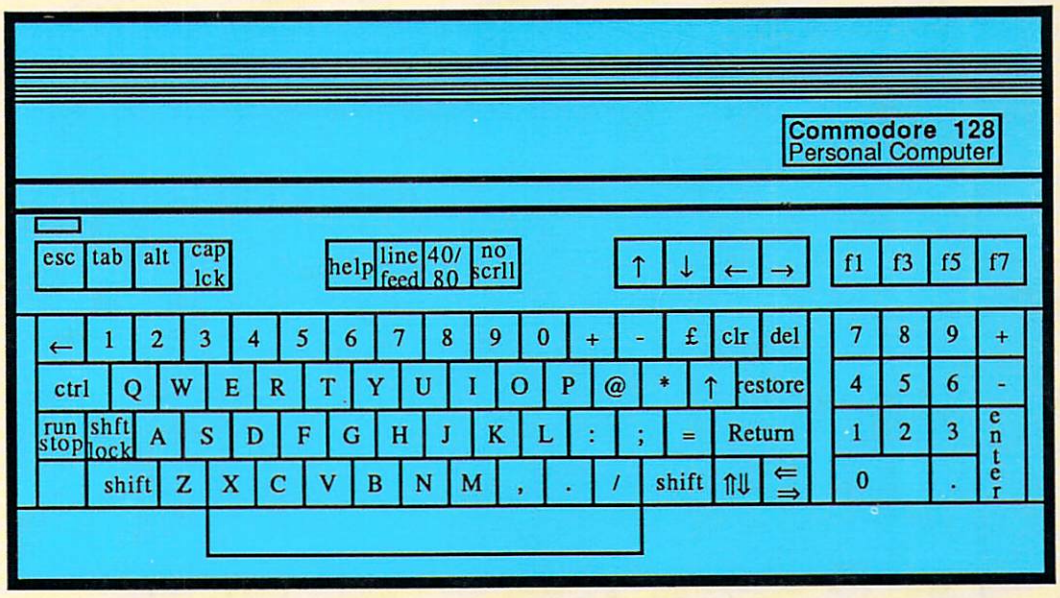
COMMODORE 128

THE AUTHORITATIVE INSIDERS' GUIDE

INTERNALS





A Data Becker book published by

You Can Count On Abacus Software

Commodore 128
Internals

an authoritative insider's guide

By K.Gerits, J.Schieb & F.Thrun

A Data Becker Book

Published by

Abacus Software

First Edition, October 1985
Printed in U.S.A.

ISBN 0-916439-42-9

Table of Contents

CHAPTER 1

Chapter 1: Fundamentals of the C-128

1.1 Introduction to the Commodore 128

After the success of the C-64, Commodore brought out the Plus 4, C-16, and C-116. These computers didn't really offer anything new, but the Commodore 128 does. It's really three computers in one: the well-known C-64, with mountains of software available for it; also, it contains a new computer based on the "success chips" (the 6510 (6502), VIC, SID, 6526, etc); and last, it is a CP/M computer. In total, it's a brand new computer with lots to offer.

The C-128 has an 80-column video controller, so it has the potential of becoming a professional machine. The VIC chip and the 6510 have been changed slightly, though they remain basically the same. It's hard to understand why the 65C02 was not selected as the microprocessor for the C-128, since it runs faster, is compatible with the 6502, and has additional useful commands. This would not have affected the C-64 mode at all. The microprocessor which Commodore did choose is the 8500, which can run twice as fast as its predecessor, the 6510.

The C-128 is also a CP/M computer, it uses CP/M 3.0+. CP/M 3.0 is the version for 128K computers. The Z-80 processor runs at 4MHz. The speed decreases when the bus is accessed, since it was not designed to handle this speed.

We'll be concentrating on both the C-64 and C-128 modes, since they are equally important and equally interesting. The most interesting is the C-128 mode. As a result, the operating system ROM listing and zero page maps are for this mode. Some things can be better explained in the C-64 mode, such as the VIC chip.

This book is the latest in a compehensive series of books from ABACUS Software & Data Becker. We'll go into each component individually and in detail so that the BASIC programmer, whether beginner or advanced, can get an in-depth look. The assembly-language programmer can get the most out of the information presented as well. Naturally, we cannot include all of the C-128's capabilities. This book is not intended as an introduction to BASIC.

Commodore has provided the 128 with an advanced version of BASIC to make use of their advanced computer, BASIC 7.0. Here are some of the important features of the C-128:

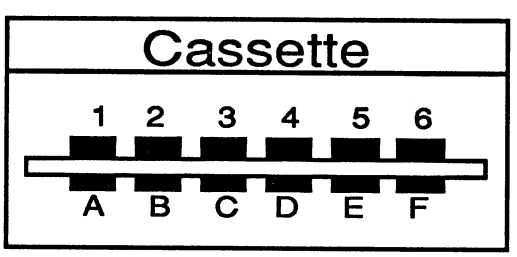
* 128K of dynamic RAM
* 2 x 4K character generator
* Color video controller (VIC) with hi-res graphics
* 80-column video controller (VDC) with RGB output
* Hi-res graphics on the 80-column monitor
* Synthesizer with three independent voices (polyphonic)
* 32K BASIC ROM
* 16K operating system
* 2 parallel I/O ports
* 2 output screens available

At this time we'll be discussing the various input and output ports of the C-128. The outputs for the monitors are not discussed here, since a special chapter is devoted to the chips that generate the video signals.

1.2 The Datasette Connection

The Datasette connections is virtually identical to that found in the C-64. The importance of the Datasette has dropped markedly since the price of the disk drive has been reduced. Only Commodore cassette recorders can be connected to this interface. The recorders are of high quality and have proven very reliable in the past.

The Datasette gets its power via its connector to the C-128. The data travels serially to and from the Datasette through the cable. In addition to the lines for read and write data, there is a line for turning the motor on and off and a line to check to see if the PLAY button is depressed. The figure gives the pin layout for this interface:



Pin	Signal
A-1	GND
B-2	+5V
C-3	CASSETTE MOTOR
D-4	CASSETTE READ
E-5	CASSETTE WRITE
F-6	CASSETTE SENSE

1.3 The User Port

The user port is a 8-bit parallel interface. The user port can be programmed to set any or all of the 8 bits to either input or output. This interface is used frequently by experimenters and individuals interested in computer hardware. The user port can be programmed from BASIC using PEEK and POKE commands. Two handshake lines are available for process control.

To give you an idea of how to program the user port, we have included a short example. Our example circuit consists of four switches, four light-emitting diodes, eight resistors, and one IC. This should be enough to teach you the basic concepts of data input and output using the user port. The circuit diagram is shown at the end of this section; it is very simple, so we have not documented it here.

Since there are so many connections on the user port, we must first explain which connections are actually available to the user. If you are not using an RS-232 cartridge, you can use the following lines without affecting the normal operation of the computer: (1, 2, 4-8, 10-12, A-N).

The layout of the user port lines:

1	GND
2	+5V; up to 100mA
3	-Reset; connected to the processor reset line
4	CNT1; connected to CNT on CIA1
5	SP1; connected to SP on CIA1
6	CNT2; CNT line on CIA2
7	SP2; connected to SP on CIA2
8	-PC2; handshake output on CIA2
9	ATN OUT; control line of the serial bus, comes from PA3 on CIA2
10	9V; 100 mA max.
11	Opposite pole for 10
12	GND
A	GND
B	-FLAG2; handshake input on CIA2
C-L	PB0-PB7; I/O lines from CIA2
M	PA2; I/O line from CIA2
N	GND

Back to our example. Data lines PB0-PB7 can be programmed individually for input or output. We will use lines PB0-PB3 as input and lines PB4-PB-7 as output. This data direction is assigned by simply setting the data direction register for data port B at address 56579. A set bit indicates output on the corresponding bit of data port B (address 56577); a cleared bit indicates input on the corresponding bit of port B. We use the following command to set the data directions for our example (bits 0-3 as input, 4-7 as output):

```
POKE 56579,240
```

This sets the high order bits and the corresponding bits of data port B are set to output while the rest are set to input.

How do we use our little circuit? Nothing could be easier!

```
PRINT PEEK(56577) AND 15
```

returns the values of the four switches and the command

```
POKE 56577,X
```

can be used to turn the LEDs on and off, where the value X may be a combination of the values 16, 32, 64, and 128--the lower bits are only used for reading.

If you have a project of your own already planned--you want to help your wife and connect the washing machine to the Commodore 128--be sure to pay attention to the following so as not to damage your computer:

When using the user port for input, the input voltage must not exceed 5 volts. A voltage from 0 to 0.6 volts is interpreted as zero, while a voltage from 1.6 to 5 volts is interpreted as one. All voltages between 0.7 and 1.5 volts will be randomly interpreted as zero or one.

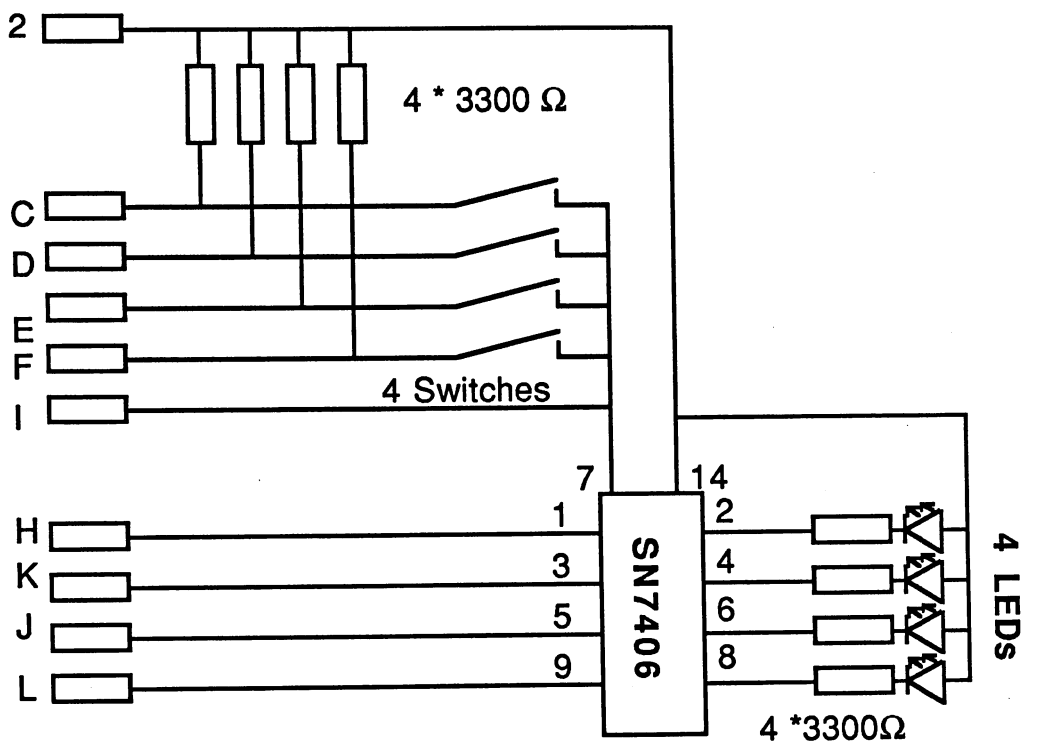
If you use the user port for output, note that the outputs can drive only one TTL input. They cannot directly drive an LED--this would lead to damage to the CIA. It is recommended that you use a buffer, as in our example.

Above all, **NEVER** connect an external voltage to a port with a bit programmed as output. Make sure you load the data direction register with the proper values so you don't mistakenly program an input bit as output.

If you want the computer to power your project, remember that no more than 100 mA of current are available. If this maximum is exceeded slightly, the cassette recorder will refuse to work properly and then the fuse inside the C-128 will blow; finally the primary fuse in the power supply will blow. Hopefully, nothing else will be damaged.

This is intended only as a brief introduction to using the user port in a simple application. If you want to use the other lines for more complex tasks, see Chapter 4 for more information on the CIA.

USER-PORT



1.4 The RS-232 interface

The RS-232 interface opens up the whole world of communications for the Commodore computer user. Most peripherals have an RS-232 interface, such as the laser printer used to print this book. Telephone modems are also connected using such an interface. RS-232 is the designation for an interface for serial data transfer only--parallel data transfer over the phone lines, for example, is not possible.

In serial transmission, the eight bits of a byte are sent one bit at a time, not all eight at once as in parallel data transmission. Serial transmission has the advantage that fewer lines are needed; the disadvantage is that it's slower. It is well-suited for transferring data via telephone lines because so few lines are required.

The software for using the RS-232 interface is built into the C-128 operating system. The interface is available from Commodore as a cartridge which is inserted in the user port. The cartridge is necessary to make the voltage conversions to ±12Volts for the true RS-232 standard.

The RS-232 interface is assigned device address 2 by the operating system. If a logical file is opened with device 2, two 256-byte buffers are allocated: an input buffer and an output buffer. In the 128 mode these buffers are placed at addresses $0C00 and $0D00. In the 64 mode, two pointers point to these buffers: $F7/$F8 points to the RS-232 input buffer and $F9/$FA points to the output buffer. You must also remember the following in C-64 mode: the buffer area is usually located in the upper area of unused memory. If a BASIC program uses the RS-232 interface, the program should begin with the OPEN command because it will erase all of the variables that BASIC stores in upper memory. Furthermore, no check is made to see if enough memory space is available. The CLOSE command frees the buffers again, but the variables are also erased since a CLR command is executed (other files are "forgotten"!). For this reason, you should not close the file until the end of the program. Only one RS-232 file may be open at a time.

When an RS-232 data channel is closed, any transmission is broken off and the buffer is reset. If you want to wait until the entire contents of the buffer have been transmitted, use the command:

SYS 61604 (JSR $F0A4) in the 64 mode or
SYS 59372 (JSR $E7EC) in the 128 mode

This command should always be used before the CLOSE command.

The parameters for data transfer are determined with a control register and a command register. These two registers are passed together with the filename when the file is opened.

The control register defines the baud rate and determines the number of data bits and stop bits transmitted. The baud rate determines the speed of the data transfer. 1000 baud means that 1000 bits are transmitted per second. The stop bits are sent after the data word (5-8 bits).

The command register determines the method of transfer, the parity checking, and the type of handshake.

In the control register, the lowest four bits determine the baud rate according to the following table:

Bit 3 2 1 0	Decimal	Baud rate
0 0 0 0	0	user baud rate, see below
0 0 0 1	1	50
0 0 1 0	2	75
0 0 1 1	3	110
0 1 0 0	4	134.5
0 1 0 1	5	150
0 1 1 0	6	300
0 1 1 1	7	600
1 0 0 0	8	1200
1 0 0 1	9	1800
1 0 1 0	10	2400
1 0 1 1	11	3600 (n.i.)
1 1 0 0	12	4800 (n.i.)
1 1 0 1	13	7200 (n.i.)
1 1 1 0	14	9600 (n.i.)
1 1 1 1	15	19200 (n.i.)

The (n.i.) means that the given baud rate is not implemented and cannot be attained by the C-128. Therefore we can program baud rates between 50 and 2400.

The number of data bits is determined by bits 5 and 6:

Bit 6 5	Decimal	Number of data bits
0 0	0	8 bits
0 1	32	7 bits
1 0	64	6 bits
1 1	96	5 bits

Bit 7 controls the number of stop bits:

Bit 7	Decimal	Number of stop bits
0	0	1 stop bit
1	128	2 stop bits

After we have defined the first byte, we must define the second byte, the command register.

Bit 0	Decimal	Handshake
0	0	3-wire handshake
1	1	X-wire handshake

Bit 4	Decimal	Transfer method
0	0	Full duplex
1	16	Half duplex

Bit 7 6 5	Decimal	Parity checking
x x 0	0	No parity checking no 8th data bit
0 0 1	32	Odd parity
0 1 1	96	Even parity
1 0 1	160	8th data bit always 1
1 1 1	224	no parity checking 8th data bit always 0

A comment about handshaking: if you select a 3-wire handshake, the control lines CTS (Clear To Send) and DSR (Data Set Ready) are not checked when sending and receiving. This means that the computer sends the data (to a printer for example) whether the receiver is ready to process the data or not. If we want the device to be able to stop the transmission, we must select X-wire handshake. The two control lines just mentioned must

be wired; the assumption is that the receiver can service these lines. If two computers are being connected, a 3-wire handshake is usually sufficient.

Let's go through an example: We want to open an RS-232 data channel with the following parameters:

* 2400 baud
* 7 data bits (ASCII)
* 2 stop bits
* No parity checking
* 8th data bit always 0
* Full duplex
* 3-wire handshake

After you have determined all the bits from the above tables, open the channel with the following OPEN instruction:

```
OPEN 1,2,0,CHR$(10+0+128)+CHR$(0+0+224)
```

The second byte in the OPEN instruction is usually CHR$(0).

1.4.1 Programming the baud rate

The various baud rates are implemented through the timers in the CIAs. You can also program baud rates that are not in the table, such as 111 baud. The maximum rate of 2400 baud cannot be exceeded, because the software in the operating system is too slow. The CIAs (or the timers) generate an NMI after a certain amount of time dependent on the baud rate. If we want to use our own baud rate, we can pass the corresponding timer values as the third and fourth characters of the filename in the OPEN command. The timer values can be obtained from the following formula:

$$T = 492662/BAUD - 101$$

The value which we get from this formula must be split into high and low bytes and then passed as the third and fourth characters of the filename. In the control register we use a zero instead of the baud rate (user baud rate), so that the operating system knows that we want to use our own baud rate.

The following example uses the same parameters as the previous example, except that the baud rate is set to 1000.

```
100 BAUD=1000
110 T=492662/BAUD-101
120 TH=INT(T/256): TL=T AND 255
130 OPEN 1,2,0,CHR$(128)+CHR$(224)+CHR$(TL)+
    CHR$(TH)
```

Baud rates between 8 and 2400 baud can be obtained with the user baud-rate programming option.

1.4.2 Reading the status variable ST

The status variable ST is used to determine if any errors occurred while transferring data via the RS-232, just as with the serial bus. The meaning of ST is somewhat different for the RS-232, however. The variable ST is reset (to zero) each time it is read in BASIC. Therefore, if you'll be checking the status variable multiple times you must store the value in a temporary value: `A=ST`. Now `A` can be checked multiple times without resetting the status variable ST. The status value should be available for multiple checks, so it must be stored in a temporary variable.

Here is the bit by bit breakdown of the status variable ST. A set bit indicates that the given event occurred.

Bit	Description
0	Parity error
1	Framing error
2	Receiving buffer full
3	Receiving buffer empty
4	CTS (Clear To Send) signal missing
5	Unused
6	DSR (Data Set Ready) signal missing
7	Break signal received

In the C-64 mode you can assign the memory area the RS-232 input and output buffers will be located. In the C-128 mode these buffers have preassigned locations. The pointers for these buffers are at addresses $F7-$FA.

1.5 Cartridge Port

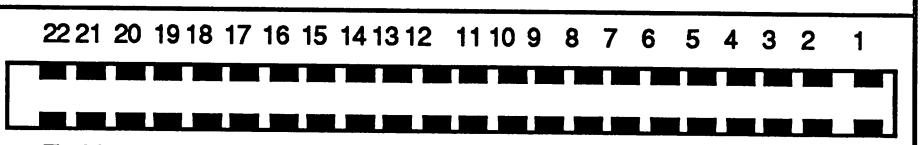
The cartridge port--also known as the expansion bus--is one of the most useful interfaces on the C-128. ROM cartridges can be inserted in this port; they might be games, BASIC extensions or something altogether different such as a MIDI interface. The address lines as well as the data lines of the computer are available on this interface. For this reason the computer is also very sensitive to damage here.

First the pinout of the 44-pin connector:

1	GND
2-3	+5V
4	-IRQ; connected to the processor IRQ line
5	CR/-W; connected to the processor R/-W line
6	DOT CLOCK; dot raster clock for the VIC, about 7.83 MHz
7	-I/O1; usually =0 in address range $DE00 to $DEFF
8	-GAME; input to AM (Address Manager)
9	-EXROM; as above
10	-I/O2; usually =0 in area $DF00 to $DFFF
11	-ROML; output from AM
12	BA; signal from VIC, indicates the validity of read data
13	-DMA; input. 0=bus system reserved for external access
14-21	CD7-CD0; data bus
22	GND
A	GND
B	-ROMH; output from AM
C	-RESET
D	-NMI
E	02; system clock output
F-Y	CA15-CA0; address bus
Z	GND

Both the 128 and 64 modes test to see if the cartridge port is occupied when the computer is turned on or reset. If the cartridge port is occupied, the memory configuration is set appropriately in the address manager, and control of the computer is given to the cartridge and not the built-in ROM operating system. This is a very user-friendly feature, since the user need only insert the cartridge and turn the computer on to start the application.

EXPANSION PORT



PIN	SIGNAL
22	GND
21	CD0
20	CD1
19	CD2
18	CD3
17	CD4
16	CD5
15	CD6
14	CD7
13	$\overline{\text{DMA}}$
12	BA
11	$\overline{\text{ROML}}$
10	I/O2
9	$\overline{\text{EXROM}}$
8	$\overline{\text{GAME}}$
7	I/O1
6	DOT CLOCK
5	CR/W
4	$\overline{\text{IRQ}}$
3	+5V
2	+5V
1	GND

PIN	SIGNAL
Z	GND
X	CA0
Y	CA1
W	CA2
V	CA3
U	CA4
T	CA5
S	CA6
R	CA7
P	CA8
N	CA9
M	CA10
L	CA11
K	CA12
J	CA13
H	CA14
F	CA15
E	02
D	$\overline{\text{NMI}}$
C	$\overline{\text{RESET}}$
B	$\overline{\text{ROMH}}$
A	GND

USER PORT

PIN	SIGNAL	CAPACITY
1	GND	
2	+5V	MAX. 100mA
3	RESET	
4	CNT1	
5	SP1	
6	CNT2	
7	SP2	
8	PC2	
9	SER. ATN IN	
10	9 VAC	
11	9 VAC	
12	GND	

PIN	SIGNAL
A	GND
B	FLAG2
C	PB0
D	PB1
E	PB2
F	PB3
H	PB4
J	PB5
K	PB6
L	PB7
M	PA2
N	GND

CHAPTER 2

Chapter 2: The VIC Chip

As you already know, the Commodore 128 has three plugs for connecting monitors. Theoretically, all three can be used at once, but this wouldn't be terribly useful, since the two 40 column screens would be identical.

Two of the three connectors are connected to the VIC chip. The VIC chip has been well-proven in the Commodore 64. The VIC chip is well liked since it has many fine features like the ability to display sprites. The VIC chip in the Commodore 128 has two additional registers which will be described later. It runs the display in the 40-column mode as well as BASIC 7.0's representation of graphics.

A television can be connected via the RF connector. This is a relatively popular solution because of the low cost. Depending on the television, the screen quality may also be satisfactory, though it is not suited for long periods of working with the computer. This is because the carrier frequency is first modulated by the computer (it must be "broadcast") and then demodulated by the television receiver. The picture quality naturally suffers as a result of all this manipulation.

If your wallet has recovered from the purchase of the Commodore 128, you might consider a color monitor such as the Commodore 1702. This monitor uses the second connection: the composite video output. Here the signal does not need to be modulated or demodulated--pure screen information plus the synchronization pulse is sent to the monitor. These monitors are a bit more expensive, but they offer significantly better screen quality because the screen resolution is better.

The VIC chip in the Commodore 128 has the same address as the 64, which makes sense, since it must also be accessed in the 64 mode. For the sake of compatibility the addresses must remain the same.

Start address: $D000

The VIC-II chip (we will call it VIC-II since it is not identical to its predecessor) cannot function with the 2MHz clock frequency (fast mode). The VIC-II chip contains the system clock. As you may know, the VIC chip uses the clock gaps (times in which the processor does not access the memory) in order to get characters out of the video RAM to refresh the

picture. This is done so as not to slow down the processor. If the processor is clocked at 2MHz, the operating speed is doubled and the clock gaps are halved. These clock gaps aren't long enough to access memory. The VIC-II chip switches the video output off and you get a single color picture (which you may recognize from cassette loading). The video controller responsible for the 80-column screen is not affected by this. It continues to display its 80 columns per line. Switching from 1 to 2MHz can also be done in the 64 mode! To do this, you must set bit 0 in register 48 of the VIC.

POKE 53296,1 corresponds to the command FAST
POKE 53296,0 corresponds to the command SLOW

These two POKEs can also be used in the 64 mode. The FAST command is a bit different from the POKE command; the BASIC 7.0 command FAST also causes the 40-column screen to be automatically switched off, so that the colorful garbage caused by the 2MHz mode does not appear on the screen.

The VIC chip not only performs all the tasks required to create a screen, it also handles the timing for the dynamic memory.

Here are some features of the VIC chip:

* 16 colors
* graphics-capable with 320x200 pixels (hi-res mode)
* Four color graphics with 160x200 pixels (multi-color mode)
* Multi-color mode possible in text mode
* Display and management of 8 sprites
* Raster and sprite-collision interrupt
* Creation of a standard NTSC signal
* Movable video RAM and character generator
* Independent handling of 16K of dynamic RAM

The pin layout of the VIC-II chip:

1-7	D6-D0;	Processor data bus
8	-IRQ;	0 when one bit of the IMR and the IRR are equal
9	-LP;	Input, Light pen strobe
10	-CS;	Processor-bus action only takes place if CS=0
11	R/W;	0 = taking over data from bus
12	BA;	0 = data not ready at receiving device
13	VDD;	+12VDC
14	COLOR;	Color information output
15	SYNC;	Impulses to synchronize lines and screen
16	AEC;	0 = VIC uses system bus, 1 = bus free
17	0OUT;	Clock output
18	-RAS;	Dynamic RAM control
19	-CAS;	as above
20	GND	
21	0COLOR;	Input color frequency
22	0IN;	Input dot frequency
23	A11;	Processor address-bus
24-29	A0/A8--A5/A13;	Multiplexed (video-) RAM address-bus
30-31	A6-A7;	(video-) RAM address-bus
32-34	A8-A10;	Processor address-bus
35-38	D11-D8;	Data from color RAM
39	D7;	Processor data-bus
40	VCC;	+5V
41-44	K0-K3;	Keyboard-Interface-Control. These pins go directly to the (expanded) keyboard.

2.1 Register Layout of the VIC Chip

The VIC-II chip has 49 registers at the address $D000+(the register number). These registers are individually described:

REG 0 Sprite register 0: X-coordinate
Here are 8 bits of the X screen coordinate of sprite 0. Bit 9 (overflow bit) is found in register 16 of the VIC chip.

REG 1 Sprite register 0: Y-coordinate
This register contains the Y-position of sprite 0. The Y-coordinate does not need an overflow (9th bit) because the maximum Y-value is 199.

Registers 2 through 15 correspond to registers 0 and 1 for sprites 1 to 7. Each sprite has a register pair in the VIC chip: Sprite 0 has register pair 0/1, sprite 1 the pair 2/3 ... sprite 7 the pair 14/15.

REG 16 MSb of the X-coordinates (note that the lower-case b in MSb is intentional! [This is to indicate bit, not byte]). This register contains the overflow bits from the X-coordinates of the sprites. A set bit means that the MSb (9th bit) of the corresponding sprite is set, 0 means not set. The MSb of sprite 0 is represented by bit 0, the MSb of sprite 7 is set by bit 7.

REG 17 Control register 1
Bit 0-2 : Offset of the upper screen border in raster lines.
Bit 3 : 0=24 lines, 1=25 lines
Bit 4 : 0=screen off
Bit 5 : 1=standard bit-map mode (graphics)
Bit 6 : 1=extended color mode (text)
Bit 7 : Carry from register 18.

REG 18 Raster IRQ
Number of the raster line at which a raster IRQ should be generated. The 9th bit of the raster line is found in register 17.

REG 19 X-portion of the screen position at which the beam was found when a strobe was generated.

REG 20 As register 19, but the Y-portion.

REG 21 Sprite enable
This register indicates whether a sprite is turned on (bit = 1) or off (bit = 0). Sprite 0 is represented by bit position 0, sprite 7 by bit 7 of the register.

REG 22 Control register 2
Bits 0-2: Offset of the left screen border in raster dots.
Bit 3: 0=38 characters, 1=40 characters (horizontal)
Bit 4: Multi-color mode (graphics)

REG 23 Sprite expand X
The sprites can be doubled in the x direction by setting the corresponding bit in this register.

REG 24 Base address of the character generator and video RAM
Bits 1-3: Address bits 11-13 for the character RAM base
Bits 4-7: Address bits 10-13 for the video RAM

REG 25 IRR: Interrupt Request Register
This register indicates which register generated an interrupt.
Bit 0: generator is REG 18
Bit 1: generator is REG 31
Bit 2: generator is REG 30
Bit 3: generator is pin LP
Bit 7: =1 when at least one other bit is one

REG 26 IMR: Interrupt Mask Register
Layout like REG 25. If at least one bit in the IRR and IMR agree (IRR AND IMR<>0), an interrupt is generated (pin IRQ=0).

REG 27 Priority register (sprites)
If the corresponding bit is set, the background character has precedence over the sprite.

REG 28 Multi-color register (sprites)
If the bit representing a given sprite is set, that sprite is represented in multi-color mode.

REG 29 Sprite expand Y
The sprites can be doubled in the Y-direction by setting the appropriate bit in this register.

REG 30 Sprite/sprite collision
Each sprite is assigned a bit. If two sprites touch each other, the two corresponding bits are set. These bits remain set until they are explicitly cleared! At the same time, bit 2 in the IRR is set. If bit 2 in the IMR is also set, an interrupt will be generated.

REG 31 Sprite/background collision
Each sprite is assigned a bit. If a sprite touches the background, the corresponding bit is set. The bits remain set until they are explicitly reset! Bit 3 in the IRR is set; if bit 3 in the IMR is also set, an interrupt is generated.

REG 32 Exterior color (border color)
The border color is set in this register (0-15).

REG 33 to REG 36 Background color registers 0-3
Background color register 0 determines the background color in the "normal" text mode. If the multi-color mode is enabled, it accesses registers 1-3.

REG 37 and REG 38 Sprite multi-color color 0/1
Sprites which are represented in multi-color can assume the background color, the sprite color, or the multi-color 0 and 1.

REG 39 to REG 46 Color sprite 0-8
The colors for the individual sprites are placed in these registers.

REG 47 Keyboard control register
This register contains the status of the four keyboard interface pins K0 to K3. Bits 0 to 3 are responsible for this. Bits 4-7 are unused and are always 1.

REG 48 2MHz bit
Bit 0 of this register determines whether the computer operates at 2MHz or 1MHz. Bits 1-7 are unused. If the bit is set, all accesses from the VIC-II chip to the memory are halted, except for refreshing the dynamic RAM.

NOTE: All of the following example programs must be entered in the 64 mode. This is necessary because the BASIC 7.0 interpreter makes inputs to the VIC-II chip practically "ineffective". For example, if you switch the graphics on with the necessary POKE instructions, you will see only a flash on the screen. The same applies to programming sprites, etc. The reason for this is that the BASIC 7.0 interpreter must have its own method of interrupt control. You can, for example, create a moving sprite with the MOVSPR command; this can be done only with BASIC 7.0 using the interrupts. We will tell you how you can get around this interrupt control in Section 7.5.

But even when the sprites aren't moving, the coordinates are always corrected by the BASIC 7.0 interpreter. You are probably asking yourself why you should program in the 64 mode when you own a 128. This is a good question, but the VIC chip can be programmed just as well from the 64 mode as it can from the 128 mode. We will use "simple" POKE commands in the following sections, in order to give examples as close to assembly language as possible. Since programming the VIC chip would be ruined by the BASIC 7.0 interpreter, we will try out the following examples in the 64 mode. This will allow us to learn and understand the operation of

the VIC chip. Machine language programmers have to feel their way through step by step. In machine language (in the 128 mode), you can get around the annoying sprite corrections by changing the IRQ vector.

2.2 The VIC Operating Modes

As you may already know and can gather from the many registers, there are a number of possible ways to arrange the screen with the VIC chip. It is quite easy to do this in the 128 mode thanks to easy-to-use BASIC 7.0 commands. In the 64 mode, it is somewhat more difficult to switch between the various modes since it must be done with POKE commands. Programming sprites in the 64 mode is also more complicated than it is in the 128 mode, in which you can easily move them about with the MOVSPR command. If you think the layout of the VIC chip doesn't interest you since you don't want to program in the 64 mode, you may not be right. If you want to program in machine language, you will need to learn more about the register layout of the VIC, which is what we want to do now.

2.3 Sprites

Sprites are movable, freely-definable figures with a resolution of 24 by 21 points. Sprites can be represented in either the two-color mode (sprite color and background color) or the multi-color mode (four colors, but the resolution is cut to 12 by 21 points). The VIC chip can manage 8 sprites, which can be moved simultaneously on the screen. The sprites can assume their positions in a frame of 512 by 256 raster points, which means that sprites can be moved completely outside of the screen.

If a sprite is defined in the two-color mode, a set bit means a set point in the color defined for this sprite. An unset bit means transparent (the background color will be displayed). In the multi-color mode, two bits apply to one point, which means that one can define four colors. The possible bit combinations refer to the following colors:

00: Transparent, background color	(REG 33)
01: Multi-color register 0	(REG 37)
11: Multi-color register 1	(REG 38)
10: Sprite-color register	(REG 39-46)

You see that two colors (multi-color registers 0 and 1) are defined to be the same for all sprites. The sprites can differ from each other in at most one color. But let's define a sprite "from scratch". We won't use the BASIC 7.0 commands, but only the commands available to us in the 64 mode (which can be used in the 128 mode as well). First we must define a sprite by means of DATA statements (the sprite editor does not exist in the 64 mode). These DATA lines should look like the following:

```
1000 DATA 000,000,000
1010 DATA 000,000,000
1020 DATA 000,000,000
1030 DATA 000,000,000
1040 DATA 000,000,000
1050 DATA 000,000,000
1060 DATA 000,000,000
1070 DATA 003,255,255
1080 DATA 000,002,000
1090 DATA 192,170,128
1100 DATA 194,150,080
1110 DATA 234,150,080
1120 DATA 194,170,168
1130 DATA 192,170,168
1140 DATA 000,032,128
1150 DATA 000,170,160
1160 DATA 000,000,000
1170 DATA 000,000,000
1180 DATA 000,000,000
1190 DATA 000,000,000
1200 DATA 000,000,000
```

In the normal development of a sprite, you would draw out the figure on paper before programming, and divide the paper up into a grid of 24 by 21 points. This gives 21 lines of 24 points each. These 24 points are then grouped into three 8-bit groups which can then be stored as bytes. Every filled box means a set bit, an empty box means an unset bit. In the multi-color mode this is more difficult. You must insert one of four bit combinations from a self-defined color table.

Note: You must first consider what colors you will define in common to all sprites, and which you want to have as the individual color for each sprite.

Once you have done this you can calculate the individual bytes and write them down. These values are then given in rows of DATA lines, as in our example. Our example sprite is a helicopter. You probably didn't recognize it in the DATA statements.

2.3.1 Address of the sprites

We have our data and now we need to store it someplace. There is a pointer for each sprite which tells the VIC chip where it can find the sprite. These pointers are found in addresses 2040 to 2047, immediately following the video RAM. Each sprite needs 3x21=63 bytes. You have probably already noticed that each pointer need only be one byte long and does not give an absolute address. It gives the position "pointer times 64," which accounts for exactly 16K. If you move the start address of the video RAM, the sprite pointers also move as well as their start addresses. For the sake of simplicity, let us assume that sprite number 1 is defined at address 13*64=832.

POKE 2041,13

Address: 2040 2041 2042 2043 2044 2045 2046 2047
Sprite #: 0 1 2 3 4 5 6 7

You can assign this address to other sprites, meaning that several sprites will have the same appearance. But to display our sprite, we first need to POKE the values from the DATA statements into the correct memory addresses.

```
10 FOR I=0 TO 63
20 READ D
30 POKE 13*64+I,D
40 NEXT
50 POKE 2041,13: REM SPRITE 1 AT ADDRESS 832
```

2.3.2 Turning on the sprite

When you start the program, you will notice that something is still missing. We need to explicitly turn our sprite on! The best way to do this is with a logical OR of the corresponding bit in register 21, since a direct POKE would erase any other sprites.

POKE 53248+21,PEEK(53248+21) OR 2

turns sprite 1 on. If you want to turn on sprites 0 and 7, for example: POKE53248+21,PEEK(53248+21) OR 1 OR 128, or better yet: POKE53248+21,PEEK(53248+21) OR 129.

To turn off sprite 1:

POKE 53248+21,PEEK(53248+21) AND NOT(2)

If you want to turn off several sprites at once, such as sprites 0 and 7,

POKE 53281+21,PEEK(53248+21) AND NOT(1 OR 128)

it can be done by a logical OR of the sprites to be turned off, which is then negated and then ANDed with the original value. In our example program, we want to turn our sprite on:

```
60 POKE 53248+21,1: REM TURN ON SPRITE 1
```

Sprite: 7 6 5 4 3 2 1 0
Bit: 7 6 5 4 3 2 1 0

2.3.3 Color

We want to be able to define the color of our sprite, otherwise we might not be able to see it:

```
70 POKE 53248+39+1, 5: REM COLOR = GREEN
```

This is done in registers 39 through 46: register 39 defines the color for sprite 0; register 46, correspondingly, defines the color of sprite 7.

The following colors are available:

0	Black	8	Orange
1	White	9	Brown
2	Red	10	Light red
3	Cyan	11	Grey 1
4	Purple	12	Grey 2
5	Green	13	Light green
6	Blue	14	Light blue
7	Yellow	15	Grey 3

2.3.4 Position

After you have made the color specification and started the program with RUN, you still won't see anything because the sprite is positioned outside of the screen area. Registers 2 and 3 must be loaded with the appropriate values in order to assign a position to sprite 1:

```
80 POKE 53248+2, 50 : REM X-COORDINATE
90 POKE 53248+3, 70 : REM Y-COORDINATE
```

You can move your sprite across the whole screen with a loop. Many readers may start to groan here. You know that BASIC 7.0 handles all of the work with sprites for you. But there's even more that must be done in 64 mode. If you want to position the sprite at X-coordinate 310, for example, eight bits aren't enough. Here you must set the ninth bit of the corresponding sprite in register 16 (or reset it if you are moving the sprite from right to left). We position our sprite at X-coordinate 310:

```
POKE 53248+16, 2: REM SPRITE 1 - SET 9TH BIT
```

If you want to avoid disturbing other sprites with this command, you must again address the appropriate bit explicitly:

```
POKE 53248+16, PEEK(53248+16) OR 2
```

Let's move our sprite from left to right across the screen:

```
FOR I=0 TO 400
POKE 53248+2, I AND 255 :
            REM MASK OUT LOWER 8 BITS
POKE 53248+16,PEEK(53248+16) AND NOT 2 OR
2*ABS(I>255)
NEXT I
```

The line just before the last is a bit complicated: The most-significant bit of sprite 1 is reset to zero by AND NOT 2. The corresponding bit is again set if necessary (X-coordinate greater than 255) by OR 2*ABS(I>255). This is all done without disturbing the other bits.

2.3.5 Expanding a sprite

Another important and useful capability is the ability to display sprites twice as large in the horizontal and/or vertical directions. The VIC chip has two registers available for this purpose: X-expand and Y-expand. Again, each sprite is represented by a bit. By setting this bit, the corresponding sprite is expanded in the X or Y direction. In our example we will expand our sprite in both the X and Y directions:

```
POKE 53248+23,2 : REM DOUBLE SPRITE 1 IN Y-DIRECTION
POKE 53248+29,2 : REM DOUBLE SPRITE 1 IN X-DIRECTION
```

Since we can expand a sprite in both the X and Y directions, we have the ability to enlarge our sprite by a factor of four.

2.3.6 Background

You have no doubt noticed when entering or changing the example program that the sprite does not scroll along with the rest of the screen. Sprites also remain visible when the screen is cleared. The sprites are ultimately determined by their position. If you want to remove a sprite from the screen, you can either a) turn it off, or b) position it outside the screen.

Sprites have another noteworthy property. If you move the text cursor over a sprite and start typing, the sprite covers the letters--the letters are visible only where the sprite is transparent. It almost has the appearance of a three-dimensional picture.

The sprites and the background can be imagined as two separate layers. It is possible to inform the VIC chip that we do not want to have individual sprites in the foreground. There is a priority level for each sprite that tells the VIC whether the sprite has precedence over the background or not. In our example, the letters would appear on top and the sprite would be covered up. In order to move a sprite behind the background, the corresponding bit in register 27 must be set. We want to take away the priority of our helicopter:

```
POKE 53248+27,2
```

Now the helicopter appears behind the letters. In order to put it in front again, we need only reset the bit:

```
POKE 53248+27,0
```

Register 27 : Background priority

```
Bit:   7 6 5 4 3 2 1 0
Prior: 7 6 5 4 3 2 1 0
```

You have no doubt noticed that all registers are organized in the same manner. One byte is all that is required in order to represent all eight possible sprites. Bit 0, the lowest order bit, always stands for sprite 0 while bit 7 always corresponds to sprite 7.

You may be wondering what happens when several sprites occupy the same space on the screen. There are set rules for determining the appearance of the result. The sprite with the lowest number appears on "top" of the others. If sprites 0 and 6 come in contact with each other, for example, all of sprite 0 will be visible, while at best only an outline of sprite 6 will be visible. Sprite 6 will appear on top of sprite 7, sprite 5 on top of sprite 6, up to sprite 0 on top of sprite 1. The lower the sprite number, the higher the priority.

2.3.7 Collision: Sprite-sprite

It is also possible that two sprites will come into contact with each other, that is, they have at least one point in common. Often it is desirable to be able to detect such contact, especially for games. The VIC has a register just for this purpose: Register 30 gives the information if sprites

have collided, and if so, which sprites were involved. If, for example, sprites 0 and 6 collide, bits 0 and 6 of register 30 are set. If more than two sprites encounter each other, the bits of all the sprites involved are set. In our example--if sprites 0 and 6 encounter each other--we would get the following result:

```
PRINT PEEK(53248+30)
65
```

The number 65 is a combination of bits 0 and 6 set: 64+1=65. After you have read register 30, you must set it back to 0, or you will not be able to detect future collisions since the register is not automatically reset.

```
POKE 53248+30,0
```

2.3.8. Collision: Sprite-background

Sprites can also come into contact with the background characters. It is possible to check to see if our helicopter comes into contact with the cursor or not. This test is independent of whether the sprite has precedence over the background or not. If a sprite does contact some part of the background, the corresponding bit in register 31 is set. Here the same applies as for register 30: You must clear the register after reading it. The register can only tell that the given sprite has come into contact with a background character, it cannot tell you which character, though that usually doesn't matter. This can be determined by the position of the sprite.

2.3.9 Multi-color sprites

Certainly the "icing on the cake" of sprite programming is the ability to define sprites in multi-color. Multi-color simply means four-color. One color is the background color; two additional colors are the same for all eight sprites. If you want to display several sprites in multi-color, you must consider carefully what colors you will choose. You must then define these in the two fixed sprite color registers. The multi-color mode does have a price: the resolution is cut in half. This usually does not present a problem since the resolution is usually more than enough. This gives you a resolution of 12x21 points. The size of the sprites remains the same since the points themselves become twice as large--two bits define one color.

The various bit combinations have the following meanings:

00 The point has the background color (no point is visible)
01 The color is taken from register 37
10 The color is taken from the given sprite color register
11 The color is taken from register 38

We must tell the VIC chip which sprites are multi-color. This is naturally done bit by bit, in register 22. To display our helicopter as multi-color:

POKE 53248+22,2

And look: it appears in shimmering color. The helicopter looks so ugly because we defined it as a single color sprite. The various bit combinations of a monochrome sprite naturally have a different character than they do with a multi-color sprite. We'll now list the entire program responsible for bringing our helicopter to life. This program will help show you how sprites are programmed, whether in BASIC or machine language.

```
10 REM SPRITE DEMONSTRATION PROGRAM
20 V = 53248: REM START ADDRESS OF THE VIC CHIP
30 POKE V+32, 15; POKE V+33,14:REM BACKGROUND COLOR
40 PRINT"<CTRL-7>": REM <CRTL> KEY AND 7
50 POKE V+21, 3 :  REM ENABLE SPRITE 0 AND 1
60 POKE V+28, 3:   REM SPRITE 0 AND 1 IN MULTICOLOR
70 POKE V+39, 6 :  REM COLOR FOR SPRITE 0 = BLUE
80 POKE V+40, 2:   REM COLOR FOR SPRITE 1 = RED
90 POKE V+37, 14: REM MULTI-COLOR 1 = LIGHT BLUE
100 POKE V+39, 0: REM MULTI-COLOR 2 = WHITE
110 POKE 2040, 13: REM SPRITE 0 AT 832 TO 895
120 POKE 2041, 13 : REM SPRITE 1 THE SAME
130 FOR I = 0 TO 62: REM NUMBER OF DATA ITEMS
140 : READ X :     REM READ THE VALUES
150 : POKE I+832, X : REM STORE THE VALUES
160 NEXT I
170 POKE V+0,25:POKE V+1, 50:REM POSITION SPRITE 0
180 POKE V+2, 60:POKEV+3,50 :REM POSITION SPRITE 1
190 FOR D = I TO 2000 : NEXT:REM DELAY LOOP
200 FOR I = 0 TO 200 : REM MOVE
210:      POKE V, I=24 : REM X-COORD. SPRITE 0
220:      POKEV=2, 200-I :REM Y-COORD. SPRITE 1
```

```
230:      POKE V=1, 40+I: REM Y=COORD. SPRITE 0
240:      POKE V+3, 200-I:REM X-COORD. SPRITE 1
250 NEXT
260 GOTO 200:      REM MOVE CONTINUALLY
1000 DATA 000,000,000
1010 DATA 000,000,000
1020 DATA 000,000,000
1030 DATA 000,000,000
1040 DATA 000,000,000
1050 DATA 000,000,000
1060 DATA 000,000,000
1070 DATA 003,255,255
1080 DATA 000,002,000
1090 DATA 192,170,128
1100 DATA 194,150,080
1110 DATA 234,150,080
1120 DATA 194,170,168
1130 DATA 192,170,168
1140 DATA 000,032,128
1150 DATA 000,170,160
1160 DATA 000,000,000
1170 DATA 000,000,000
1180 DATA 000,000,000
1190 DATA 000,000,000
1200 DATA 000,000,000
```

It is certainly more complicated to prepare multi-color sprites than single-color sprites, in which a point on paper corresponds directly to a point on the screen. Fortunately there are sprite editors which make the work a good deal easier. Such an editor is built in to BASIC 7.0 (SPRDEF). But as we said before, it is very important for the machine language programmer to know how sprites are programmed without BASIC commands.

The sprites that you define and use with the sprite editor built into BASIC 7.0 are stored in RAM at $0E00-$1000.

Sprites in any of the possible modes can be covered by the background, whether it be in text, graphic, or multi-color graphic mode.

2.3.10 Interrupts through the VIC chip

The VIC chip is capable of generating interrupts. Interrupts temporaily halt the machine language program currently being executed by the microprocessor because a certain event occurred. There are four different sources of interrupt on the VIC:

* The lightpen
* The raster-line interrupt
* A sprite/sprite collision
* A sprite/background collision

Because of the VIC chip's ability to generate raster-line interrupts, it is possible for BASIC 7.0 to mix text and graphics (by means of the GRAPHIC command). To program an interrupt, you set the appropriate bits in the IMR register specifying which interrupt source(s) you want. In addition, you must change the interrupt vector to your own interrupt routine so that you can react appropriately to the interrupt.

If the interrupt comes from CIA1, you must branch to the kernal routine. The CIA1 generates interrupts every sixtieth of a second in order to read the keyboard. Otherwise you can branch to you own routine. You can determine if the CIA1 caused the interrupt by reading register 13, ICR (Interrupt Control Register).

If the interrupt came from the VIC chip, bit 7 of the IRR (Interrupt Request Register) is set in addition to the bit of the generator. You need only test for the generator bit if multiple interrupts are enabled on the VIC.

If you use only the raster-line interrupt, you must check bit 7. You can specify which raster line is to cause the interrupt by setting registers 18 and 17 (overflow). When this line is encountered while the screen is being constructed, an interrupt is generated. By the time the routine reacts, the beam creating the picture is already a few lines farther down. You must be sure to take this time delay into consideration.

The possibilities which interrupt programming offers, as well as the flood of programming tricks to be mentioned and explained would go far beyond the scope of this book.

2.3.10.1 More than 8 sprites on the screen

We will use the following program as a small example of what can be done with the raster-line interrupt. The raster-line interrupt makes it possible to display more than the usual 8 sprites on the screen at one time. The control program need only exchange the data for the sprites with an area reserved for this purpose or redefine the pointers at a specific raster-line.

If you display more than 8 sprites using the raster-line interrupt, the freedom of movement in the vertical direction is somewhat limited. If you use 16 sprites, for example, the first eight sprites must move above the middle line (0--99) while the second set of eight must be satisfied with the lower half (100-199). The sprites can move freely in the horizontal direction. For many games the vertical restriction is not a problem so you can make extensive use of the raster-line interrupt.

Our example program displays 16 sprites in various colors and moves them across the screen. Eight sprites are to be displayed in the upper half of the screen. If the video controller has displayed the upper half, we generate an interrupt. In the interrupt routine we set the parameters for the sprites which are to be displayed in the lower half of the screen. At the same time, we must prepare the next raster interrupt for the end of the screen so that we can again switch back to the upper 8 sprites.

```
1    REM 16 SPRITES
5    PRINT CHR$(147)
100  FOR I = 0 TO 7: POKE 2040+I, 15: NEXT
110  V = 53248
120  POKE V+21, 255 : POKE V+ 33, 0
130  FOR I = 0 TO 7: POKE V+2*I,(I+1)*30:
     POKE V+2*I+1,70;NEXT
140  FOR I = 0 TO 7: POKE V+39+I,I+1: NEXT
200  FOR I = 828 TO 907: READ X: POKE I, X : NEXT
300  FOR I = 960 TO 960 + 62 :READ X:POKE I, X: NEXT
350 SYS 828
430  D = D + 1; FOR I = 0 TO 7: POKE V+2*I,(I+1)* D:
     POKE V+2*I=1,I*5+60: NEXT
440  IF D> 28 THEN D=1
450  GOTO 430
900 DATA 120, 169,100,141,18,208,173,17
910  DATA 208,41,127,141,17,208,169,129
```

```
920 DATA 141, 26,208,169,91,160,3,141
930  DATA 20,3,140,21,3,88,96,173
940  DATA 25,208,141,25,208,41,1,208
950  DATA 3,76,49,234,173,18,208,201
960  DATA 200,176,22,160,200,169,170,140
970  DATA 18,208,162,14,157,1,208,202
980  DATA 202,16,249,104,168,104,170,104
990  DATA 64,160,100,169,90,76,115,3
1000 DATA 255,255,255,182,210,73,164,155
1001 DATA 109,255,255,255,164,155,109,182
1002 DATA 211,109,182,218,109,182,219,77
1003 DATA 182,219,105,182,219,109,255,255
1004 DATA 255,0,0,0,0,0,0,0
1005 DATA 0,0,0,0,0,0,0,0
1006 DATA 0,0,0,0,0,0,0,0
1007 DATA 0,0,0,0,0,0,0,0
```

Examine line 430 closely. In addition to the sprite coordinates, you can change all of the other sprite parameters as well, such as the color or size. You can also change the sprite pointers so that other sprite patterns can be displayed, even multicolor.

You can do more than display 16 sprites. If you change the display mode in the raster interrupt routine, you can display a split screen--The top half could display hi-res graphics while the lower half displays text. Superimposed effects can also be achieved in this manner.

Now that we have described the programming and use of sprites in detail, we want to look at the other operating modes of the VIC chip.

2.4 Normal Character Display

This mode is the most "normal" of all the display modes of the VIC: the text mode. It is automatically enabled when the machine is turned on. One thousand characters from the video RAM are displayed as a page of text on the screen. Each character has a code which is used as a pointer to the character generator. This pointer is used to display the bit pattern stored in the character generator at the current screen position. In this manner the computer can display 256 different characters on the screen. Two different characters sets are stored in the Commodore 128. You can select between upper/lower case and upper/graphics mode with SHIFT/Commodore. These are two of the character sets. You can also select between the 40 column and 80 column screens, giving another character set which is a combination of the upper/lower case and upper/graphics case sets.

There is a separate location in the color RAM for each character on the screen. This location determines the color of the character. When the character is displayed, the color of each set bit is fetched from the lower nibble of the color RAM. 16 colors can be defined here. If a bit is not set, the color is fetched from the background color register 0; the point is therefore transparent.

2.4.1 Moving the video RAM

A useful feature of the VIC chip is the ability to move the location of the video RAM and/or the character generator. In this manner you can have two or more text screens. For example, while you display one screen, you can build another behind the scenes. The same applies to the graphic mode. Color RAM cannot be moved, however.

As already mentioned, the VIC chip can address only 16K. Normally the first 16K of bank 0 is addressed--the video RAM is found at address $0400-$07FF. Register 24 of the VIC chip supplies the address of the video RAM in 1K increments. Bits 4-7 of this register represent the address bits 10-13 of the video RAM. The address $0400 looks like this in binary:

0000 1000 0000 0000 = $0400

The left-most bit is address bit 15, the right-most is address bit 0. Address bits 10-13 read: 0010. This bit combination is also found in register 24, bits 4-7. To move the video RAM by 1K, the new address would be $0800.

0001 0000 0000 0000 = $0800

Address bits 10-13 now read 0100. To write this address to register 24, you must first mask out (=erase) bits 4-7 and then the bit combination can be defined with a logical OR operation.

```
P=PEEK(53248+24) : REM OLD CONTENTS
POKE 53248+24,(P AND 240) OR 64
```

This OR operation is necessary to make sure you do not disturb the other bits in the register because they define the address of the character generator.

The limit of movement is reached when you try to move the video RAM by more than 16K. Registers 24 has bits 10-13 of the address available, enough for movements within a 16K range. Since address bits 14 and 15 cannot be defined in the VIC chip, these bits must be stored outside it. These two bits are found in register 0 of CIA2 (address $DD00), bits 0 and 1. Note that these two bits are active low, meaning that their values are inverted. In order to address the lowest 16K (address bits 14 and 15 are 0), bits 0 and 1 of register 0 in CIA2 must be set.

IMPORTANT!
If you change bits 0 and 1 of CIA2, not only does the video RAM move by 16K, the base of the character generator moves too. Remember this when doing graphics programing.

The following values stand for given memory ranges:

X	Bits	Range
0	00	$C000-$FFFF
1	01	$8000-$BFFF
2	10	$4000-$7FFF
3	11	$0000-$3FFF (power-up condition)

```
POKE 56576, A: REM SELECT THE 16K PAGE
```

2.4.2 Moving the character generator

The CIA2 bits define the 16K page for both the video RAM and the character generator. The character generator can also be moved, but in 2K increments instead of 1K increments. Bits 1-3 of register 24 in the VIC represent address bits 11-13 of the character generator.

Normally this pointer points to the character ROM, which is responsible for the appearance of the characters on the screen. In the graphics mode, the character generator must be moved, however, in order to define the base of the graphic page (the video RAM becomes the color RAM). The character ROM is found physically outside the readable range of the VIC chip, because the address $D000 is not addressable when a lower page is selected. This character ROM has a special status thanks to the address manager, however: If the relative addresses $1000-$1FFF or $9000-$9FFF are addressed, the character ROM is automatically accessed ($D000-$DFFF). If you disturb this by programing in the graphics mode, for example, you must use either page 1 or 3 or move the area for the character generator.

If, for example, you want to program and use a couple of self-defined characters, first copy the original character set out of the character ROM into RAM. Then you can redefine individual characters or completely redefine the entire set. You need only tell the VIC where it can find the new character set.

2.4.3 The color RAM

The color RAM is probably the only thing which you cannot redefine on the VIC. This is not a hindrance for it is important to always know where the color RAM will be. The color RAM serves as the color palette for the text display; the VIC gets the color for each character from this RAM. When you work in the hi-res mode, the color RAM is unused. You can use this RAM for other purposes. In the multi-color mode, the color RAM comes back into play--it yields color values for the entire screen area.

The color RAM begins at address $D800 and ends at address $D800+999.

2.5 Programming Color and Graphics

We will clarify the theory behind video programming by using examples.

Whenever you have the opportunity to define a color, whether it be in the color RAM for a character on your text screen or the color for a sprite, the following codes apply to the given colors:

Key	Color	Number
Ctrl-1	Black	0
Ctrl-2	White	1
Ctrl-3	Red	2
Ctrl-4	Cyan	3
Ctrl-5	Purple	4
Ctrl-6	Green	5
Ctrl-7	Blue	6
Ctrl-8	Yellow	7
C=-1	Orange	8
C=-2	Brown	9
C=-3	Light red	10
C=-4	Grey 1	11
C=-5	Grey 2	12
C=-6	Light green	13
C=-7	Light blue	14
C=-8	Grey 3	15

For example, to make the border and background black, the following instructions are necessary:

```
POKE 53280,0
POKE 53281,0
```

To fill the screen (which is now black) with white A's we must fill the video-RAM, at address $0400 to address $0400+999, with the color code 1. In addition, we must put 1 (for white) in all locations of the color RAM at address $D800 to $D800+999 :

```
10 PRINT CHR$(147); : REM CLEAR THE SCREEN
20 FOR I=0 TO 999   : REM 1000 CHARACTERS
30 POKE 55296+I,1   : REM WHITE
```

```
40 POKE I+1024,1     : REM AN A
50 NEXT I
60 GET A$: IF A$="" THEN 60
```

Line 60 prevents the screen from being scrolled. The program is stopped when a key is pressed. If this is too boring for you, try the following:

```
30 POKE 55296+I,RND(0)*16 : REM COLOR
40 POKE 1024+I,RND(0)*255 : REM CHARACTER
```

You should try it out to see what happens. But since programming the text screen is as simple as it is boring, we will now turn to graphics programing:

2.5.1 The hi-res mode

Since we wish to program at the lowest programming level, machine language, we don't have commands for drawing lines or circles--not even a command to set a point. Those who want to program in the 64 mode should get rid of the idea of using BASIC 7.0 commands. If you program in machine language, you can naturally access the routines stored in the ROM. But it usually better if you you write such routines yourself, since you can adapt these routines to meet your individual needs. In addition, the operating system routines make time-consuming checks that we can dispense with entirely in machine language.

Here is a program which plots a sine curve on the screen in the hi-res mode, without using a single command from BASIC 7.0; everything is done "by hand". This program can also be translated directly into machine language, in which only the sine calculation will present a problem.

```
5 REM 128 MODE ONLY: GRAPHIC 1,1
10 REM SINE-PLOT-PROGRAM FOR C-64 MODE AND 128 MODE
20 V=53248:       REM START ADDRESS OF VIC
30 AD=8192:       REM START ADDRESS OF HI-RES BIT
MAP
32 REM 128 MODE ONLY:  GOTO 120
40 POKE V+17,59: REM TURN ON GRAPHICS
50 POKE V+24,24: REM DEFINITION OF CHAR-GENERATORS
60 FOR I=1024 TO 2023: REM SET THE HIRES COLOR RAM
```

```
70 POKE I,16:     REM COLOR
80 NEXT I
90 FOR I=8192 TO 16383: REM CLEAR THE HIRES BIT MAT
100 : POKE I,0
110 NEXT I
120 Y=100:        REM POSITION X AXIS
130 FOR X=0 TO 319: REM MARK THE X AXIS
140 : GOSUB 1000:REM POINT SET
150 NEXT X
160 X=160:         REM POSITION Y AXIS
170 FOR Y=0 TO 199: REM MARK Y AXIS
180 : GOSUB 1000
190 NEXT Y
200 X=0
210 FOR I=-3.141592654 TO 3.141592654
    STEP 0.0196349541
220 : Y= 100+99*SIN(I): REM FUNCTION
230 : GOSUB 1000
240 : X=X+1:      REM NEXT FUNCTION
250 NEXT I
260 GET A$:IF A$="" THEN 260
265 REM C-128 MODE ONLY : GRAPHIC 0
1000 OY= 320* INT(Y/8) + (Y AND 7): REM Y-OFFSET
1010 OX= 8* INT(X/8)               : REM X-OFFSET
1020 MA = 2^(7-(X AND 7))
1020 AV = AD + OX + OY
1040 POKE AV, PEEK(AV) OR MA: REM SET POINT ON OR
1050 RETURN
```

When you start the program, you will not be very impressed by the execution speed. This is because of the time-consuming calculations and the REM commands. A very time-intensive calculation is the (2^a) calculation which can be replaced by a table in both BASIC and machine language. Naturally this all can be done in BASIC 7.0 more effectively, but you would never know a point is set **internally**. The program contains the BASIC 7.0 commands in REM statements so you can see the differences.

We'll take a closer look at the program to find out how we produced the graphics on the screen.

In order to make the calculations in the program reference the VIC chip, we have first defined the starting address of the chip. This also makes it easier to see which register is being accessed. First we change register 17

by writing the value 59 into it. Bit 5 is set to tell the VIC that we are in the graphics mode. The start addresses of the video RAM and character generator are placed in register 24. We write a 24 in this register.

24 = $18 = %0001 1000

Bits 4-7 of the register determine the address bits 10-13 of the video RAM--we get the start address $0400, the normal value of the screen. Furthermore, bits 1-3 determine address bits 11-13 of the character base:

%0010 0000 0000 0000 = $2000 = 8192

We have defined the address of the video RAM as well as the address of the bit map with one POKE command. Based on our own experience, most of the errors occur in the conversion of these two addresses. For this reason you should do everything in detail, as in our example, by writing the two addresses down and then putting together the bits that are required.

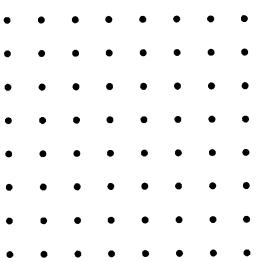
When you start the program, you return to BASIC again by pressing a key. But you can see that the graphics mode is not turned off, and you can see that the text is quite colorful. This is because the video RAM is filled with the values that refer to these colors. You should save the contents of registers 17 and 24 before you overwrite them so that you can reconstruct them later. Insert the following lines to return to the text mode when you press a key:

```
35 A1=PEEK(V+17): A2=PEEK(V+24)
270 POKE V+17, A1: POKE V+24, A2: END
```

This program makes use of the hi-res mode in which we have a resolution of 320x200 points. This gives exactly 64,000 points available to us. Since 8 points=8 bits that can be combined into one byte, we need a memory area of exactly 8000 bytes in order to display the graphics. Three hundred and twenty (320) points can be displayed in one line, or 40 bytes (320/8); we recognize this from the text mode. Further, we have 25 lines of 8 points. Notice the parallel to the text mode.

One character in the text mode consists of 8x8=64 points which can be independently set or cleared. The color for the set points comes from the color RAM while the color for the unset points is taken from the background color register 0. The graphic mode is similar. Here too 8x8 points are taken together as a unit. Two colors can be displayed in this little box of 64 points. If a memory location were provided for the color of each

point, we would need 64K of color memory! By combining the points into 8x8 groups, we only need 1000 bytes for the color definition. We will take a closer look at such an 8x8 unit.



Such a unit is also called a character matrix. All of our letters and special characters that we can see on the screen in the text mode are defined in this matrix. In the hi-res mode we can define all of the matrices ourselves and no longer have just a "pointer table" to pre-defined matrices (character generator). This may sound complicated, but it really isn't.

You see that it must be possible to mix text and graphics or to "draw" text in the graphic area without too much programming effort. Writing directly to the graphic storage naturally doesn't work. But exactly how is the graphic brought to the screen? What memory location in our graphic storage defines which 8 points in our graphic? The following figure should clarify these questions:

```
8192: . . . . . .      8200: . . . . . .
8193: . . . . . .      8201: . . . . . .
8194: . . . . . .      8202: . . . . . .
8195: . . . . . .      8203: . . . . . .
8196: . . . . . .      8204: . . . . . .
8197: . . . . . .      8205: . . . . . .
8198: . . . . . .      8206: . . . . . .
8199: . . . . . .      8207: . . . . . .

8124: . . . . . .      8257: . . . . . .
etc.
```

This figure shows the shift between columns and lines as far as the addressing goes. Our graphic storage starts at address 8192 and defines the first 8 points of our graphic with the first byte. If we want to address the ninth point in our first line, we must use the address 8200 which is where this point resides. The scheme of representation is similar to the text mode;

it is displayed character by character and line by line. But how do we address a given point? We must first calculate the address in which it is located. To establish such an algorithm we first simplify the conditions. First we will just try addressing a point in the first line:

```
AD = 8192 + INT(X/8)*8
```

For the sake of simplicity, we will call the term INT(X/8)*8, OX (or offset of the X-position). This is all we need to do for the X-coordinate. We now have the address of the point, but we don't know what bit to access. We don't want to disturb any of the others:

```
BIT = X - INT(X/8)*8
```

We need to find the remainder of X/8. This is done by masking out the lowest three bits with a logical AND operation.

```
BIT = X AND 7
```

Try it once; it works and is much faster than the division, especially in machine language. Now, however, we must consider that the left-most bit is not labeled 0, but 7. We must reverse this relationship:

```
BIT = 7 - (X AND 7)
```

Now the formula is correct. To set such a point in assembly language or BASIC we have to set the appropriate memory location with a logical OR operation. To do this, we have to calculate the power of two:

```
2^(7-(X AND 7))
```

Now we can set any point in the first line:

```
POKE 8192+OX,PEEK(8192+OX) OR 2^(7-(X AND 7))
```

To address the first eight lines, we need only add the Y-coordinate. If we want to access the ninth line, we have to skip 320 bytes. The following addition takes the Y-position into account:

```
OY = INT(Y/8)*320 + (Y AND 7)
```

In order to address a point, add the offset of the X and Y positions to the base address of the graphics memory. The following formula results for the address calculation:

$$AD = OX + OY + 8192$$

Our terms for calculating the X and Y offsets are integrated into the formula. We have now derived all of the calculations necessary to set a point. The following sequence of commands in BASIC give us the correct results:

```
OY=320*INT(Y/8) + (Y AND 7)
OX=8*INT(X/8)
BI=2^(X AND 7)
AD=8192 + OX + OY
POKE AV, PEEK(AV) OR BI
```

If we want to erase a point, the address calculation does not change, but we must modify the POKE command. We must also mask out the calculated bit:

```
POKE AV, PEEK(AV) AND NOT BI
```

Now we know how to set and clear points. But we still don't know how the colors to be displayed for set and cleared bits can be set. In our example the bit map is found at addresses 8192-16192. You recall than we have moved the normal RAM to color RAM. This means that the information to determine the color of the points on the hi-res screen will come from this memory, memory which otherwise contains the contents of the screen. This memory area is located at address 1024 thru 2023.

Since we can define two colors with one bit, we must also place these two colors in video RAM. Recall the construction of the graphic screen. We always had "matrices" of 8 bytes--eight sequential bytes in our bit map. Such a matrix has the same size as a character on the screen. The colors for our first matrix, at address 8129-8199, is defined in the first byte of the video RAM--address 1024. These two colors apply to all 64 points in this matrix. Correspondingly, the colors for the second matrix, from address 8200 to 8207, are stored in address 1025. The question remains, how are these colors defined?

Let's take another look at our example program that filled the range from 1024 to 2023 with the value 16. What does 16 look like in binary?

16 = $10 = %00010000

If we separate the upper and lower nibbles (unit of four bits) from each other, we get two values between 0 and 15--sufficient to define the available colors. In this example we get the values 1 and 0. If we look at the color table, we see that we have defined the colors white and black. In the hi-res mode you must define the colors so that sufficient contrast is retained. Often two adjacent points must be set in order to be able to see the color at all. This varies from monitor to monitor, however. The contrast between white and black is the best possible (perhaps black on white would be even better), while red and blue result in utter chaos. The color defined in the upper nibble of the color RAM is displayed for a set bit. In our example this means that the background is black (0) and the graphic is shown in white (1). The following rule applies for setting the color RAM:

POKE <color RAM>,<foreground>*16 + <background>

Naturally, you can define more than two colors across the entire screen: there are 256 possible combinations within a matrix and black and white is only one of them. Programming in hi-res mode is best learned by trial and error.

2.5.2 The multi-color mode

In addition to the hi-res mode, there is another option for displaying graphics on the screen: the multi-color mode. We are familiar with the term multi-color from sprites. In multi-color we have four colors per matrix, though as with sprites, the resolution suffers. In multi-color mode it is "only" 160x200--exactly half. A byte now defines four points instead of eight. To turn on the multi-color mode we must set bit 5 of register 17 (just as for the hi-res mode). In addition, the fourth bit in register 22 must be set. This is done by the instruction:

POKE 53248+22,PEEK(53248+22) OR 16

The addresses for the bit map and color RAM are programmed in the same manner as for the hi-res mode. The following contents should be found in address 8192 (the first byte of the bit map):

PEEK(8192)= %00011011 = $1B = 27

This byte defines the first four points of the first line. Since two bits are taken together, we get the bit pairs 00, 01, 10, and 11--all four combinations are possible.

Bits	Color information comes from
00	Background color register 0
01	Upper four bits of the video RAM
10	Lower four bits of video RAM
11	Color RAM

Here only the bit combination 00 is the same for the entire screen. Bit combinations 01 and 10 work the same way as described for the hi-res mode. The color RAM begins at address $D800 and makes one color available. Programming in multi-color mode is very attractive since it offers a wider selection of colors. Naturally our address calculation must change since only four points are defined by each byte. The formula for the X offset changes:

```
OX=8*INT(X/4)
MA=2^(6-2*(X AND 3))
POKE AV, PEEK(AV) OR MA*<bit pattern>
```

You can see that the formula for the bit determination has also changed. You must remember that a bit pair must be logically ORed with the existing contents and the power of two may only go in steps of two. The <bit pattern> is shifted left by the multiplication. Since the multi-color mode is most often used in games, you should be familiar with the programming tricks used in this mode.

2.5.3 The multi-color mode (text)

(register 22 bit 4=1)

Another relatively unused multi-color mode is the multi-color text mode. In this mode characters on the screen can have more than one color. For example, you can define a zero made up of a white circle with a blue slash through it. If the multi-color mode is enabled, the VIC checks to see if bit 3 of the color register is set. This means that the color of the character is greater than 7 (8-15). If this is the case, the character is displayed in multi-color mode. The character no longer has an 8x8 matrix, but just a 4x8 matrix with the following bit combinations:

Bits	Color register	Defined at address
00	Background register 0	$D021 (53281)
01	Background register 1	$D022 (53282)
10	Background register 2	$D023 (53283)
11	Color register	Color RAM $D800-$D800+1000

If the bit combination is 11, the color is taken from the lower three bits of the color register. If bit three is not set in the color register (color 0-7), a normal single-color 8x8 matrix is displayed. This mode is only useful if you define your own character set. This mode is used in some games because it is easier to program than the hi-res mode. Switch to this mode once: Since these characters are not intended for multi-color mode, you get a colored spectacle:

```
POKE 53248+22,PEEK(53248+22) OR 16
```

The following command is used to turn this mode off again:

```
POKE 53248+22,PEEK(53248+22) AND 239
```

2.5.4 Extended-color mode

(register 17 bit 6=1)

Even all this wasn't enough for the designers of the VIC. They created yet another mode: the extended-color mode. This mode is very similar to the normal text mode. A character can consist of only two colors, but the background color is not necessarily the same. One can choose between three background colors (for the 0-bits), while the 1-bits get their color from the color register. The background color is determined by the two most-significant bits in the video RAM:

Bits	Background color register #
00	0
01	1
10	2
11	3

Since two bits have been taken away from the video RAM, only six bits remain to define the character to be displayed. This has the result that only 64 characters can be represented--these are the lowest 64 characters. There are two sides to everything...

2.6 Smooth Scrolling

You may have seen this word in some computer literature and wondered what it means.

Smooth scrolling is beautiful as it sounds: by means of this capability you can move the screen horizontal or vertically by one pixel. Scrolling is the shifting of the screen. This can be used in games to create moving backgrounds so that one gets smooth scrolling. This movement can take place in any one of four directions (up, down, left, or right). Moving in one direction causes one row of pixels to be covered up while a new row appears at the other end. The screen can be placed in eight different positions with this scrolling, sufficient to allow a character to appear on the screen slowly. To make use of smooth scrolling, the screen must be made smaller. The VIC has two bits available to do this, in which one can select the display mode of 38/40 characters per line and 24/25 lines. The border then increases correspondingly.

If we want to move the screen vertically, we must give up a line, while if we want to move it horizontally, we lose two characters per line. To switch to the 38-column mode, bit 3 of register 22 must be cleared:

```
POKE 53248+22,PEEK(53248+22) AND 247
```

After you have entered this line, the screen shrinks in size. To switch back to the "normal" mode, we must set bit 3 again:

```
POKE 53248+22,PEEK(53248+22) OR 8
```

The same thing applies to the 24-line mode. Here bit 3 of register 17 must be cleared if we want 24 lines:

```
POKE 53248+17,PEEK(53248+17) AND 247
POKE 53248+17,PEEK(53248+17) OR 8
```

In register 22, bits 0-2 indicate what offset the left edge of the screen has. By varying these three bits one can achieve soft scrolling in the horizontal direction. If you want to scroll vertically, the offset in register 17 must be changed accordingly.

But we don't want to keep you in suspense any longer. Here is a demo program to clarify what effects can be achieved with smooth scrolling:

```
10 PRINT CHR$(147) : REM CLR SCREEN
20 POKE 52348+17,PEEK(53248+17) AND 247
30 FOR I=1 TO 24
40 : PRINT "            HELLO !!": REM 12 SPACES
50 NEXT I: PRINT "            HELLO !!";
   : REM NO SCROLLING AND 12 SPACES
60 POKE 53248+17,PEEK(53248+17) AND 248 OR 7
   : REM SET FIRST POSITION
70 FOR I=6 TO 0 STEP-1
80 POKE 53248+17,PEEK(53248+17) AND 248 OR I
90 FOR I1=1 TO 60: NEXT I1: REM DELAY LOOP
100 NEXT:                   REM END OF LOOP
110 GOTO 60:                REM AGAIN
```

Naturally this smooth scrolling works in the graphic mode too. It is in the graphic mode that the most refined effects can be created. For example, you can have a space ship moving soundlessly through a never-ending universe. After all eight rows of points have been scrolled, you must fill a graphic column or row with new values.

You can see that the VIC-II chip offers a great deal. Not everything is covered by the BASIC 7.0 commands. This chapter covers all of the features of the VIC-II so that you won't miss out on anything.

CHAPTER 3

Chapter 3: Input and Output Control

3.1 General Information about the CIA 6526

CIA stands for Central Intelligence Agency, though that really doesn't concern us here. For us, CIA stands for Complex Interface Adapter, and that should be more interesting. The Commodore 128 uses the CIA 6526. A brief run-down of its main features:

* 16 individually programmable input/output lines
* 8 or 16-bit handshake for input and output
* 2 independent, cascadable 16-bit interval timers
* 24-hour (AM/PM) clock with programmable alarm time
* 8-bit shift register for the serial I/O

3.1.1 Pin Configuration

1	GND
2-9	I/O PA (port A); 8-bit directional
10-17	I/O PB (port B); 8-bit directional Bits 6&7 can be programmed to signal the time-out of both timers
18	-PC (port control); output only; signals the availability of data on port B or both ports
19	TOD (Time Of Day); input only, 50/60 Hz; triggers the real-time clock
20	+5V; operating voltage
21	-IRQ (interrupt request); output only; 0 if a set bit in the ICR matches the occurrence of the given event
22	R/W (read/write); input only; 0=input from data bus 1=output to data bus
23	-CS (chip select); input only; 0=data bus valid, 1=data bus high-impedance (tri-state)
24	-FLAG; input only; meaning same as -PC
25	02 (system clock 2); input only all data bus actions occur only on 02=1

26-33	DB7-DB0 (data bus); bidirectional; interface to processor
34	-RES (reset); input only; 0=reset CIA
35-38	RS3-RS0 (register select); input only; serves to select a 16-bit register; valid only if -CS=0
39	SP (serial port); bidirectional; input/output of the shift register
40	CNT (count); bidirectional; input/output of the shift register clock or trigger input for the interval counter.

3.2 Register Description of the CIA

REG 0 PRA (port register A)
Access: read/write
Bits 0-7: This register corresponds to the condition of pins PA0-PA7.

REG 1 PRB (port register B)
Access: read/write
Bits 0-7: This register corresponds to the condition of PB0-PB7.

REG 2 DDRA (data direction register A)
Access: read/write
Bits 0-7: These bits determine the direction of data on the corresponding data bits of port A.
0=input, 1=output

REG 3 DDRB (data direction register B)
Access: read/write
Bits 0-7: These bits determine the direction of data on the corresponding data bits of port B.
0=input, 1=output

REG 4 TA LO (Timer A, low byte)
Access: read
Bits 0-7: This register returns the current condition of the low-order byte of time A.
Access: write
Bits 0-7: This register is loaded with the low-order byte of the value from which timer is supposed to count down to zero.

REG 5 TA HI (Timer A, high byte)
Access: Read
Bits 0-7: This register returns the current condition of the high-order byte of time A.
Access: Write
Bits 0-7: This register is loaded with the high-order byte of the value which timer is supposed to count down to zero.

REG 6 TB LO (Timer B, low byte)
Same as register 4.

REG 7 TB HI (Timer B, high byte)
Same as register 5.

REG 8 TOD 10ths (Clock tenths of a second)
Access: Read
Bits 0-3: Tenths of a second in BCD format
Bits 4-7: Always 0
Access: Write and CRB bit 7=0
Bits 0-3: Tenths of a second in BCD format
Bits 4-7: Must be 0!

REG 9 TOD SEC (Clock seconds)
Access: Read
Bits 0-3: Seconds (one's digit) in BCD format
Bits 4-6: Tens of seconds in BCD
Bit 7: always zero

REG 10 TOD MIN (Clock minutes)
Access: Read
Bits 0-3: Minutes (one's digit) in BCD format
Bits 4-6: Tens of minutes in BCD
Bit 7: always zero
Write access as per REG 8.

REG 11 TOD HR (Clock hours)
Access: Read
Bits 0-3: Hours (one's digit) in BCD format
Bits 4: Tens of hours
Bits 5-6: Always zero
Bit 7: 0=AM, 1=PM
Write access as per REG 8

REG 12 SDR (Serial data register)
Access: Read/write
Bits 0-7: The data are shifted out to or shifted in from pin SP from/to this register.

REG 13 ICR (Interrupt control register)
Access: Read (INT DATA)
Bit 0: 1=Timer A timeout
Bit 1: 1=Timer B timeout
Bit 2: 1=Alarm time equals clock time
Bit 3: 1=SDR full/empty (depending on operating mode)
Bit 4: 1=Signal on FLAG pin
Bits 5-6: Always zero
Bit 7: At least one bit in INT MASK matches a bit in INT DATA
Note: Reading this register erases all of the bits!
Access: Write (INT MASK)
Meaning of bits as above, except bit7:
Bit 7: 1=Every 1-bit sets the corresponding mask bit. The other remain unchanged.
0=Every 1-bit clears the corresponding mask bit. The other remain unchanged.

REG 14 CRA (Control Register A)
Access: Read/write
Bit 0: 1=Timer A start, 0=stop
Bit 1: 1=Signal timer A timeout on pin B6
Bit 2: 1=Every timeout on timer A inverts PB6
0=Every timeout on PB6 creates a high signal on PB6 for the length of the system clock
Bit 3: 1=Timer A counts down to zero and stops
0=Timer A counts down to zero and repeats continuously

Bit 4: 1=Absolute loading of start value in timer A. This bit functions as a strobe. It must be set for each absolute load.
Bit 5: This bit determines the source of the timer trigger. 1=timer counts rising CNT edges, 0=timer counts system clock pulses.
Bit 6: 1=SP is output, 0=SP is input
Bit 7: 1=Real-time clock trigger is 50Hz
0=Real-time clock trigger is 60Hz

REG 15 CRB (Control register B)
Access: Read/write
Bits 0-4: These bits have the same meaning as in REG 14, except they apply to timer B and PB7.
Bits 5-6: These determine the source of the trigger for timer B. 00=timer counts system clocks, 01=timer counts rising CNT edges, 10=timer counts timeouts of timer A, 11=timer counts timeouts of timer A when CNT=1.

3.3 I/O Ports

Ports A and B each consist of an 8-bit data register (PRA or PRB) and an 8-bit data direction register (DDRA or DDRB). When a bit is set in the DDR, the corresponding bit in the PR functions as an output. If a bit in the DDR=0, the corresponding bit in the PR is defined as an input.

During a read access, the PR returns the current condition of the corresponding pins (PA0-7, PB0-7); it does this for both input and output pins. PB6 and PB7 can assume output functions for the two timers.

The data transfer between the CIA and the "outside" world connected to PA/PB can be accomplished with handshaking. PC and FLAG are used for this. PC goes low for one clock period when a read or write access occurs on PRB. This signal can indicate the availability of data on PRB or indicate receipt of data by PRB. FLAG is a trailing-edge triggered input which can be connected to the PC of another CIA, for example. A trailing edge on FLAG sets the FLAG interrupt bit.

The serial data port SDR is a synchronous 8-bit shift register. CRA bit 6 determines the input or output mode. In the input mode the data are accepted into the shift register on a rising edge on CNT. After 8 CNT pulses

the contents of the shift register are placed in SDR and the SP bit in ICR is set. In the output mode timer A functions as a baud rate generator. The data are shifted out of SDR to SP at half the timeout frequency of timer A. The theoretical limit to the baud rate is 1/4 of the system clock.

The transfer begins after data are written to the SDR, assuming timer A is running and is in the continuous mode (CRA bit 0=1 and bit 3=0). The clock derived from timer A appears on CNT. The data from SDR are loaded into the shift register and are shifted out on every trailing edge on CNT. After 8 CNT pulses, the SP signal is created. If the SDR is loaded with new data before this event, these are automatically loaded into the shift register and shifted out. No interrupt is generated in this case.

The data in SDR are shifted out high-order bit first. Data going into the register must following the same format.

3.4 The Timers

Both timers have a 16-bit timer (read-only) and a 16-bit temporary storage (write-only). If a timer is read, its current contents are returned. When writing, the data are first written to the temporary storage.

Both timers can be used independently of each other or in connection. The various operating modes allow long time delays, variable pulse lengths, and pulse chains. By using the CNT input, the timer can measure external pulses or frequencies.

Each timer has a control register (CRA and CRB) assigned to it, which allows the following functions:

Start/Stop (Bit 0)
This bit allows the timer to be started or stopped at any time.

PB ON/OFF (Bit 1)
This bit directs the timeout to PB (PB6 for timer A, PB7 for timer B). This function has precedence over the data direction set in DDRB.

Toggle/Pulse (Bit 2)
This bit determines the method in which the timeout signals will appear on PB. Either the condition of PB is inverted at every timeout, or a positive pulse is created for the duration of the clock.

One-shot/Continuous (Bit 3)

In the one-shot mode the timer counts from the temporary storage value down to zero, sets the IRC bit, reloads the timer with the temporary storage value and stops. In the continuous mode, this procedure does not stop.

Force-load (Bit 4)

This bit allows the timer to be loaded at any time, independent of whether it is running or not.

Input mode (Bit 5 CRA, Bits 5-6 CRB)

These bits select the clock which determines the rate at which the timers will count down. Timer A can be clocked either by the system clock or by a clock supplied on CNT. Timer B can be further clocked by the timeout pulses from timer A, either absolutely or dependent on CNT=1.

3.5 The Real-time Clock

There is a 24-hour real-time clock (TOD) in the CIA with a resolution of 1/10 second. In consists of four registers: Hours, minutes, seconds, and 1/10ths of second. In the hour's register, the highest bit (bit 7) indicates whether it is AM or PM. All registers are given in BCD format so that the clock can be used without a lot of processor effort, even in machine language.

The clock is a 50/60 Hz signal at the pin TOD, which can be programmed in CRA bit 7. In addition, there is an alarm register that can be used to generate an interrupt at any desired time. The alarm register occupies the same address as the TOD register, so the access is controlled by CRB bit 7.

Note that the alarm register is **write only!** Any read access returns the TOD register regardless of the state of CRB bit 7.

In order to be able to properly set and read the alarm time, the following order must be preserved:

If the hours register is written, the clock automatically stops--it starts to run when the tenth of second register is loaded. The starting of the clock can be controlled exactly in this manner.

Since a carry can occur in a register already read when reading the clock, the registers are stored in temporary storage. This temporary storage is freed again when the tenths of a second are read.

3.5.1 Real-time in BASIC

Most of you probably know about the "clock" available from BASIC, TI$ and TI. Unfortunately the long-time accuracy of this clock leaves much to be desired; it is off about 1/2 hour per day.

If you need a more exact time indication, you can use the real-time clock built into the CIA. Thie CIA clock uses the line frequency, which has excellent long-term accuracy.

Here are two BASIC programs, one for setting the clock time, and one for reading it. Since it doesn't make a whole lot of sense to read the tenths, the register is always set to zero.

```
10 C=56328: REM BASE ADDRESS OF THE CLOCK IN CIA1
20 REM C=56584 FOR THE CLOCK IN CIA2
30 POKE C+7,PEEK(C+7) AND 127: REM SET CLOCK TIME
40 POKE C+6,PEEK(C+6) AND 128: REM LINE FREQ=60HZ
50 INPUT "PLEASE ENTER THE TIME IN THE FORMAT
   HHMMSS: ";A$
60 H=VAL(LEFT$(A$,2))
70 M=VAL(MID$(A$,3,2))
80 S=VAL(MID$(A$,5))
90 IF H>23 THEN 40 : REM ERROR
100 IF H>11 THEN H=H+68 : REM SET PM FLAG IF
    NECESSARY
110 POKE C+3,16*INT(H/10)+H-INT(H/10)*10
120 IF M>59 THEN 40 : REM ERROR
130 POKE C+2,16*INT(M/10)+M-INT(M/10)*10
140 IF S>59 THEN 40 : REM ERROR
150 POKE C+1,16*INT(S/59)+S-INT(S/59)*10
160 POKE C,0 : REM TENTHS -- START CLOCK
```

The values are converted to BCD format in lines 110,130, and 150. You can use the following program to read the clock:

```
10 C=56328 : REM BASE ADDRESS OF THE CLOCK IN CIA1
20 PRINT CHR$(147) : REM C=56584 FOR CLOCK IN CIA2
30 H=PEEK(C+3):M=PEEK(C+2):S=PEEK(C+1):T=PEEK(C)
40 FL=1
50 IF H>32 THEN H=H AND 127: FL=0: REM FLAG FOR PM
60 H=INT(H/16)*10+H-INT(H/16)*16:ON FL GOTO 80
70 IF H=12 THEN 90: ELSE H=H+12
80 IF H=12 THEN H=0
90 M=INT(M/16)*10+M-INT(M/16)*16
100 S=INT(S/16)*10+S-INT(S/16)*16
110 T$=MID$(STR$(T),2)
120 H$=RIGHT$("0"+MID$(STR$(H),2),2)
130 M$=RIGHT$("0"+MID$(STR$(M),2),2)
140 S$=RIGHT$("0"+MID$(STR$(S),2),2)
150 PRINT "<Home>";
160 PRINT H$;":";M$;":";S$;":";T$
170 GOTO 30 : REM LOOP
```

If you press the STOP/RESTORE key combination, the clock must be reset because the operating system sets all of the registers back to the starting values. Unfortunately, the bit responsible for the clock (50/60Hz) is also affected by this.

3.6 The CIAs in the Commodore 128

If you want to make use of the CIAs in the Commodore 128, you must remember that the CIAs have predetermined tasks to perform. Its first priority is to handle the interrupts, which the operating system requires for a number of routines. If possible, refrain from changing the ICR register.

CIA 1: Base address $DC00 (56320)

REG 0 (PRA)

Bits 0-7: In normal operation the row selection of the keyboard matrix is found here. Some bits are also connected to controller port 1 on the outside of the computer. This is used to connect joysticks or paddles.

Bits 0-4: Joystick 0, order: up, down, (left right, and fire button).

Bits 6-7: Select paddle set A/B. Only one of the two bits may be 1.

REG 1 (PRB)
Bits 0-7: In normal operation the column selection of the keyboard matrix is found here, if a key was pressed.
Bits 0-4: The same function as REG 0, but for control port 2 (joystick 1).

REG 13 (ICR)
Bit 4: Input data on cassette port.

Timer A and CRA are required for the disk operation, timer B & CRB for the cassette operation.

CIA 2: Base address $DD00 (56576)

REG 0 (PRA)
Bits 0-1: VA 14-15 (highest-order address bits of the video RAM),
Bit 2: TXD (only in connection with an RS-232 cartridge, else free),
Bit 3: ATN (serial bus output)
Bit 4: CLOCK (serial bus output)
Bit 5: DATA (serial bus output)
Bit 6: CLOCK (serial bus input)
Bit 7: DATA (serial bus input)

REG 1 (PRB)
Bits 0-7: User port/RS-232. These bits have following meaning when an RS-232 cartridge is inserted:
Bit 0: RXD (Receive Data)
Bit 1: RTS (Request To Send)
Bit 2: DTR (Data Terminal Ready)
Bit 3: RI (Ring Indicator)
Bit 4: DCD (Data Carrier Detect)
Bit 6: CTS (Clear To Send)
Bit 7: DSR (Data Set Ready)

REG 13 (ICR)
Bit 4: RXD (only for RS-232 operation, else free).

Timer A & CRA are required for the RS-232 baud rate, timer B & CRB for the RS-232 bit checking.

3.7 The Joystick

In addition to the BASIC 7.0 commands for reading the joystick you can use the following BASIC program for interpreting the data:

```
10 J1=56320 : REM JOYSTICK PORT 1
20 J2=56321 : REM JOYSTICK PORT 2
30 J=PEEK(J1) : REM READ FROM PORT
40 IF (J AND 1)=0 THEN PRINT "UP ";
50 IF (J AND 2)=0 THEN PRINT "DOWN ";
60 IF (J AND 4)=0 THEN PRINT "LEFT ";
70 IF (J AND 8)=0 THEN PRINT "RIGHT ";
80 IF (J AND 16)=0 THEN PRINT "FIRE";
90 PRINT: GOTO 30
```

The program reads from joystick port 1; if you want to read from port 2, you need only replace J1with J2 in line 30.

If you want control in two directions at once, such as up and right, this can also be read--in our example both directions are displayed on the screen. This increases the number of directions from 4 to 8.

3.8 The Commodore 128 Serial Bus

Peripheral devices are connected to the computer via the serial bus. These can be such things as a printer or disk drives. You can think of a bus as working like this: Data is transported from the computer over the bus to specific stops (peripheral) and they return via the same path. The serial bus built into the Commodore 64 and 128 is a trimmed-down version of the bus included in the "larger" Commodore computers. The "big" bus has 24 lines while the "smaller" bus has only 6. This reduction may have been made for reasons of cost or space, but this bus has definitely contributed to the success of the Commodore computers (Many even think that it is Commodore's secret recipe).

Here is the pinout of the bus:

1 SRQ; Service request. If a device has completed a task and now needs new data, or has some to send, or requires some kind of action, it can signal the controller by means of this line (like in the hospital where you can ring for a nurse). This initiates an identify cycle (by means of EOI or ATN), in order to determine which device is involved. This function is not used on the Commodore.

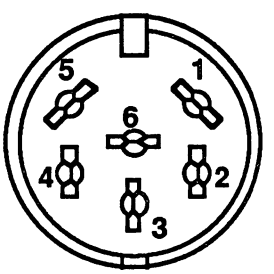
2 GND; ground connection

3 ATN; (In) ATtentioN. Whenever the controller wants to send a command, it activates this line. It must still be determined for which device the command is intended (all of the devices should "listen"). This is done when the device address is transmitted so that the other devices can get off the bus.

4 CLK; (In/Out) CLocK. Since the data travel through the bus bit by bit in serial and not in parallel, the TALKER sends a CLK pulse along with each bit, which indicates the validity of the data line.

5 DATA (In/Out) is the sole data line, over which a data byte is shifted with the lowest-order byte first.

6 RESET; sends a reset to the connected devices.



All of the additional lines found on the larger bus, like EOI, NDAC, etc., are simulated or replaced by the two lines CLK and DATA. The time between the signal jumps of the two lines gives information about the signal.

3.8.1 Fast and slow modes

You may think it a waste to leave one line unused on the already puny bus. But unfortunately, that's the way it is--at least in the "normal" mode.

If there is a "normal" mode, you know there must be some other "abnormal" mode. This is true! As you know, the 1541 can hardly be described as a fast disk drive (quite the opposite). This is because each byte must be picked to pieces and then sent over the bus bit by bit. This deplorable state of affairs must be corrected--what good is a super machine like the Commodore 128 when it has such a handicap? Commodore developed the 1571 disk drive which loads up to eight times (!) faster than the 1541 (you can find out more in the book 1571 Internals by Abacus Software). Other things have been added in the CP/M mode as well. The speed advantage is possible **only** in the 128 mode, not in the 64 mode. The 1541 can be operated as usual in the 128 mode.

You may have already given some thought as to how this speed increase was accomplished; with the help of the unused SRQ signal. In the fast serial mode this line is used as second CLK line, as a fast, bidirectional CLOCK line.

On power-up, the 1571 is always in the slow mode, which is why you can connect it to a C-64. The user can then specify the "fast" mode, which will remain in effect until it is turned off. The existing kernal routines in the C-128 have been changed in order to recognize the fast and slow modes. There is a special flag in the kernal to indicate if the current peripheral device is fast or slow.

In order to declare the 1571 as a fast device, the user must send an HRF signal (Host Request Fast). This is done by sending eight CLOCK pulses over the SRQ line. The 6526 on the control board of the 1571 disk drive recognizes this signal and generates an interrupt. A flag is then set in the drive which indicates the fast mode. If the disk drive is the LISTENER and receives data, it sends a DRF signal (Device Request Fast). By means of this signal the computer recognizes that the disk drive can send and receive data in the fast mode. A 1541 can't send this signal, of course. The fast-mode flag in the computer can be reset by the following occurrences:

UNLISTEN, UNTALK, bus error, and <RUN/STOP><RESTORE>

3.8.2 The device addresses

It's possible to connect a variety of devices to the serial bus, such as two disk drives and a printer. This makes it necessary to be able to distinguish between the different devices so that the data know where they have to "get off the bus." You can imagine a device address as a house number. The values 0-30 are possible as device addresses.

Device address	
0-3	Internal device (keyboard, screen, user port, cassette port)
4-7	Normally CBM printer
8-11	Normally CBM disk drives
12-30	Not used

The device address contains additional information besides the actual device number: the action which is to be performed. The possible actions are the following:

32 The device is addressed as a LISTENER, which means that it is to receive data.
This action is called for by the BASIC command PRINT# or DSAVE, for instance.

64 The device is supposed to be the TALKER; it is supposed to send data.
This is used, for example, by the BASIC commands INPUT# or DLOAD.

48 The operating mode LISTEN is ended (UNLISTEN). The lower half-byte (device) is always 15.

80 The operating mode TALK is ended (UNTALK). The lower half-byte is always 15.

For example, if you want to address a printer with the device address 4 for printing, the whole device address is 32+4=36 ($24).

3.8.3 The secondary address

The secondary address does not select a device on the serial bus--it is used to select a mode in the device addressed. For example, a specific printing mode can be selected on most printers by specifying a secondary address. On the CBM printers, secondary address 0 selects the upper/graphics mode, secondary address 7 selects the upper/lowercase mode. With a disk drive one can choose a data channel with the secondary address.

The secondary address is also composed of the actual secondary address and the connection in which the secondary address occurs.

96	PRINT, INPUT, or GET
224	CLOSE
240	OPEN

This next table will also prove useful. It shows the bit patterns for the individual device and secondary addresses.

Command	Abbreviation	Binary value
Host Request Fast	HRF	%1111 1111
Device Request Fast	DRF	%0000 0000
Talk address	(TA)	%010x xxxx
Listen address	(LA)	%001x xxxx
UNTALK	(UNTLK)	%0101 1111
UNLISTEN	(UNLSN)	%0011 1111
SA OPEN	(SA(O))	%1111 yyyy
SA CLOSE	(SA(C))	%1110 yyyy
SA normal	(SA)	%011z zzzz

The normal secondary address (zzzz) may have a value between 0 and 31. The channel address (yyyy) may have a value between 0 and 15. As an example, the secondary addresses and their meaning for the 1541 disk drives:

00 - PRG type (read data channel)
01 - PRG type (write data channel)
02-14 - Channels for all file types
15 - Command channel

3.8.4 The system variable ST

When peripheral devices are connected, errors can naturally occur. The system variable ST gives information about whether the last action on the serial bus was successful or not. If it was not successful, the error can be analyzed by means of the error code passed in the status variable ST. ST can have the following values:

1 Can occur after OPEN or PRINT. After transmission of a byte, no acknowledgement was received via NDAC within 64milliseconds (ms), and it will probably not come.

2 Can occur during INPUT or GET. If a device is addresses as a TALKER and does not send a byte within 64ms, ST contains this value.

64 The data byte last transmitted was sent in connection with an EOI (End Of Information), which means the end of the file (EOF) for the disk drive.

-128 An addressing attempt produced no reaction on the drive. In this case a BASIC program will display the error message DEVICE NOT PRESENT; in machine language you can react in whatever manner is appropriate.

A combination of these values can also occur. Here it is advisable not to read the absolute value in a BASIC program, but just the appropriate bit:

```
1000 IF (ST AND 64) THEN PRINT "<EOF>"
```

To read the status word ST in machine language, it is necessary to get it from the zero page. Fortunately, it is at the same address in both the 64 and 128 modes: $90 (144 decimal). Reading the value in machine language would look like this:

```
LDA $90  ;Get status variable
AND #$40 ;bit 6 set?
BNE EOF  ;EOF reached
   .
   .
```

CHAPTER 4

Chapter 4: The Sound Chip SID

4.1 The Sound Controller

4.1.1 General information about the SID

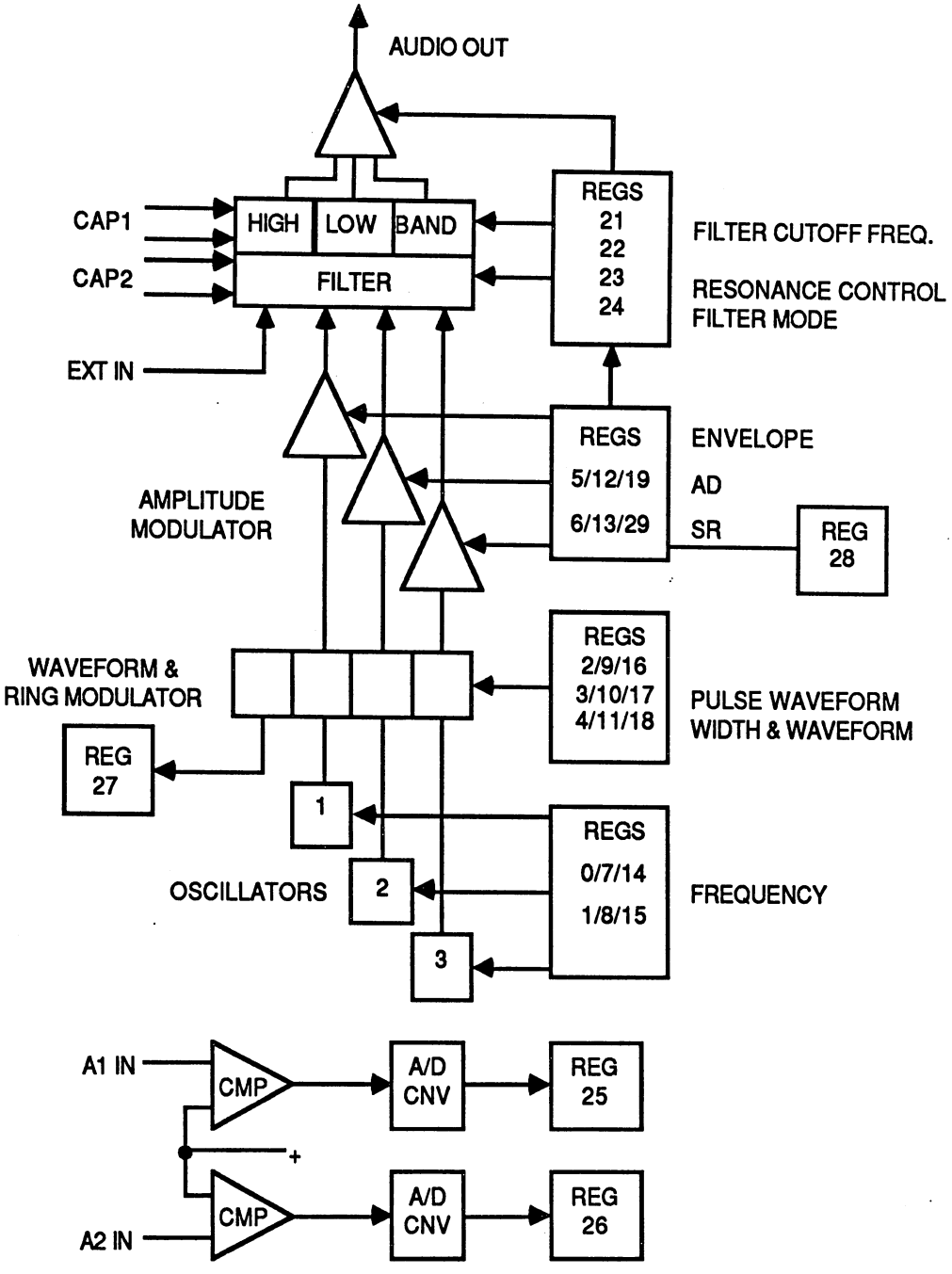
Music is an interesting computer applications area. You are fortunate that such a powerful synthesizer (the SID chip) is contained inside the C-128. It is the same component contained in the Commodore 64. Almost every game uses some of the SID's soundmaking capabilities, but none really push the chip to its limits. Often the best-known melodies can be heard coming from the computer in all possible and impossible tone colors. The computer can also talk, thanks to the SID, without additional hardware. All it needs is the right program.

SID stands for Sound Interface Device. While many synthesizers have only one voice (monophonic), the SID has three completely independent, freely programmable voices (polyphonic). Competing computers have also adopted this element and installed polyphonic synthesizers.

Here are the important features of the SID 6581:

* 3 independent, freely programmable voices
* 4 mixable wave types for each voice
* 3 mixable filters (highpass, lowpass, bandpass)
* Envelope generator (ADSR control) for each voice
* 2 cascadable ring modulators
* alternation option for external signal sources
* Two 8-bit A/D converters

The Block Diagram of the SID



4.1.2 Pinout of the 28-pin device:

1-2	CAP1A, CAP1B; connection for capacitor for programmable filter. Recommended capacitance: 2200pF.
3-4	CAP2A, CAP2B; like 1-2
5	-RES (reset); =0 brings the SID back to start-up state
6	02 (system clock); all data bus actions occur only while 02=1
7	R/W (read/write); 0=write access, 1=read access
8	-CS (chip select); 0=data bus valid, 1=data bus high-Z (tri-state)
9-13	A0-A4 (address bits 0-4); serve to select one of the 29 registers
14	GND (ground); Note: The SID should have its own ground connection for power in order to reduce interference with or from other system components.
15-22	D0-D7; data lines to and from the processor system
23	A2IN (analog input 2); operation described in Section 4.1.4
24	A1IN (analog input 1); as 23, except for A/D converter 1
25	VCC; supply voltage +5V
26	EXT IN (external input); input for external audio signals to be alienated through the SID.
27	AUDIO OUT; summed output of all signals created in the SID
28	VDD; supply voltage +12V

As we already mentioned, the SID 6581 has three independently programmable voices.

No doubt some of our readers have already programmed sounds or sound sequences in BASIC 7.0. However, complex sound and music cannot be produced using the BASIC 7.0 commands. Also, the easy-to-use commands are not available in the 64 mode; this is no reason to give up since you can get a lot out of the SID with POKE commands; in principle the BASIC 7.0 interpreter does the same thing when it executes your commands.

Those of you who have programmed some sounds in BASIC are familiar with or aware of terms like "envelope" and "amplitude modulation." We will explain these terms for everyone because they are very important when working with the SID.

Each voice consists of an oscillator, an envelope generator, an amplitude modulator, and waveform generator. With a clock frequency of 1MHz, the oscillator creates a fundamental frequency in the range 0-8200Hz

with a resolution of 16 bits. Four different waveforms are possible: sawtooth, square (with variable duty cycle), triangle, and the "white noise" familiar to every hi-fi freak. The waveform is an important criterion for the tone picture of the created sound, since every waveform has its own set of harmonics. A triangle wave is very soft, like a wood flute. The sawtooth waveform sounds more metallic, like a trumpet. A clarinet resembles a square wave; it sounds very hollow. This leaves the white noise, which doesn't really resemble any instrument, but can be used to simulate drums. Special noise effects can be best created by superimposing another waveform on the noise. Noise is achieved through the superimposition of many random frequencies.

The amplitude modulator affects the volume while the tone is being generated. This modulator is controlled by the envelope generator, which you can program directly. We will see how the envelope generator is programmed later.

In addition, the outputs of the all the devices can be sent to a programmable filter where you can further influence the tone color. Another possibility for SID fans: Voices 1 and 2 can be ring-modulated by voice 3. That means that it consists of the fundamental voice together with the sum and difference with voice 3. With voice 3 you can read out the current value of the envelope generator during the course of a sound and then change the filter based on this data, for instance.

4.1.3 Register description of the SID

The base address of the SID 6581 is $D400 (54272).

REG 0 Lower byte of oscillator frequency for voice 1.

REG 1 Upper byte of oscillator frequency for voice 1.

REG 2 Pulse width LSB for voice 1.

REG 3 Pulse width MSB for voice 1.
Registers 2 and 3 determine the on/off duty cycle of the square output on voice 1. Only bits 0-3 of register 3 are used.

REG 4 Control register for voice 1

Bit 0: KEY; Control bit for the course of the envelope generator. When changed from 0 to 1, the volume of voice 1 increases from zero to the maximum value (REG 24) within the "attack" time specified in REG 5 and then within the "decay" time specified in REG 5 falls to the "sustain" level programmed in REG 6, at which it remains until the control bit is changed to zero again. Then the volume falls to zero within the "release" time specified in REG 6.

Bit 1: SYNC; 1=oscillator 1 is synchronized with oscillator 3. This bit also has effect when voice three is supposed to be silent.

Bit 2: RING; 1=the triangle waveform output of oscillator 1 is replaced by a frequency mix (sum and difference of the frequencies of voices 1 and 3). This effect also occurs when voice three is silent.

Bit 3: TEST; When another waveform is selected along with the noise generator in the same oscillator, it can occur that the noise generator is disabled. It can be re-enabled with this bit.

Bit 4: TRI; 1=triangle wave form selected.

Bit 5: SAW; 1=sawtooth waveform selected.

Bit 6; PUL; 1=square waveform selected. The on/off relationship of this waveform is controlled in REG 2 and REG 3.

Bit 7: NSE; 1=noise generator selected.

Note for bits 4-7: It is possible in practice to select multiple waveforms at the same time. In addition to what was said for bit 3, it should be noted that result is not exaclty the sum of all of the forms but more of a logical AND of the components.

REG 5 ATTAC/DECAY

Bits 0-3: These bits determine the time it takes until the volume falls from the maximum value to the sustain level. The selectable range is from 6ms to 24 seconds.

Bits 4-7: Here the time is takes for the volume to reach the maximum value after the KEY bit is set is defined. The selectable range is from 2ms to 8 seconds.

REG 6 SUSTAIN/RELEASE
Bits 0-3: These bits determine the time within which the volume will fall from the sustain level after the KEY bit is cleared (end of the tone). The selectable range is 6ms to 24 seconds.
Bits 4-7: These bits specify the sustain level, the volume which will be maintained after the maximum value is reached and before it falls back.

REG 7-13 These registers control voice 2 in the same manner as do register 0-6, with the following exceptions:
SYNC synchronizes oscillator 2 with oscillator 3.
RING replaces the triangle output of oscillator three with the frequency mix of oscillators 2 and 3.

REG 14-20 These registers control voice 3 in the same manner as do registers 0-6 for voice 1, with the following exceptions:
SYNC synchronizes oscillator 3 with oscillator 2.
RING replaces the triangle wave from oscillator 3 with the frequency mix from oscillators 2 and 3.

REG 21 Filter frequency, low-order byte
Only bits 0-2 are used.

REG 22 Filter frequency, high-order byte
The 11-bit number in registers 21 and 22 determines the frequency.
In the Commdore 128 this frequency is determined as follows:
F=(30+W*5.8) Hz, whereby W is the 11-bit number.

REG 23 Filter resonance and switch
Bit 0: 1=voice 1 is directed to the filter
Bit 1: 1=voice 2 is directed to the filter
Bit 2: 1=voice 3 is directed to the filter
Bit 3: 1=the external source is directed to the filter
Bits 4-7: These bits determine the resonance frequency of the filter. These are used to enhance specific sections of the frequency spectrum. The effect is especially noticeable on the sawtooth waveform.

REG 24 This register has the following purposes:
Bits 0-3: Total volume
Bit 4: Switches the lowpass filter on
Bit 5: Switches the bandpass filter on
Bit 6: Switches the highpass filter on
The high and lowpass filters have a slope of 12 dB/octave. The bandpass filter has a slope of 6 dB/octave.
More than one filter can be enable at a time. If, for example, the high and lowpass filters are enabled, a notch filter results. In order to hear the effects of the filter, at least one filter must be enabled and at least one voice must be directed to the filter.
In general, the filter is used to filter out specific ranges of the frequency spectrum.
Filtering allows much finer and more ingenious manipulation of the tone picture than simply selecting the waveform permits.
Different instruments can be simulated perfectly by changing the filter frequency during the tone.

Bit 7: 1=voice 3 silent. This should be used whenever voice 3 is used to control the other voices.

All of the register described so far can only be written to. A read access returns no useful information. Only read accesses may be made to the following registers:

REG 25 A/D Converter 1

REG 26 A/D Converter 2

REG 27 Noise generator for voice 3
This register returns a random number which corresponds to the current state of the noise generator 3. The generator must be enabled, but voice 3 can be made inaudible (bit 7 in REG 24 = 1).

REG 28 Envelope generator for voice 3
This register returns the current condition of the relative volume of voice 3. This can be used to vary the frequency or filter parameters during the tone creation, for example. An example of this can be found in section 4.2.2.

Now that we have seen the table of registers, we want to clarify their use by means of short examples. We will place the emphasis on the tone-producing registers in section 4.1.5. Now we will examine the A/D converters.

4.1.4 The analog/digital converter

The words analog and digital are widely known. For example, clocks and watches with hands are called analog, while ones which display the time using numerals are called digital. These terms are derived from the way in which the time is displayed.

An A/D converter is a device for converting an analog signal, such as a voltage, to a digital value. The problem is that one must convert an analog value with theoretically an infinite number of levels to a finite digital value with predetermined levels. In this conversion there is a maximum error of +/- the smallest digital step.

As you can gather from the registers, the SID 6581 contains two A/D converters. These are designed with an internal reference voltage of about 2.5 volts.

The measuring procedure consists of charging an external capacitance and then placing a value in register 25 or 26 corresponding to the time required for a new charge of the capacitor to reach the reference voltage. This process is carried out repeatedly.

4.1.4.1 The operation of the A/D converter

A requirement of this type of A/D converter is that only resistance values can be measured, such as the position of a potentiometer, a light-sensitive resistance, or a temperature sensor.

If voltages are to be measured, they must first be converted to the appropriate form, possibly with the help of a unijunction transistor. The measurement is made simply by connecting +5V to one end of the resistance and the other end to the analog input of the SID (available on the control port, the designations are POTX and POTY). The values read from register 25 and 26 are measures of the resistances.

In order to use the entire scale of 8 bits, the resistance must range from 200 ohms (no smaller) to 200 Kohms. The programming aspects of the A/D converter are handled in the next section.

4.1.4.2 Using paddles

Paddles are nothing more than potentiometers in handheld form and are therefore well suited for the A/D converters. The generic Atari type paddles can be connected to the Commodore 128. These are connected to control port 1 or 2 where you connect a joystick.

Since some bits in CIA 1 and 2 are responsible for reading the keyboard as well as the paddles, writing a program to read the paddles is not all that simple. The best thing to do is to turn the keyboard off to inhibit nonsensical values, but only during the exact time of access of the paddles, since otherwise the keyboard will not be read.

We want to show you a short machine language program that makes it possible to read the paddles with ease. The best thing to do is to include it in your BASIC programs in the form of a BASIC loader. The program occupies the area from $OC00to $0C41. This area was chosen because it is free in C-128 mode. You can of course move it if you want to use it in C-64 mode, remembering to change the address $0C03 and $0C16 accordingly.

```
0C00   SEI           ;INHIBIT KEYBOARD
0C01   LDA #$80      ;PARAMETERS FOR PADDLE SET A
0C03   JSR $0C2E     ;GET A/D VALUES A1 AND A2
0C06   STX $0201     ;AND STORE
0C09   STY $0202
0C0C   LDA $DC00     ;GET KEYS A FROM CIA1
0C0F   AND #$0C      ;FILTER OUT REQUIRED BITS
0C11   STA $0200     ;AND STORE
0C14   LDA #$40      ;PARAMETERS FOR PADDLE SET B
0C16   JSR $0C2E     ;GET A/D VALUES B1 AND B2
0C19   STX $0203     ;AND STORE
0C1C   STY $0204
0C1F   LDA $DC01     ;GET KEYS B FROM CIA2
0C22   AND #$0C      ;FILTER OUT REQUIRED BITS
0C24   STA $0205     ;AND STORE
```

```
0C27   LDA #$FF     ;ALL BITS OUTPUT IN CIA 1
0C29   STA $DC92    ;TO REENABLE KEYBOARD READ
0C2C   CLI
0C2D   RTS          ;RETURN TO BASIC PROGRAM
0C2E   STA $DC00    ;SELECT PADDLE SET
0C31   ORA #$C0     ;AND SET CORRESPONDING BITS
0C33   STA $DC02    ;TO OUTPUT
0C36   LDX #$00     ;DELAY LOOP
0C38   DEX          ;TO QUIET THE
0C39   BNE $CFF6    ;A/D INPUT
0C3B   LDX $D419    ;GET A/D 1
0C3E   LDY $D41A    ;GET A/D 2
0C41   RTS          ;BACK TO MAIN PROGRAM
```

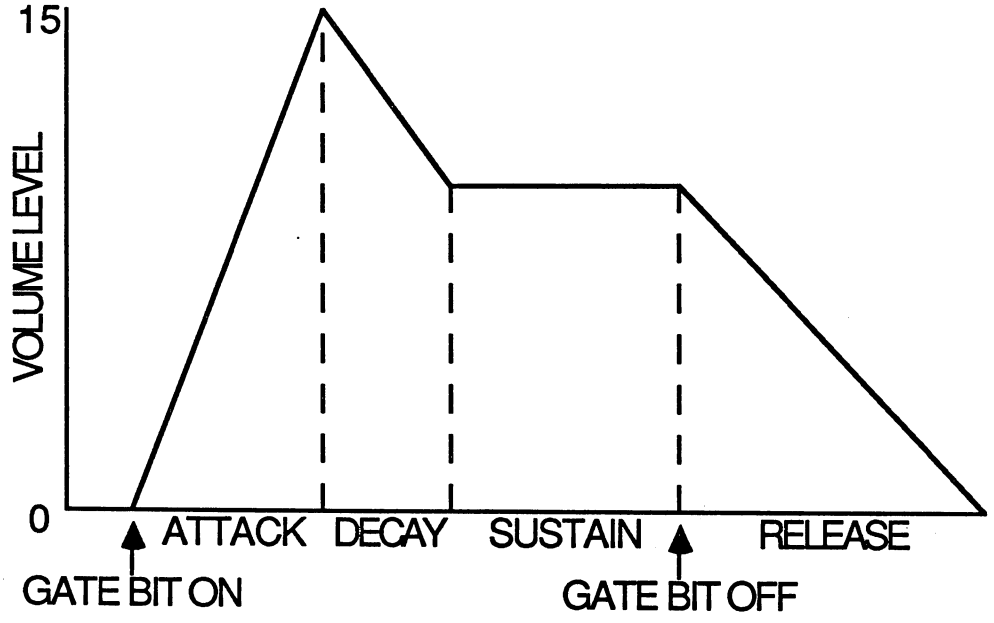
Here is the BASIC loader with an example program. Connect the paddles, start the program, and see what it does.

```
1  POKE 54528, 32: REM SET CONFIGURATION 128 ONLY
10 DATA 120,169,128,32,46,12,142,1,2,140,2,2,173
20 DATA 0,220,41,12,141,0,2,169,64,32,46,12,142
30 DATA 3,2,140,4,2,173,1,220,41,12,141,5,2,169
40 DATA 255,141,2,220,88,96,141,0,220,9,192,141,2
50 DATA 220,162,0,202,208,253,174,25,212,172,26,
   212,96
60 FOR M = 3072 TO 3072 + 65
70 READ A: POKE M,A: NEXT : REM LOAD MACHINE
   LANGUAGE
80 AX = 515 : REM PADDLE 1 CONTROL PORT 1
90 AY = 516 : REM PADDLE 2 CONTROL PORT 1
100 BA = 517 : REM BUTTON PADDLE 1
110 BX = 513 : REM PADDLE 2 CONTROL PORT 2
120 BY = 514 : REM PADDLE 2 CONTROL PORT 2
130 BB = 512 : REM BUTTON PADDLE 2
135 PRINT"<CLR>"
140 SYS 3072 : REM START M/L
150 PRINT"<HOME>" PEEK(AX)" "PEEK(AY)" "PEEK(BA)
160 PRINT" <CRS DOWN TWO>" PEEK(BX)" "PEEK(BY)"
   "PEEK(BB)
170 GOTO 140
```

4.1.5 Programming the SID

We have already talked about terms like envelope and ADSR control; we will now look at how we can program the SID directly in machine language.

The tone color is determined by the selection of the waveform; filters can further be used to change the tone picture. The envelope determines the course of the tone, the volume, the length of the rise, etc. The following figure should clarify the individual stages that a sound goes through:



We can recognize from the figure that sound is divided into four basic stages: attack, decay to sustain level, sustain, and release to zero. The duration of individual stages can be set for each voice independently. The attack of the tone starts when the KEY bit is set (bit 0, register 4 for voice 1). All values, including frequency, attack, decay, sustain, and release, must be defined before the KEY bit is set!

The tone rises from zero to the maximum volume (REG 14) within the time frame defined in attack (REG 5, bits 4-7). After the maximum value is attained, the volume drops to the sustain volume (REG 6, bits 4-7) within the decay (REG 5, bits 0-3) time. This volume is maintained until the KEY

bit is cleared. Once this happens, the volume falls back to zero within the release time (REG 6, bits 0-3). The register numbers given in parentheses refer to voice 1. For voice 2 you must add 7, and add 14 for voice 3.

The duration of the attack can be defined in a time frame from 2ms to 8 seconds. The values for decay and release lie in the range 6ms to 24 seconds. These time frames are divided into 16 steps, which you see in this table:

Value	Attack	Decay/Release
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 s
11	800 ms	2.4 s
12	1 s	3 s
13	3 s	9 s
14	5 s	15 s
15	8 s	24 s

The following program is designed to familiarize you with the waveforms and sound range of the SID 6581:

```
10  S1 = 54272 : REM VOICE 1
20  S2 = 54279 : REM VOICE 2
30  S3 = 54286 : REM VOICE 3
40  FL = 54293 : REM FILTER LO-BYTE
50  FH = 54295 : REM FILTER HIGH BYTE
60  RS = 54295 : REM RESONANCE AND COUNTER
70  PL = 54296 : REM VOLUME
80  POKE S1+4,0: POKE S2+4,0: POKE S3+4,0: REM
    CONTROL REGISTERS AT 0
90  POKE S1+2,0: POKE S2+2,0: POKE S3+2,0: REM
    PULSE AT 0
```

```
100 POKE S1+5,0: POKE S1+6,240: REM ATTACK/DECAY
    VOICE 1
120 POKE RS,0: POKE PL,15: REM RESONANCE/ VOLUME
    =15
130 PRINT "TRIANGLE..."
140 T = 16: GOSUB 400
150 PRINT "SAWTOOTH..."
160 T = 32 : GOSUB 300
170 PRINT "SQUARE..."
180 T = 64: GOSUB 300
190 PRINT "NOISE..."
200 T = 128: GOSUB 300
210 PRINT END"
220 END
300 POKE S1,0: POKE S1+1,0: REM FREQUENCY
310 POKE S1+4, T+1: REM TONE, WAVE DEFINATION
320 FOR I = 0 TO 255 : RFOR J = 0 TO 255 STEP 50
330 POKE S1,J:POKE S1+1,I
340 NEXT J,I
350 POKE S1+4,T; REM TONE
360 RETURN
```

Lines 10 to 80 should be included in every program using sound. After you have typed the program and started it, you will hear the frequency spectrum and the various waveforms of the SID. We want to give you an example of what happens when you change the envelope. For the sake of simplicity, take lines 10 to 80 from our example and add the following lines:

```
100 A=9: D=9: S=8: R=9: H=400
110 POKE S1+15,16*A+D: POKE S1+16,16*S+R
120 POKE RS,0: POKE PL,15
130 POKE S1,37: POKE S1+1,17: REM FREQUENCY
140 POKE S1+4,33 : REM SOUND ON AND SAWTOOTH
150 FOR I=0 TO H: NEXT
160 POKE S1+4,32: REM RELEASE TONE
```

You have no doubt noticed the significance of the individual variables: A=attack, D=decay, S=sustain, and R=release. The variable H is the duration of the sustain. Change the variables to get a feeling for the various sounds that different values can produce. Note that no variable, with the exception of H, may contain a value greater than 15. If you want to use the envelope, do not load register 4 with zero after the delay loop which defines the duration of the tone; this causes the tone to die. Do it like we did in the

example: When turning the tone on, load register 4 with the waveform+1. To turn the tone off, just load register 4 with the value for the waveform again.

The best way to learn how anything works is to try it out. We would like to present a few more examples for you to experiment with. Feel free to change the tone parameters to see what sort of effects you can get. The next example program uses all three voices of the SID. Again, add lines 10-80 to this example.

```
100 A=0: D=1: S=13: R=10: H=100
110 POKE S1+15,16*A+D: POKE S1+6,16*S+R
120 POKE S2+15,16*A+D: POKE S2+6,16*S+R
130 POKE S3+15,16*A+D: POKE S3+6,16*S+R
140 POKE RS,0: POKE PL,15
150 POKE S1,37: POKE S1+1,17
160 POKE S2,154: POKE S2+1,21
170 POKE S3,177: POKE S3+1,25
180 POKE S1+4,33: POKE S2+4,33: POKE S3+4,33
190 FOR I=0 TO H: NEXT
200 POKE S1+4,32: POKE S2+4,32: POKE S3+4,32
```

With the notes in DATA lines, you can use such a routine to play some music. At the end of this section is a program to play a song.

The next example will demonstrate how the frequency of a tone can be changed in relationship to the envelope. Here we use voice 3 since it is the only one from which we can read the envelope.

```
100 A=9: D=9: S=9: H=30
110 POKE RS,0: POKE P,15
120 POKE S3+5,16*A+D: POKE S3+6,16*S+R
130 POKE S3+4,33
140 FOR I=0 TO H: POKE S3+1,PEEK(54300): NEXT
150 POKE S3+4,32
160 FOR I=0 TO R*4: POKE S3+1,PEEK(54300): NEXT
```

We want to give you an example of a special effect created with "white noise". We'll let the Federation Starship Enterprise roar through our living room:

```
100 A=15: D=0: S=8: R=13: H=800
110 POKE RS,0: POKE PL,15
120 POKE S1,0: POKE S1+1,30
130 POKE S2,0: POKE S2+1,1
140 POKE S3,0: POKE S3+1,100
150 POKE S1+5,16*A+D: POKE S1+6,16*S+R
160 POKE S1+4,129: POKE S3+4,23
170 FOR I=0 TO H: NEXT
180 POKE S1+4,128: POKE S3+4,16
```

To convert a note for the SID, you must insert th frequency of the note into the following formula:

F=Freq/0.06097

Since this value consists of a high and low value, we must process the calculated value further:

Fl=F AND 15: Fh=INT(F/256)

4.2 The Filters

The SID offers three filters which you can use individually or in combination. The harmonic content of a sound wave (which is what a tone is) is controlled by means of filters. The highpass filter dampens frequencies below a defined cutoff frequency. The tones then sound somewhat metallic. The opposite of a highpass filter is the lowpass filter. Frequencies above a defined cutoff point are damped by this filter. There is also a bandpass filter which allows only a narrow band of frequencies through. If the highpass and lowpass filters are combined, only the cutoff frequency is damped, all other frequencies are undisturbed. This is called a notch filter.

In addition to filter type and filter frequency, you can also set the filter resonance. In order to understand the significance of this parameter, you should imagine the filter as a fourth oscillator in the sound chip. Filters, like oscillators, can be set to a specific frequency.

The resonance value that determines the filter itself works like an oscillator. If the resonance is set to zero, the filter simply cuts frequencies off (as already discussed). If the resonance value is increased step by step, the filter begins to oscillate more and more at the filter frequency.

The maximum value of the filter resonance is 15--the sound of the oscillator directed through the filter is then radically changed and influenced by the filter frequency. It is easy to see that a whole spectrum of new sounds can be obtained using the filters.

The following register table shows which SID registers influence the filters:

Register			---- Contents ----					
#	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
21						freq 2	freq 1	freq 0
22	freq10	freq 9	freq 8	freq 7	freq 6	freq 5	freq 4	freq 3
23	res 3	res 2	res 1	res 0	filtext	filt 3	filt 2	filt 1
24	3 OFF	highp	bandp	lowp	vol 3	vol 2	vol 1	vol 0

4.3 Synchronization and Ring Modulation

The filters allow use to change the signals produced by the individual oscillators. There is another way to change the oscillator signal in the SID: the synchronization and ring modulation.

While the only the signal of a single oscillator can be affected by the filter, synchonization and ring modulation give us the ability to change the signal of one or two oscillators in relation to their signals. An oscillator is a tone source, but its signal is determined by the signal of another oscillator.

For ring modulation, the digital number values of the oscillations of a given oscillator and the oscillator to be affected are multiplied together within the SID and output through the affected oscillator. When the frequencies of the two oscillators is close, a very complex waveform results containing many non-harmonic overtones, so that it often sounds metallic or bell-like.

Here is the program we promised that will play a song:

```
0 REM ***  SONG   ***
4 FOR I= 54272 TO 54296: POKE I,0:NEXT
10 FIRST=54272
11 VL    =FIRST+24
12 AN    =FIRST+5
13 OUT   =FIRST+6
14 H1    =FIRST
15 H2    =FIRST+1
16 VC1   =FIRST+4
20 POKE VL,15
21 POKEAN,23
22 POKE OUT,123
30 READ NTE,DUR
40 IF NTE=0 THEN END
50 F2=NTE /256:F1=NTE  AND 255
60 POKE H2,F2:POKE H1,F1
70 POKE VC1,33:FOR I = 0 TO DUR*100:NEXT
80 POKE VC1,32:FOR I = 0 TO DUR*100:NEXT
90 GOTO 30
100 REM *** NOTES ***
130 DATA 6430,1,6430,1,6430,3,7217,2,5407,2
140 DATA 6430,1,6430,2,8583,3,9634,2,9634,1,
   10814,2
150 DATA 10814,3,9634,2,9634,2,8583,1,8583,5,
   10814,1
160 DATA 11457,3,10820,2,10814,2,9636,4,10814,1,
   9634,1
170 DATA 9634,1,8583,11,10814,1,9634,2,8534,5,
   10814,1
180 DATA 12860,2,14435,5,12860,1,12860,2,10814,3
   9634,2
190 DATA 9634,1,9634,2,10814,9,10814,2,11457,3
   10820,2
200 DATA 10814,2,9634,4,10814,1,9634,1,9634,1,
   8585,15
210 DATA 0,0
```

This concludes out chapter on the SID. We hope that you have found enough information and suggestions to start working with this chip. This applies particularly to those of you who can and want to program in machine language. Have fun!

CHAPTER 5

Chapter 5: The 8563 VDC Chip

5.1 General Information

As mentioned in Chapter 2, you can connect two monitors to your Commodore 128. The 40-column monitor is contrilled by the VIC chip. The 80 column RGB monitor, is driven by the 8563 VDC. The 80-column screen is well suited for professional applications that are impossible or more difficult with a 40-column screen. RGB stands for Red Green Blue, which means that the colors red, green, and blue can be displayed on the screen in various combinations. The color white, for example, is achieved with an equal mix of all three colors; the color yellow can be made with a combination of red and green. But don't worry--you don't have to figure out which colors you have to mix to obtain the one you want. We will come back to the color codes for the 15 possible colors.

An important bonus of the VDC chip is that it doesn't use up any of the main memory for storing its screen contents. It has 16K of its own memory which it uses for video RAM and attribute RAM. Even the character generator is copied into this 16K.

On the international models of the C-128, pressing the <ASCII/DIN> key copies a foreign language character set into memory. You will notice that it takes a little while before the cursor is ready again.This is because all 4096 bytes of the charater generator are copied from ROM into the video controller RAM. Stop and think for a minute: Why 4096 bytes? There are two character sets. 2048 bytes are all that are required to define 256 characters! You are right of course, but *both* character sets selected with the Commodore key on the 40-column screen are stored in the VDC memory. These two character sets can be displayed simultaneously on the 80-column screen. A bit in the attribute RAM determintes which character set is to be used. Since the character set is in the VDC RAM, it is easy to change the appearance of individual characters by simply changing the contents of the RAM.

But all of these advantages that this separate video RAM offers us has another side to it. Addressing this RAM is quite complicated--it has to be done indirectly via two registers on the VDC chip. We will talk about this more later.

Those who think it would be boring to take a closer look at this chip are deceiving themselves. This chip offers an enormous number of possibilities; to describe them all would far exceed the scope of this book. Hackers are advised to take a closer look at this chip, since it seems that you always find something new that can be done with it. We will limit ourselves to the most important, most interesting possibilities. The expectations that one has for an 80-column controller are far exceeded: this video controller can display hi-resolution graphics with a resolution of 640x200 points!

5.2 The Pinout:

1	CCLK; Character Clock
2	-DCLK; Dot Clock
3	HSYNC; Horizontal Synchronization
4	CS; System time
5-6	Not connected
7	-CS; Chip Select
8	-RS; Resister Select (Address Line A0)
9	-R/W; Read-Write Selection
10-11	D7-D6; Data Lines D7-D6
12	GND;
13-18	D5-D0: Data Line D5-D0
19	DISPEN; Display Enable (not wired)
20	VSYNC; Vertical Synchronization
21	DR/-W; Display-RAM READ/WRITE
22	-RES; Reset Line (output) - meaning unknown
23	-RES; Reset Line (input)
24	TST; meaning unkown
25	LPEN; Light Pen
26-33	DA0-DA&; address Display-RAM
34-42	DD0-DD&; Data Lines Display-RAM
37	VCC; operating voltage +5V
43	I; Intensity
44	B; Blue
45	G; Green
46	R; Red
47	-RAS; Low-Address Select
48	-CAS; Column Address Select

5.3 The VDC Registers

The 8563 VDC chip has a total of 37 registers available, which have the following meanings: (The values in parentheses indicate the default values that are loaded into the registers after a warm start.)

REG 0 HORIZONTAL TOTAL; (126) This register specifies the total number of characters per line, including the beam return. This register should be loaded with an 8-bit value corresponding to the technical data of the monitor.

REG 1 HORIZONTAL DISPLAYED; (80) In this register the number of actual characters per line is programmed. All 8-bit values smaller than REG 0 are possible. The standard value is 80.

REG 2 HORIZONTAL SYNC POSITION; (102) In this the left border is sychronized. All 8-bit values smaller than REG 0 are possible. If the register value is reduced, the left border moves right; if the contents are increased, the left border moves left.

REG 3 SYNC WIDTH; (73) Bits 0-3 determine the horizontal sync pulse width in characters. The value zero cannot be programmed. Bits 4-7 determine the vertical sync pulse width multiples of a raster period. If zero is programmed, it means 16.

REG 4 VERTICAL TOTAL; (39) This register contains the number of total lines including the vertical beam return. This register should be programmed according to the technical data of the monitor used.

REG 5 VERTICAL TOTAL ADJUST; (224) Bits 0-4 serve as a fine adjustment for REG 4. Bits 5-7 are always set. The default value 224 means that bits 0-4 are cleared.

REG 6 VERTICAL DISPLAYED; (25) Contains the number of representable characters. Any value smaller than REG 4 is possible.

REG 7 VERTICAL SYNC POSITION; (32) This register defines the upper border of the screen. If the contents of this register are increased, the screen moves up. Correspondingly, the screen moves down when the value is decreased.

REG 8 INTERLACE MODE; (252) Bits 0-1 determine the interlace mode. Normally these bits are cleared. 00 and 10= non-interlace mode, 01=interlace-sync mode (the screen appears to flicker), 11=interlace-sync and video mode. Try this once!

REG 9 CHARACTER TOTAL VERTICAL; (231) Bits 0-4 determine the number of raster lines per character (vertical) minus one. Bits 5-7 are always set. The default value 231 stands for 7, or 7+1=8 raster lines per character.

REG 10 CURSOR MODE/START RASTER; (160) Bits 5-6 set the cursor mode: 00=non-blinking, 01=cursor not displayed, 10=blink fast, 11=normal blink.

REG 11 CURSOR END SCAN LINE; (231) Only bits 0-4 are relevant; the others are always set. This register contains the line at which the cursor will stop. For a block cursor for example, the cursor starts at line 0 and stops at line 7. For an underline cursor: start and end at 7.

REG 12 DISPLAY START ADDRESS HI; (0) The high byte of the start of the video RAM is stored in this register. Normally the video RAM lies at address $0000 in the special VDC memory.

REG 13 DISPLAY START ADDRESS LO; (0) The low-bye of the video RAM corresponding to REG 12 is defined here.

REG 14 CURSOR POSITION HI; The high byte of the cursor is defined in this register. The cursor address must be specified because the VDC will let it blink on its own.

REG 15 CURSOR POSITION LO; The low byte of the cursor address corresponding to REG 14 is defined here.

REG 16 LIGHT PEN VERTICAL; This and the following register can only be read. The two high-order bytes in register 16 are always zero. This register returns the vertical address of the light pen. The value must be corrected by the software because the raster beam will have moved by the time the raster line is determined.

REG 17 LIGHT PEN HORIZONTAL; Corresponding to register 16, this register contains the horizontal address of the light pen.

REG 18 UPDATE ADDRESS HI; The high byte of the address to be manipulated is given in this register. It doesn't make any difference if the address is in video RAM, attribute RAM, or somewhere else.

REG 19 UPDATE ADDRESS LO; The low byte of the address to be manipulated is given here in connection with register 18.

REG 20 ATTRIBUTE ADDRESS HI; (4) The high-order byte of the start address of the attribute memory is placed in this register. The attribute RAM defines the color and status of each character on the screen.

REG 21 ATTRIBUTE ADDRESS LO; (0) In connection with register 20, this register sets the low-order byte of the start address. In the normal mode the attribute RAM starts at address $0400.

REG 22 CHARACTER TOTAL & DISPLAYED; (120) Bits 4-7 determine the total number of displayed horizontal lines (7). Bits 0-3 set the displayed number of lines (8). This defines the width of a character.

REG 23 CHARACTER DSP(V); (232) Number of vertical lines displayed (8); this defines the height of a character.

REG 24 VERTICAL SMOOTH SCROLL; (32)
Bit 7: COPY bit; when this bit is set, the range at the block-start address is copied to the update address when the word count register is written. If this bit is cleared, the update address is filled with the data register (REG 31)
Bit 6: RVS bit; If this bit is set, the entire screen display is reversed. A set point is cleared and a cleared point is set.

Bit 5: CBRATE; meaning is not yet known.
Bits 0-4: Here the vertical edge of the screen can be moved (smooth scrolling).

REG 25 HORIZONTAL SMOOTH SCROLLING; (64)
Bit 7: TEXT; if this bit is cleared, the text mode is enabled. The information for the characters is taken from the CHARROM. If this bit is set, single-point graphics are enabled.
Bit 6: ATR; This bit indicates whether the color information for a character should come from the attribute RAM (set bit) or if all points should appear in monochrome (color is in REG 26).
Bit 5: SEMI; semi-graphic operating mode;
1: the existing horizontal space between two characters is filled with the color of the character last displayed.
0: like (1), but the space is filled with the background color.
Bit 4: DBL; If this bit is set, the characters appear in double width.
0: Pixel size=1 dot clock
1: Pixel size=2 dot clocks
bits 0-3: Here the horizontal edge can be moved in raster lines (smooth scrolling).

REG 26 FORGND/BACKGND; (240)
Bits 0-3 determines the background color.
Bits 4-7 determine the foreground color for graphic or monchrome mode.

REG 27 ADDRESS INCREMENT ROW; (0)
This register defines the number of bytes are to be added to the video RAM for each column. Normally this is zero. If you redefine the character width, for instance, (and thereby the the number of characters/line), this value must be reprogrammed.

REG 28 CHARACTER BASE ADDRESS; (47)
Bits 5-7 determine the base of the character generator, address bits 13 to 15; the character generator can only be moved in 8K steps.
Bit 4: RAM; This bit defines the RAM type:
1: 4164; 0: 4416

REG 29 UNDERLINE SCAN LINE; (231)
Bits 0-4 indicate the line in which to underline. The default value is 8. This register can be used to change underlining to overlining.

REG 30 WORD COUNT;
In this register you write the number of characters which are to be written to the update address, or if the COPY bit is set, the number of bytes to be copied.

REG 31 DATA;
This register contains the data to be written to a memory location. If a memory location is read, the contents will appear in this register.

REG 32 BLOCK START ADDRESS HI;
This register (and the following) defines the start address of the block to be copied.

REG 33 BLOCK START ADDRESS LO;
Corresponding to register 32, this register defines the low-order byte.

REG 34 DISPLAY ENABLE BEGIN; (125)
Number of characters from the start of the displayed line to the postive edge on the display enable pin.

REG 35 DISPLAY ENABLE END; (64)
Like REG 34, but until the negative edge.

REG 36 DRAM REFRESH RATE; (245)
Bits 0-3 specify the rate at which the VDC memory must be refreshed (refresh cycles per screen line).

5.4 General Information About the VDC Registers

To look at each register individually is not very informative. At best, you can recognize what the individual registers do when you simply write values to them and see what happens. Not all of the registers are useful to the programmer, as is the case with the VIC or SID chip. The VDC contains a number of registers that are present simply for screen display and synchronization. You should never change these registers.

The base address of the 80-column video controller is $D600. A little tip: At least in our prototype, the VDC could also be manipulated in the 64 mode; this means that 80-column mode is possible in the 64 mode as well! In addition to the ability to program in the 2MHz mode, this presents another small gap in the compatibility of the 64 mode.

You cannot address the various registers of the VDC as simply as with the VIC or SID. Using the VIC or SID, you simply add the register number to the base address. In the VDC, register manipulation is relative, meaning that you have to tell the controller which register you want to read or write and then perform this operation. This is certainly a complicated method, but you get used to it quickly. If, for example, you want to change a byte in the video RAM, you must address this memory location relatively via the registers, since they are not directly addressable.

Now we'll describe the technique. The VDC can be accessed at address $D600 and $D601.

If you want to read a register, for instance, you must write the register address in $D600. The VDC then returns the current contents of the register in address $D601.

If you want to write to a register, write the register number in address $D600 and the new register value in address $D601.

Address $D600								
(Write)	----	----	R5	R4	R3	R2	R1	R0
(Read)	Status	LP	VBLANK	----	----	----	----	----

Address $D601								
(Read/write)	D7	D6	D5	D4	D3	D2	D1	D0

If you write to address $D600, the register is selected. Bits 0 to 5 are used for this. You can also read from $D600; this will return a status report of the VDC. Bit 7, the status bit, indicates if the VDC is finished with its last action or not. If this bit is set, the video controller is not yet done, and you must wait until it gives the green light or data will disappear. It is necessary to test this bit only in machine language since BASIC is far too slow for this to be a problem. If, for example, we want to write to the DATA regiser in the VDC in machine language, it would look like this:

```
        LDA #$1F    ;DATA REGISTER
        STA $D600   ;SELECT
WAIT    BIT $D600   ;TEST STATUS BIT
        BMI WAIT    ;NOT SET, THEN NOT DONE
        LDA #$21    ;ASCII CODE FOR "!"
        STA $D601   ;AND WRITE
        RTS         ;RETURN
```

In this routine, we have placed the value $1F into the VDC select register. We loop at `WAIT` until the VDC tells us that it has accepted our value. Then we can write into the register at $D601. Another delay routine should be included after writing to address $D601, though this depends on the program.

Bit 6 of address $D600 is reserved for the light pen and does not interest us at the moment. Bit 5 tells us if the cathode beam is on its return course (bit is set) or not. This can be used for synchronizing various activities to the beam. The rest of the bits are not used.

To summarize, writing to address $D600 selects the VDC register. Writing to address $D601transfers the data.

You can use the following machine language code to read the value of the DATA register:

```
        LDA #$1F    ;DATA REGISTER
        STA $D600   ;ADDRESS REGISTER
WAIT    BIT $D600   ;STATUS BIT STILL SET?
        BPL WAIT    ;NOT DONE
        LDA $D601   ;GET CURRENT CONTENTS
```

We can also manipulate the VDC from BASIC. But because of BASIC's slowness, there may be some problems, so you shouldn't be annoyed if things don't work right away.

Read and writing the DATA register in BASIC would look like this:

```
10 A=DEC("D600"): D=A+1: REM BASE ADDRESS VDC
20 POKE A,31: PRINT PEEK(D): REM GET REG CONTENTS
30 POKE A,31: POKE D,33: WRITE TO REGISTER
```

But now you may want to know how to work with screen addresses. We know that the video RAM starts at address $0000 and consists of 2000 characters. To manipulate an address in RAM, you must first define whether you want to read or write in the `update register`.

Let's show you with a short BASIC program:

```
10 A=DEC("D600"): D=A+1
20 POKE A,18: POKE D,0: REM UPDATE ADDRESS HI BYTE
30 POKE A,19: POKE D,0: REM UPDATE ADDRESS LO BYTE
40 POKE A,31: POKE D,1: REM A 1 FOR "A"
50 POKE A,30: POKE D,1: REM SET CHAR COUNTER
```

It demonstrates several key points. The order in which you `POKE` is important. First the `update address` is selected. Next the character to be displayed is sent. Finally the number of times the character is to be displayed is sent. If you haven't sent the update address, you won't get your desired results.

Unfortunately this routine probably won't work! Not in the FAST mode nor the SLOW mode. You can see this more clearly by adding the following lines to the program:

```
5 PRINT CHR$(19);"  ": REM TWO SPACES
60 GETKEY A$: RUN
```

Each time you press a key, the first two positions on the screen are erased. After this, the video controller is "requested" to display an "A" in the first screen position. So we can check to see if an "A" is really displayed at the correct position.

When we start the program, we see that the result does not correspond to our expectations. The A moves from left to right. It is not always placed at the right location. Sometimes an "@" even appears on the screen instead of the A.

Unfortunately we can't achieve any better results here. In BASIC, it appears to be impossible. We have tried various methods, all without success. BASIC is simply too slow. What we can't accomplish in BASIC, we should at least be able to do in machine language. So let's look at a short machine language program which does the same thing as our BASIC program.

Below is the assembly language listing of this routine, which is designed to display an "A" on the screen. Press the reset button on your computer to make sure all the VDC registers are reset before entering this program.

```
00D00    8E 00 D6    STX $D600
00D03    2C 00 D6    BIT $D600
00D06    10 FB       BPL $0D03
00D08    8D 01 D6    STA $D601
00D0B    60          RTS
00D0C    A2 12       LDX #$12
00D0E    A9 00       LDA #$00
00D10    20 00 24    JSR $0D00
00D13    E8          INX
00D14    20 00 24    JSR $0D00
00D17    A2 1F       LDX #$1F
00D19    A9 01       LDA #$01
00D1B    20 00 24    JSR $0D00
00D1E    CA          DEX
00D1F    4C 00 24    JMP $0D00
```

This little machine language routine can be entered with the built-in monitor and tested with the following BASIC program:

```
10 PRINT CHR$(147);
20 SYS DEC("0D0C"): GETKEY$: RUN
```

Start the program with RUN. The result will probably surprise you. The position is right, but now we have two "A's" instead of one. The VDC displays word count+1 many characters, though it does this very carefully and at the correct address. If we had wanted to display two "A's", we would be all set, but we wanted just one. Loading the word count register with zero causes 256 characters to be printed.

The solution is quite simple: If you want to display just one character, do not write to the word count register after selecting the update address and

the DATA register. Just load the update address with a new value or read from this register--then it works.

To try this out we need to change our machine language program at address $00D1E:

```
00D1E A2 12    LDX #$12
00D20 4C 00 24 JMP $0D00
```

You see that it doesn't matter what value you write to the update register. The sample program is located in the output buffer for the RS-232 ($0D00-$0DFF). Now we'll change the machine language routine so we can write any character to any position, even in BASIC.

```
10 REM ========================================
20 REM  BASIC LOADER FOR 80-COLUMN POKE ROUTINE
30 REM ========================================
40 :
50 FOR I=0 TO 36
60 : READ X
70 : POKE DEC("D00")+I,X
80 : S=S+X
90 NEXT
100 IF S<>2850 THEN PRINT "*** ERROR IN DATA ***":
    END
110 :
120 DATA 142,0,214,44,0,214,16,251,141,1,214,96,
    162,18,169,0
130 DATA 32,0,13,232,169,0,32,0,13,162,31,169,1,
    32,0,13
140 DATA 162,18,76,0,13
150 :
160 REM *** TRY IT OUT ***
170 :
180 PRINT CHR$(147);
190 SYS DEC("D0C"): GETKEY A$: GOTO 180
```

Now we have the program we wanted, even if it can't be done in "pure" BASIC. Maybe there is some algorithm which works in BASIC and permits manipulations to be made on the 80-column screen.

As already mentioned, this routine can display any character at any location on the screen. To make it do this, you have to write the high byte of

the address to address $0D0F, the low byte to address $0D15, and the character to address $0D1C. Try it once with the following sample program:

```
 10 REM ======================================
 20 REM    EXAMPLE PROGRAM FOR POKE ROUTINE
 30 REM ======================================
 40 :
 50 LO=DEC("D15"): HI=DEC("D0F"): PO=DEC("D1C")
 60 FOR I=0 TO 1999
 70 : POKE LO, I AND 255 : REM POKE LOW BYTE
 80 : POKE HI, I/256     : REM POKE HIGH BYTE
 90 : POKE PO, I AND 255 : REM FOR EXAMPLE
100 : SYS DEC("D0C")
110 NEXT
120 GETKEY A$
```

But we don't want to display just one character. Sometimes it would be practical if we could display 80 characters at once (with the help of the word count register), for example, to erase a line or something similar. But the VDC might display one character too many. Imagine a word processing program that had this problem: it would be quite aggravating.

This error must have been compensated for in the operating system, though. The solution is (what, again?) rather simple and works very well.

You know the starting address of the area to be filled with characters. Let's say that you want to display `n` characters. So you can calculate the address in video RAM where they will be written. Simply let the video controller fill `n-1` characters.

Next we can read the `update address` (which the VDC automatically increments) to determine if it has displayed the correct number of characters. If so then we are done. Otherwise we must display one more character. This method is always faster than writing each character by itself. You can use an operating system routine that outputs a character based on the update address and DATA register as many times as the value in the accumulator specifies. This routine is found at the address $C53E. Place the calculated address in $0A3C/$0A3D. We'll add the routine to the one already existing:

```
00D25    A2 12       LDX #$12
00D27    A9 00       LDA #$00
00D29    20 00 0D    JSR $0D00
00D2C    8D 3D 0A    STA $0A3D
00D2F    E8          INX
00D30    A9 00       LDA #$00
00D32    20 00 0D    JSR $0D00
00D35    8D 3C 0A    STA $0A3C
00D38    A9 00       LDA #$00
00D3A    A2 1F       LDX #$1F
00D3C    20 00 0D    JSR $0D00
00D3F    A9 00       LDA #$00
00D41    18          CLC
00D42    48          PHA
00D43    6D 3C 0A    ADC $0A3C
00D46    8D 3C 0A    STA $0A3C
00D49    90 03       BCC $0D4E
00D4B    EE 3D 0A    INC $0A3D
00D4E    68          PLA
00D4F    4C 3E C5    JMP $C53EA
```

You can add the following DATA lines to the BASIC loader:

```
150 DATA 162,18,169,0,32,0,13,141,61,10,232,169,0,
    32,0,13
160 DATA 141,60,10,169,0,162,31,32,0,13,169,0,24,
    72,109,60
170 DATA 10,144,3,238,61,10,104,76,62,197
```

Lines 50 and 100 must also be changed:

```
 50 FOR I=0 TO 81
100 IF S<>5859 THEN PRINT "*** ERROR IN DATA ***":
    END
```

Store the high byte of the starting address at address $0D28, the low byte at address $0D31. You must POKE the fill character into address $0D39 and the number at address $0D40. Example:

```
POKE DEC("0D28"),0 : POKE DEC("0D31"),0: REM ADDR
POKE DEC("0D39"),33: REM FILL CHARACTER
POKE DEC("0D40"),79: REM FILL QUANTITY-1
SYS DEC("0D25")    : REM CALL THE ROUTINE
```

Once you enter these lines, the first line will be filled with exclamation points.

As already mentioned, you can change the attribute RAM in the same way as we changed the screen contents. For example, if you want to display the first line in flashing white, you must fill the attribute RAM with $1F=31. To do this we enter the following lines:

```
POKE DEC("0D28"),8 : POKE DEC("0D31"),0:
                     REM ATTRIBUTE RAM
POKE DEC("0D39"),31: REM FILL CHARACTER
POKE DEC("0D40"),80: REM FILL QUANTITY
SYS DEC("0D25")    : REM CALL THE ROUTINE
```

5.4.1 The character set

The character set in the VDC can be easily changed. Sixteen bytes of RAM must be defined per character. Eight bytes are copied from the CHARROM, and eight additional zero-bytes are appended for reasons internal to the VDC. The character set starts at address $2000 for the VDC. To read a character out or to change it, you can find it with this address:

2*4096 + <code>*16

The VDC, unlike the VIC, can display the two character sets, obtained with <SHIFT><Commodore> in 40 column mode, on the screen at the same time since these are both found in the VDC RAM. The reverse characters are also defined, though these aren't really necessary since a bit in the attribute RAM controls whether a character is displayed normal or in reverse. Both of these features can be utilized if you want define additional characters.

The memory layout of the VDC RAM looks like this:

```
$0000-$07CF:Video RAM
$0800-$0FCF:Attribute RAM
$2000-$3FFF:CHARRAM(character generator)
```

5.4.2 The character attribute

The attribute of a character is composed of several criteria: The first is the RGB signal, whether red, green, or blue are active (all bits here are set for white, for instance), then the intensity signal (which determines the two levels of brightness of the character). Then there is a bit which determines if a character should flash on and off, a bit to underline a character, a bit for reverse, and a bit for the alternate character set. You can see that the reverse characters really need not be defined at all, since a corresponding bit is provided in the attribute RAM. But to make things simpler, the reverse character set was simply copied along with the rest of the characters.

But now we come back to the actual attribute RAM: The eight bits of an attribute byte are arranged as follows:

ALT	RVS	UL	FLASH	R	G	B	I
7	6	5	4	3	2	1	0

ALT stands for ALTernate. If the second character set is selected (the one obtained with <SHIFT><Commodore> on the keyboard), the ALT bit in the attribute RAM is set.

RVS stands for ReVerSe and means that the character will be displayed in reverse. Unfortunately, no direct use is made of this bit. Professional software programmers can make better use of the reverse characters.

UL stands for UnderLine. If this bit is set, the corresponding character is underlined in the raster line defined in register 29; normally this is line 7.

FLASH is self-explanatory. If this bit is set, the character defined by the given attribute byte will flash on and off. Color and any underlining is retained.

R stands for Red, G for Green, and B for Blue. The color signal consists of the set and cleared bits. There is also an intensity signal I that is used to set the brightness; a set bit means bright.

Here is a table of the 15 possible color and intensity combinations:

R	G	B	I	Color
0	0	0	0	Black
0	0	0	1	Dark grey
0	0	1	0	Blue
0	0	1	1	Light blue
0	1	0	0	Green
0	1	0	1	Light green
0	1	1	0	Cyan
0	1	1	1	Light Cyan
1	0	0	0	Red
1	0	0	0	Light red
1	0	1	0	Purple
1	0	1	1	Light Purple
1	1	0	0	Brown
1	1	0	1	Yellow
1	1	1	0	Light grey
1	1	1	1	White

5.5 Using the VDC Registers

As already mentioned, the 37 VDC registers account for a very flexible 80-column controller. We want to take a closer look at and demonstrate their use with some examples. One of the more useful is the ability to display 30 lines on the screen instead of 25 a second is the ability to use the high-resolution graphics with a resolution of 640x200 points. We will concentrate on these two examples.

But first we present a program which is very useful for exploring the world of the VDC registers. When testing, you may often find that your screen displays nothing but garbage. This means you have confused the controller so much that it can no longer display a meaningful picture. The best thing to do is to press the <RUN/STOP><RESTORE> keys.

On international models of the C-128 that include a foreign character set, the character generator may be overwritten. The best thing to do then is to press the <ASCII/DIN> key to copy the character generator back to the normal mode.

This program shows you the current register contents on the screen and then lets you write to any of the registers. After you have entered the values, you can observe the results directly on the screen (if in fact there are results). The current register contents are then displayed again.

```
 10 REM *** TESTING THE VDC REGISTERS ***
 20 :
 30 A=DEC("D600"): D=A+1
 40 PRINT CHR$(147)"CURRENT REGISTER CONTENTS -"
 50 FOR I=0 TO 37
 60   POKE A,I: C=PEEK(D)
 70   PRINT "#";I;RIGHT$(HEX$(C),2),
 80 NEXT I
 90 PRINT: PRINT
100 INPUT "REGISTER, VALUE --- ";RE,VA
110 POKE A,RE: POKE D,VA: GOTO 40
```

5.5.1 Smooth scrolling

As with the VIC chip, you can move the screen vertically or horizontally in raster line increments on the VDC. VDC register 24 (bits 0-4) and 25 (bits 0-3) are used for this purpose. Contrary to the way smooth scrolling is done on the VIC, you don't lose any columns or lines on the VDC. The VDC is not well-suited for games--it has very good resolution, but its complicated addressing is far too slow--but you can use smooth scrolling to create many useful effects. Here is a short demonstration program which shows the operation of smooth scrolling on the 80-column screen.

```
 10 REM *** DEMO PROGRAM FOR SMOOTH SCROLLING ***
 20 A=DEC("D600"): D=DEC("D601")
 30 VE=24: HO=25
 40 PRINT CHR$(147)CHR$(27);"M"; : REM SCREEN CLR
    AND SCROLL OFF
 50 A$="Hello C-128 fans!"
 60 FOR I=0 TO 24
 70   PRINT A$
 80 NEXT
 90 :
100 FOR I0=0 TO 6
110   POKE A,VE: V=PEEK(D) AND 240 OR I0
```

```
120    POKE A,VE: POKE D,V
130    FOR I1=1 TO 20: NEXT
140    POKE A,HO: H=PEEK(D) AND 240 OR I0
150    POKE A,HO: POKE D,H
160    FOR I1=1 TO 20: NEXT
170 NEXT
180 GOTO 100
```

If this goes too fast for you or not fast enough, change the delay loops in lines 130 and 160 correspondingly.

If bit 3 is cleared, 25 lines are displayed and the following (or preceding) RAM is scrolled on the screen. If you set bit 3, only 22 lines are displayed and you can scroll the last three lines of the screen by means of smooth scrolling.

5.5.2 Block copying

If the controller is so hard to access, why is screen scrolling so fast? The solution is simple: The VDC is intelligent enough to move entire blocks in its memory. If this had to be done via the relative addressing, it would take a considerably longer time.

If you want the VDC to move an area of memory, you must tell it this via the COPY bit (bit 7 in REG 24). If this bit is set, the VDC copies instead of filling. The starting address of the block to be copied is defined in registers 32 and 33; the destination address of the copying procedure must be defined in the update register (REG 18 and 19); the copy process begins when you write to the word count register. This also specifies the number of characters to be copied.

NOTE: The word count register specifies the exact number of characters to be copied. For example, if you want to copy the first text line on the screen to the line below and preserve the attributes, you must first copy the text line and then the attributes. We will do an upward-scroll in our example program--in BASIC it goes quite slowly, but in machine language it is fast enough.

```
 10 REM *** DEMO PROGRAM FOR BLOCK COPYING ***
 20 A=DEC("D600"): D=DEC("D601")
 30 POKE A,24: C=PEEK(D):REM *** CONTENTS OF REG 24
 40 POKE A,24: POKE D,C OR 128:REM *** SET COPY BIT
 50 FOR Z=24 TO 0 STEP -1
 60   AQ=Z*80: AZ=AQ+80: REM *** SOURCE AND DEST
 70   POKE A,18: POKE D,AZ/256: POKE A,19: POKE D,
      AZ AND 255
 80   POKE A,32: POKE D,AQ/256: POKE A,33: POKE D,
      AQ AND 255
 90   POKE A,30: POKE D,79: REM *** COPY TEXT
100   AQ=2048+AQ: AZ=2048+AZ:REM *** ATTRIBUTE ADDR
110   POKE A,18: POKE D,AZ/256: POKE A,19: POKE D,
      AZ AND 255
120   POKE A,32: POKE D,AQ/256: POKE A,33: POKE D,
      AQ AND 255
130   POKE A,30: POKE D,79: REM *** COPY ATTRIBUTE
140 NEXT
150 PRINT CHR$(19);CHR$(27)"D"; :REM CLEAR 1ST LINE
160 POKE A,24: POKE D,C: REM *** CLEAR COPY BIT
```

This routine does nothing more than the ESC sequence CHR$(27);"W", but it shows the operation of block copying.

5.5.3 Foreground and background color

You can define the background color of the 80-column screen in register 26 (bits 0-3). The foreground color has effect in the graphic mode and--provided the ATR bit in register 25 is not set--also in the text mode.

The definition of the register:

```
POKE DEC("D600"),26
POKE DEC("D601"),<foreground>*16 + <background>
```

5.5.4 The cursor mode

You can also determine the appearance of the cursor yourself. You can turn it off completely, make it blink fast or slow, and define it as a block or underline cursor. You can make these definitions using ESC sequences, but there are situations where this is not possible--such as in machine language. The cursor mode is set in register 10. Further, register 10 indicates in which raster line the block cursor is to begin. With the starting and ending line of the block cursor you can turn the cursor into a broad stripe in the middle, etc. (The underline cursor is defined in the same manner). Here are the four possible bit combinations of the cursor mode:

00 - non-blinking cursor
01 - cursor off
10 - slow cursor (cursor flashes at 1/16 SRF)
11 - fast cursor (cursor flashes at 1/32 SRF)

SRF = Screen Refresh Frequency

As already mentioned, the VDC takes over all functions of displaying the character under the cursor and does not burden the CPU with it.

For a block cursor, the start line is line 0; the end line, defined in register 11, is line 7. In order to define a underline cursor, one need only change the start line to 7.

To demonstrate the effects, simply try out the following:

```
10 REM *** DEMO FOR CURSOR ***
20 A=DEC("D600"): D=DEC("D601")
30 FOR X = 1 to 7: REM LINE 1 TO 7
40 :POKE A,10: POKE D,X: REM *** NON-BLINKING-START
50 POKE A,11: POKE D,7: REM END LINE=7
60 FOR I = 1 TO 100 : NEXT I
70 NEXT X
```

The cursor address is defined in registers 14 and 15; the cursor is then displayed at this location where it blinks if so instructed and negates the character found underneath it. These two registers have no other function.

5.5.5 The character length and width

The matrix of the characters found in VDC RAM is 8x8 points; this means that the characters displayed on the screen are 8 points wide and 8 lines tall. This can be changed. The height and width of the characters can be set in registers 22 and 23. The following BASIC program demonstrates this:

```
10 REM *** DEMO PROGRAM FOR CHARACTER MATRIX ***
20 :
30 A=DEC("D600"): D=A+1
40 FOR I0=0 TO 8: POKE A,22: POKE D,112+I0
50 FOR I1=0 TO 8: POKE A,23: POKE D,I1
60 FOR I2=1 TO 30: NEXT I2,I1
70 FOR I2=1 TO 30: NEXT I2
80 NEXT I0
90 GOTO 40
```

You must always add 112 to register 22 because the upper nibble must always be $7.

5.5.6 More than 25 lines on the screen

Yes, you read it right! It is possible to display 25 lines with a total of 2000 characters on the screen, but you can even display 28 lines with 2240 characters and more. This is no trick of the imagination; every programmer who wants to write a word processor or database for the C-128, for example, will be pleased at this capability.

The technique we will present can manage 25 lines in BASIC. This means that the other 3 lines remain when scrolling and clearing the screen and are therefore well-suited for status lines. These three lines (including attribute) can be changed with an appropriate machine language program. But first to the theory:

In register 6 of the video controller, you can specify how many lines are to appear on the screen. The default value here is 25. Let's change this value to 10:

```
10 A=DEC("D600"): D=A+1
20 POKE A,6: POKE D,10
```

You see that the controller now displays only 10 lines on the screen and the remaining lines are simply "swallowed up." Just as we can make the screen smaller, we also have the ability to increase the number of lines. We do this by simply correcting line 20:

```
20 POKE A,6: POKE D,28
```

And now we have 28 lines on the screen. You also see some lines that will usually flash in various colors. We can now (provided the monitor is good enough) see all 28 lines on the screen--even if the last three lines don't contain any useful information.

A small note: On a very well-adjusted IBM color monitor we have been able to display up to 30 lines. It wouldn't make any sense to use this though, since most monitors would not be able to display it. We have been able to display 2 or 3 additional lines on every monitor. So we can say in general that at least two additional lines are possible, which you can then use for status lines, etc.

We already know that the video RAM lies at address $0000 and the attribute RAM at address $0800. We must change this since we have displayed 2240 characters; the end of the video RAM then lies at address $0960 and part of the attribute RAM is overwritten (and vice versa). There is enough space between the attribute RAM and the character generator. Address $0A00 is then available for the start address of the attribute RAM.

But when we want to write to the 80-column screen with BASIC, we have a small problem: The interpreter gets the base address of the attribute RAM from address $0A2F in the zero page. This isn't so bad--we just inform the BASIC interpreter of the new base address. This is correct--but if we take a closer look at the kernal, we see that the base address is not added but logically ORed. Bits 0 and 1 are affected by this; these two bits may not be relevant; that is, they may not be set. This is why it is advisable to define address $1000 as the start address of the video RAM. We do this with the two instructions:

```
POKE DEC("0A2F"),16
POKE DEC("D600"),20: POKE DEC("D601"),16
```

When this is done, everything works as it should. We'll use these ideas in our next program:

```
10 REM *** DEMO PROGRAM FOR 28-LINE SCREEN **
20 :
30 A=DEC("D600"): D=DEC("D601")
40 POKE A,20: POKE D,16:REM *** VDC RECEIVES NEW
   BASE ADDRESS
50 POKE DEC("0A2F"),16: REM *** KERNAL RECEIVES NEW
   BASE ADDRESS
60 POKE A,6: POKE D,28: REM *** 28 LINES
80 PRINT CHR$(147)
```

When you start this program, 28 lines appear on the screen--though the last three lines still have no meaningful content. Unfortunately, we cannot write to these lines with the PRINT statement. The operating system is not prepared for such things. It becomes clear that we must POKE characters (strings) into memory. This is done by a small machine language routine so that the characters to be printed can be put into a string.

This machine language routine is passed the address of the string to be printed. The address of a variable can be obtained with the POINTER(var) command. Before this, the low and high bytes of the screen address at which the string is to be printed are stored in memory locations $FA (250) and $FB (251). The current attribute is used as the color or attribute which you may change. You cannot integrate any control characters in the strings. These are accepted, but result in a gap in the screen. It is possible to allow for execution of control sequences, but we have not included this feature for space reasons. The routine is intended to output strings in our new window without requiring a lot of effort on the part of the programmer. The following commands are necessary in order to display a string on the first line of our new window:

```
T$="This is a test string!"
POKE 250,(2000 AND 255)
POKE 251,(2000/256)
A=POINTER(T$)
SYS DEC("D27"),A AND 255,A/256
```

First the string variable is defined which contains the string to be printed. Then we POKE the start address in $FA and $FB, low byte first. We then indicate the address at which the string T$ is stored in bank 1. This address is then, divided into low and high bytes, passed to the output

routine at address $0D27. The routine then gets each character and outputs it. That's it. Here is the machine language program:

```
00D00  8E 00 D6 STX $D600  ;ACC. OF THE REGISTER
00D03  2C 00 D6 BIT $D600  ;TEST STATUS
00D06  10 FB    BPL $0D03  ;NO YET READY
00D08  8D 01 D6 STA $D601  ;STORE THE VALUES
00D0B  60       RTS        ;END THE ROUTINE
00D0C  A2 12    LDX #$12   ;UPDATE REGISTER HI
00D0E  A9 00    LDA #$00   ;LOAD THE HI VALUE
00D10  20 00 0D JSR $0D00  ;SET THE HI ADDRESS
00D13  E8       INX        ;UPDATE ADDRESS LO
00D14  A9 00    LDA #$00   ;LOAD THE LO-BYTE
00D16  20 00 0D JSR $0D00  ;AND THE ACCUMULATOR
00D19  A2 1F    LDX #$1F   ;DATA REGISTER OF VDC
00D1B  A9 00    LDA #$00   ;LOAD THE POKE VALUE
00D1D  20 00 0D JSR $0D00  ;SET THE VALUES
00D20  A2 12    LDX #$12   ;DUMMY VALUE
00D22  A9 00    LDA #$00   ;UPDATE ADDRESS
00D24  4C 00 0D JMP $0D00  ;SET THE VALUES
00D27  85 FC    STA $FC    ;MARK LO-BYTE OF STRING
00D29  86 FD    STX $FD    ;MARK HI-BYTE OF STRING
00D2B  A0 00    LDY #$00   ;OFFSET - STRING LENGTH
00D2D  A2 01    LDX #$01   ;BANK 1 FOR VARIABLES
00D2F  A9 FC    LDA #$FC   ;$FC WITH THE ADDRESS
00D31  20 74 FF JSR $FF74  ;AND FAR FETCH
00D34  85 FE    STA $FE    ;MARK LENGTH
00D36  A0 01    LDY #$01   ;OFFSET LOW-BYTE ADDRESS
00D38  A2 01    LDX #$01   ;BANK 1 FOR VARIABLES
00D3A  A9 FC    LDA #$FC   ;$FC WITH THE ADDRESS
00D3C  20 74 FF JSR $FF74  ;FAR FETCH
00D3F  48       PHA        ;LO-BYTE OF STACK
00D40  C8       INY        ;POINTER OF HI-BYTE
00D41  A2 01    LDX #$01   ;ADDRESS
00D43  A9 FC    LDA #$FC   ;FOR VARIABLE
00D45  20 74 FF JSR $FF74  ;FAR FETCH
00D48  85 FD    STA $FD    ;MARK THE HI-BYTE
00D4A  68       PLA        ;GET LO-BYTE
00D4B  85 FC    STA $FC    ;STORE THE LO-BYTE
00D4D  A5 FC    LDA $FC    ;GET LO-BYTE
00D4F  D0 02    BNE $0D53  ;WHEN NOT NULL;DECREMENT
00D51  C6 FD    DEC $FD    :THE LO-BYTE, ELSE DEC
00D53  C6 FC    DEC $FC    ;ALSO THE HI-BYTE
```

```
00D55  A5 FA     LDA $FA     ;ALSO THE SOURCE ADDRESS
00D57  D0 02     BNE $0D5B   ;DECREMENT THE LO-BYTE
00D59  C6 FB     DEC $FB     ;AND DEC
00D5B  C6 FA     DEC $FA     ;ALSO HI-BYTE
00D5D  A5 FA     LDA $FA     ;GET LO-BYTE
00D5F  85 E0     STA $E0     ;LO-BYTE LINE ADDRESS
00D61  A5 FB     LDA $FB     ;GET HI-BYTE
00D63  85 E1     STA $E1     ;HI-BYTE LINE ADDRESS
00D65  A2 01     LDX #$01    ;BANK 1 FOR VARIABLES
00D67  A4 FE     LDY $FE     ;POSITION IN STRING
00D69  A9 FC     LDA #$FC    ;ADDRESS IN ZERO PAGE
00D6B  20 74 FF  JSR $FF74   ;FAR FETCH
00D6E  A4 FE     LDY $FE     ;GET POSITION IN STRING
00D70  84 EC     STY $EC     ;ALSO CURSOR COLUMN
00D72  20 0C C0  JSR $C00C   ;AND CHARACTER OUTPUT
00D75  C6 FE     DEC $FE     ;DEC THE POINTER
00D77  D0 E4     BNE $0D5D   ;IF NOT END OF STRING
00D79  60        RTS         ;END ROUTINE
```

At first glance the routine may appear rather long, but it really isn't. Remember that this routine and a few short BASIC lines give you three additional lines to use. Furthermore, there is another short routine at the start of this one that writes a character to a location in the VDC memory. The BASIC loader for this routine is found after the example program. Here is the example program, which allows displays 28 lines using both of the new routines.

```
 10 REM *** DEMO PROGRAM FOR 28 LINE SCREEN ***
 20 :
 30 A=DEC("D600"): D=DEC("D601")
 40 POKE A,20: POKE D,16: REM *** VDC GETS NEW BASE
 50 POKE DEC("0A2F"),16:  REM *** KERNAL GETS NEW
                                  BASE ADDRESS
 60 POKE A,7: POKE D,28:  REM *** 28 LINES
 70 POKE A,6: POKE D,33:  REM *** NEW SYNC
 80 :
 90 PRINT CHR$(147);
100 T$+"                          ": REM 20 SPACES
110 FOR X=0 TO 79 STEP 20: FOR Y=0 TO 2
120 GOSUB 1000: NEXT: NEXT
130 INPUT "Enter your name: ";T$
140 FOR Y=0 TO 2: X=2*Y: GOSUB 1000: NEXT
150 END
```

```
1000 REM *** OUTPUT T$ AT X,Y COORDINATE; Y=0 MEANS
             1ST LINE ***
1010 AZ=2000+Y*80+X: REM DESTINATION ADDRESS
1020 POKE 250,AZ AND 255: REM LOW BYTE
1030 POKE 251,AZ/256: REM HIGH BYTE
1040 T%=POINTER(T$): REM ADDRESS OF THE STRING
1050 SYS DEC("D27"),T% AND 255,T%/256: REM PASS
1060 RETURN
1070 :
```

This program first enables the three additional three lines (lines 30-70). Then the window is cleared and the name you entered is printed on each line.

If you don't want to enter the machine language program with the assembler, you can use the following BASIC loader and then save the machine language program on disk as a BINary file.

```
10 REM BASIC LOADER FOR PRINT STRING
20 :
30 FOR I= DEC("D00") TO DEC("D79")
40 READ A$
50 POKE I, DEC(A$)
60 S=S+DEC(A$)
70 NEXT
80 IF S<>16613 THEN PRINT"ERROR IN DATA STATEMENTS"
90 INPUT "SAVE PROGRAM ON DISKETTE Y/N";A$
100 IF A$<>"Y" THEN END
110 INPUT "FILE NAME";F$
120 BSAVE""+ F$ +"",B1,P3328 TO P3449 :END
130 :
200 DATA 8E,00,D6,2C,00,D6,10,FB,8D,01,D6,60,A2,12,A9,00
210 DATA 20,00,0D,E8,A9,00,20,00,0D,A2,1F,A9,00,20,00,0D
220 DATA A2,12,A9,00,4C,00,0D,85,FC,86,FD,A0,00,A2,01,A9
230 DATA FC,20,74,FF,85,FE,A0,01,A2,01,A9,FC,20,74,FF,48
240 DATA C8,A2,01,A9,FC,20,74,FF,85,FD,68,85,FC,A5,FC,D0
250 DATA 02,C6,FD,C6,FC,A5,FA,D0,02,C6,FB,C6,FA,A5,FA,85
260 DATA E0,A5,FB,85,E1,A2,01,A4,FE,A9,FC,20,74,FF,A4,FE
270 DATA 84,EC,20,0C,C0,C6,FE,D0,E4,60
```

5.5.7 Hi-res graphics

We probably got you excited when we mentioned that a graphics display is also possible on the 80-column screen. The resolution of these graphics is 640x200 points, exactly twice as great as the hi-res mode of the VIC chip. There is no multi-color mode. The brillance of the graphics is quite impressive (if the monitor can display it properly). Here you don't have to set two points next to each other in order to see one point, as on the VIC. There is "only" one color available, but this is completely sufficient for most graphics (such as mathematical curves).

This graphic mode is not supported by the BASIC 7.0 graphics commands. We again offer you a small machine language package that can perform the following elementary functions:

* turn graphic mode on and off
* clear the graphic page
* set and clear points

We could have integrated more features into the machine language routine package, but we don't want to turn the *C-128 Internals* into a collection of programs!

The *how* of the VDC graphic mode is also interesting. The bit-map mode is enabled by setting bit 7 of register 25. There are then 16Kbytes of the VDC memory available for graphics on the screen. If you clear the graphics, the character generator is also cleared.

On the international models of the C-128 if you exit with <RUN/STOP> <RESTORE>, you must also press <ASCII/DIN> or you will see nothing on the screen because the character set has been erased. The character set can also be copied under program control when switching from the graphic mode to the text mode. You can also press <ASCII/DIN> while the graphic mode is enabled--you will be surprised.

The graphic mode is enabled by setting bit 7. The attribute RAM becomes nonfunctional as it is required for graphic display, we must also clear the ATR bit in register 25. We can combine these two actions by loading register 25 with 128. This is all that is necessary to enable the graphic mode. We can leave the attribute and video RAM addresses alone since they play no role.

The graphic memory is defined at address $0000. The logic for setting and clearing points is similar to that described for the VIC chip; here setting and clearing are accomplished through logical OR and AND. One byte also defined eight points (pixels) for the VDC. The first point, which has the coordinates 0/0, is located in the upper left-hand corner, and thereby at address $0000. The rest of the procedure is simpler than for the VIC chip. The graphics are defined line by line. The memory layout is clarified in the following figure:

```
$0000 $0001 $0002 $0003 ..... $027F (639 decimal)
$0280 $0281 $0282 $0283 ..... $04FF (1279 decimal)
  .     .     .     .     .     .          .
  .     .     .     .     .     .          .
```

On the VDC the memory is not divided into matrices of eight, so that addressing a point is much easier. The following formula is needed to address a given point (X/Y):

$$AD = INT(X/8) + Y*802$$

The point in this byte is addressed in the same manner as with the VIC, by the following formula:

$$2\text{^}(7-(X \text{ AND } 7))$$

Since this addressing is so simple, the machine language program is correspondingly shorter. First the assembly language listing, followed by the BASIC loader:

```
00C00   4C CD 0C JMP $0CCD  ;SWITCH ON THE GRAPHICS
00C03   4C D0 0C JMP $0CD0  ;TURN OFF GRAPHICS
00C06   4C D3 0C JMP $0CD3  ;BACK TO TEXT MODE
00C09   4C E0 0C JMP $0CE0  ;SET A POINT
00C0C   4C DD 0C JMP $0CDD  ;ERASE A POINT
00C0F   8E 00 D6 STX $D600  ;STORE IN REGISTER
00C12   2C 00 D6 BIT $D600  ;TEST STATUS
00C15   10 FB    BPL $0C12  ;NOT FINISHED YET
00C17   8D 01 D6 STA $D601  ;STORE VALUE
00C1A   60       RTS        ;RETURN TO PROGRAM
00C1B   8E 00 D6 STX $D600  ;LOAD REGISTER
00C1E   2C 00 D6 BIT $D600  ;TEST STATUS
00C21   10 FB    BPL $0C1E  ;NOT FINISHED YET
00C23   AD 01 D6 LDA $D601  ;GET REGISTER VALUE
```

```
00C26  60        RTS          ;RETURN TO PROGRAM
00C27  A2 19     LDX #$19     ;REGISTER 25 CHOSEN
00C29  A9 80     LDA #$80     ;BIT 7 SET
00C2B  20 0F 0C  JSR $0C0F    ;REGISTER 25 SET
00C2E  A0 40     LDY #$40     ;$40 FOR OFF
00C30  A2 12     LDX #$12     :REGISTER 18 UPDATE HI
00C32  98        TYA          ;HI BYTE TO ACCU.
00C33  20 0F 0C  JSR $0C0F    ;SET UPDATE HI
00C36  A2 1F     LDX #$1F     ;REGISTER 31 DATA REG.
00C38  A9 00     LDA #$00     ;
00C3A  20 0F 0C  JSR $0C0F    ;DATA REGISTER WRITTEN
00C3D  A2 1E     LDX #$1E     ;WORDCOUNT REGISTER
00C3F  20 0F 0C  JSR $0C0F    ;WITH NO FILL
00C42  88        DEY          ;DECREMENT THE NUMBER
00C43  10 EB     BPL $0C30    ;FOLLOW BLOCK OFF
00C45  60        RTS          ;RETURN TO OFF ROUTINE
00C46  08        PHP          ;RETURN CARRY # SET/OFF
00C47  A5 FA     LDA $FA      ;LO-BYTE X-COORD.
00C49  85 FE     STA $FE      ;TEMP. STORAGE
00C4B  46 FB     LSR $FB      ;HI-BYTE WITH X OVER TWO
00C4D  66 FA     ROR $FA      ;COPY CARRY LOW-BYTE
00C4F  46 FB     LSR $FB      ;S.O.
00C51  66 FA     ROR $FA      ;S.O.
00C53  46 FB     LSR $FB      ;PUT TOGETHER INT(X/8)
00C55  66 FA     ROR $FA      ;
00C57  A9 00     LDA #$00     ;HI-BYTE  OF ADDRESS ON
00C59  85 FD     STA $FD      ; NULL SET
00C5B  A5 FC     LDA $FC      ;Y-COORD. IN ACC.
00C5D  06 FC     ASL $FC      ;Y TIMES 2
00C5F  26 FD     ROL $FD      ;COPY CARRY
00C61  06 FC     ASL $FC      ;TIMES TWO OPTION
00C63  26 FD     ROL $FD      ;AMT * 4, PLUS 1*Y
00C65  65 FC     ADC $FC      ;OPTION Y*5
00C67  85 FC     STA $FC      :LO-BYTE
00C69  90 02     BCC $0C6D    ;NO CARRY
00C6B  E6 FD     INC $FD      ;CARRY INTO HI-BYTE
00C6D  A2 04     LDX #$04     ;IS WORD WITH 4 TIMES
00C6F  06 FC     ASL $FC      ;WITH 2 MULTIPLER THIS
00C71  26 FD     ROL $FD      ;OPTION ONE * 16
00C73  CA        DEX          ;AND 16*5 FOR 80 OPTION
00C74  D0 F9     BNE $0C6F    ;WITH 80 MULTIPLER
00C76  A5 FA     LDA $FA      ;INT(X/8)
00C78  65 FC     ADC $FC      ;ADD TO Y*80
```

```
00C7A  85 FC     STA $FC     ;AND STORE
00C7C  90 02     BCC $0C80   ;NO CARRY
00C7E  E6 FD     INC $FD     ;REM CARRY
00C80  A2 12     LDX #$12    ;REGISTER 18 UPDATE HI
00C82  A5 FD     LDA $FD     ;HI-BYTE OF ADDRESS
00C84  20 0F 0C  JSR $0C0F   ;SET
00C87  E8        INX         ;UPDATE LO
00C88  A5 FC     LDA $FC     ;LO-BYTE OF ADDRESS
00C8A  20 0F 0C  JSR $0C0F   ;SET THE LO-BYTE
00C8D  A2 1F     LDX #$1F    ;DATA REGISTER
00C8F  20 1B 0C  JSR $0C1B   ;GET THE STORED VALUE
00C92  48        PHA         ;STACK
00C93  A5 FE     LDA $FE     ;GET X-COORD. (LO)
00C95  29 07     AND #$07    ;X AND 7
00C97  AA        TAX         ;POINTER NOT X
00C98  68        PLA         ;GET VALUE BACK
00C99  28        PLP         ;GET CARRY BACK
00C9A  B0 05     BCS $0CA1   ;SET  POINT
00C9C  3D C5 0C  AND $0CC5,X;CLEAR POINT
00C9F  90 03     BCC $0CA4   ;UNCONDITIONAL JUMP
00CA1  1D BD 0C  ORA $0CBD,X;SET POINT
00CA4  48        PHA         ;STACK
00CA5  A2 12     LDX #$12    ;UPDATE HI
00CA7  A5 FD     LDA $FD     ;HI-BYTE OF LINE ADDRESS
00CA9  20 0F 0C  JSR $0C0F   ;SET THE VALUE
00CAC  E8        INX         ;UPDATE LO
00CAD  A5 FC     LDA $FC     ;LO-BYTE OF ADDRESS
00CAF  20 0F 0C  JSR $0C0F   ;SET THE LO-BYTE
00CB2  A2 1F     LDX #$1F    ;DATA REGISTER
00CB4  68        PLA         ;RECOVER STACK
00CB5  20 0F 0C  JSR $0C0F   ;SET NEW VALUE
00CB8  A2 12     LDX #$12    ;UPDATE ADDRESS HI
00CBA  4C 1B 0C  JMP $0C1B   ;AND POINT SET
00CBD  80 40 20 10 08 04 02 01;  TABLE SETTING PTS
00CC5  7F BF DF EF F7 FB FD FE;  TABLE CLEAR POINTS
00CCD  20 27 0C  JSR $0C27   ;SET THE GRAPHIC MODE
00CD0  4C 2E 0C  JMP $0C2E   ;TURN OFF GRAPHICS
00CD3  A2 19     LDX #$19    ;REGISTER 25 SELECT
00CD5  A9 40     LDA #$40    'ATR-BIT SET,TXT-BIT OFF
00CD7  20 0F 0C  JSR $0C0F   ;SET THE TEXT MODE
00CDA  4C 0C CE  JMP $CE0C   ;COPY CHAR ROM
00CDD  18        CLC         ;CLR CARRY FOR POINT OFF
00CDE  90 01     BCC $0CE1   ;UNCONDITIONAL JUMP
```

```
00CE0   38          SEC          ;SET CARRY FOR POINT SET
00CE1   85 FA       STA $FA      ;STORE X-LOW
00CE3   86 FB       STX $FB      ;STORE X-HI
00CE5   84 FC       STY $FC      ;STORE Y-COORD.
00CE7   4C 46 0C    JMP $0C46    ;POINT SET/CLEAR
```

As you see, there are five entry points available. The graphic page is automatically cleared when the graphic mode is enabled. If you only want to enable the graphic page, you can do this with the following BASIC commands:

POKE DEC("D600"),25: POKE DEC("D601"),128

The following subroutines are reached with the five entry point addresses:

$0C00	Enable and clear graphic page
$0C03	Clear the graphics
$0C06	Back to text mode
$0C09	Set a point
$0C0C	Clear point

The coordinates for setting or clearing a point can be passed directly with the SYS command. The syntax looks like this:

SYS <ENTRY POINT>,<X LOW>,<X HIGH>,<Y>

For example, the command

SYS DEC("0C09"),0,185,191

is necessary to set the point with the coordinate (185,191). The general call looks like this:

SYS DEC("0C09"),X AND 255,X/256,Y

By the way, it pays to append the % sign to the variable names whenever possible because then the variable is treated as an integer variable--leading to great increases in speed. Unfortunately, this doesn't work for loop variables. The constants 255 and 256 should be defined as integer variables--this also increases the speed because the values do not have to be recalculated by the interpreter each time. We have made use of this in our example program.

Here is the BASIC loader for the graphics package:

```
10 REM *** BASIC LOADER FOR 80 COLUMN GRAPHICS***
20 :
30 FOR I= DEC("0C00") TO DEC("0CE9")
40 : READ X$:X=DEC(X$)
50 : POKE I,X
60 : S=S+X
70 NEXT
80 IF S<> 25905 THEN PRINT"***** ERROR IN DATA
    STATEMENTS *****"
90 INPUT"SAVE PROGRAM TO DISKETTE";A$
100 IF A$<>"Y" THEN END
110 PRINT:INPUT "FILE NAME";F$
120 BSAVE""+F$+"",B0,P3072 TO P3306
130 END
140 :
1000 DATA 4C,CD,0C,4C,D0,0C,4C,D3,0C,4C,E0,0C,4C,DD,0C,8E
1010 DATA 00,D6,2C,00,D6,10,FB,8D,01,D6,60,8E,00,D6,2C,00
1020 DATA D6,10,FB,AD,01,D6,60,A2,19,A9,80,20,0F,0C,A0,40
1030 DATA A2,12,98,20,0F,0C,A2,1F,A9,00,20,0F,0C,A2,1E,20
1040 DATA 0F,0C,88,10,EB,60,08,A5,FA,85,FE,46,FB,66,FA,46
1050 DATA FB,66,FA,46,FB,66,FA,A9,00,85,FD,A5,FC,06,FC,26
1060 DATA FD,06,FC,26,FD,65,FC,85,FC,90,02,E6,FD,A2,04,06
1070 DATA FC,26,FD,CA,D0,F9,A5,FA,65,FC,85,FC,90,02,E6,FD
1080 DATA A2,12,A5,FD,20,0F,0C,E8,A5,FC,20,0F,0C,A2,1F,20
1090 DATA 1B,0C,48,A5,FE,29,07,AA,68,28,B0,05,3D,C5,0C,90
1100 DATA 03,1D,BD,0C,48,A2,12,A5,FD,20,0F,0C,E8,A5,FC,20
1110 DATA 0F,0C,A2,1F,68,20,0F,0C,A2,12,4C,1B,0C,80,40,20
1120 DATA 10,08,04,02,01,7F,BF,DF,EF,F7,FB,FD,FE,20,27,0C
1130 DATA 4C,2E,0C,A2,19,A9,40,20,0F,0C,4C,0C,CE,18,90,01
1140 DATA 38,85,FA,86,FB,84,FC,4C,46,0C
```

This routine is located in the RS-232 input buffer and can therefore be called from any bank configuration. This memory area was chosen because it is seldom used. If you do need it, you must move the routine to a new area and make the appropriate changes to the program.

In conclusion, we do not want to leave you with the graphics package alone, we we wrote a short example program in BASIC which draws a damped oscillation on the 80-column screen. We find that the execution speed is quite satisfactory. You can also learn more about the operation of

the graphic routines from the example program. Naturally you can change the function in line 30 to see what "your" function looks like.

```
 10 REM ** EXAMPLE PROGRAM FOR GRAPHICS PACKAGE **
 20 :
 30 DEFFNR(X)=40*SIN(X)*EXP(-0.5*X)+100
 40 FAST: TRAP 1000: REM IN CASE OF ERROR IN FNR(X)
 50 F%=256: FF%=255: SE=DEC("C09"): RE=DEC("C0C")
 60 SYS DEC("C00"): REM GRAPHICS ON
 70 Y%=100: REM DRAW X-COORDINATE
 80 FOR X=0 TO 639 STEP 3: REM DOTTED LINE
 90 : SYS SE,X AND FF%, X/F%, Y%
100 NEXT
110 X%=320: REM DRAW Y-COORDINATE
120 FOR Y=0 TO 199 STEP 2 : REM DOTTED LINE
130 : SYS SE,X% AND FF%, X%/F%, Y
140 NEXT
150 C=-32
160 FOR X=0 TO 639
170 : FU%=FNR(C): IF FU%<0 OR FU%>199 THEN 190
180 : SYS SE,X AND FF%, X/F%, FU%
190 C=C+.1
200 NEXT
210 GETKEY A$: REM *** DONE, WAIT FOR KEY, BACK TO
    TEXT
220 SYS DEC("C06"): PRINT CHR$(147): SLOW
1000 PRINT ERR$ (ER);EL
```

There are an unlimited number of applications for graphics. We will let your imagination run free here. We wish you much success with the use of the 80 column graphics routines.

CHAPTER 6

Chapter 6: The Memory Management Unit

6.1 Introduction to the MMU

The Memory Management Unit (MMU) was designed to handle the complex addressing tasks in the C-128. As you may know, the 8502 and the Z-80 can address only 64K. You know from BASIC that the two RAM banks can only be addressed separately. Each 64K of RAM overlays the ROM and the I/O components. For example, there are two different RAMs at address $D600, the I/O provided by the 80-column controller and the ROMs. If a cartridges is inserted into the expansion slot, the MMU must differentiate this too.

The MMU is also used in the 64 mode and is completely compatible with the C-64. In addition it can handle the tasks that come up in the C-128 and CP/M modes. It also performs the computer mode selection and selects between the 8502 and the Z-80. Here is a list of its features:

* Manages the translated address bus (TA8-TA15)
* Selects the computer mode (C-64, C-128, CP/M)
* Selects the processor (Z-80, 8502)
* Prepares and manages the CAS selection lines for bank-switching the RAM.

The MMU has a total of 11 registers that are found starting at address $D500. Since the I/O range is not always enabled, the memory configuration register and load registers A-D are copied into the memory range $FF00 to $FF05. This way there are four set configurations found in the preconfiguration registers A-D. They can be selected simply by loading a load register into the configuration register, without having to enable the I/O range. This is a very useful feature and saves both time and programming effort. But more about this later.

Here is a graphic representation of the available registers:

$FF04	LCRD	Load Configuration Register D
$FF03	LCRC	Load Configuration Register C
$FF02	LCRB	Load Configuration Register B
$FF01	LCRA	Load Configuration Register A
$FF00	CR	CONFIGURATION REGISTER (Copy at $D501)
$D50B	VR	Version Register
$D50A	P1H	Page 1 Pointer -High
$D509	P1L	Page 1 Pointer-Low
$D508	POH	Page 0 Pointer-High
$D507	POL	Page 0 Pointer-Low
$D506	RCR	RAM Configuration Register
$D505	MCR	Mode Configuration Register
$D504	PCRD	Preconfiguration Register D
$D503	PCRC	Preconfiguration Register C
$D502	PCRB	Preconfiguration Register B
$D501	PCRA	Preconfiguration Register A
$D500	CR	CONFIGURATION REGISTER (Copy at $FF00)

6.2 The Configuration Register

As already mentioned, there is a copy of some of the MMU registers at address $FF00 (independent of the enabled RAM bank). This is not quite correct. In reality there is a copy of *one* register at address $FF00; this is the configuration register CR. If you read memory location $FF00, you get the current contents of the configuration register. If you write to address $FF00, the contents of the configuration register at $D500 in the MMU change at the same time. The registers $FF01 to $FF04 are just "half" copies of the MMU registers. Half because when reading them they return the current contents of the corresponding MMU preconfiguration register, but when writing to these registers, the contents not of the corresponding MMU registers, but the configuration register is changed.

This is not a disadvantage--quite the opposite. If you write to an LCRx register, the CR will be loaded with the corresponding PCR. An example: We write to LCRA at address $FF01. The contents of this register doesn't change, but the contents of the CR does. The PCRA ($D501) is copied to the CR. This is a very practical feature: We can change the CR register without having to bother with the I/O range. We can select between four configurations stored in the MMU. This means the programmer need only say, "Select configuration #1," and the MMU switches this configuration on. In machine language this selection looks simply like this:

```
STA $FF01 ;Acc. contents irrelevant--enable
           configuration 1
```

At the start of a program one can pre-program the most-used configurations into the four PCRs. But "manual" reconfiguration isn't much harder. Load the accumulator with the bit pattern necessary and store this at address $FF00. Example for bank 15:

```
LDA #00   ;corresponds to BANK 15 command
STA $FF00 ;select configuration
```

All eight bits of the configuration register are relevant:

Bits 7,6 Select RAM bank. The bit combinations 00 and 01 are possible in the 128K version. But since memory expansion up to 256K is allowed, the possibilities 10 and 11 exist for this expansion. If these RAM banks are not present, 10 means the same as 00 and 11 the same as 01.

Bits 5,4 Select what will be accessed when the memory range $C000 to $FFFF is addressed:
00 - System ROM (kernal)
01 - Internal function ROM
10 - External function ROM
11 - RAM (bank, see bits 6 and 7)

Bits 2,3 Select what will be accessed when the memory range $8000 to $BFFF is addressed:
00 - System ROM (BASIC)
01 - Internal function ROM
10 - External function ROM
11 - RAM (bank, see bits 6 and 7)

Bit 1 Select what will be accessed when the memory range $4000 to $7FFF is addressed:
0 - System ROM (BASIC)
1 - RAM (bank, see bits 6 and 7)

Bit 0 Select what will be accessed when the memory range $D000 to $DFFF is addressed:
0 - System I/O
1 - RAM/ROM, dependent on bits 4 and 5

It should be noted that when ROM is enabled in the range $C000 to $FFFF (bits 4 and 5) there is always a gap in the range $D000 to $DFFF. If the system I/O is enabled, the system I/O components occupy this range. If bit 0 is set, the character generator is found here.

6.2.1 The preconfiguration registers

The preconfiguration registers are found at addresses $D501 to $D504 and copies of them are found at addresses $FF01 to $FF04. They have no significance in the C-64 mode. Preconfiguration registers are registers in the MMU which can be pre-programmed with specific memory configurations. These pre-programmed configurations can be transported to the configuration register by means of the "Load Configuration Register".The use of these registers was described in section 6.2. The bits are encoded in the same manner as for the configuration register. The encoding is also found in section 6.2.

6.3 The Mode Configuration Register

The mode configuration register is found at address $D505. It sets the current computer mode, that is, which CPU is operational (8502 or Z-80) and whether the 64 or 128 mode is active.

Bit 7 Indicates if the 40/80 column key was pressed at reset. 0=80 column, 1=40 column mode.

Bit 6 This bit indicates whether the 64 or 128 mode is active; 0=128 mode. After power-up or RESET the 128 mode is always active.

Bits 4,5 These two bi-directional bits indicates whether or not the cartridge lines GAME or EXROMIN are being used. If so, the 64 mode must be enabled and control passed to the cartridge. In the 128 mode these lines are not used.

Bit 3 FSDIR control bit. This bit is used as the output bit for the fast serial data bus buffer as well as the input bit for the disk enable signal.

Bits 1,2 These bits have no significance.

Bit 0 This bit selects the processor; 1=Z-80, 0=8502.

If bit 0 of this register is written to, the contents are temporarily buffered until the current clock cycle is done. Otherwise, complications could occur when the processors are switched.

By looking at bit 0 we can determine whether the Z-80 or the 8502 is operational. When writing to this register, the bit is stored until the clock pulse falls. If the bit is set, the Z-80 is active and the range $D000 to $DFFF is mirrored in the range $0000 to $0FFF. The BIOS ROM is also physically located at the range $0000 to $0FFF. The BIOS ROM can't be read (via software) when the 8502 is enabled.

For example, if the Z-80 is enabled, the 8502 is stopped and the Z-80 continues where it left off. This simply means that the PC (Program Counter) continues with the course of the program. The same is true when the 8502 is switched on: it picks up its work where it left off and this takes place immediately after the instruction to switch on the Z-80.

In the 64 mode the Z-80 enable line (defined by bit 0) is always zero so that the Z-80 mode cannot be enabled in the C-64 mode. Furthermore, there are no copies of the MMU registers in the addresses at $FF00 in the 64 mode.

6.4 The RAM Configuration Register

The RAM configuration register is found at address $D506 of the MMU. It is used to define the common RAM areas. But why define common RAM areas?

Quite simple: The interpreter, for example, stores all of the required system variables and pointers in the zero page (although there really isn't a zero page anymore). If the interpreter now switches to bank 1, for instance, in order to read or write variables, these system pointers would no longer be available since they are found in bank 0. It would be a lot of bother to have to make changes in both memory banks every time a zero-page location was changed.

To avoid this, the Commodore engineers thought it would be very practical to be able to define a certain memory range so that it looked the same in all RAM banks. In reality, the zero page is stored in only one RAM bank, bank 0. If you then address this memory range in RAM bank 1 (or another bank), the MMU recognizes this and addresses the corresponding area in bank 0.

These common memory ranges are called *common areas*. The MMU offers the programmer the option of defining whether or not he wants a common area, and if so, how large it should be and where it should be located. But first the register layout before we take a closer look at the individual bits:

Bits 6,7 These two bits store the RAM bank for the VIC chip, where the text or a graphic page will be stored. Normally the video RAM is found in bank 0.
00=RAM bank 0, 01=RAM bank 1, 10=RAM bank 2(0), 11=RAM bank 3(1)

Bits 4,5 These two bits are still unused in the present version. They are intended for expansion to 1Mbyte of RAM. Then selecting these would address a 256K block.

Bits 2,3 Bits 2 and 3 indicate if and where a common area is defined.
00=no common area, independent of bits 0 and 1
01=lower area is common
10=upper area is common
11=both upper and lower areas are common

Bits 0,1 Here is defined how many Kbytes will be used as a common area. These two bits are valid only when bits 2 and 3 are not equal to 00.
00=1 Kbyte common area
01=4 Kbyte common area
10=8 Kbyte common area
11=16 Kbyte common area

When a common area is defined, the minimum area possible is 1K. But is also possible to declare no area as common. To do this, set bits 2 and 3 to zero. If only one of bits 0 and 1 are set, bits 4 and 5 will have effect. Normally, only the lower area with 1Kbyte is defined as a common area. In the 64 mode, this register has no effect.

If a 1Kbyte area is defined as a common area, the range $0000 to $03FF is identical in both RAM banks if the lower area is enabled. If both the upper and lower areas are enabled as the common area, the ranges $0000 to $03FF and $FC00 to $FFFF are identical in both RAM banks. You can define up to 32K as a common area by defining both areas and 16K as the common area.

Bits 6 and 7 determine from which RAM bank the VIC chip should get its information regarding the video RAM. This offers us fantastic capabilities. It is very easy to manage two 40-column screens without having to move the address of the video RAM, which is more complicated than switching the RAM bank. In RAM bank 0 you can define screen number 1 at address $0400 and screen 2 at the same address in bank 1. You can then switch between the two by setting or clearing bit 6.

```
LDA #00    ;system I/O
STA $FF00  ;enable
LDA $D506  ;old RCR value
ORA #$40   ;screen in RAM bank 1
STA $D506  ;enable
```

6.5 The Page Pointer

There are two page pointers: one page pointer for the zero page, and a page pointer for page 1, in which the stack normally lies.

$D507/$D508: Page pointer 0
$D509/$D50A: Page pointer 1

The low-order byte of these pointers represents the address bits 8 to 15. The high-order byte determines the RAM bank which will be used, representing address bits 16 to 19. Bits 7-4 are not used in the high-order byte.

If the high-order byte of a page pointer is written, it is stored temporarily until the low-order byte of the pointer is also written.

If the zero page or page 1 is moved to another address, the MMU adds the base address automatically to access the zero page or for stack actions.

Higher bytes ($D508/$D50A)

Bits 7-4 unused
Bits 3-0 Address bits 16 to 19 for translated address (TA)
In the present version, only bit 0 is of interest; the remaining bits 1-3 are ignored.

Lower byte ($D507/$D509)

Bits 0-7 These bits represent the high-order byte of the page pointer, that is, address bits 8-15. For the zero-page pointer this byte is usually 0; for the page-1 pointer it is 1.

The advantages are clear. You can have a separate stack for each subroutine as well as a separate system-variable area if you don't call the kernal routines. Moving the zero page has two advantages: Programs will be shorter and faster.

Assembly language programmers are often searching for free memory locations in the zeropage. As an example, the `LDA ($xx),Y` instructions function only with zero-page addresses. Using the page pointers you can move zero page to a free memory area.

The ability to move page 1 is also practical. This makes it possible to give each subroutine its own stack. This is a very useful feature. You need only save the stack pointer and then have a new stack available for the subroutine. This results in more space on the stack, and the stack need not be completely reconstructed when the routine is exited. You need only to restore the page 1 pointer to the normal value ($01) and reset the stack pointer SP. This is a particularly useful feature for PASCAL compilers.

Moving the stack might look something like this:

```
LDA #$00    ;system I/O
STA $FF00   ;enable
LDA #$F0    ;stack at address $F000
STA $D507   ;in RAM bank 0
TSX         ;and save SP
STX $FD     ;in zero-page $FD
LDX #$FF    ;initialize
TXS         ;new stack
```

Since the stack has been redefined, the stack must be reconstructed the at the end of the routine, otherwise it is no longer possible to exit from the subroutine with RTS. This reconstruction looks like this:

```
LDX $FD     ;get old stack pointer
TXS         ;and reset SP
LDA #$01    ;stack again at address $0100
STA $D507   ;default value
RTS         ;return now possible
```

Here is a rather unconventional way to clear the screen. It is used often in professional programs because it is very fast. It is used in graphics programs for filling surfaces, for example.

The whole thing is done by "misusing" the stack pointer for addressing. A PHA instruction writes the contents of the accumulator to the current stack address and the stack address is automatically decremented--all of this in a one-byte command. This is much faster since it's all done in hardware. In the "normal" assembler notation this looks like this:

```
STA ($xx),Y
DEY
```

The addressing mode is more complicated for the CPU, so it needs more time. The same action requires three bytes, and it is slower since the code must be fetched, interpreted, and executed.

Our new screen-clear routine saves the stack pointer, places it at the screen start $0400, and then pushes the accumulator onto the new stack 1024 times. After each 256 bytes the high-order byte must naturally be incremented. The interrupts should also be disabled during the routine in order to prevent stack overflow.

```
        LDA #$00   ;BANK 15
        STA $FF00
        SEI        ;DISABLE INTERRUPTS
        LDA #$04   ;NEW START ADDRESS OF THE SP
        STA $D509  ;IS $0400 IN RAM BANK 0
        TSX        ;STACK POINTER TO X
        STX $FD    ;AND SAVE CURRENT POINTER
        LDX #$FF   ;SP TO START OF STACK
        TXS
        LDY #$03   ;256 BYTES TIMES 4
        LDX #$00   ;256-BYTE COUNTER
        LDA #$20   ;FILL CHARACTER
NEXT    PHA        ;SAVE CHARACTER
        DEX        ;DECREMENT LOOP
        BNE NEXT   ;NEXT CHARACTER
        INC $D509  ;INCREMENT SP HIGH BYTE
        DEY        ;ALL FOUR BLOCKS FILLED?
        BNE NEXT   ;NO, NOT YET
        LDX $FD    ;GET OLD SP
        TXS        ;AND STORE AGAIN
        LDA #$01   ;HIGH BYTE OF ORIGINAL STACK
        STA $D509  ;AND SET
        CLI        ;REENABLE INTERRUPTS
        RTS        ;END OF THE CLEAR ROUTINE
```

This routine isn't much longer than a "traditional" screen-clear routine and it is much faster. It also demonstrates the capabilities that are possible by changing the page-pointer base addresses.

6.6 The Version Register

Bits 7-4 Bank version; These bits give information as to the total available memory space. As already mentioned, the computer has the possibility to expanded to 1Mbyte. The standard contents of this register for the 128 are $20. The "2" stands for two 64K blocks. A 1M version would contain sixteen 64K blocks. Bits 7-4 of this register would then contain a 0.

Bits 3-0 MMU version; These bits indicate the version number of the MMU.

The system version register is quite uninteresting for actual memory management. The low-order nibble contains a specification of the MMU version. In the high-order nibble the existing memory construction can be read. Here programs can determine how much memory they can access and set themselves accordingly. Future programs will undoubtedly do this.

CHAPTER 7

Chapter 7: Assembly Language Programming

7.1 Introduction to Assembly Language Programming

We hardly need to explain to an *Internals* reader what assembly or machine language is. This chapter is designed to show you how to use the operating system routines in your own programs. Why keep reinventing the wheel? There is a whole set of useful routines available which can be very easily accessed. This makes your programs shorter and easier to read.

We want to make the work easier for you and explain the kernal routines by means of short examples. Naturally, we cannot go into all of the kernal routines; there are simply too many.

7.2 The CPU - The 8502

The heart of a computer is the CPU and in the C-128 it's the 8502, in addition to the Z-80. It is fully software-compatible to the 6510 and its predecessor, the 6502. In contrast to the 6510, the 8502 can be driven at 2MHz--making it twice as fast.

The pinout:

1 0IN System clock input; selectable 1 or 2MHz (approximately)

2 RDY Ready; 0=processor stops on the next clock cycle until RDY=1. This can be used to operate slow memory, for example.

3 -IRQ Interrupt request; 0=processor gets the next commands address from $FFFE and continues from there. This occurs only when interrupts are enabled.

4 -NMI Non-maskable interrupt; 0=processor gets the next command address from $FFFA and continues from there. This interrupt cannot be disabled.

5 AEC Address enable control; 0=processor brings data, address, and control bus to the high-Z state (tri-state). The bus can then be driven by other devices, such as a second processor.

6 VCC Operating voltage +5V.

7-20	A0-A13; Address bus.
21	GND
22-23	A14-A15; Address bus
24-29	P5-P0; I/O pins
30-37	D7-D0; data bus
38	R/-W; 0=write access, 1=read access All access occur only when 02=1.
39	02OUT; System clock output for supplying other components
40	RES Reset; 0=processor goes into the rest state. When the signal goes from 0 to 1, the processor gets an address from $FFFC executes the program at that address.

7.3 The Kernal Routines

First we like to dedicate ourselves to the routines that are found in part in the extended zero page. These include the very important routines which allow you to read from, write to, or peform a comparison with any memory location in any bank.

7.3.1 FETCH, STASH, and CMPARE

These three routines are used to read, write, and compare memory locations in any bank, regardless of the memory configuration. The configuration is unchanged after calling one of these routines.

When calling the routines, you must pass the configuration index in the X register. The operating system has 16 configurations of the 128 possible stored in a table at $F7F0.

Find the desired memory configuration from the table on the next page and load it into the X register.

X-Register	Memory Configuration
0	only RAM 0
1	only RAM 1
2	only RAM 2 (RAM 0)
3	only RAM 3 (RAM 1)
4	Int. ROM, RAM 0, I/O
5	Int. ROM, RAM 1, I/O
6	Int. ROM, RAM 2, I/O
7	Int. ROM, RAM 3, I/O
8	Ext. ROM, RAM 0, I/O
9	Ext. ROM, RAM 1, I/O
10	Ext. ROM, RAM 2, I/O
11	Ext. ROM, RAM 3, I/O
12	Kernal, Int (Lo), RAM 0, I/O
13	Kernal, Ext (Lo), RAM 1, I/0
14	Kernal, BASIC, RAM 0, CHARROM
15	Kernal, BASIC, RAM 0, I/O

7.3.1.1 FETCH

Part of the FETCH routine is found at address $02A2 in RAM. To read from a memory location, the following parameters are passed to this routine:

```
Acc   : pointer to zero-page address
X-reg : configuration index (see above)
Y-reg : offset for the address
```

Before calling FETCH, place the two byte address of the memory location to be read into a zero-page location. Then pass the address of the zero-page location into the Accumulator.

The following example program reads from address $1000 in bank 1:

```
LDA #$00  ;LOW BYTE OF $1000
```

```
LDA #$00   ;LOW BYTE OF $1000
STA $FC    ;IN ZERO PAGE
LDA #$10   ;HIGH BYTE OF $1000
STA $FD    ;IN ZERO PAGE
LDA #$FC   ;ADDRESS IN ZERO PAGE
LDY #$00   ;OFFSET IS ZERO
LDX #$01   ;SELECT RAM BANK 1
JSR $FF74  ;FETCH--RETURN IN ACC.
```

After the call the accumulator returns the contents of the memory address. The X-register contains the current configuration and the Y-register remains unchanged.

7.3.1.2 STASH

The STASH routine is essentially the opposite of the FETCH routine. Since you must pass in the accumulator the value to be stored, the accumulator can no longer be used to pass the address of the zero-page pointer. This is why you have to do "by hand" what the FETCH routine did for you automatically.

Writing to the memory address $1000 in bank RAM looks like this:

```
LDA #$00   ;LOW BYTE OF $1000
STA $FC    ;IN ZERO PAGE
LDA #$10   ;HIGH BYTE OF $1000
STA $FD    ;IN ZERO PAGE
LDA #$FC   ;ZERO PAGE ADDRESS OF THE POINTER
STA $02B9  ;WRITE TO STASH ROUTINE
LDA #$FF   ;VALUE TO BE WRITTEN
LDX #$01   ;RAM BANK 1
LDY #$00   ;OFFSET FOR ADDRESS
JSR $FF77  ;CALL STASH
```

After calling the STASH routine, the accumulator and the Y-register are unchanged; the X-register contains the current configuration.

The MMU register can be written in the same manner, without having to change the configuration. The same applies to the I/O components.

7.3.1.3 CMPARE

The CMPARE routine compares the contents of a memory location with the contents of the accumulator. To do this, you have to write the address of the memory location to be compared into the CMPARE routine before calling it. Comparing the memory location $1000 in bank 1 with the value $22 would look like this:

```
LDA #$00  ;LOW BYTE OF $1000
STA $FC   ;IN ZERO PAGE
LDA #$10  ;HIGH BYTE OF $1000
STA $FD   ;IN ZERO PAGE
LDA #$FC  ;ADDRESS OF THE PTR IN THE ZERO PAGE
STA $02C8 ;WRITE TO CMPARE ROUTINE
LDA #$22  ;VALUE TO BE COMPARED TO
LDX #$01  ;RAM BANK 1
LDY #$00  ;OFFSET
JSR $FF7A ;COMPARE ($1000) IN RAM 1 TO $22
```

After the routine has been called, the flags (zero, minus, and carry) are set according to the comparison. The accumulator and the Y-register remain unchanged, the X-register contains the current memory configuration.

7.3.2 GETCONF

This routine does nothing more than get the configuration byte from the table at $FF70 corresponding to the configuration index in the X-register. This value is simply returned; it is not set. Since the kernal ROM must be enabled in order to jump to this routine, it's recommended that you read the configuration byte from the table; it goes faster.

```
LDX #$0F  ;SELECT CONFIGURATION 15
JSR $FF6B ;GETCONF
STA $FF00 ;SET CONFIGURATION
```

would be the same as:

```
LDX #$0F  ;SELECT CONFIGURATION 15
LDA $F7F0,X ;GET CONFIGURATION BYTE
STA $FF00 ;SET CONFIGURATION
```

The length of the routine is the same--it can be shortened by doing it directly (without the X-register):

```
LDA $F7F0 + $0F
```

7.3.3 JSRFAR and JMPFAR

If, for example, you have blocked out part of the ROM and want to jump to a kernal routine, you can do this via the JSRFAR routine. Here the CPU registers are not used for passing parameters but the zero-page addresses $02 to $09.

$02 Configuration index
$03, $04 New PC - jump address
$05 New processor status
$06 Accumulator
$07 X-register
$08 Y-register
$09 SP - stack pointer

The corresponding values are found at $05 as the output parameters. Let us assume that we have configuration 1 enabled--that is, only RAM 1. We want to determine the contents of address $0400 in RAM bank 0 (the left-hand corner of the 40-character screen). We must use the FETCH routine for this. For example:

```
LDA #$7F   ;ENABLE RAM 1 AND KERNAL
STA $FF00  ;INTO CONFIGURATION REGISTER
LDA #$0F   ;CONFIGURATION IDEX KERNAL & RAM 0
STA $02    ;PASS
LDA #$FF   ;HIGH BYTE OF $FF74
STA $03    ;PASS
LDA #$74   ;LOW BYTE OF THE DESTINATION ADDRESS
STA $04    ;PASS $FF74
LDA #$00   ;LOW BYTE OF $0400
STA $FC    ;SAVE
LDA #$04   ;HIGH BYTE OF $0400
SAT $FD    ;PASS
LDA #$FC   ;ZERO-PAGE ADDRESS OF THE POINTER
STA $06    ;AND PASS
LDA #$00   ;ADDRESS RAM BANK 0
```

```
LDA #$00   ;VALUE FOR Y-REGISTER FOR FETCH
STA $08    ;SAVE OFFSET
JSR $FF6E  ;CALL JSRFAR
LDA $06    ;HERE IS THE VALUE FROM $0400 IN
            RAM 0
```

A lot of parameters to pass, right? First it's very important to ensure that the kernal range $C000 to $FFFF is enabled. No RAM may be addressed here or the JSRFAR routine will hang up (even if you call the JSRFAR routine directly at $02CD--it simply branches back to the kernal). So you should always check this before calling JSRFAR, which we do in our example routine first. RAM bank 1 is enabled by the byte $7F and all memory areas except for $C000 to $FFFF are switched to RAM. Then the new configuration register is defined.

The second important point: The program counter PC is defined with the high byte at address $03 and the low byte at address $04; note that this is not the usual machine language convention.

All specifications that are not absolutely necessary can be omitted. Usually all that is required is to define the memory configuration in $02 and then the new program counter in $03/$04. All the others are options which may be useful at various times.

The routine JSRFAR writes the corresponding values at addresses $05 to $09 when it returns. In our example, use is also made of parameter passing in the accumulator.

We now want to show you another short example. Imagine that you have program code in RAM bank 0 as well as RAM bank 1. This first routine is the "subroutine" in bank 1 which in our example does nothing more than add the accumulator and X-register. The carry is indicated in the carry flag. Enter the routine in the monitor with A 12000. You then select RAM bank 1.

```
12000          LDA $06    ;ACC PARAMETER
12002          CLC        ;CLEAR CARRY FOR ADDITION
12003          ADC $07    ;ADD TO X REGISTER
12005          RTS        ;END OF THE ROUTINE
```

The routine gets the contents of the accumulator from address $06 and then adds it to the X-register. The contents of the accumulator are returned in address $06. In this example it is important that the processor status

in address $06. In this example it is important that the processor status register, containing the flags, is passed to address $05. In the main program the carry flag can be tested with BCC or BCS. But here is the routine in RAM bank 0, which calls the routine in RAM bank 1 by means of the JSRFAR routine:

```
02000           LDA #$3F   ;RAM 0 AND KERNAL
02002           STA $FF00  ;SET AS CONFIGURATION
02005           LDA #$0D   ;RAM 1 AND KERNAL
02007           STA $02    ;NEW CONFIGURATION
02009           LDA #$20   ;ACC IS $20
0200B           STA $06    ;PASS
0200D           LDA #$FF   ;ADD $FF
0200F           STA $07    ;PASS
02011           LDA #$20   ;HIGH BYTE OF $2000
02013           STA $03    ;PASS AS PC
02015           LDA #$00   ;LOW BYTE OF $2000
02017           STA $04    ;PASS AS PC
02019           JSR $FF74  ;CALL JSRFAR
0201C           LDA $05    ;GET FLAGS
0201E           PHA        ;ON STACK
0201F           PLP        ;AND IN FLAG REGISTER
02020           LDA $06    ;LOW BYTE OF ADDITION
02022           STA $FD    ;STORE AS LOW BYTE
02024           LDA #$00   ;HIGH BYTE
02026           STA $FE    ;STORE
02028           BCC $202C  ;NO CARRY, THEN JUMP
0202A           INC $FE    ;COMPENSATE FOR CARRY
0202C           BRK        ;TO MONITOR
```

When you enter and start this routine, you will find the result of the addition $FF+$20 = $11F at address $FD/$FE. This routine shows how to get the flags which are passed in memory location $05 actually into the status register: Load the accumulator with the contents of $05, push it onto the stack, and then pull it into the status register.

The JMPFAR routine works the same way as JSRFAR. Here however there is no return via RTS, but that is also why this routine is called JMPFAR. Naturally, no output parameters can be checked since there is no return.

7.4 The Important Kernal Routines

7.4.1 Kernal routines with vectors at $FF4D

First we want to look at the kernal routines defined via jump vectors at address $FF4D. These include the most important routines, from input and output of characters to the RS-232 routines.

The routines are introduced in the order of their definition at $FF4D. Whenever possible, the input/output parameters are given, as well as a short description. Where appropriate, a short example routine accompanies the description. The entry addresses are given in both decimal (in parentheses) and hexadecimal.

When vectors are present, you should **always** use them to access the routine--it's why they are there. Should the operating system ever be changed or extended, the location of these vectors will not be changed so your program will not crash or go crazy.

C64 MODE

Purpose: Enable the 64 mode
Address: $FF4D (65357)

Description: A jump to this routine causes the computer to switch from the 128 mode to the 64 mode. The clock frequency is reduced to 1MHz and the MMU locks all of the necessary registers so that they cannot be manipulated in the 64 mode. **There is no return!**

Input parameters: None
Output parameters: -none, since no return-

DMA-CALL

Purpose: Initialize external RAM components
Address: $FF50 (65360)

Description: In order to have direct memory access (DMA) to external RAM, it must be first initialized with this routine. The new configuration is passed in the X-register.
Input parameter: .X
Output parameters:

BOOTCALL

Purpose: Boot the disk
Address: $FF53 (65363)

Description: When this routine is called, the computer attempts to boot from the disk inserted in the drive--the same as when the computer is turned on. If the routine cannot find a boot file, it returns control. The device address is passed in the X-register so you can boot from device 8 or 9.

Input parameter: .X
Output parameters:

PHOENIX

Purpose: Cold start
Address: $FF56 (65366)

Description: Cold start the 128 mode. If a memory card is found in the expansion cartridge, control is passed to this card. Otherwise an attempt is made to boot the disk. Tabs, key definitions, etc. are all reset.

LKUPLA

Purpose: Search in the table for logical file number
Address $FF59 (65369)

Description: The routine searches in the table for the device and secondary addresses of the logical file number given in the accumulator. The status variable ST is set according to the results of the routine. If the logical file number is found, the carry is cleared and the following parameters are transmitted: A:LFN, X:device address, Y:secondary address. If the routine does not succeed, the carry is set. Only logical file numbers opened with OPEN can be found.

Input parameter: .A contains the LFN to find
Output parameters: Status ST at $90, .A, .X, .Y, carry
Zero-page address $B8 to $BA

Example:

```
 ;Search for LFN
LDA #$01  ;SEARCH FOR LFN 1
JSR $FF59
BCS ERROR ;NOT OPENED--OUTPUT ERROR
TAX       ;LFN TO X
JSR $FF59 ;CKOUT - SET FILE AS OUTPUT FILE
```

LKUPSA

Purpose: Search for a secondary address
Address: $FF5C (65372)

Description: This routine looks in the table of opened channels for the secondary address passed in the Y-register. As for the LKUPLA routine, the carry flag is set if the search failed. The carry is cleared if the search succeeded and the accumulator contains the LFN, the X-register contains the device address, and the Y-register the secondary address.

Input parameters: .Y contains the SA to search for
Output parameters: Status ST at $90, .A, .X, .Y, carry
Zero-page addresses $B8 to $BA

Example:

```
 ;Search for LFN of disk command channel
LDY #$0F  ;SEARCH FOR LFN WITH
JSR $FF5C ;SECONDARY ADDRESS 15
BCS ERROR ;NOT FOUND, RETURN ERROR
TAX       ;LFN TO X
JSR CKOUT ;OPEN AS OUTPUT DEVICE
JSR INITD ;INITIALIZE DISKETTE
```

SWAPPER

Purpose: Switch 40/80 columns
Address: $FF5F (65375)

Description: This routine exchanges the 40/80 column mode. The information in the zero page for the active screen must be exchanged with that of the passive screen. The memory range $E0 to $FA is exchanged with the area $0A40 to $0A5A. No input parameters are necessary.

Example:

```
 ;Clear both screens
JSR $C142 ;CLEAR SCREEN
JSR $FF5F ;EXCHANGE 40/80 COLUMN MODE
JSR $C142 ;CLEAR PASSIVE SCREEN TOO
JSR $FF5F ;BACK TO CURRENT SCREEN
```

DLCHR

Purpose: Copy the CHARROM
Address: $FF62 (65378)

Description: The character set is copied into the VDC RAM when the <40/80 DISPLAY> key is pressed because the 80-column controller does not get the character information from ROM. The graphics package, for example, makes use of this routine because the character set in VDC RAM is overwritten when graphics are used. The character set selected by the <40/80 DISPLAY> key and is copied into VDC RAM by this routine. There are neither input nor output parameters.

PFKEY

Purpose: Redfine a key
Address: $FF65 (65381)

Description: This routine allows you to define the function keys (F1 to F8 as well as SHIFT/RUN-STOP and HELP). The address in the zero page which points to the KEY text is passed in the accumulator. The X-register contains the number of the function key (1 to 10) and Y contains the length of the string. Then you can call the routine PFKEY, which inserts this string into the table.
Input parameters: Zero page, .A, .X, .Y

Example: (at address $2100)

```
      ;Redefine the HELP key
     LDA #$00   ;LOW BYTE OF $2000
     STA $FC    ;STORE IN ZERO PAGE
     LDA #$20   ;HIGH BYTE OF $2000
     STA $FD    ;STORE IN ZERO PAGE
     LDA #$FC   ;POINTER
     LDX #$0C   ;REDEFINE HELP KEY
     LDY #4     ;LENGTH OF STRING AT $2000
     JSR $FF65  ;REDFINE KEY
```

And at address $2000:

```
02000 52 55 4E 0D ....
```

SETBNK

Purpose: Define memory bank for disk operation
Address: $FF68 (65384)

Description: This routine must be called before LOAD, SAVE, VERIFY, and every OPEN command. The configuration index of the filename is passed to it in the Y-register, as well as the configuration index of the memory area to be processed in the accumulator. The Y-register is stored in zero-page address $C6 and the accumulator in $C7. See also the example for SETNAM (FFBD).

Input parameters: .A, .Y

GETCONF

Purpose: Get the configuration byte
Address: $FF6B (65387)

Description: There is a table of 16 of the memory configurations required for normal operation. This table is found at address $F7F0. You pass the configuration index to this routine in the X-register and you get the configuration byte back in the accumulator. Normally this byte is then written in the configuration register at address $FF00 of the MMU.

Input parameter: .X
Output parameter: .A

Example:

```
    ;Set RAM bank 1
LDX #$01  ;ONLY RAM BANK 1
JSR $FF6B ;GET CONFIGURATION BYTE
STA $FF00 ;AND SET
```

JSRFAR

Purpose: Jump to a subroutine in any bank
Address: $FF6E (65390)

Description: The routine JSRFAR is used to jump to a subroutine in any configuration. The parameters are passed through zero-page locations $02 to $09. After the routine returns, the old configuration is re-enabled. A precise description including example program is found in Section 7.3.3.

Input parameters: Zero page $02 to $09
Output parameters: Zero page $05 to $09

JMPFAR

Purpose: Jump to any bank
Address $FF71 (65393)

Description: Here again the parameters are passed through zero-page addresses $02 to $09. JMPFAR is not a subroutine call but just a jump to an

address in a bank; JMPFAR combines switching the configuration byte with the jump. Since there is no return here, no parameters are returned. You can find more about this routine in Section 7.3.3.

Input parameters: Zero page $02 to $09

INDFET

Purpose: Get a byte from any bank
Address: $FF74 (65396)

Description: This routine, completely contained in the zero page, allows you to read any memory address in any configuration without having to change the current configuration. To do this you must first define a pointer in a zero-page address to the memory location to be read. This zero-page address is then passed in the accumulator, while the configuration index is passed in the X-register and the offset to the zero-page pointer in the Y-register. You can find more information about the FETCH (=INDFET), STASH, and CMPARE routines in Section 7.3.1.

Input parameters: .A, .X, .Y, 1 zero-page address
Output parameter: .A

Example:

```
 ;Get $1000 from RAM bank 1
LDA #$00  ;LOW BYTE OF $1000
STA $FC   ;STORE IN ZERO PAGE
LDA #$10  ;HIGH BYTE OF $1000
STA $FD   ;STORE IN ZERO PAGE
LDA #$FC  ;POINTER IN ZERO PAGE
LDX #$0D  ;RAM 1 AND KERNAL
LDY #$00  ;OFFSET IS ZERO
JSR $FF74 ;GET BYTE FROM $1000, RAM BANK 1
```

INDSTA

Purpose: Store accumulator in any bank
Address: $FF77 (65399)

Description: Similar to the INDFET routine, this routine stores the contents of the accumulator in any memory configuration. The parameters must be

passed in the accumulator, and the X and Y registers. The character to be stored must be passed in the accumulator. The zero-page address at which the pointer is stored must be defined at address $02B9. You can get more detailed information about this routine in Section 7.3.1.

Input parameters: .A, .X, .Y, zero page, $02B9

Example:

```
;Store $FF at $1000 in RAM bank 1
LDA #$00  ;LOW BYTE OF $1000
STA $FC   ;STORE
LDA #$10  ;HIGH BYTE OF $1000
STA $FD   ;STORE
LDA #$FC  ;ADDRESS IN ZERO PAGE
STA $02B9 ;PASS TO INDSTA ROUTINE
LDA #$FF  ;VALUE TO BE WRITTEN
LDX #$0D  ;RAM 1 AND KERNAL
LDY #$00  ;OFFSET IS ZERO
JSR $FF77 ;CALL INDSTA
```

INDCMP

Purpose: Compare the accumulator with memory in any bank
Address: $FF7A (65402)

Description: This routine compares the accumulator with any memory location in any bank. Just as with the INDSTA routine, you must pass the address of a zero-page pointer to the INDCMP routine. This is done at address $02C8. The byte to be compared is passed in the accumulator while the configuration index is passed in X and the offset in the Y-register. After calling the routine, the result of the comparison--the processor status byte--is found at address $05. The example below shows how you can react accordingly to the result of the comparison. More information is in Section 7.3.1.

Input parameters: .A, .X, .Y, zero page, $02C8
Output parameters: $05 (status)

Example:

```
;Compare <acc> with <$1000> in bank 1
LDA #$00  ;LOW BYTE OF $1000
STA $FC   ;STORE
```

```
LDA #$10   ;HIGH BYTE OF $1000
STA $FD    ;STORE
LDA #$FC   ;POINTER IN ZERO PAGE
STA $02C8  ;PASS TO INDCMP ROUTINE
LDA #$FF   ;COMPARISON OPERAND
LDX #$0D   ;RAM BANK 1 AND KERNAL
LDY #$00   ;OFFSET
JSR $FF7A  ;CALL INDCMP
LDA $05    ;GET STATUS (RESULT OF COMPARE)
PHA        ;ON STACK AND THEN
PLP        ;IN PROCESSOR STATUS REGISTER
BEQ EQUAL  ;JUMP IF EQUAL
;--- NOT EQUAL ---
```

PRIMM

Purpose: Output text
Address: $FF7D (65405)

Description: This routine is very practical because it's simple to use. No parameters need be passed. All characters following the call are sent to the current output device via BSOUT. A zero-byte is used as the terminating character. The program execution is then continued immediately following the zero-byte. One disadvantage of this routine: The program will be unreadable if it is disassembled.

Example:

```
JSR $FF7D ;OUTPUT FOLLOWING CHARACTER
.ASC "This is a string!"
.BYT $0D,$0A,$0D,$00
LDA #$00  ;THE PROGRAM CONTINUES HERE
```

See also the example in the ROM listing at $F908.

CINIT

Purpose: Initialize video controller and editor
Address: $FF81 (65409)

Description: The function keys are returned to the defaults, both video controllers are initialized and the 40/80 column mode is enabled dependent

on the 40/80 column key. The keyboard buffer is cleared, all flags are reset, and a CLRCH is performed.

IOINIT

Purpose: Initialize the input/output device
Address: $FF84 (65412)

Description: The input/output devices are initalized, meaning that the RESET line on the serial bus is activated. Any printers connected are set to their initial states and the disk drive clears its channels--it is like it had just been turned on.

RAMTAS

Purpose: BASIC warm start
Address: $FF87 (65415)

Description: This routine initializes the zero page, resets the pointers for SYSTOP and SYSBOT (the memory upper and lower boundaries), resets the pointers for the RS-232 input/output buffers, and resets the cassette buffer.

RESTOR

Purpose: Initialize system vectors
Address: $FF8A (65418)

Description: The system vectors at address $0314 to $0332 (inclusive) are set to the default values. This routine should be called when you modified many of the vectors and want to set them back. This routine calls the following VECTOR routine with the carry cleared.

VECTOR

Purpose: Copy or reset system vectors
Address: $FF8D (65421)

Description: This routine copies the 16 vectors at $0314 to the address defined by the X (low) and Y (high) registers, provided the carry flag is set. If the carry flag is cleared, the vectors at $0314 are loaded with the area given by the X and Y registers.

Input parameters: .X, .Y, carry

Example:

```
LDX #$00  ;LOW BYTE OF $1000
LDY #$10  ;HIGH BYTE OF $1000
CLC       ;CLEAR CARRY FOR COPY ($1000)->($0314)
JSR $FF8D ;LOAD VECTORS
```

SETMSG

Purpose: Enable/disable DOS messages
Address: $FF90 (65424)

Description: The routine stores the value of the accumulator in the zero-page address $9D. If system messages should be printed, set bit 7 of the accumulator. If $9D is positive, system messages are inhibited.

Input parameter: .A

SECND

Purpose: Send secondary address to LISTEN
Address: $FF93 (65427)

Description: The secondary address to be sent is passed in the accumulator. The routine outputs the contents of the accumulator on the serial bus as the secondary address.

Input parameters: .A

Example:

```
;SEND LISTEN
LDA #$F0  ;SECONDARY ADDRESS 0 FOR CLOSE
JSR $FF93 ;SET SECONDARY ADDRESS
```

TKSA

Purpose: Send secondary address to TALK
Address: $FF96 (65430)

Description: This routine sends the secondary address given in the accumulator on the bus preceded by a TALK signal.

Input parameter: .A

MEMTOP

Purpose: Set/get the memory top
Address: $FF99 (65433)

Description: If the carry flag is set, the maximum available memory location is returned in the X-register (low) and Y-register (high). If the routine is called with the carry cleared, the memory top is set with the two registers.

Input parameters: .X, .Y (for cleared carry), carry
Output parameters: .X, .Y (for set carry)

Example:

```
;Read the memory top
SEC       ;READ THE TOP
JSR $FF99 ;GET TOP
STX $FC   ;STORE
STY $FD   ;STORE
LDX #$00  ;LOW BYTE OF $1000
LDY #$10  ;HIGH BYTE OF $1000
CLC       ;FLAG TO SET MEMTOP
JSR $FF99 ;SET MEMORY TOP
```

MEMBOT

Purpose: Set/get the memory bottom
Address: $FF9C (65436)

Description: Similar to MEMTOP, the lower boundary of the available memory is set with the two registers X (low) and Y (high) if the carry flag is cleared. If the carry flag is set, the memory bottom is read and returned in the two registers.

Input parameters: .X, .Y (for cleared carry), carry
Output parameters: .X, .Y (for set carry)

KEY

Purpose: Return key pressed
Address: $FF9F (65439)

Description: This routine is elementary to keyboard decoding. The keyboard is checked for a pressed key by means of the keyboard decoding table. If a pressed key is returned, the ASCII value is determined and placed into the keyboard buffer at ($034A).

SETTMO

Purpose: Set the time-out flag for IEEE
Address: $FFA2 (65442)

Description: The routine saves the value passed in the accumulator at address $0A0E as the timeout flag for the IEEE routines. In order to permit the timeout in the IEEE routines, bit 7 of the accumulator must be set.

Input parameters: .A

ACPTR

Purpose: Get a byte from the serial bus
Address: $FFA5 (65445)

Description: The routine gets a byte from the serial bus. This character is returned in the accumulator. The status byte ST at $90 is set according to the action.

Output parameter: .A

CIOUT

Purpose: Output a character to the serial bus
Address: $FFA8 (65448)

Description: This routine is counterpart of ACPTR. The character passed in the accumulator is output on the serial bus. Here too the status byte ST at $90 is changed according to the action.

Input parameter: .A

UNTLK

Purpose: Send UNTALK on the serial bus
Address: $FFAB (65451)

Description: This routine is called when closing or redirecting an input channel. It silences a "talking" device.

UNLSN

Purpose: Send UNLISTEN on the serial bus
Address: $FFAE (65454)

Description: Corresponding to UNTALK, this routine shuts off a receiving device. This is done when closing or redirecting an output channel.

LISTN

Purpose: Send LISTEN to a device
Address: $FFB1 (65457)

Description: A device on the serial bus is requested for input. The LISTEN signal is sent over the serial bus to do this. The device address of the appropriate device is passed in the accumulator. For example, a LISTEN is sent to a printer before characters are sent to it over the serial bus. If you use LISTEN, you must output the characters via the routine CIOUT (not via BSOUT!). Use the routine UNLISTEN to close the channel. Only one device may be active on the serial bus. To simplify all this, you can open and close channels in the operating system. BSOUT and BASIN then take care of sending LISTEN and UNLISTEN as well as TALK and UNTALK.

Input parameter: .A

Example:

```
;Send LISTEN to printer
LDA #$24   ;DEVICE ADDRESS FOR PRINTER AND
            LISTEN ON
JSR $FFB1
```

TALK

Purpose: Send TALK to a device
Address: $FFB4 (65460)

Description: This routine sends the command TALK to a device. The device address is to be passed in the accumulator. The TALK command requests a device connected to the serial bus for talking, i.e. for sending information.

Input parameters: .A

READST

Purpose: Get the I/O status byte
Address $FFB7 (65463)

Description: The current system status is returned in the accumulator. If the RS-232 is active, the status byte is returned and immediately cleared in memory. If you need the status byte more often, save it somewhere. If a channel other than the RS-232 channel is open, the status byte is returned in address $90.

Output parameter: .A

SETLFS

Purpose: Set file parameters
Address: $FFBA (65466)

Description: This routine is required to open a file. The logical file number is passed in the accumulator, the device address in the X-register, and the secondary address in the Y-register. The routine stores these values in the zero-page addresses from $B8 to $BA.

Input parameters: .A, .X, .Y

SETNAM

Purpose: Set the filename parameters
Address: $FFBD (65469)

Description: Information for the filename is stored in the zero page in this routine. These specifications must all be made before the channel is opened. The length of the filename is passed in the accumulator, the low byte of the address at which the filename is stored in the X-register, and the high byte in the Y-register. Furthermore, you must pass with the SETBNK routine the configuration indices for the filename and the memory range to be processed.

Input parameters: .A, .X, .Y

Example:

```
;Open one of the directory files on the disk
LDA #$0C  ;AREA IN RAM BANK 0
TAX       ;FILENAME ALSO IN RAM BANK 0
JSR $FF68 ;CALL SETBNK
LDA #$01  ;LOGICAL FILENUMBER
LDX #$08  ;DEVICE ADDRESS
LDY #$00  ;SECONDARY ADDRESS FOR READING
JSR $FFBA ;SETFLS
LDA #$01  ;LENGTH OF THE FILENAME
LDX #$00  ;LOW BYTE OF THE ADDRESS AT WHICH
LDY #$10  ;THE FILENAME IS STORED ($1000)
JSR $FFBD ;OPEN - OPEN THE CHANNEL
```

and at address $1000:

```
01000 24 ....
```

OPEN

Purpose: Open a file
Address: $FFC0 (65472)

Description: The file defined by the routines SETNAM, SETLFS, and SETBNK is entered into the list of logical file numbers. Not until this is done can the logical file number be used for the routines CKOUT and CHKIN. A maximum of nine files can be open at one time.

CLOSE

Purpose: Close a logical file
Address: $FFC3 (65475)

Description: The logical file specified in the accumulator is closed. All stored values like the device address, secondary address, etc. are erased from the table. If an error is encountered, the carry flag will be set.

Input parameter: .A
Output parameter: carry

Example:

```
;Example for CLOSE
LDA #$01  ;CLOSE THE EXAMPLE FILE FROM SETNAM
JSR $FFC3 ;CALL CLOSE
BCS ERROR ;ERROR ENCOUNTERED
```

CHKIN

Purpose: Define a logical file as the input channel
Address: $FFC6 (65478)

Description: The logical file number to be used as the input channel is passed in the X-register. The given logical file number must have already been opened with the OPEN command. If the BASIN routine is called after the OPEN command, the input is not done from the keyboard but from the opened file; this can be from the disk drive. It should be noted that **no** CHKIN is required when reading from the keyboard because it is the standard input device. After a CLOSE or CLRCH, the keyboard is automatically again the input device. The carry flag is also used as the OK flag for this routine.

Input parameter: .X
Output parameter: carry

Example:

```
;Read the directory
JSR DIROP ;OPEN 1,8,0,"$"(SELF-DEFINED ROUTINE)
LDX #$01  ;LFN OF THE OPENED FILE
JSR $FFC6 ;EXECUTE CHKIN
JSR $FFCF ;BASIN--GET CHARACTER
```

CKOUT

Purpose: Define a logical file as the output file
Address: $FFC9 (65481)

Description: This routine defines a file passed in the X-register as the output file. It must have been previously opened properly. A file opened with OPEN 1,8,0,"$" and then defined as the output file with CKOUT would result in an error because this file was opened for reading and not for writing. After defining an output file, the screen is no longer the output

device -- the output file is. All characters output via BSOUT are sent to this device. The carry flag is used to indicate an error. If it is cleared, the operation was successful.

Input parameters: .X
Output parameters: carry

CLRCH

Purpose: Close input/output channel
Address: $FFCC (65484)

Description: This routine clears any input or output files defined with CHKIN and/or CHKIN. An UNTALK is sent to the input device and UNLISTEN is sent to the output device. The screen again becomes the output device and the keyboard the input device. The files are *not* closed. Neither input nor output parameters are passed.

BASIN

Purpose: Get a character from the input channel
Address: $FFCF (65487)

Description: The file opened and defined as the input file by CHKIN (otherwise the keyboard) returns a character in the accumulator.

Output parameter: .A

BSOUT

Purpose: Output a character to the output channel
Address: $FFD2 (65490)

Description: The character passed in the accumulator is sent to the open file defined as the output file by CKOUT. If the screen is the output file (default), the ASCII character is converted to a printable POKE code (This is an extensive procedure. Those interested should look at the appropriate code in the C range of the kernal).

Input parameter: .A

Example:

```
;Switch the 40/80 column mode
LDA #$1B  ;<ESC>
JSR BSOUT ;$FFD2, OUTPUT CHARACTER
LDA #"X"  ;<ESC>X TO EXCHANGE THE SCREEN STATUS
JSR BSOUT ;OUTPUT
```

(There is also a special routine to which you can jump.)

LOADSP

Purpose: Load a file into memory
Address: $FFD5 (65493)

Description: Before a file can be loaded with LOADSP, the device, secondary address, filename, etc. must be defined by the routines SETLFS, SETNAM, and SETBNK. The address at which the file is to be loaded is passed in the X (low) and Y (high) registers.

Input parameters: .X, .Y

Example:

```
;Load an overlay
JSR PREP  ;SETLFS, SETBNK, SETNAM, ETC.
LDX #$00  ;LOW BYTE OF $1000
LDY #$10  ;HIGH BYTE OF $1000 (LOAD ADDRESS)
JSR $FFD5 ;LOAD FILE AT $1000
```

SAVESP

Purpose: Save memory to a file
Address: $FFD8 (65496)

Description: This routine saves a memory range to a file (disk, cassette). As with the LOADSP routine, you must first define the device address, secondary address, RAM bank, filename, etc. with the routines SETBNK, SETLFS, and SETNAM. The zero-page address at which the start address of the area to be saved is stored and passed in the accumulator. The end address of the range is passed in the X (low) and Y (high) registers.

Input parameters: .A, .X, .Y, zero page

Example:

```
;Save the range $1000 to $1100
JSR PREP  ;CALL SETLFS, SETNAM, SETBNK
LDA #$00  ;LOW BYTE OF $1000
STA $FC   ;STORE IN ZERO PAGE
LDA #$10  ;HIGH BYTE OF $1000
STA $FD   ;STORE IN ZERO PAGE
LDA #$FC  ;THE POINTER IS LOCATED IN $FC
LDX #$00  ;LOW BYTE OF THE END ADDRESS $1100
LDY #$11  ;HIGH BYTE OF THE END ADDRESS $1100
JSR $FFD8 ;SAVESP--SAVE THE RANGE $1000-$1100
```

SETTIM

Purpose: Set the system clock TI
Address: $FFDB (65499)

Description: This routine sets the system clock TI, which is defined at address $A0. This clock is controlled by the kernal IRQ routine and is not very accurate. If want an accurate clock, use the timers in the two CIAs (see Chapter 3). The high-order byte of the 24-hour clock is passed in the Y-register.

Input parameters: .A, .X, .Y

Example:

```
;Reset the system clock
LDA #$00  ;RESET MEANS
TAY       ;SET TO 0,0,0
TAX       ;ALL THREE REGISTERS TO ZERO
JSR $FFDB ;SETTIM
```

RDTIM

Purpose: Read the system clock
Address: $FFDE (65502)

Description: This routine reads from the 24-hour clock and passes the three bytes in registers Y (highest-order), X, and the accumulator (lowest).

Output parameters: .A, .X, .Y

Example:

```
;Read the 24-hour clock
JSR $FFDE ;CALL RDTIM
STY $FC   ;STORE MSB
STX $FD   ;STORE MIDDLE BYTE
STA $FE   ;STORE LSB
```

STOP

Purpose: Poll the STOP key
Address: $FFE1 (65505)

Description: If the STOP key was pressed since the last IRQ call, the zero flag will be set and a CLRCH will be executed. If the STOP key was not pressed, the zero flag will be cleared.

Output parameters: zero flag

Example:

```
;Check for STOP
JSR $FFE1 ;STOP KEY PRESSED
BEQ YES   ;PRESSED
```

GETIN

Purpose: Get a character from the keyboard buffer or RS-232
Address: $FFE4 (65508)

Description: Gets a character from the defined input file. If no character is ready, the accumulator is returned with zero.

Output parameter: .A

CLALL

Purpose: Close all open files
Address: $FFE7 (65511)

Description: All of the files opened with OPEN are closed, actually CCALL deletes the files by clearing the table index--no CLOSE is actually performed. This can be particularly annoying for open disk files (WRITE FILE OPEN ERROR results). After erasing the logical files, a CLRCH is executed. CLALL should therefore be used with caution.

UDTIM

Purpose: Update system clock
Address: $FFEA (65514)

Description: This routine is usually called by the IRQ routine. The three-byte 24-hour clock is incremented by one unit.

SCRORG

Purpose: Get the size of the current window
Address: $FFED (65117)

Description: The routine SCRORG gets the current window values in the registers. After the call, the accumulator contains the maximum column number, the number of lines in the window is found in the Y-register, and the X-register contains the number of columns in the window.

Output parameters: .A, .X, .Y

PLOT

Purpose: Get/set cursor position
Address: $FFF0 (65120)

Description: The cursor position is either fetched or set based on the condition of the carry flag. The X and Y registers are the communication registers. The Y-register defines the line (the first line in the window is

zero) and the X-register the column of the cursor. If the carry flag is set, the current cursor position in the window is returned in the X and Y registers.

Input parameters: .X, .Y, carry

Example:

```
;Set an asterisk in the middle of the window
JSR $FFED ;CALL SCRORG
TXA       ;COLUMN NUMBER TO ACC
LSR A     ;DIVISION BY TWO (MIDDLE)
TAX       ;AND AS COLUMN BACK TO X
TYA       ;LINE NUMBER TO ACC
LSR A     ;DIVISION BY TWO (MIDDLE)
TAY       ;AND AS LINE TO Y
CLC       ;CLEAR CARRY=SET CURSOR POSITION
JSR $FFF0 ;SET CURSOR POSITION
LDA #"*"  ;LOAD ACC WITH ASTERISK
JSR $FFD2 ;AND OUTPUT
```

IOBASE

Purpose: Get the base address of the I/O area
Address: $FFF3 (65123)

Description: The address of the input/output area is returns in the X (low) and Y (high) registers. This address is always $D000 for the 128. For later expansions or movements, we advise you in order to maintain compatibility to integrate this routine into the software and make reference to it.

Output parameters: .X, .Y

Example:

```
;Start of the program
JSR $FFD3 ;IOBASE
STX $FD   ;STORE LOW BYTE
STY $FE   ;STORE HIGH BYTE
```

This address is referenced in the program as follows:

```
STA ($FD),Y ;IN I/O AREA
```

7.4.2 Other useful kernal routines

There are some other routines in the kernal which can help save time and program memory. These routines are found particularly in the $C000 block of ROM and are used for input/output on the two screens. Here are some of the routines we feel are useful.

CLRWIN

Purpose: Clear the window (screen)
Address: $C142 (49474)

Description: If no window is defined, the entire screen is cleared. If a window is defined, only the screen area inside the boundaries of the window is erased.

CURHOM

Purpose: Cursor to HOME position in window
Address: $C150 (49482)

Description: The cursor is positioned in the upper left-hand corner of the window. If no window is defined, the cursor is placed in the upper left-hand corner of the screen. Note that position 0/0 always defines the upper left-hand corner of the window.

GETLIN

Purpose: Get an input character
Address: $C258 (49752)

Description: Characters are taken from the keyboard and displayed on the screen at the current cursor position until the <RETURN> key is pressed.

BSOUT SCRN

Purpose: Output a character to the current screen
Address: $C72D (50989)

Description: This routine is the continuation of the BSOUT routine at $FFD2. The routine is faster since it does not have all of the checks that are built into BSOUT. The character is passed to the routine in the accumulator and output to the currently active screen--at the current cursor position.

Input parameters: .A

CLQIR

Purpose: Clear the quote, insert, and reverse modes
Address: $C77D (51069)

Description: This routine clears the flags for the quote, insert, and reverse modes. It works somewhat faster than outputting the necessary control sequences via BSOUT.

Here is a list of other important routines and their address:

$C854	(51284)	Cursor right in window
$C85A	(51290)	Cursor down in window
$C867	(51303)	Cursor up in window
$C875	(51317)	Cursor left in window
$C880	(51328)	Enable second character set
$C8BF	(51391)	Clear RVS mode
$C8C1	(51393)	Set RVS mode
$C8C7	(51399)	Enable underlining
$C8CE	(51406)	Disable underlining
$C91B	(51483)	Delete character to the left of the cursor
$C93D	(51517)	Delete character under cursor
$C94F	(51535)	Jump to tab
$C980	(51584)	Clear all tabs
$C98E	(51598)	BELL - create bell tone
$CA14	(51732)	Cursor pos. defined left/top of window
$CA16	(51734)	Cursor pos. defined right/top of window
$CA24	(51748)	Define screen as window
$CA52	(51794)	Clear current line

$CA76	(51830)	Clear from cursor to end of line
$CA8B	(51851)	Clear from start of line to cursor pos.
$CA9F	(51871)	Clear from cursor pos. to end of screen
$CABC	(51900)	Scroll up
$CAF2	(51954)	Enable block cursor
$CAFE	(51966)	Enable underline cursor
$CB0B	(51979)	Cursor flash off
$CB21	(52001)	Cursor flash on
$CB3F	(52031)	Invert 80-column screen
$CB48	(52040)	80-column screen normal
$CC27	(52263)	<space> at current cursor position
$CC2F	(52271)	Character <acc> at current cursor position
$CC4A	(52298)	Output character <acc>, <X>:color, <Y>:column to 80-column screen (without moving the cursor)
$CC6A	(52330)	Get/set cursor position
$CD2C	(52524)	SWAPPER - switch 40/80-column

7.5 Tips & Tricks

Naturally, this section cannot replace our book Tips & Tricks, but we want to explain to you the most important and/or useful things which we have found out.

By use of these examples, you'll be able to see how to use the documented zero-page and ROM listings--since the information ultimately comes from these listings.

7.5.1 Disabling the STOP key

Frequently you may want to prevent the user from interrupting the program by pressing the STOP key--in many situations this can be dangerous if the STOP key is pressed accidentally.

To solve the problem, we look in zero page. Here, at address $0300 is a table of jump commands for the most important kernal routines. Practically speaking, this area is an interface between the programmer and the operating system, because it allows the programmer to cause other things to happen simply by redirecting the jump commands (usually to a routine he writes).

The address for the kernal STOP routine is found at address $0328--it points to $F66E. The current status of the STOP key is read from zero-page address $91 at this address $F66E. Address $91 is always loaded with the latest condition by the IRQ routine. If we skip this test, we achieve the effect that pressing the STOP key is no longer recognized by the STOP test routine. We need only modify the address at $0328. We write the low byte of the address of the command following the STOP routine to this address.We do this in BASIC with the following POKE:

```
POKE DEC("0328"),112 : REM DISABLE STOP KEY
```

The vector $0328/$0329 no longer points to $F66E but to $F670. The operating system no longer recognizes the STOP key, not during a program, nor while listing, or many other actions.

We have now done what we set out to do. There is a still a bug in the system, however. If someone is clever enough to press the STOP and RESTORE keys at the same time, our program will be interrupted anyway! The STOP test routine at address $F66E is also called in the NMI routine, though it does not use the vector $0328, so pressing the STOP key will be recognized.

7.5.2 Disable STOP-RESTORE combination

If this combination is pressed on the keyboard, the NMI service routine is called. NMI stands for Non-Maskable Interrupt--an interrupt is generated which cannot be disabled with the SEI command.

But there is a vector for this routine also in the zero-page area. The vector responsible for the NMI routine is found at address $0318 and points to the NMI routine in the kernal at address $FAF0.

If you do not want a BASIC warm-start to be executed when the STOP-RESTORE key combination is pressed, you must set the NMI vector to the end of the NMI routine. It is advisable to set the vector to $FA62, since this jumps to the IRQ return routine, reseting the registers and executes an RTI.

The following BASIC command is necessary to redirect the NMI routine:

```
POKE DEC("0318"),98 : REM REDIRECT NMI
```

After you have integrated this POKE command into your program (together with the STOP-key disable) it is impossible for anyone to exit your program unless they build a RESET switch on the user or expansion port, but this too can be intercepted...

7.5.3 The IRQ vector

The IRQ routine in the kernal is called every 1/60 of a second. The CIA is responsible for generating this interrupt with its timers. The vector for the IRQ routine is found at address $0314 and normally points to the kernal address $FA65. If you want to link into the IRQ routine, for your own sprite control, or to change the border color every second, etc., in can be done in this way.

Redirect the IRQ vector to your own routine and jump to the "remaining" kernal IRQ routine after executing yours. But be careful when you redirect the IRQ vector. The interrupts must be disabled when changing the vector or the computer may crash.

Here is a short example program which changes the border color of the 40-column screen by one color code every 60th IRQ call.

```
02000  78          SEI         ;Disable interrupts
02001  A9 0C       LDA #$0C    ;Store low byte of new
02003  8D 14 03    STA $0314   ;IRQ routine in vector
02006  A9 20       LDA #$20    ;Store high byte of new
02008  8D 15 03    STA $0315   ;IRQ routine in vector
0200B  58          CLI         ;Enable int. again
0200C  E6 FD       INC $FD     ;Increment counter
0200E  A5 FD       LDA $FD     ;Get counter
02010  C9 3C       CMP #$3C    ;60 already?
02012  D0 07       BNE $201B   ;Not yet reached
02014  EE 20 D0    INC $D020   ;Increment border color
02017  A9 00       LDA #$00    ;And counter again
02019  85 FD       STA $FD     ;Set to zero
0201B  4C 65 FA    JMP $FA65   ;Remaining IRQ routine
```

This routine is enabled by calling the enable routine at address $2000. This is done by:

SYS DEC("2000")

Now the color of the border is changed at regular intervals. This is one example (even though trivial), of what you can do with the IRQ routine.

7.5.4 Disabling the BASIC interrupt

As we mentioned in the chapter on the VIC chip, it can be very annoying when the interpreter is always getting in the way. There is a way around this. The interrupts stop working if you tell the interpreter not to jump to the BASIC IRQ routine. This can be done at address $0A04. If bit 0 is set, the BASIC IRQ routines for graphics and sound are executed. If we clear this bit, these routines will no longer be executed and the sprites will stop moving, etc.

This is a welcome option for all machine language programmers who want to program the sprites themselves. The text/graphic mode is not affected by all of this; it is still switched automatically. This is because this switch occurs in the kernal IRQ routine. If, for example, you want to enable the graphic mode, but don't want to use the BASIC commands, you must either make corresponding changes in the zero-page addresses, or you must sneak into the kernal routine.

To demonstrate the effect of this disabling, first define a sprite and enable it:

```
SPRITE 1,1,2,0,1,1 : REM TURN SPRITE 1 ON
MOVSPR 1,90#9      : REM MOVE SPRITE 1
```

Whatever your sprite may look like, it is now moving across the screen. If you now try to write to the VIC registers and change the appearance or the position of the sprite, you will see a brief flash on the screen and then the sprite will do what it wants or what the operating system wants.

The sprites can be stopped once and for all by clearing bit 0 in address $0A04. This is done with the following instruction:

```
POKE DEC("0A04"), PEEK(DEC("0A04")) AND 254
```

The sprite stops where it is and moves no further. Now the VIC chip can be manipulated without interference.

7.5.5 Positioning the cursor

You will often want to position the cursor at a given location on the screen/window from within BASIC. Unfortunately, there is no command which does this. You can only set the graphic cursor at a position X,Y by means of the LOCATE command. Of course this positioning is possible by outputting cursor-movement codes, but this method is:

a) slow,
b) memory-consuming, and
c) cumbersome

We offer you a way of positioning the cursor by calling the kernal routine that sets the cursor position. Normally the cursor line is passed in the X-register and the column in the Y-register. You can also pass these parameters as (optional) parameters in the SYS command.

As you can probably gather from the kernal listing, the routine for setting the cursor position is found at address $CC6A. Since we want to set the cursor position and not determine it, we can skip the carry-flag test at the start of the routine. We will use address $CC6C as the entry point.

The syntax for positioning the cursor looks like this:

BANK 15: SYS DEC("CC6C"),,<line>,<column>

The first line and the first column in the window is line zero, column zero. The two commas are required before the <line>.

As an example of how you can make use of this positioning routine, take a look at the following program:

```
10 REM *** DEMO PROGRAM FOR CURSOR POSITIONING ***
30 CL=40-40*(PEEK(DEC("D7")):REM 40 OR 80 COL?
40 PRINT CHR$(147);: REM CLEAR SCREEN
50 X=INT(RND(TI)*24): REM LINE
60 Y=INT(RND(TI)*CL): REM COLUMN
70 BANK 15: SYS DEC("CC6C"),,X,Y
75 PRINT "X"
80 GET G$: IF G$="" THEN 50
```

7.6 The Z-80

As you already know, there is a Z-80A built into your C-128. Most Z-80 fans will be interested in finding out how to switch this processor on. Here's a quick answer. The currently-active processor can be selected in bit 0 of the mode configuration register. If this bit 0 is set, the Z-80 is activated. A set bit means that the 8502 is working. If one switches to the Z-80 in this manner, the computer will never return from this mode.

In the C-128 there is a ROM containing 4K of Z-80 code. After power-up or RESET this Z-80 code is executed, meaning that the Z-80 is enabled. This ROM is located at $D000, but is mirrored down to $0000 for the Z-80. After a RESET, the Z-80 begins its work at address $0000. This ROM **cannot** be read by software.

In section 7.6.1 the first part of this ROM disassembled. We will not present a complete listing. It should be noted that these 4K bytes do not really have anything to do with CP/M itself, but only with booting CP/M.

After the configuration ($3E) has been selected, a test is made to see if there is a cartridge (/GAME or /EXROM line set) in the expansion port. If this is the case, control is passed to this cartridge. First, the 64 mode is enabled and the 8502 is activated.

If there is no cartridge in the expansion port, the Commodore key is tested. If you hold down the Commodore key during power-up or RESET, the 64 mode is entered directly, without making a BOOT attempt and without having to enter GO 64. If the Commodore key is not pressed, the various memory areas are copied, in the common area at $FFD0. It should be noted that the Z-80 as well as 8502 code is copied. After both routines are copied, control is passed to the (just-copied) routine at $FFE0. In this routine the 8502 is enabled and control is again passed to our "normal" operating system. If the Z-80 is enabled by the programmer, processing continues here (at address $FFEE). And it is precisely here that we find the interface. If you replaces the RST 8 with a JMP command, the Z-80 can be made to execute your own Z-80 program.

Let's go through a very simple example. We want to enable the Z-80 and change a memory location through Z-80 assembly language. This machine language program is to be located at address $3000:

```
3E 3F     LD   A,$3F     ;Select configuration
32 00 FF  LD   ($FF00),A;Set configuration
3E 1E     LD   A,$1E     ;Any value
32 00 22  LD   ($2200),A;Write in mem loc $2200
C3 E0 FF  JMP  $FFE0     ;And enable the 8502 again
```

We'll enter the Z-80 codes at address $3000 with the monitor. Use the M command to do this.

```
M 3000
>03000: 3E 3F 32 00 FF 3E 1E 32 00 22 C3 E0 FF
```

We must not forget to change the jump at $FFEE or otherwise the (normal) RST 8 will be executed. A jump to our routine must be placed at address $FFEE. We must insert the following three bytes at this address:

```
M FFEE
>FFEE: C3 00 30
```

Now we must write a routine in 8502 code which enables the Z-80 and continues after the return from the Z-80 execution. The routine looks like this:

```
SEI          ;Disable interrupts
LDA #$3E     ;Configuration byte
STA $FF00    ;Store
LDA #$B0     ;Enable Z-80 in the
STA $D505    ;Mode configuration register
NOP          ;Delay (buffer)
BRK          ;End, return to monitor
```

Enter this routine with the assembler at address $2100. Set the memory location $2200 to zero with the monitor and start the whole routine with:

```
G 2100
```

The computer returns immediately to the monitor. Read memory location $2200 and you will see that this address contains the value $1E.

7.6.1 The Z-80-ROM

Here is the first section of the Z-80 ROM, with comments:

```
****************************************  RST 00 (cold start)

0000:  3E 3E      LD  A,$3E       Configuration byte(RAM,I/O)
0002:  32 00 FF   LD  ($FF00),A   In configuration register
0005:  C3 3B 00   JP  $003B       Remainder of cold start

****************************************  RST 08

0008:  31 77 3C   LD  SP,$3C77
000B:  3E 3F      LD  A,$3F
000D:  C3 8C 01   JP  $018C       Remainder of RST 08

****************************************  RST 10

0010:  E1         POP HL          Return address from stack
0011:  6E         LD  L,(HL)      Low byte of the return address
0012:  C3 20 00   JP  $0020       Jump to RST 20 routine

0015:  00         NOP             Fill bytes
0016:  00         NOP
0017:  00         NOP

****************************************  RST 18

0018:  E1         POP HL          Return address from stack
0019:  6E         LD  L,(HL)      Low byte of return address
001A:  C3 28 00   JP  $0028       Jump to RST 28 routine

001D:  00         NOP             Fill bytes
001E:  00         NOP
001F:  00         NOP

****************************************  RST 20

0020:  3A 0F FD   LD  A,($FD0F)
0023:  A7         AND  A
```

```
0024:  28 02        JR   Z,$+4 >$0028
0026:  2C           INC  L
0027:  2C           INC  L

**************************************** RST 28

0028:  26 01        LD   H,$01
002A:  7E           LD   A,(HL)
002B:  23           INC  HL
002C:  66           LD   H,(HL)
002D:  6F           LD   L,A
002E:  E9           JP   (HL)

002F:  00           NOP

**************************************** RST 30

0030:  30 35        JR   NC,$+55 >$0067
0032:  2F           CPL
0033:  31 32 2F     LD   SP,$2F32
0036:  38 35        JR   C,$+55 >$006D

**************************************** RST 38

0038:  C3 FD FD     JP   $FDFD        Continue RST 38 at $FDFD

**************************************** RST 0 Contn'd

003B:  01 2F D0     LD   BC,$D02F     Register 47 of VIC Chip
                                            (keyboard)
003E:  11 FC FF     LD   DE,$FFFC     Write $FF in the keyboard
0041:  ED 51        OUT  (C),D        No extension keys
0043:  03           INC  BC           Register 48=clock register
0044:  ED 59        OUT  (C),E        Set to $FC -> 1 MHz mode
0046:  01 05 D5     LD   BC,$D505     Mode config. register
0049:  3E B0        LD   A,$B0        Test /EXROM and /GAME
004B:  ED 79        OUT  (C),A        Enable 128 mode
004D:  ED 78        IN   A,(C)        Mode config. register
004F:  2F           CPL               Read again and negate
0050:  E6 30        AND  $30          /EXROM or /GAME set?
0052:  28 05        JR   Z,$+7 >$0059 No, then no cartridge
```

```
********************************** Enable 64 mode and pass
                                     Control to the cartridge
0054:  3E F1       LD  A,$F1       Enable 8502 and select the
0056:  ED 79       OUT (C),A       64 mode
0058:  C7          RST $00         And execute cold start
0059:  01 0F DC    LD  BC,$DC0F    Select CRB reg. in CIA1
005C:  3E 08       LD  A,$08       And then stop
005E:  ED 79       OUT (C),A       Timer B as well as
0060:  0D          DEC C           Timer A of
0061:  ED 79       OUT (C),A       CIA 1
0063:  0E 03       LD  C,$03       DDRB--data direction reg.
0065:  AF          XOR A           For port B: Set all bits
0066:  ED 79       OUT (C),A       to Input
0068:  0D          DEC C           Pointer to DDRA and
0069:  3D          DEC A           Put all bits to
006A:  ED 79       OUT (C),A       Output.
006C:  0D          DEC C           Decrementing BC causes it
006D:  0D          DEC C           to Point to port A
006E:  3E 7F       LD  A,$7F       Write $7F to port A (See
0070:  ED 79       OU  (C),A       also Keyboard matrix)
0072:  03          INC BC          Pointer to port B (input)
0073:  ED 78       IN  A,(C)       And read
0075:  E6 20       AND $20         Mask out Commodore key
0077:  01 05 D5    LD  BC,$D505    Pointer for mode config reg
007A:  28 D8       JR  Z,$-38 >$0054 Key pressed> 64 mode
007C:  21 B4 0F    LD  HL,$0FB4    Load the MMU reg. with the
007F:  01 0A D5    LD  BC,$D50A    Values at
0082:  16 0B       LD  D,$0B       $0FAA
0084:  7E          LD  A,(HL)      Note that the
0085:  ED 79       OUT (C),A       11 MMU registers
0087:  2B          DEC HL          Are loaded with the values
0088:  0D          DEC C           At $0FB4 downwards!
0089:  15          DEC D
008A:  20 F8       JR  NZ,$-6 >$0084 End of the loop
008C:  21 1A 0D    LD  HL,$0D1A    Copy the area from $0D1A
008F:  11 00 11    LD  DE,$1100    To $1100
0092:  01 08 00    LD  BC,$0008    Copy eight bytes
0095:  ED B0       LDIR            (8502 code!)
0097:  21 E5 0E    LD  HL,$0EE5    Also copy the area
009A:  11 D0 FF    LD  DE,$FFD0    From $0EE5 to the common
```

```
009D:  01 1F 00    LD  BC,$001F    Area at $FFD0
00A0:  ED B0       LDIR            Copy 31 bytes
00A2:  21 00 11    LD  HL,$1100    $1100 as jump vector
00A5:  22 FA FF    LD  ($FFFA),HL  Copy jump vector in
00A8:  22 FC FF    LD  ($FFFC),HL  All four addresses
00AB:  22 FE FF    LD  ($FFFE),HL  Including address
00AE:  22 DD FF    LD  ($FFDD),HL  $FFDD (just copied!)
00B1:  C3 E0 FF    JP  $FFE0       And jump to the Z-80 part
```

The following section is copied to $FFD0 at the start and contains 8502 code to switch over to the Z-80 mode:

```
**************************  also copy to $FFD0

0EE5: 78          SEI          Disable interrupts
0EE6: A9 3E       LDA #$3E     Configuration index
0EE8: 8D 00 FF    STA $FF00    Set configuration index
0EEB: A9 B0       LDA #$B0     Enable Z-80
0EED: 8D 05 D5    STA $D505    Write to mode config. register
0EF0: EA          NOP          Delay
0EF1: 4C 00 30    JMP $3000    Jump to continuation
0EF4: EA          NOP
```

The jump at address $0EF1 is changed or replaced by a RETURN in most cases.

The following section--again in Z-80 mnemonics--is also copied to $FFE0. The RST 0 routine jumps to this address when it is done. Then the computer is again in the 8502 mode. If the Z-80 is re-enabled, the Z-80 continues at precisely the same location (NOP).

```
**************************  This area is copied to $FFE0

0EF5: F3          DI           Disable interrupts
0EF6: 3E 3E       LDA #$3E     Configuration index
0EF8: 32 00 FF    STA $FF00    Into configuration register
0EFB: 01 05 D5    LD BC,$D505  Mode configuration register
0EFE: 3E B1       LD  A,$B1    Enable 8502
0F00: ED 79       OUT (C),A    Into mode config. register
0F02: 00          NOP          Delay
0F03: CF          RST $08      Continuation
```

The address $0F03 is found at address $FFEE after the copy. If you want to run your own Z-80 program, you must define a jump to your routine at this point. In our example, our Z-80 program is located at address $3000. We must then branch to this routine at address $FFEE:

```
FFEE: C3 00 30 JMP $3000 branch to routine
```

To enable the Z-80 in 8502 assembly language, you should call the routine at address $FFD0. To enable the 8502 in Z-80 assembly language,you should call the routine at $FFE0 to enable the 8502 when the Z-80 is running.

7.7 Boot Sector and Boot Routine

Those of you who have worked with an IBM PC are well aware of the advantage of a boot sector. The first thing to clarify is what a "boot" has to do with a modern computer like the C-128. The answer is not a difficult one. As an article of clothing, the boot is the "lowest part" of a person. It has the actual contact to the ground on which we walk and stand. The boot sector of a computer is similar. It is also the lowest part of a program, the connection between the computer program and the machine.

When you turn your C-128 on, you will notice that the disk drive (assuming you have one) makes some noises and then is quiet. Even when you have inserted a disk, the disk drive always runs before the computer responds.

The reason for this action is that the computer tries to load this so-called "boot sector". This sector can be used to load a program as soon as the computer is turned on, without the user having to press a single key. The boot sector can also be a program of its own, which is then started automatically. This sector has many uses, but in order to make full use of it, it is important to be familiar with the internal structure of the sector and the action of the boot routine.

Since the boot routine is controlled by the operating system and cannot search the entire diskette for such a sector, there is only one pre-determined place on the diskette that can be used as a boot sector. This is:

Side 1, track 01, sector 00

But be careful since this sector is also physically the first data block on a diskette, it's possible that this space is already used by other files. Before you install a boot sector on a diskette, you should always check to see if this sector is already occupied.

In order to be able to understand the makeup of the boot sector, you should become familiar with the operation of the boot routine. This kernal routine performs the following steps:

1) A block-read command to track 1, sector 0 is constructed in the DOS buffer of the expanded zero page.

2) The command is executed and the block read (provided a formatted disk is in the drive) is loaded into the cassette buffer.

3) The first three bytes of the block are checked to see if they contain the required identification code for a boot sector. This identification code is **CBM**. If this code is not present, the boot routine is stopped.

4) The four bytes following the CBM code are loaded into four zero-page pointers. Generally these 4 bytes are set to the value $00. The first two bytes can contain a starting address, which has nothing to do with the address at which the program is to be loaded. The third byte is the corresponding configuration index of the start address. But all of the first three entries are ignored if the fourth byte contains the value $00. It contains the number of blocks, in addition to the boot sector, that are to be loaded from the disk.

5) Independent from whether the block counter in the boot block is set or not, the bytes following these four address and control bytes are read and displayed on the screen via the BSOUT routine. Here the screen can be cleared or an appropriate boot-up message can be displayed. This character output continues until the computer comes across a byte with the value $00.

6) Now the control bytes read in step 4 have a meaning. If the block counter is set to zero, this routine is skipped. If this is not the case a new command string is formed in the DOS buffer which instructs the drive to load another boot block from the diskette. The determination of this boot block is quite simple. The sector number is incremented by 1. If the sector number is greater than 20 (there is a maximum of only 21 sectors per track, numbered 0-20), the track number is incremented by 1 and the sector number is reset to 0. A block-read command to read

this block is executed, whereby the block read is stored at the address and configuration created by the first three bytes. The memory address of the following boot blocks is incremented and the block counter is decremented by 1. This is done until the block counter is counted down to zero.

7) The boot routine then returns to the code following the text constants (if present) in the original boot sector in the cassette buffer. A filename, as indicated in the disk directory, may reside here. Except the fact that the characters of the filename are not displayed on the screen, all of the bytes here are read until the boot routine encounters the $00 terminating code. The length of the filename is recorded in a counter.

8) Now we come to another option. If the length of the filename in the counter is a value other than zero, the characters "0:" are prefixed to the filename. Then the filename counter is incremented by 2, and a branch is made to the kernal LOAD routine in order to read this program into memory. If this happens, or if the length of the filename is zero, the boot routine goes back to behind the code $00 indicating the end of the filename.

9) The bytes following the filename are interpreted as a machine language program and the boot routine passes control to this program. From this point on, the programmer is responsible for starting the program loaded, or for loading another program, or for branching to another of the boot blocks.

If you make note of the above steps when creating your own boot sectors, you will soon see that it is not difficult, provided you know what the operating system expects. Here again are the most important points and instructions:

Bytes 0,1,2 : CBM identification code
Bytes 3,4 : Memory address for the following boot sectors
Byte 5 : Configuration index for the following boot sectors
Byte 6 : Block counter for the number of following boot sectors
Byte 7 to 1st terminating code ($00) : boot message
Name of the program to load, followed by the second terminator ($00)
Your own machine language program entry

Address of the boot sector: Side 0, track 1, sector 0

CHAPTER 8

Chapter 8: The ROM Listing

The ROM listing is probably the most important tool for the real machine language programmer. For those of you who don't know what we mean by the term "ROM listing," it is simply this; the operating system is found in ROM. If this operating system is disassembled, the result is called a ROM listing.

The real art is not in reading the operating system and disassembling it, but in documenting it. The documentation should make it simpler for the reader to make use of the individual routines. You can find more information about the most important kernal routines in Chapter 7.

The entire operating system comprises a total of 44Kbytes in the Commodore 128. 28K of this is for the BASIC and the other 16K is for the kernal. This book documents the *kernal*. A complete documentation of the whole 44K would far exceed the capacity of a single book.

The *kernal* contains the most important elementary routines which the computer needs to display characters on the screen, decode the keyboard, control the cassette recorder, etc.

Below are some of the abbrevations used in the the ROM listings.

pntr.	pointer	disp.	display
krnl.	kernal	addr.	address
w/	with	f/	from
clr	clear	dev.	device
acc.	accumulator	sys.	system
char.	character	subt.	subtract
inc.	increment	dec.	decimal
decr.	decrement	Z-P	zero page
y-reg.	y-register	x-reg.	x-register
rout.	routine	#	number
prgm	program	ctrl.	control
cmd.	command	max.	maximum
crsr.	cursor	bnk.	bank

8.1 ROM Listings Start $FB00

```
****************************      Monitor entry vectors
B000:  4C 21 B0   JMP  $B021      Regular monitor entry
B003:  4C 09 B0   JMP  $B009      Monitor BREAK entry
B006:  4C B2 B0   JMP  $B0B2      Exmon monitor entry
B009:  20 7D FF   JSR  $FF7D      Kernal PRINT: string output

****************************      Initial monitor message
                                  produced by BREAK entry

B00C:  0D 42 52 45 41 4B 07 00    <C/R> BREAK <Bell>

****************************      Monitor initalization after
                                  BREAK entry
B014:  68         PLA             Place BANK no. on stack in
B015:  85 02      STA  * $02      appropriate zero-page byte
B017:  A2 05      LDX  # $05      Get the contents of x-reg, y-reg,
B019:  68         PLA             accumulator, processor status
B01A:  95 03      STA  * $03,X    & program counter from stack &
B01C:  CA         DEX             put in corresponding zero-page
B01D:  10 FA      BPL  $B019      bytes.
B01F:  30 25      BMI  $B046      Jump to general initialization

****************************      Initialization for regular entry

B021:  A9 00      LDA  # $00      Load configuraton register with
B023:  8D 00 FF   STA  $FF00      $00 and enable all system ROMs
B026:  85 06      STA  * $06      Clear zero-page memory for acc.
B028:  85 07      STA  * $07      Clear Z-P memory for x-reg
B02C:  85 05      STA  * $05      Clr memory for processor status
B02E:  A9 00      LDA  # $00      Load Acc.- lo-addr for monitor
B030:  A0 B0      LDY  # $B0      Load Y-reg with hi-addr monitor
B032:  85 04      STA  * $04      Acc in memory: prgm counter lo
B034:  84 03      STY  * $03      Y-reg in memory: prgm cntr hi
B036:  A9 0F      LDA  # $0F      Set Z-P memory for BANK# at
B038:  85 02      STA  * $02      $0F-Krnl+BASIC,RAM 0, I/O
B03A:  20 7D FF   JSR  $FF7D      Kernal PRINT: string output
```

```
******************************   Text constants for initial monitor
                                 message

B03D:  0D 4D 4F 4E 49 54 4F 52   <C/R> MONITOR
B045:  00

******************************   General monitor initalization

B046:  D8         CLD            Reset decimal mode
B047:  BA         TSX            Store stack pntr in X-reg
B048:  86 09      STX  * $09     and in memory for stack pointer
B04A:  A9 C0      LDA  # $C0     Sys/control messages enabled
B04C:  20 90 FF   JSR  $FF90     Kernal SETMSG:Sys/ctrl-messages
B04F:  58         CLI            All system interrupts enabled

******************************   Monitor command: R
                                 (Register contents)
B050:  20 7D FF   JSR  $FF7D     Kernal PRINT: output string

******************************   Text constants for processor
                                 memory

B053:  0D 20 20 20 20 50 43 20   C/R PC SR AC XR YR SP C/R
B05B:  20 53 52 20 41 43 20 58
B063:  52 20 59 52 20 53 50 0D
B06B:  3B 20 1B 51 00            ; <Esc - Q>

******************************   Output contents of registers, st
                                 stacks, & prgm cntr status

B070:  A5 02      LDA  * $02     Get current BANK # in acc.
B072:  20 D2 B8   JSR  $B8D2     Acc.= 2-byte ASCII: hi=A,lo=X
B075:  8A         TXA            ASCII for lower nibble in Accu
B076:  20 D2 FF   JSR  $FFD2     Kernal BSOUT: output a char
B079:  A5 03      LDA  * $03     Z-P memory for PC hi in accu
B07B:  20 C2 B8   JSR  $B8C2     Acc in 2-byte ASCII and output
B07E:  A0 02      LDY  # $02     Displ. points to ZP byte - PC Lo
B080:  B9 02 00   LDA  $0002,Y   PC Lo, P, A, X, Y, S in Accu
B083:  20 A5 B8   JSR  $B8A5     Acc output as 2-byte
                                 ASCII+<BLANK>
B086:  C8         INY            Increment displ.
```

```
B087:  C0 08      CPY  # $08      Bytes $04-$09 already output?
B089:  90 F5      BCC  $B080      no, then read next byte
B08B:  20 B4 B8   JSR  $B8B4      Linefeed + clear rest of line
B08E:  A2 00      LDX  # $00      Displacement pntr, input buffer
B090:  86 7A      STX  * $7A      reset to 0
B092:  20 CF FF   JSR  $FFCF      Kernal BASIN: read out char
B095:  9D 00 02   STA  $0200,X    & put in monitor input buffer
B098:  E8         INX             Displ. increment to input buffer
B099:  E0 A1      CPX  # $A1      Have 160 chars been printed?
B09B:  B0 1F      BCS  $B0BC      yes, then output error message
B09D:  C9 0D      CMP  # $0D      <RETURN> entered?
B09F:  D0 F1      BNE  $B092      no, then wait for next character
B0A1:  A9 00      LDA  # $00      When <RETURN> entered,mark
B0A3:  9D FF 01   STA  $01FF,X    command-string end with $00.
B0A6:  20 E9 B8   JSR  $B8E9      Test input buffer for cmd end,
B0A9:  F0 E0      BEQ  $B08B      If <:>,<?>, cmd end, wait input.
B0AB:  C9 20      CMP  # $20      Was character a <SPACE> ?
B0AD:  F0 F7      BEQ  $B0A6      Read next character.
B0AF:  6C 2E 03   JMP  ($032E)    Vector to MONITOR routine
B0B2:  A2 15      LDX  # $15      Number of keywords in X-reg
B0B4:  DD E6 B0   CMP  $B0E6,X    compared with keyword table.
B0B7:  F0 0C      BEQ  $B0C5      If found, go to keyword table
B0B9:  CA         DEX             pointer -- decrement by 1, until
B0BA:  10 F8      BPL  $B0B4      entire table is searched
B0BC:  20 7D FF   JSR  $FF7D      Kernal PRINT: output

******************************    ? constant for monitor error
                                  messages
B0BF:  1D 3F 00                   <Crsr Right> ?

******************************    Return to input wait loop

B0C2:  4C 8B B0   JMP  $B08B      jump to input wait loop

******************************    Establish monitor command
                                  addresses

B0C5:  E0 13      CPX  # $13      Is keyword <L>, <S>, <V>?
B0C7:  B0 12      BCS  $B0DB      yes, then perform task
B0C9:  E0 0F      CPX  # $0F      Is keyword a conversion char?
```

```
B0CB:  B0 13      BCS  $B0E0     ($,+,&,%) YES--then do task.
B0CD:  8A         TXA            Keyword number to accu and
B0CE:  0A         ASL  A         multiplied by 2
B0CF:  AA         TAX            This value as offset in X-reg
B0D0:  BD FD B0   LDA  $B0FD,X   Monitor routine (hi) addr. got
B0D3:  48         PHA            & treated as quasi-RTS on stack.
B0D4:  BD FC B0   LDA  $B0FC,X   Monitor routine (hi) addr. got
B0D7:  48         PHA            & treated as quasi RTS on stack.
B0D8:  4C A7 B7   JMP  $B7A7     Command parameter utilization.

*****************************    Release LSV and conversions

B0DB:  85 93      STA  * $93     Store char of command keyword
B0DD:  4C 37 B3   JMP  $B337     Execution of L,S,V commands
B0E0:  4C B1 B9   JMP  $B9B1     Execution of conversion chars.

*****************************    Monitor command: X (Exit)

B0E3:  6C 00 0A   JMP  ($0A00)   Vector: BASIC warm-start
                                 ($4003)

*****************************    Monitor keywords

B0E6:  41 43 44 46 47 48 4A 4D   A C D F G H J M
B0EE:  52 54 58 40 2E 3E 3B 24   R T X @ . > ; $
B0F6:  2B 26 25 4C 53 56         + & % L S V

*****************************    Addresses of monitor
                                 commands (-1)

B0FC:  05 B4      ($B406)        A = Assemble
B0FE:  30 B2      ($B231)        C = Compare
B100:  98 B5      ($B599)        D = Disassemble
B102:  DA B3      ($B3DB)        F = Fill
B104:  D5 B1      ($B1D6)        G = Go to
B106:  CD B2      ($B2CE)        H = Hunt
B108:  DE B1      ($B1DF)        J = Jump
B10A:  51 B1      ($B152)        M = Monitor
B10C:  4F B0      ($B050)        R = Register
B10E:  33 B2      ($B234)        T = Transfer
B110:  E2 B0      ($B0E3)        X = Exit
```

```
B112:  8F BA      ($BA90)      @ = Disc Command
B114:  05 B4      ($B406)      . = Assemble
B116:  AA B1      ($B1AB)      > = Modify Memory
B118:  93 B1      ($B194)      ; = Modify Register

*****************************  LDA routine for acc from any
                               bank  FETVEC=bank byte of the
                               OP3 operand

B11A:  8E B2 0A   STX  $0AB2   X-reg temporary storage
B11D:  A6 68      LDX  * $68   Bank no. taken from OP3
B11F:  A9 66      LDA  # $66   FETVEC addr. for indfet in A
B121:  78         SEI          All system interrupts disabled
B122:  20 74 FF   JSR  $FF74   Kernal INDFET:LDA(fetvec), Y
                               any bank
B125:  58         CLI          All system interrupts enabled
B126:  AE B2 0A   LDX  $0AB2   X-reg loaded with saved value
B129:  60         RTS          Return from subroutine

*****************************  STA routine places acc contents
                               in any bank.
                               STAVEC=OP3 bank byte

B12A:  8E B2 0A   STX  $0AB2   X-reg temporary storage
B12D:  A2 66      LDX  # $66   Load STAVEC (lo addr) into
B12F:  8E B9 02   STX  $02B9   X-reg and put Indsta routine in
                               STAVEC
B132:  A6 68      LDX  * $68   Get bank # from 'from' OP3
B134:  78         SEI          All system interrupts disabled
B135:  20 77 FF   JSR  $FF77   Kernal INDSTA:STA(stavec), Y
                               bank
B138:  58         CLI          All system interrupts enabled
B139:  AE B2 0A   LDX  $0AB2   X-Reg loaded with stored value
B13C:  60         RTS          Return from subprogram

*****************************  CMP routine--acc contents w/
                               specified bank.
                               CMPVEC=OP3 bank byte

B13D:  8E B2 0A   STX  $0AB2   X-reg temp. storage
```

```
B140:  A2 66      LDX  # $66      Load CMPVEC addr in Y-reg &
B142:  8E C8 02   STX  $02C8      CMPVEC mem for Indcmp
B145:  A6 68      LDX  * $68      Get bank # 'from' OP3
B147:  78         SEI             All system interrupts disabled
B148:  20 7A FF   JSR  $FF7A      Kernal INDCMP:
                                  CMP(CMPVEC), Y bank
B14B:  58         CLI             All system interrupts enabled
B14C:  08         PHP             Secure result of CMP
B14D:  AE B2 0A   LDX  $0AB2      X-Reg loaded w/ secured value
B150:  28         PLP             Set back comparison result
B151:  60         RTS             Return from subprogram

******************************    Monitor command: M
                                  (Memory display)

B152:  B0 08      BCS  $B15C      No parameter, then set default
B154:  20 01 B9   JSR  $B901      Copy contents of OP1 into OP3
B157:  20 A7 B7   JSR  $B7A7      Get 'to' in OP1
B15A:  90 06      BCC  $B162      Convey from-to step number
B15C:  A9 0B      LDA  # $0B      Load OP1 (lo) with default
B15E:  85 60      STA  * $60      load step count 12
B160:  D0 15      BNE  $B177      Goto exec. of memory display
B162:  20 0E B9   JSR  $B90E      Difference: OP1-OP3  in OP1
B165:  90 2A      BCC  $B191      If 'from'>'to' then ERROR
B167:  A2 03      LDX  # $03      Step # divided by 2 three times
B169:  24 D7      BIT  * $D7      Check for 40/80-col. mode
B16B:  10 01      BPL  $B16E      40-col, to step division
B16D:  E8         INX             80-col, to step number
B16E:  46 62      LSR  * $62      Div. of OP1 (3-byte operand)
B170:  66 61      ROR  # $61      by 2, for memory display values
B172:  66 60      ROR  # $60      of 8 or 16.
B174:  CA         DEX             Division # for step #-1
B175:  D0 F7      BNE  $B16E      OP1 divided by 8/16
B177:  20 E1 FF   JSR  $FFE1      Kernal STOP: test for STOP key
B17A:  F0 12      BEQ  $B18E      STOP pressed, go EXIT routine
B17C:  20 E8 B1   JSR  $B1E8      Display a line of memory
B17F:  A9 08      LDA  # $08      + constant from 'from' operand
B181:  24 D7      BIT  * $D7      Check for 40/80-col. mode
B183:  10 01      BPL  $B186      40-col, add constant of 8 OK
B185:  0A         ASL  A          80-col, add constant *2 (=16)
```

```
B186:  20 52 B9   JSR  $B952      Addition: Acc contents + OP3
B189:  20 22 B9   JSR  $B922      Subtraction: OP1 - constant <1>
B18C:  B0 E9      BCS  $B177      Loop, 'til OP1 < 0
B18E:  4C 8B B0   JMP  $B08B      Jump to input wait loop
B191:  4C BC B0   JMP  $B0BC      <?> Output and go to input wait
                                  loop

******************************    Monitor command : ;
                                  (Modify reg)

B194:  20 74 B9   JSR  $B974      C=0--OP1 in ZP
                                  bank/PCHi/PCLo
B197:  A0 00      LDY  # $00      Set displacement for zero page
B199:  20 A7 B7   JSR  $B7A7      Get OP1's modifier
B19C:  B0 0A      BCS  $B1A8      Carry set=identifier for exit rout.
B19E:  A5 60      LDA  * $60      Get lo-add from OP1 as modifer
B1A0:  99 05 00   STA  $0005,Y    Modify status;B,A,X,Y stat. ptr.
B1A3:  C8         INY             Display Z-P CPU memory +1
B1A4:  C0 05      CPY  # $05      All CPU memory changed?
B1A6:  90 F1      BCC  $B199      no, then jump to next routine
B1A8:  4C 8B B0   JMP  $B08B      Jump to input wait loop

******************************    Monitor command:
                                  > (Modify mem)

B1AB:  B0 1C      BCS  $B1C9      No parameter, then no change
B1AD:  20 01 B9   JSR  $B901      Copy contents of OP1 into OP3
B1B0:  A0 00      LDY  # $00      Set modify display ptr. to 0
B1B2:  20 A7 B7   JSR  $B7A7      Get modify value in OP1
B1B5:  B0 12      BCS  $B1C9      No other value=print line
B1B7:  A5 60      LDA  * $60      Get value from OP1 (low)
B1B9:  20 2A B1   JSR  $B12A      STA routine in any bank
B1BC:  C8         INY             Display pntr for modify byte+1
B1BD:  24 D7      BIT  * $D7      Test for 40/80-col. mode
B1BF:  10 04      BPL  $B1C5      Max. param reading of 40 chars.
B1C1:  C0 10      CPY  # $10      16 chars. read/changed?
B1C3:  90 ED      BCC  $B1B2      no, goto next parameter
B1C5:  C0 08      CPY  # $08      8 chars read/changed?
B1C7:  90 E9      BCC  $B1B2      no, get next parameter
B1C9:  20 7D FF   JSR  $FF7D      Kernal PRINT: output string
```

```
*******************************   Clear insert, RVS, quote modes

B1CC:  1B 4F 91 00                <Esc - O> <Crsr Up>

*******************************   Display changed memory line

B1D0:  20 E8 B1   JSR  $B1E8      Outputs:<8/16 hex values, 8/16
                                  ASCII
B1D3:  4C 8B B0   JMP  $B08B      Jump to input wait loop

*******************************   Monitor command : G  (Go to)

B1D6:  20 74 B9   JSR  $B974      C=0 OP1 in zeropage
                                  bank/PCHi/PCLo
B1D9:  A6 09      LDX  * $09      Load X w/ Z-P byte for stack ptr
B1DB:  9A         TXS             Modify stack ptr. w/ X-reg.
B1DC:  4C 71 FF   JMP  $FF71      Krnl JMPFAR: JMP to any bank

*******************************   Monitor command : J (Jump to)

B1DF:  20 74 B9   JSR  $B974      C=0 OP1 in zero page
                                  bank/PCHi/PCLo
B1E2:  20 6E FF   JSR  $FF6E      Kernal JSRFAR:JSR
B1E5:  4C 8B B0   JMP  $B08B      Jump to input wait loop

*******************************   Display '<',8/16 hex values &
                                  8/16 ASCII characters for
                                  memory display

B1E8:  20 B4 B8   JSR  $B8B4      Line feed + clear rest of line
B1EB:  A9 3E      LDA  # $3E      Load acc with '<' char.
B1ED:  20 D2 FF   JSR  $FFD2      Kernal BSOUT: output one char
B1F0:  20 92 B8   JSR  $B892      Output OP3 in 5-byte ASCII
B1F3:  A0 00      LDY  # $00      Loop # set to 0
B1F5:  F0 03      BEQ  $B1FA      1 hex value skip space
B1F7:  20 A8 B8   JSR  $B8A8      Output <SPACE> <CR>
B1FA:  20 1A B1   JSR  $B11A      <Crsr-up>.LDA from any bank
B1FD:  20 C2 B8   JSR  $B8C2      A displayed as 2-byte ASCII
B200:  C8         INY             Loop+displacement #+1
B201:  C0 08      CPY  # $08      8 hex values printed?
```

```
B203:  24 D7      BIT  * $D7      Test for 40/80-col. screen
B205:  10 02      BPL  $B209      Output to 40-col.
B207:  C0 10      CPY  # $10      16 hex values printed?
B209:  90 EC      BCC  $B1F7      Get next hex value
B20B:  20 7D FF   JSR  $FF7D      Kernal PRINT: output string

******************************    Constant: colon, RVS-on

B20E:  3A 12 00                   : <Rvs On>

******************************    Output 8/16 bytes in ASCII

B211:  A0 00      LDY  # $00      Loop and display counter to 0
B213:  20 1A B1   JSR  $B11A      LDA from any bank
B216:  48         PHA             Put char. on stack
B217:  29 7F      AND  # $7F      Mask bit 7 (no RVS char.)
B219:  C9 20      CMP  # $20      Check for ctrl char.
B21B:  68         PLA             Get char. from stack again
B21C:  B0 02      BCS  $B220      Not ctrl char, then normal output
B21E:  A9 2E      LDA  # $2E      Load accumulator with <.>
B220:  20 D2 FF   JSR  $FFD2      Kernal BSOUT: outpt character
B223:  C8         INY             Loop & displacement counter +1
B224:  24 D7      BIT  * $D7      Check for 40/80-col. screen
B226:  10 04      BPL  $B22C      Continue display if 40-col.
B228:  C0 10      CPY  # $10      16 characters printed?(80-col)
B22A:  90 E7      BCC  $B213      no, output next char.
B22C:  C0 08      CPY  # $08      8 characters printed? (40-col)
B22E:  90 E3      BCC  $B213      no, print next char.
B230:  60         RTS             Return to subroutine

******************************    Monitor command : C
                                  (Compare)

B231:  A9 00      LDA  # $00      Set char. for COMPARE
B233:  2C        .Byte   $2C      skip to $B236

******************************    Monitor command : T
                                  (Transform)

B234:  A9 80      LDA  # $80      Set TRANSFORM marker
```

```
B236:  85 93      STA  * $93      and put into cmd byte memory
B238:  A9 00      LDA  # $00      Direction ptr for C/T cmd to $00
B23A:  8D B3 0A   STA  $0AB3      (=forward) set ($80=backward)
B23D:  20 83 B9   JSR  $B983      Get 'til' & step cnt in OPH,OP2
B240:  B0 05      BCS  $B247      Carry set= error marker found
B242:  20 A7 B7   JSR  $B7A7      Get 'to'/'with' (in OP1)
B245:  90 03      BCC  $B24A      'To'/'with' operand is OK
B247:  4C BC B0   JMP  $B0BC      <?> displayed go input wait loop
B24A:  24 93      BIT  * $93      Was it transferred (-) or com-
B24C:  10 2C      BPL  $B27A      pared (+) in CMP routine?
B24E:  38         SEC             Set carry for subtraction
B24F:  A5 66      LDA  * $66      Test whether contents of both
B251:  E5 60      SBC  * $60      bytes (addr lo), (addr hi) are
B253:  A5 67      LDA  * $67      larger than operand OP3, or the
B255:  E5 61      SBC  * $61      address bytes of OP1.
B257:  B0 21      BCS  $B27A      'To'<'from'=direction OK
B259:  A5 63      LDA  * $63      Add the contents of the 3-byte
B25B:  65 60      ADC  * $60      operand OP2 in locations
B25D:  85 60      STA  * $60      $65-$64-$63 to the contents of
B25F:  A5 64      LDA  * $64      the 3-byte operand OP1
B261:  65 61      ADC  * $61      in locations $62-$61-$60.
B263:  85 61      STA  * $61      Put any addition overflow
B265:  A5 65      LDA  * $65      results in OP1.
B267:  65 62      ADC  * $62      Store addition result
B269:  85 62      STA  * $62      in OP1
B26B:  A2 02      LDX  # $02      Copy the contents of the
B26D:  BD B7 0A   LDA  $0AB7,X    3-byte help operands
B270:  95 66      STA  * $66,X    in memory locations
B272:  CA         DEX             $0AB9-$0AB8-$0AB7 into the
B273:  10 F8      BPL  $B26D      operand OP3 ($68-$67-$66)
B275:  A9 80      LDA  # $80      When 'til' is greater than 'from'
B277:  8D B3 0A   STA  $0AB3      set direction marker to backward
B27A:  20 B4 B8   JSR  $B8B4      <CR> & clear rest of line
B27D:  A0 00      LDY  # $00      Set displacement ptr. to 0
B27F:  20 E1 FF   JSR  $FFE1      Kernal STOP: check STOP key.
B282:  F0 47      BEQ  $B2CB      If STOPkey goto Exit routine.
B284:  20 1A B1   JSR  $B11A      LDA from any bank
B287:  A2 60      LDX  # $60      $60 is lo-addr. 'with' 'til'-OP1
B289:  8E B9 02   STX  $02B9      Set STAVEC at this addr.
B28C:  8E C8 02   STX  $02C8      Set CMPVEC at this addr.
```

```
B28F:  A6 62      LDX  * $62      Load X-reg w/ bank byte 'til'
B291:  78         SEI               All system interrupts disabled
B292:  24 93      BIT  * $93      Was it transfer or comparison?
B294:  10 03      BPL  $B299      Compare in appropriate routine
B296:  20 77 FF   JSR  $FF77      Kernal INDSTA:
                                  STA(STAVEC), Y any bank
B299:  A6 62      LDX  * $62      Load X w/ bank byte 'with'
B29B:  20 7A FF   JSR  $FF7A      Kernal INDCMP:
                                  CMP(CMPVEC),Y any bank
B29E:  58         CLI             All system interrupts enabled
B29F:  F0 09      BEQ  $B2AA      'Equal' not given, and
B2A1:  20 92 B8   JSR  $B892      OP3 output as 5-byte ASCII
B2A4:  20 A8 B8   JSR  $B8A8      <SPACE>,<C/R>,<crsr-up>
B2A7:  20 A8 B8   JSR  $B8A8      output
B2AA:  2C B3 0A   BIT  $0AB3      Test for transfer direction
B2AD:  30 0B      BMI  $B2BA      Send new return address
B2AF:  E6 60      INC  * $60      Fwd. transfer of 'til' address
B2B1:  D0 10      BNE  $B2C3      raised by 1 and monitored
B2B3:  E6 61      INC  * $61      for overflow
B2B5:  D0 0C      BNE  $B2C3      If hi-addr. overflow, then error
B2B7:  4C BC B0   JMP  $B0BC      <?> output - to input wait loop
B2BA:  20 22 B9   JSR  $B922      Subtraction: OP1 - constant <1>
B2BD:  20 60 B9   JSR  $B960      Subtraction: OP3 - constant <1>
B2C0:  4C C6 B2   JMP  $B2C6      Jump to subtraction OP2 - <1>

******************************    Set step number & 'from'

B2C3:  20 50 B9   JSR  $B950      Addition: constant <1> to OP3
B2C6:  20 3C B9   JSR  $B93C      Subtraction: OP2 - constant <1>
B2C9:  B0 B4      BCS  $B27F      Loop until all steps done
B2CB:  4C 8B B0   JMP  $B08B      Jump to input wait loop

******************************    Monitor command : H  (Hunt)

B2CE:  20 83 B9   JSR  $B983      Get 'til' step value in OP1
B2D1:  B0 61      BCS  $B334      Carry set=identifier - found error
B2D3:  A0 00      LDY  # $00      Display hunt char in CMP buffer
B2D5:  20 E9 B8   JSR  $B8E9      Read a char from input buffer
B2D8:  C9 27      CMP  # $27      Was character read a <.> ?
B2DA:  D0 16      BNE  $B2F2      no, don't look for string
```

```
B2DC:  20 E9 B8   JSR  $B8E9      Read a character to input buffer
B2DF:  C9 00      CMP  # $00      Has command-end been found?
B2E1:  F0 51      BEQ  $B334      yes, then output error <?>
B2E3:  99 80 0A   STA  $0A80,Y    Put char in CMP buffer
B2E6:  C8         INY             Displace CMP buffer +1
B2E7:  20 E9 B8   JSR  $B8E9      Test input buffer for cmd-end,
B2EA:  F0 1B      BEQ  $B307      <:>, <?>; if so, execute HUNT
B2EC:  C0 20      CPY  # $20      32 in CMP buffer?
B2EE:  D0 F3      BNE  $B2E3      no, get next CMP value for
B2F0:  F0 15      BEQ  $B307      hunt routine
B2F2:  8C 00 01   STY  $0100      Store displ. in CMP buffer
B2F5:  20 A5 B7   JSR  $B7A5      Put CMP operand in OP1
                                  (like CHRGOT)
B2F8:  A5 60      LDA  * $60      Transmit OP1 byte into
B2FA:  99 80 0A   STA  $0A80,Y    CMP buffer
B2FD:  C8         INY             Displace CMP buffer +1
B2FE:  20 A7 B7   JSR  $B7A7      Get more CMP values in OP1
B301:  B0 04      BCS  $B307      NONE FOUND-execute HUNT
B303:  C0 20      CPY  # $20      32 values in CMP buffer?
B305:  D0 F1      BNE  $B2F8      No, get next CMP value
B307:  84 93      STY  * $93      Store cnt of CMP buffer values
B309:  20 B4 B8   JSR  $B8B4      <CR> & clear rest of line
B30C:  A0 00      LDY  # $00      Display 1st char in CMP buffer
B30E:  20 1A B1   JSR  $B11A      LDA from any bank
B311:  D9 80 0A   CMP  $0A80,Y    CMP w/ char from CMP buffer
B314:  D0 0E      BNE  $B324      Unequal--on to next step
B316:  C8         INY             Display next CMP buffer value
B317:  C4 93      CPY  * $93      All individual comps run?
B319:  D0 F3      BNE  $B30E      No, next step of comparison
B31B:  20 92 B8   JSR  $B892      Contents of OP3, 5-byte ASCII
B31E:  20 A8 B8   JSR  $B8A8      <SPACE>, <CR>, <crsr-up>
B321:  20 A8 B8   JSR  $B8A8      <SPACE>, <CR>, <crsr-up>
B324:  20 E1 FF   JSR  $FFE1      Kernal STOP: check STOP key
B327:  F0 08      BEQ  $B331      If STOP, goto Exit routine.
B329:  20 50 B9   JSR  $B950      Addition: constant <1> to OP3
B32C:  20 3C B9   JSR  $B93C      Subtraction: OP2 - constant <1>
B32F:  B0 DB      BCS  $B30C      Loop until all steps done
B331:  4C 8B B0   JMP  $B08B      Jump to input wait loop
B334:  4C BC B0   JMP  $B0BC      Output <?> -to input wait loop
```

```
*****************************   Jumps to monitor commands:
                                L = Load, S = Save, V = Verify

B337:  A0 01      LDY  # $01    Load Y-reg with $01
B339:  84 BA      STY  * $BA    Set device number (1=Datasette)
B33B:  84 B9      STY  * $B9    Set secondary address (1=write)
B33D:  88         DEY           Y-reg counts down to $00
B33E:  84 C6      STY  * $C6    Set BANK no. for LSV call
B340:  84 B7      STY  * $B7    Length of filename set to 0
B342:  84 C7      STY  * $C7    Set BANK for addr. of filename
B344:  84 90      STY  * $90    Clear status byte (0 = all OK)
B346:  A9 0A      LDA  # $0A    Zero-page memory for hi addr.
B348:  85 BC      STA  * $BC    of filename loaded w/ $0A
B34A:  A9 80      LDA  # $80    Zero-page memory for lo addr.
B34C:  85 BB      STA  * $BB    of Filename w/ $80 (= $0A80)
B34E:  20 E9 B8   JSR  $B8E9    Test input buffer;
B351:  F0 58      BEQ  $B3AB    If cmd-end; go to input loop
B353:  C9 20      CMP  # $20    Was char. read a <SPACE>?
B355:  F0 F7      BEQ  $B34E    Yes, continue, read next char.
B357:  C9 22      CMP  # $22    Was char. a <">?
B359:  D0 15      BNE  $B370    No, error in command string
B35B:  A6 7A      LDX  * $7A    X-reg loaded w/ display from
                                input buffer
B35D:  BD 00 02   LDA  $0200,X  Read 1st " in-buffer(=filename)
B360:  F0 49      BEQ  $B3AB    $00 = End of command string
B362:  E8         INX           Input buffer pointer to next char.
B363:  C9 22      CMP  # $22    Has 2nd <"> been found?
B365:  F0 0C      BEQ  $B373    Yes, further evaluation
B367:  91 BB      STA  ($BB),Y  Filename placed at $0A80
B369:  E6 B7      INC  * $B7    Counter for filename length + 1
B36B:  C8         INY           Filename memory pntr increment
B36C:  C0 11      CPY  # $11    Filename longer than 16 chars ?
B36E:  90 ED      BCC  $B35D    No, read next character
B370:  4C BC B0   JMP  $B0BC    Display <?> &go input wait loop

*****************************   LSV parameter evaluation after
                                2nd <">

B373:  86 7A      STX  * $7A    Input buffer pointer after 2nd "
B375:  20 E9 B8   JSR  $B8E9    Check buffer cmd-end, <:><?>
```

```
B378:  F0 31      BEQ  $B3AB     LV can run w/o parameters
B37A:  20 A7 B7   JSR  $B7A7     Get parameter from OP1 (dev #)
B37D:  B0 2C      BCS  $B3AB     No param, goto LV expression
B37F:  A5 60      LDA  * $60     Get OP1 (lo)(dev address)
B381:  85 BA      STA  * $BA     and put in zero page
B383:  20 A7 B7   JSR  $B7A7     Get OP1 parameters (start addr.)
B386:  B0 23      BCS  $B3AB     No param, goto LV expression
B388:  20 01 B9   JSR  $B901     Copy OP1 contents into OP3
B38B:  85 C6      STA  * $C6     Get bank# in zeropage bank B
B38D:  20 A7 B7   JSR  $B7A7     LSV parameters (end addr.)
B390:  B0 3F      BCS  $B3D1     No parameter, to LV expression
B392:  20 B4 B8   JSR  $B8B4     Line feed & clear rest of line
B395:  A6 60      LDX  * $60     OP1(low) is 'til' value for SAVE
B397:  A4 61      LDY  * $61     OP1(hi) is 'til' value for SAVE
B399:  A5 93      LDA  * $93     Get command-/keyword
B39B:  C9 53      CMP  # $53     Was there an <S> for Save ?
B39D:  D0 D1      BNE  $B370     No=error, no 'til' for SAVE
B39F:  A9 00      LDA  # $00     Load acc with 0,  to zero page
B3A1:  85 B9      STA  * $B9     for secondary addr.
B3A3:  A9 66      LDA  # $66     Bank # 'from' operand (OP3)
B3A5:  20 D8 FF   JSR  $FFD8     Kernal SAVESP: Save data
B3A8:  4C 8B B0   JMP  $B08B     Jump to input wait loop

******************************   Execute valid LV commands

B3AB:  A5 93      LDA  * $93     Get command-/keyword
B3AD:  C9 56      CMP  # $56      <V> for Verify  ?
B3AF:  F0 06      BEQ  $B3B7     Accu <> 0, verify in LOADSP
B3B1:  C9 4C      CMP  # $4C      <L> for Load
B3B3:  D0 BB      BNE  $B370     no, then was it Save <S>?
B3B5:  A9 00      LDA  # $00     u = 0 is load marker in LOADSP
B3B7:  20 D5 FF   JSR  $FFD5     Kernal LOADSP: Load data
B3BA:  A5 90      LDA  * $90     Load system STATUS in acc
B3BC:  29 10      AND  # $10     Mask bit for read error
B3BE:  F0 E8      BEQ  $B3A8     No LV error -go input wait loop
B3C0:  A5 93      LDA  * $93     Get char for command/keyword
B3C2:  F0 AC      BEQ  $B370     No cmd/keyword--then ERROR
B3C4:  20 7D FF   JSR  $FF7D     Kernal PRINT: output string
```

```
******************************   Monitor constant for <ERROR>

B3C7:  20 45 52 52 4F 52 00      ERROR

******************************   Goto input wait loop after

B3CE:  4C 8B B0   JMP  $B08B     Jump to input wait loop

******************************   Extension of LV commands
                                 w/ device & starting addrs.

B3D1:  A6 66      LDX  * $66     lo-addr. (start addr. in X-reg)
B3D3:  A4 67      LDY  * $67     hi-addr.(start addr. in Y-reg)
B3D5:  A9 00      LDA  # $00     Write sec. address $00 = read
B3D7:  85 B9      STA  * $B9     in zero page mem. for sec. addr.
B3D9:  F0 D0      BEQ  $B3AB     to execute LV commands

******************************   Monitor command : F (Fill)

B3DB:  20 83 B9   JSR  $B983     Get 'til' and stepsize in
                                 OPH,OP2
B3DE:  B0 23      BCS  $B403     Carry set=error output identifier
B3E0:  A5 68      LDA  * $68     Get bank no. from 'from' (OP3)
B3E2:  CD B9 0A   CMP  $0AB9     Cmp w/ bank # of 'til' operand
B3E5:  D0 1C      BNE  $B403     Unequal =error output identifier
B3E7:  20 A7 B7   JSR  $B7A7     Get cmd parameter (fill value)
B3EA:  B0 17      BCS  $B403     Carry set=error output identifier
B3EC:  A0 00      LDY  # $00     Set display for fill command, 0
B3EE:  A5 60      LDA  * $60     into OP1(lo)
B3F0:  20 2A B1   JSR  $B12A     STA routine (accu in any bank)
B3F3:  20 E1 FF   JSR  $FFE1     Kernal STOP: check STOP key
B3F6:  F0 08      BEQ  $B400     If pressed, then input wait loop
B3F8:  20 50 B9   JSR  $B950     Addition: constant <1> to OP3
B3FB:  20 3C B9   JSR  $B93C     Subtraction: OP2 - constant <1>
B3FE:  B0 EE      BCS  $B3EE     Kernal STOP:Test for STOP key
B400:  4C 8B B0   JMP  $B08B     Jump to input wait loop
B403:  4C BC B0   JMP  $B0BC     Display <?> &go input wait loop
```

```
*******************************     Monitor command: A
                                    (Assemble)

B406:  B0 3A      BCS  $B442        Carry set=error output identifier
B408:  20 01 B9   JSR  $B901        Copy OP1 to OP3
B40B:  A2 00      LDX  # $00        Clear mnemonic buffer display
B40D:  8E A1 0A   STX  $0AA1        Bit 0 for compressed cmd code 0
B410:  8E B4 0A   STX  $0AB4        Set loop counter to 0
B413:  20 E9 B8   JSR  $B8E9        Test input buffer for cmd-end,
                                    <:>, <?>
B416:  D0 07      BNE  $B41F        Not cmd-end, then go on
B418:  E0 00      CPX  # $00        Display still 0, no commands
B41A:  D0 03      BNE  $B41F        No, continue
B41C:  4C 8B B0   JMP  $B08B        Jump to input wait loop
B41F:  C9 20      CMP  # $20        Is char. read a <space>?
B421:  F0 E8      BEQ  $B40B        Yes, read and initialize
B423:  9D AC 0A   STA  $0AAC,X      Put char. in mnemonic buffer
B426:  E8         INX               Mnem. buffer display ptr. +1
B427:  E0 03      CPX  # $03        3 mnemonic chars. given?
B429:  D0 E8      BNE  $B413        No, get next char.
B42B:  CA         DEX               Displ. pointer to last character
B42C:  30 17      BMI  $B445        3 characters processed, continue
B42E:  BD AC 0A   LDA  $0AAC,X      Read 3 mnem. chars backward
B431:  38         SEC               Set carry for subtraction
B432:  E9 3F      SBC  # $3F        Alpha char values; A=1,B=2,etc
B434:  A0 05      LDY  # $05        Counter shifted 5x for 1 bit
B436:  4A         LSR  A            Shift 1 bit of the letter value out
B437:  6E A1 0A   ROR  $0AA1        of acc into byte pair $AA1-$AA0
B43A:  6E A0 0A   ROR  $0AA0        The three mnemonic chars. will
B43D:  88         DEY               be shifted into the byte pair
B43E:  D0 F6      BNE  $B436        mentioned above and occupy 3
B440:  F0 E9      BEQ  $B42B        sets of 5 bits in these bytes
B442:  4C BC B0   JMP  $B0BC        Display <?>; go input wait loop
B445:  A2 02      LDX  # $02        Set displacemnt of output buffer
B447:  AD B4 0A   LDA  $0AB4        Load loop counter into acc
B44A:  D0 30      BNE  $B47C        If not equal to 0, then skip
B44C:  20 CE B7   JSR  $B7CE        Get cmd parameters in OP1
B44F:  F0 29      BEQ  $B47A        If 0, then test for cmd-end
B451:  B0 EF      BCS  $B442        Carry set=char. for error output
B453:  A9 24      LDA  # $24        Load <$> into acc and bring to
```

```
B455:  9D A0 0A   STA  $0AA0,X   output buffer
B458:  E8         INX            Displace output buffer +1
B459:  A5 62      LDA  * $62     Get OP1's bank byte
B45B:  D0 E5      BNE  $B442     >0=error output indicator
B45D:  A0 04      LDY  # $04     Hex division factor
B45F:  AD B6 0A   LDA  $0AB6     Get number base of operand
B462:  C9 08      CMP  # $08     Compare w/ <8>
B464:  90 05      BCC  $B46B     <8--get high addr. (OP1)
B466:  CC B4 0A   CPY  $0AB4     Cmp. with loop counter
B469:  F0 06      BEQ  $B471     Equal, then skip
B46B:  A5 61      LDA  * $61     Get high addr. byte of OP1
B46D:  D0 02      BNE  $B471     If not equal to 0, then skip
B46F:  A0 02      LDY  # $02     Set loop counter for null bytes
B471:  A9 30      LDA  # $30     Load ASCII <0> into acc and
B473:  9D A0 0A   STA  $0AA0,X   store in assem-cmd temp storage
B476:  E8         INX            Incr. assem-cmd length counter
B477:  88         DEY            Loop count for OP nullbytes -1
B478:  D0 F9      BNE  $B473     Loop till counter=0
B47A:  C6 7A      DEC  * $7A     Display pntr on previous char.
B47C:  20 E9 B8   JSR  $B8E9     Test input buffer for cmd-end,
B47F:  F0 0E      BEQ  $B48F     If cmd-end, then to expression
B481:  C9 20      CMP  # $20     Char a <SPACE>?
B483:  F0 C2      BEQ  $B447     Yes, new parameter expression
B485:  9D A0 0A   STA  $0AA0,X   in asmblr-cmd temp. storage
B488:  E8         INX            Command >9 chars?
B489:  E0 0A      CPX  # $0A     No, then get next char.
B48B:  90 BA      BCC  $B447     Yes, then display <?> error
B48D:  B0 B3      BCS  $B442     Asmblr-cmd OP2 (low) is length
B48F:  86 63      STX  * $63     Byte length of cmd in OP2 (low)
B491:  A2 00      LDX  # $00     Load X-reg w/ 0 and bring up
B493:  8E B1 0A   STX  $0AB1     Cmd-comparison loop counter
B496:  A2 00      LDX  # $00     Load X-reg w/ 0 and use as
B498:  86 9F      STX  * $9F     display for asmblr-cmd buffer
B49A:  AD B1 0A   LDA  $0AB1     Get cmd-comp. counter
B49D:  20 59 B6   JSR  $B659     Addr. & length for cmd counter
B4A0:  AE AA 0A   LDX  $0AAA     Get cmd length pointer (0,1,2)
B4A3:  86 64      STX  * $64     and store in OP2 (high)
B4A5:  AA         TAX            Test result for mnem. compare
B4A6:  BD 61 B7   LDA  $B761,X   Byte at mnemonic keyword tab 2
B4A9:  20 7F B5   JSR  $B57F     Compare w/ byte in asm buffer
```

```
B4AC:  BD 21 B7   LDA  $B721,X   Byte at mnemonic keyword tab 1
B4AF:  20 7F B5   JSR  $B57F     Compare w/ byte in asm buffer
B4B2:  A2 06      LDX  # $06     Loop counter for address cmp.
B4B4:  E0 03      CPX  # $03     3 loops completed?
B4B6:  D0 14      BNE  $B4CC     No, then only addressing cmp.
B4B8:  AC AB 0A   LDY  $0AAB     Get cmd length pointer (0,1,2)
B4BB:  F0 0F      BEQ  $B4CC     Handle as a 1-byte cmd
B4BD:  AD AA 0A   LDA  $0AAA     Get addressing key
B4C0:  C9 E8      CMP  # $E8     Compare w/ $E8
B4C2:  A9 30      LDA  # $30     ASCII for <0> in acc
B4C4:  B0 1E      BCS  $B4E4     carry set, in corresponding eval.
B4C6:  20 7C B5   JSR  $B57C     Compare w/ byte in asm buffer
B4C9:  88         DEY            Decrement cmd length cnt by 1
B4CA:  D0 F1      BNE  $B4BD     If not equal to 0, then skip
B4CC:  0E AA 0A   ASL  $0AAA     Shift addressing key
B4CF:  90 0E      BCC  $B4DF     Bit=0, then skip cmp.
B4D1:  BD 14 B7   LDA  $B714,X   Get addressing char 1 at tab
B4D4:  20 7F B5   JSR  $B57F     Compare w/ byte in asm buffer
B4D7:  BD 1A B7   LDA  $B71A,X   Get addressing char 2 at tab
B4DA:  F0 03      BEQ  $B4DF     If $00, then no comp.
B4DC:  20 7F B5   JSR  $B57F     Compare w/ byte in asm buffer
B4DF:  CA         DEX            Addressing loop counter -1
B4E0:  D0 D2      BNE  $B4B4     Not equal to 0, continue loop
B4E2:  F0 06      BEQ  $B4EA     0, continue evaluation
B4E4:  20 7C B5   JSR  $B57C     Compare w/ byte in asm buffer
B4E7:  20 7C B5   JSR  $B57C     Compare w/ byte in asm buffer
B4EA:  A5 63      LDA  * $63     Get stored length of asmblr cmd
B4EC:  C5 9F      CMP  * $9F     Compare w/ display
                                 (asmblr-cmd buffer)
B4EE:  F0 03      BEQ  $B4F3     If equal then skip
B4F0:  4C 8B B5   JMP  $B58B     Increment cmd-loop counter
B4F3:  AC AB 0A   LDY  $0AAB     Get cmd length pointer
B4F6:  F0 32      BEQ  $B52A     If 0, then a 1-byte cmd
B4F8:  A5 64      LDA  * $64     Get hi-addr. byte from OP2
B4FA:  C9 9D      CMP  # $9D     and compare it with $9D
B4FC:  D0 23      BNE  $B521     Not equal, then skip
B4FE:  A5 60      LDA  * $60     Get low operand addr. and
B500:  E5 66      SBC  * $66     subtract low-cmd addr.
B502:  AA         TAX            Put result in X-reg
B503:  A5 61      LDA  * $61     Get high operand addr. and
```

```
B505:  E5 67      SBC  * $67      subtract high-cmd addr.
B507:  90 08      BCC  $B511      To evaluate of backward branch
B509:  D0 6E      BNE  $B579      "BRANCH OUT OF RANGE",<?>
B50B:  E0 82      CPX  # $82      Check whether branch is valid
B50D:  B0 6A      BCS  $B579      If off by more than $82 give <?>
B50F:  90 08      BCC  $B519      In corresponding expression
B511:  A8         TAY             Copy accu into y-reg and
B512:  C8         INY             increment from 0 to 1
B513:  D0 64      BNE  $B579      Unequal to 0, output <?> error
B515:  E0 82      CPX  # $82      Compare to $02
B517:  90 60      BCC  $B579      Less than 2, then output <?>
B519:  CA         DEX             Addr. balance: decrement X-reg
B51A:  CA         DEX             Addr. balance: decrement X-reg
B51B:  8A         TXA             Bring value to accumulator
B51C:  AC AB 0A   LDY  $0AAB      Get cmd-length counter in Y-reg
B51F:  D0 03      BNE  $B524      <>0, then skip
B521:  B9 5F 00   LDA  $005F,Y    Get value from operand OP1
B524:  20 2A B1   JSR  $B12A      STA routine for acc in any bank
B527:  88         DEY             Decrement cmd length pntr by 1
B528:  D0 F7      BNE  $B521      <>0--skip
B52A:  AD B1 0A   LDA  $0AB1      Get value from OP1
B52D:  20 2A B1   JSR  $B12A      STA routine for acc in any bank
B530:  20 AD B8   JSR  $B8AD      <CR><crsr-up>
B533:  20 7D FF   JSR  $FF7D      Kernal PRINT: output string

******************************    Monitor constant:
                                  assemble output

B536:  41 20 1B 51 00             A <SPACE> <Esc - Q>

******************************    Generate chars. and address
                                  stagger for next assembly
                                  procedure

B53B:  20 DC B5   JSR  $B5DC      Output address and get byte
B53E:  EE AB 0A   INC  $0AAB      Increment opcode lngth ptr. by 1
B541:  AD AB 0A   LDA  $0AAB      Add length to 'from' operand
B544:  20 52 B9   JSR  $B952      Addition: acc contents + OP3
B547:  A9 41      LDA  # $41      Load accu with <A> (assemble)
B549:  8D 4A 03   STA  $034A      in procedure buffer for next line
```

```
B54C:  A9 20      LDA  # $20      Load accu with <SPACE>
B54E:  8D 4B 03   STA  $034B      in procedure buffer for next line
B551:  8D 51 03   STA  $0351      in procedure buffer for next line
B554:  A5 68      LDA  * $68      Bank byte of 'from' addr. in acc
B556:  20 D2 B8   JSR  $B8D2      Acc in 2-byte ASCII: hi=A,lo=X
B559:  8E 4C 03   STX  $034C      In procedure buffer for next line
B55C:  A5 67      LDA  * $67      hi-addr byte(OP3)of 'from' addr
B55E:  20 D2 B8   JSR  $B8D2      Acc in 2-byte ASCII: hi=A,lo=X
B561:  8D 4D 03   STA  $034D      In proc. buffer for next line
B564:  8E 4E 03   STX  $034E      In proc. buffer for next line
B567:  A5 66      LDA  * $66      lo-addr byte(OP3)of 'from' addr
B569:  20 D2 B8   JSR  $B8D2      Acc in 2-byte ASCII: hi=A,lo=X
B56C:  8D 4F 03   STA  $034F      in proc. buffer for next line
B56F:  8E 50 03   STX  $0350      in proc. buffer for next line
B572:  A9 08      LDA  # $08      Keyboard buffer set for
B574:  85 D0      STA  * $D0      8 chars (=length of proc. line)
B576:  4C 8B B0   JMP  $B08B      Jump to input wait loop
B579:  4C BC B0   JMP  $B0BC      Display <?> , go input wait loop

****************************      Compare acc contents w/ a char.
                                  from asmblr-cmd temp. storage

B57C:  20 7F B5   JSR  $B57F      Execute following routine twice
B57F:  8E AF 0A   STX  $0AAF      Store x-reg contents
B582:  A6 9F      LDX  * $9F      Load asmbr-cmd display pointer
B584:  DD A0 0A   CMP  $0AA0,X    Cmp w/ char from asmblr buffer
B587:  F0 0A      BEQ  $B593      If equal, then exit
B589:  68         PLA             Get RTS addr. from stack
B58A:  68         PLA             Get RTS addr. from stack
B58B:  EE B1 0A   INC  $0AB1      Increment cmd-comparison loop
B58E:  F0 E9      BEQ  $B579      >255--output errors
B590:  4C 96 B4   JMP  $B496      Jump to correspond expression
B593:  E6 9F      INC  * $9F      Asmblr-cmd display pointer +1
B595:  AE AF 0A   LDX  $0AAF      Return old X-reg contents
B598:  60         RTS             Return to subroutine
```

```
*****************************   Monitor command: D
                                (Disassemble)

B599:  B0 08      BCS  $B5A3    No valid 'from' operand
B59B:  20 01 B9   JSR  $B901    Copy OP1 to OP3
B59E:  20 A7 B7   JSR  $B7A7    Get OP1 operand
B5A1:  90 06      BCC  $B5A9    If valid, then send step number
B5A3:  A9 14      LDA  # $14    Standard step value $14
                                (=20 bytes to disassemble)
B5A5:  85 60      STA  * $60    in low step counter
B5A7:  D0 05      BNE  $B5AE    Uncond jump to disasmblr.
B5A9:  20 0E B9   JSR  $B90E    Store diff of OP1-OP3 in OP1
B5AC:  90 23      BCC  $B5D1    Carry clear=marker for error out
B5AE:  20 7D FF   JSR  $FF7D    Kernal PRINT: string output

*****************************   Monitor constant: Clear 1 line

B5B1:  0D 1B 51 00              <Cr> <Esc - Q>

*****************************   Disassembly dependent on
                                'from' operand & step size

B5B5:  20 E1 FF   JSR  $FFE1    Kernal STOP: test for STOP key
B5B8:  F0 14      BEQ  $B5CE    If pressed, goto input wait loop
B5BA:  20 D4 B5   JSR  $B5D4    Prep. and output disasmbld line
B5BD:  EE AB 0A   INC  $0AAB    Increment opcode lgth pntr by 1
B5C0:  AD AB 0A   LDA  $0AAB     and for 'from' addr. calc in acc
B5C3:  20 52 B9   JSR  $B952    Addition: acc contents + OP3
B5C6:  AD AB 0A   LDA  $0AAB    Lgth ptr. for step size calc in acc
B5C9:  20 24 B9   JSR  $B924    Subtraction: OP1 - acc contents
B5CC:  B0 E0      BCS  $B5AE    Continue disassem. if necessary
B5CE:  4C 8B B0   JMP  $B08B    Jump to input wait loop
B5D1:  4C BC B0   JMP  $B0BC    Display <?>; go input wait loop
B5D4:  A9 2E      LDA  # $2E    Load accu with <.>
B5D6:  20 D2 FF   JSR  $FFD2    Kernal BSOUT: char. output
B5D9:  20 A8 B8   JSR  $B8A8    <SPACE><CR><crsr-up>
B5DC:  20 92 B8   JSR  $B892    Output 'from' addr.(OP3) as
                                5-byte ASCII
B5DF:  20 A8 B8   JSR  $B8A8    <SPACE><CR><crsr-up>
B5E2:  A0 00      LDY  # $00    Load displacement for FETCH
```

```
B5E4:  20 1A B1   JSR  $B11A      LDA routine from any bank
B5E7:  20 59 B6   JSR  $B659      Validity check of opcode bytes
B5EA:  48         PHA             Put result on stack
B5EB:  AE AB 0A   LDX  $0AAB      Get command length loop
B5EE:  E8         INX             Increment length key by 1
B5EF:  CA         DEX             Decrement length key by 1
B5F0:  10 0A      BPL  $B5FC      Output cmd value < 0 constant
B5F2:  20 7D FF   JSR  $FF7D      Kernal PRINT: output string

****************************      Monitor constants: 3 spaces

B5F5:  20 20 20 00                <SPACE><SPACE><SPACE>

****************************      Assembly/disassembly sub.

B5F9:  4C 02 B6   JMP  $B602      Skip LDA for routine
B5FC:  20 1A B1   JSR  $B11A      LDA routine for acc - any bank
B5FF:  20 A5 B8   JSR  $B8A5      Output acc as 2-byte ASCII
                                  +<SPACE>
B602:  C8         INY             Increment Y-reg contents by 1
B603:  C0 03      CPY  # $03      Compare to $03
B605:  90 E8      BCC  $B5EF      <3, then continue loop
B607:  68         PLA             Get result from stack
B608:  A2 03      LDX  # $03      Put 3 chars for mnem. output in
B60A:  20 A1 B6   JSR  $B6A1      X-reg, and to char. output
B60D:  A2 06      LDX  # $06      Initialize loop count with 6
B60F:  E0 03      CPX  # $03      After 3 loops, the actual address
B611:  D0 17      BNE  $B62A      value will be output
B613:  AC AB 0A   LDY  $0AAB      Number of cmd operand bytes
B616:  F0 12      BEQ  $B62A      No operand bytes, then skip
B618:  AD AA 0A   LDA  $0AAA      Command addr. key
B61B:  C9 E8      CMP  # $E8      Check for branch
B61D:  08         PHP             Put carry flag on stack
B61E:  20 1A B1   JSR  $B11A      LDA routine for any bank
B621:  28         PLP             Reset carry flag
B622:  B0 1D      BCS  $B641      If carry set, then BRANCH
B624:  20 C2 B8   JSR  $B8C2      Acc conveyed as 2-byte ASCII
B627:  88         DEY             If cmd has two operand bytes,
B628:  D0 EE      BNE  $B618      then expression of the second bit
B62A:  0E AA 0A   ASL  $0AAA      is masked by addressing key
```

```
B62D:  90 0E      BCC  $B63D      Bit not set, skip
B62F:  BD 14 B7   LDA  $B714,X    Get char. for addr. type
B632:  20 D2 FF   JSR  $FFD2      Kernal BSOUT: output one char
B635:  BD 1A B7   LDA  $B71A,X    Get char. for addr. type
B638:  F0 03      BEQ  $B63D      Not equal to 0--then output
B63A:  20 D2 FF   JSR  $FFD2      Kernal BSOUT: output one char
B63D:  CA         DEX             Address output loop -1
B63E:  D0 CF      BNE  $B60F      All 6 loops out
B640:  60         RTS             Return to subroutine

******************************    Address from BRANCH cmd

B641:  20 4D B6   JSR  $B64D      Addr. calc. X=high, A=low

B644:  18         CLC             Clear carry for addition
B645:  69 01      ADC  # $01      Add 1 for low-addr. correction
B647:  D0 01      BNE  $B64A      No overflow skip hi-correction
B649:  E8         INX             Add 1 for high correction
B64A:  4C 9F B8   JMP  $B89F      Give acc + X-reg. as 4-bytes
B64D:  A6 67      LDX  * $67      Get high addr. of 'from'
                                  operand (OP3)
B64F:  A8         TAY             Bring BRANCH offset in x-reg
B650:  10 01      BPL  $B653      BRANCH 'forward' continues
B652:  CA         DEX             Decrement high addr. for
                                  'backward'-1
B653:  65 66      ADC  * $66      +branch offest to low addr(OP3)
B655:  90 01      BCC  $B658      No overflow skip hi correction
B657:  E8         INX             Overflow correction for hi-addr.
B658:  60         RTS             Return from subroutine

******************************    Determine addressing and length
                                  of the test code passed in A

B659:  A8         TAY             Put test code in Y-reg
B65A:  4A         LSR  A          Shift bit 0 out & test
B65B:  90 0B      BCC  $B668      If bit 0=0 then OK
B65D:  4A         LSR  A          Shift & test bit 1
B65E:  B0 17      BCS  $B677      If bit 1=1 then no good
B660:  C9 22      CMP  # $22      Test whether exit code $89 used
```

```
B664:  29 07      AND  # $07      Mask bits 3-7
B666:  09 80      ORA  # $80      Mask bit 7 in
B668:  4A         LSR  A          Divide acc contents by 2
B669:  AA         TAX             and display in X-reg
B66A:  BD C3 B6   LDA  $B6C3,X    Load byte as addressing ref. tab
B66D:  B0 04      BCS  $B673      If remainder left from div., skip
B66F:  4A         LSR  A          Copy contents of the
B670:  4A         LSR  A          upper nibble
B671:  4A         LSR  A          (bits 4-7) into
B672:  4A         LSR  A          lower nibble (bits 4-3)
B673:  29 0F      AND  # $0F      Mask out upper nibble (Bit 4-7)
B675:  D0 04      BNE  $B67B      Not equal 0 is valid
B677:  A0 80      LDY  # $80      If code is invalid, load Y w/ $80
B679:  A9 00      LDA  # $00      and acc with $00
B67B:  AA         TAX             Displacement not transfer to X
B67C:  BD 07 B7   LDA  $B707,X    Get addressing key from tab
B67F:  8D AA 0A   STA  $0AAA      and put in $0AAA
B682:  29 03      AND  # $03      Mask out bits 2-7
B684:  8D AB 0A   STA  $0AAB      Store bits 0,1(cmd. length)
B687:  98         TYA             Copy test code in acc
B688:  29 8F      AND  # $8F      Mask out bits 4,5,6
B68A:  AA         TAX             Store masked out value in X
B68B:  98         TYA             Copy test code in acc
B68C:  A0 03      LDY  # $03      Initialize loop counter with 3
B68E:  E0 8A      CPX  # $8A      Cmp masked-out value w/ $8A
B690:  F0 0B      BEQ  $B69D      Equal, then skip
B692:  4A         LSR  A          Divide acc contents by 2
B693:  90 08      BCC  $B69D      If no remainder, then skip
B695:  4A         LSR  A          Divide acc contents by 2
B696:  4A         LSR  A          Divide acc contents by 2
B697:  09 20      ORA  # $20      Set bit 5 in acc
B699:  88         DEY             Decrement loop counter by 1
B69A:  D0 FA      BNE  $B696      Not equal to 0, continue loop
B69C:  C8         INY             Loop number incremented by 1
B69D:  88         DEY             Decrement loop counter by 1
B69E:  D0 F2      BNE  $B692      Not equal to 0, divide further
B6A0:  60         RTS             Return from subroutine
```

```
*****************************
```

Prepare and send a char. for mnemonic display

```
B6A1:  A8          TAY
B6A2:  B9 21 B7    LDA  $B721,Y
B6A5:  85 63       STA  * $63
B6A7:  B9 61 B7    LDA  $B761,Y
B6AA:  85 64       STA  * $64
B6AC:  A9 00       LDA  # $00
B6AE:  A0 05       LDY  # $05
B6B0:  06 64       ASL  * $64
B6B2:  26 63       ROL  * $63
B6B4:  2A          ROL  A
B6B5:  88          DEY
B6B6:  D0 F8       BNE  $B6B0
B6B8:  69 3F       ADC  # $3F
B6BA:  20 D2 FF    JSR  $FFD2
B6BD:  CA          DEX
B6BE:  D0 EC       BNE  $B6AC
B6C0:  4C A8 B8    JMP  $B8A8
```

Cmd code as display to Y-reg
Get byte from mnemonic table 1
and put in OP2 (low)
Get byte from mnemonic table 2
and put into OP2 (high)
Load accu w/ 0
Shift 5 bits of OP2 2-byte addr.
to the left; put bits into accu
Loop until all five
bits are shifted. The
addition of
the number $3F gives a valid
char or a <?>
Kernal BSOUT: output one char
3 loops for the 3 letters from the
16-bit value in addr lo/hi in OP2
<SPACE><CR><crsr-up>:RTS

```
*****************************
```

Address reference table

```
B6C3:  40 02 45 03 D0 08 40 09
B6CB:  30 22 45 33 D0 08 40 09
B6D3:  40 02 45 33 D0 08 40 09
B6DB:  40 02 45 B3 D0 08 40 09
B6E3:  00 22 44 33 D0 8C 44 00
B6EB:  11 22 44 33 D0 8C 44 9A
B6F3:  10 22 44 33 D0 08 40 09
B6FB:  10 22 44 33 D0 08 40 09
B703:  62 13 78 A9
```

```
*****************************
```

Address types & length key

```
B707:  00 21 81 82
B70B:  00 00 59 4D
B70F:  91 92 86 4A
B713:  85
```

-1 / #$ -2 / *$ -2 / $ -3
-1 / -1 / ($,X)-2 / ($),Y-2
*$,X-2 / $,X -3 / $,Y -3 / ($) -3
*$,Y-2

```
B714:  9D          .Byte $9D
```

Backspace control code

```
*****************************
```
Display addressing modes

```
B715:  2C 29 2C 23 28 24
B71B:  59 00 58 24 24 00
```
<,><)><,><#><(><$>
<Y>< ><X><$><$><>

```
*****************************
```
Mnemonic keyword table 1

```
B721:  1C 8A 1C 23 5D 8B 1B A1
B729:  9D 8A 1D 23 9D 8B 1D A1
B731:  00 29 19 AE 69 A8 19 23
B739:  24 53 1B 23 24 53 19 A1
B741:  00 1A 5B 5B A5 69 24 24
B749:  AE AE A8 AD 29 00 7C 00
B751:  15 9C 6D 9C A5 69 29 53
B759:  84 13 34 11 A5 69 23 A0
```
BRK PHP BPL CLC JSR PLP BMI SEC
RTI PHA BVC CLI RTS PLA BVS SEI
??? DEY BCC TYA LDY TAY BCS CLV
CPY INY BNE CLD CPX INX BEQ SED
??? BIT JMP JMP STY LDY CPY CPX
TXA TXS TAX TSX DEX ??? NOP ???
ASL ROL LSR ROR STX LDX DEC INC
ORAANDEOR ADC STA LDA CMP SBC

```
*****************************
```
Mnemonic keyword table 2

```
B761:  D8 62 5A 48 26 62 94 88
B769:  54 44 C8 54 68 44 E8 94
B771:  00 B4 08 84 74 B4 28 6E
B779:  74 F4 CC 4A 72 F2 A4 8A
B781:  00 AA A2 A2 74 74 74 72
B789:  44 68 B2 32 B2 00 22 00
B791:  1A 1A 26 26 72 72 88 C8
B799:  C4 CA 26 48 44 44 A2 C8
```
A byte in table 1 returns, with the corresponding value in table 2, a 16-bit value coded as a 3-character mnemonic. The 16-bit argument is divided into three sections of 5 bits. Bit 0 is unused in coding.

```
*****************************
```
Monitor constant: 3 spaces

```
B7A1:  0D 20 20 20
```
<CR><SPACE><SPACE><SPACE>

```
*****************************
```
Test for valid separator between the command's operands

```
B7A5:  C6 7A      DEC  * $7A
B7A7:  20 CE B7   JSR  $B7CE
B7AA:  B0 16      BCS  $B7C2
B7AC:  20 E7 B8   JSR  $B8E7
B7AF:  D0 09      BNE  $B7BA
B7B1:  C6 7A      DEC  * $7A
B7B3:  AD B4 0A   LDA  $0AB4
```
Input buff pntr to previous char.
Get OP1 operand
Carry set=error signal
Renew last-read char.
If cmd-end, then continue
Input buff pntr to previous char.
Get error-recognition flag

```
B7B6:  D0 11      BNE  $B7C9     If not equal to 0, then OK exit
B7B8:  F0 0D      BEQ  $B7C7     No valid operand, then error exit
B7BA:  C9 20      CMP  # $20     Was char. read a <SPACE>?
B7BC:  F0 0B      BEQ  $B7C9     Valid separator, OK exit
B7BE:  C9 2C      CMP  # $2C     Was the char. a comma?
B7C0:  F0 07      BEQ  $B7C9     Valid separator, OK exit
B7C2:  68         PLA            The addresses on the stack are
B7C3:  68         PLA            cleared, <?> is displayed, and
B7C4:  4C BC B0   JMP  $B0BC     prg goes to the input wait loop

******************************   Exit for error in cmd-operands

B7C7:  38         SEC            Set carry=error signal
B7C8:  24        .Byte $24       skip to $B7CA

******************************   Command operand/separator OK

B7C9:  18         CLC            Carry clear=signal for OK
B7CA:  AD B4 0A   LDA  $0AB4     Load error-recognition help flag
B7CD:  60         RTS            What appears to be RTS is the
                                 command entry

******************************   Init. & evaluation of a command
                                 parameter in OP1

B7CE:  A9 00      LDA  # $00     Load acc w/ $0 for param. init.
B7D0:  85 60      STA  * $60     Clear the 3-byte cmd parameter
B7D2:  85 61      STA  * $61      No. 1 (OP1), in zero page from
B7D4:  85 62      STA  * $62     $62 (highest) to $60 (lowest)
B7D6:  8D B4 0A   STA  $0AB4     Temp. memory for error control
B7D9:  8A         TXA            Put X-reg in accumulator
B7DA:  48         PHA            and save on stack
B7DB:  98         TYA            Put Y-reg in accumulator
B7DC:  48         PHA            and save on stack
B7DD:  20 E9 B8   JSR  $B8E9     Test input buffer for cmd-end,
B7E0:  D0 03      BNE  $B7E5     <:>,<?>.No end marker, go on
B7E2:  4C 7E B8   JMP  $B87E     Exit routine w/ clear-carry
B7E5:  C9 20      CMP  # $20     marker. Was it a <SPACE> ?
B7E7:  F0 F4      BEQ  $B7DD     Yes, then read next char.
B7E9:  A2 03      LDX  # $03     Get display for 4 conver chars.
```

```
B7EB:  DD F5 B0   CMP  $B0F5,X   Check for conversion (%&+$)
B7EE:  F0 06      BEQ  $B7F6     until conversion char. is found
B7F0:  CA         DEX            Display calc. table - 1
B7F1:  10 F8      BPL  $B7EB     Loop till table through
B7F3:  E8         INX            X-reg set to $0 (= HEX)
B7F4:  C6 7A      DEC  * $7A     Displacement ptr to input buff -1
B7F6:  BC 8A B8   LDY  $B88A,X   Load Y-reg w/ num system base
B7F9:  BD 8E B8   LDA  $B88E,X   Load accu w/ multip. factor
B7FC:  8D B6 0A   STA  $0AB6     for the num. system, & store it
B7FF:  20 E9 B8   JSR  $B8E9     Test inp. buf cmd-end, <:>, <?>
B802:  F0 7A      BEQ $B87E      Exit from operand determination
B804:  38         SEC            Set carry for subtraction
B805:  E9 30      SBC  # $30     Convert to fixed-point values
B807:  90 75      BCC  $B87E     If char <0 then exit
B809:  C9 0A      CMP  # $0A     Was char. a no. between 0 - 9 ?
B80B:  90 06      BCC  $B813     Yes, jump to hex adaptation
B80D:  E9 07      SBC  # $07     Adaptation of hex numbers A - F
B80F:  C9 10      CMP  # $10     If value isn't between 0 - F, then
B811:  B0 6B      BCS  $B87E     Exit from operand determ't rtne.
B813:  8D B5 0A   STA  $0AB5     Store established hex numbers
B816:  CC B5 0A   CPY  $0AB5     Compare base w/ hex value
B819:  90 61      BCC  $B87C     If base < char, then error
B81B:  F0 5F      BEQ  $B87C     If base = char, then error
B81D:  EE B4 0A   INC  $0AB4     Byte for error recognition +1
B820:  C0 0A      CPY  # $0A     Was decimal input chosen ?
B822:  D0 0A      BNE  $B82E     No, then jump to decimal init.
B824:  A2 02      LDX  # $02     Set loop counter to 2
B826:  B5 60      LDA  * $60,X   Copy the 3-byte operand (OP1)
B828:  9D B7 0A   STA  $0AB7,X   in the 3-byte temp operand for
B82B:  CA         DEX            decimal address input
B82C:  10 F8      BPL  $B826     ($0AB9=highest, $0AB7= lowest)
B82E:  AE B6 0A   LDX  $0AB6     get counter for multip. factor
B831:  06 60      ASL  * $60     3-byte
B833:  26 61      ROL  * $61     operand (OP1)
B835:  26 62      ROL  * $62     multiplied by 2
B837:  B0 43      BCS  $B87C     If overflow present, then error
B839:  CA         DEX            Loop counter mult by 2 -1
B83A:  D0 F5      BNE  $B831     Loop to OP1 multiplication
B83C:  C0 0A      CPY  # $0A     Is number base the dec. system?
B83E:  D0 22      BNE  $B862     No, jump to decimal conversion
```

```
B840:  0E B7 0A   ASL  $0AB7      Decimal conversion: the 3-byte
B843:  2E B8 0A   ROL  $0AB8      temp operand in $AB9 - $AB7
B846:  2E B9 0A   ROL  $0AB9      is multiplied by 2
B849:  B0 31      BCS  $B87C      If overflow occurs, then error
B84B:  AD B7 0A   LDA  $0AB7      Addition of 3-byte
B84E:  65 60      ADC  * $60      temp operand in
B850:  85 60      STA  * $60      memory locations
B852:  AD B8 0A   LDA  $0AB8      $0AB9-$0AB8-$0AB7
B855:  65 61      ADC  * $61      to contents of 3-byte
B857:  85 61      STA  * $61      operand OP1 under observation
B859:  AD B9 0A   LDA  $0AB9      for possible overflow.
B85C:  65 62      ADC  * $62      Result of the addition will be
B85E:  85 62      STA  * $62      put into OP1.
B860:  B0 1A      BCS  $B87C      If overflow occurs, then error
B862:  18         CLC             Clear w/ carry (for bin,oct,hex)
B863:  AD B5 0A   LDA  $0AB5      Get determined char. value
B866:  65 60      ADC  * $60      Add values of least significant
B868:  85 60      STA  * $60      OP1 place
B86A:  8A         TXA             Load accumulator with 0
B86B:  65 61      ADC  * $61      Check for overflow by adding of
B86D:  85 61      STA  * $61      least significant OP1 place
B86F:  8A         TXA             Load accu with 0
B870:  65 62      ADC  * $62      Check for overflow at
B872:  85 62      STA  * $62      place of OP1 addition
B874:  B0 06      BCS  $B87C      If overflow occurs, then error
B876:  29 F0      AND  # $F0      Mask out lower nibble (B. 0-3)
B878:  D0 02      BNE  $B87C      If top nibble <> 0, then error
B87A:  F0 83      BEQ  $B7FF      evaluate next operand position

******************************    Exit param. evaluate w/ error

B87C:  38         SEC             Set carry = error-found marker
B87D:  24        .Byte $24        Skip to $B87F

******************************    Exit parameter evaluation if OK

B87E:  18         CLC             Clear carry = param-OK marker
B87F:  8C B6 0A   STY  $0AB6      Store base of number system
B882:  68         PLA             Restore old Y contents from
B883:  A8         TAY             stack
```

```
B884:  68          PLA          Restore old X contents from
B885:  AA          TAX          stack
B886:  AD B4 0A    LDA  $0AB4   Load acc with error help pointer
B889:  60          RTS          Return from subroutine

******************************  Number system bases

B88A:  10 0A 08 02               Hex, decimal, octal, binary

******************************  Number of multiplications with
                                 the factor 2 for number systems

B88E:  04 03 03 01               Hex, decimal, octal, binary

******************************  OP3 contents displayed as
                                 5-byte ASCII

B892:  A5 68       LDA  * $68    Load A w/ hi (bank) byte (OP3)
B894:  20 D2 B8    JSR  $B8D2    Acc in 2-byte ASCII: hi=A,lo=X
B897:  8A          TXA           ASCII code of low value in acc
B898:  20 D2 FF    JSR  $FFD2    Kernal BSOUT: print a character
B89B:  A5 66       LDA  * $66    Load A w/ lo(Addr-lo)byte(OP3)
B89D:  A6 67       LDX  * $67    Load Xmid(Addr-hi)byte(OP3)
B89F:  48          PHA           Store acc on stack
B8A0:  8A          TXA           Addr-hi value from OP3 in acc
B8A1:  20 C2 B8    JSR  $B8C2    Display acc in 2-char ASCII
B8A4:  68          PLA           Load acc again w/ addr-lo (OP3)

******************************  Prepare acc in ASCII, output,
                                 output <Blank>, for start-of-line

B8A5:  20 C2 B8    JSR  $B8C2    Acc displayed as 2-char ASCII
B8A8:  A9 20       LDA  # $20    Put <Blank> in accumulator
B8AA:  4C D2 FF    JMP  $FFD2    Kernal BSOUT: output a char
B8AD:  20 7D FF    JSR  $FF7D    Kernal PRIMM: output string
```

```
*******************************   Monitor output constants

B8B0:  0D 91 00                   <Cr> <Crsr Up>

*******************************   End of output routine

B8B3:  60         RTS             Return from subroutine

*******************************   <Cr> <Cr> <Esc-Q> blank
                                  output

B8B4:  A9 0D      LDA  # $0D      Load <Cr> code into accu.
B8B6:  4C D2 FF   JMP  $FFD2      Kernal BSOUT: output a char
B8B9:  20 7D FF   JSR  $FF7D      Kernal PRIMM: output string

*******************************   Monitor constants for carriage
                                  return and clear next line, blank

B8BC:  0D 1B 51 20 00             <Cr> <Esc-Q> Blank

*******************************   End of output routine

B8C1:  60         RTS             Return to subroutine

*******************************   Convert acc contents contents to
                                  2-byte char. and output via
                                  BSOUT

B8C2:  8E AF 0A   STX  $0AAF      Store old X-reg contents
B8C5:  20 D2 B8   JSR  $B8D2      Acc in 2-byte ASCII: hi=A,lo=X
B8C8:  20 D2 FF   JSR  $FFD2      Kernal BSOUT: output a char
B8CB:  8A         TXA             Load char. from X into accu.
B8CC:  AE AF 0A   LDX  $0AAF      Restore X-register
B8CF:  4C D2 FF   JMP  $FFD2      Kernal BSOUT: output a char

*******************************   Split acc contents and convert to
                                  2-byte ASCII code (X=lo, A=hi)

B8D2:  48         PHA             Store acc contents temporarily
B8D3:  20 DC B8   JSR  $B8DC      Convert low nibble to ASCII
B8D6:  AA         TAX             ASCII for low nibble in X-reg
```

```
B8D7:  68          PLA          Restore acc contents
B8D8:  4A          LSR  A       Shift right 4 times so that
B8D9:  4A          LSR  A       the highest nibble (bits 4-7) is
B8DA:  4A          LSR  A       shifted into the lower nibble
B8DB:  4A          LSR  A       (bits 0-3) position

******************************  Convert the lower nibble in the
                                 acc to ASCII code

B8DC:  29 0F       AND  # $0F   Mask high nibble out (bits 4-7)
B8DE:  C9 0A       CMP  # $0A   Is it a number from 0-9?
B8E0:  90 02       BCC  $B8E4   Yes, create ASCII code
B8E2:  69 06       ADC  # $06   Character adaptation for A-F
B8E4:  69 30       ADC  # $30   Generate ASCII for acc contents
B8E6:  60          RTS          Return from subroutine

******************************  Get 1 char. from input buffer
                                 and check for cmd-end,<:>,<?>
                                 equal flag.

B8E7:  C6 7A       DEC  * $7A   Display to input buffer - 1 (like
                                CHRGOT)
B8E9:  8E AF 0A    STX  $0AAF   Store X-reg contents
B8EC:  A6 7A       LDX  * $7A   Load X-reg w/ display to in. buf
B8EE:  BD 00 02    LDA  $0200,X Get char. from cmd input buffer
B8F1:  F0 06       BEQ  $B8F9   Has $00 (cmd-end) been found?
B8F3:  C9 3A       CMP  # $3A   Is char. read a <:> ?
B8F5:  F0 02       BEQ  $B8F9   Yes, exit with set equal flag
B8F7:  C9 3F       CMP  # $3F   Is char. read a <?> ?
B8F9:  08          PHP          Store status of equal flag
B8FA:  E6 7A       INC  * $7A   Displ. to input buffer+1 (next
B8FC:  AE AF 0A    LDX  $0AAF   char.). Restore X-register
B8FF:  28          PLP          Restore equal flag status
B900:  60          RTS          Return from subroutine
```

```
*****************************      Copy contents of OP1
                                   ($62-$61-$60)
                                   into OP3 ($68-$67-$66)

B901:  A5 60      LDA  * $60       Get OP1 lo(addr-lo) and copy
B903:  85 66      STA  * $66       into OP3 lowest (addr-lo)
B905:  A5 61      LDA  * $61       Get OP1 middle (addr-hi) and
B907:  85 67      STA  * $67       copy into  OP3 middle (addr-hi)
B909:  A5 62      LDA  * $62       OP1 highest (bank-byte) copies
B90B:  85 68      STA  * $68       into OP3 highest (bank-byte)
B90D:  60         RTS              Return to subroutine

*****************************      Store diff. OP1-OP3 in OP1

B90E:  38         SEC              Set carry for subtraction
B90F:  A5 60      LDA  * $60       Load accu with OP1 lowest
B911:  E5 66      SBC  * $66       Subtract OP3 lowest from it
B913:  85 60      STA  * $60       Store result in OP1 lowest
B915:  A5 61      LDA  * $61       Load acc w/ OP1 middle
B917:  E5 67      SBC  * $67       Subtr OP3 middle (+ underflow)
B919:  85 61      STA  * $61       Store result in OP3 middle
B91B:  A5 62      LDA  * $62       Load acc w/ OP1 highest
B91D:  E5 68      SBC  * $68       Subtr OP3 highest (+underflow)
B91F:  85 62      STA  * $62       Store result in OP1 highest
B921:  60         RTS              Return from subroutine

*****************************      Subtraction: OP1 - Minuend in
                                   $0AAF

B922:  A9 01      LDA  # $01       Load acc w/ 1 and store as
B924:  8D AF 0A   STA  $0AAF       Minuend in $0AAF
B927:  38         SEC              Set carry for subtraction
B928:  A5 60      LDA  * $60       Load accu w/ OP1 Lowest
B92A:  ED AF 0A   SBC  $0AAF       Subtr minuend from OP1 lowest
B92D:  85 60      STA  * $60       Write result of subtr. back
B92F:  A5 61      LDA  * $61       Load acc w/ OP1 middle
B931:  E9 00      SBC  # $00       Note underflow of lowest subtr.
B933:  85 61      STA  * $61       Write result of subtr. back
B935:  A5 62      LDA  * $62       Load acc w/ OP1 highest
B937:  E9 00      SBC  # $00       Note underlow of middle subtr.
```

```
B939:  85 62      STA  * $62      Write result of subtr. back
B93B:  60         RTS             Return from subroutine

******************************    Subtraction of constant 1 from
                                  operand 2 (OP2) in $65-$64-$63

B93C:  38         SEC             Set carry for subtraction
B93D:  A5 63      LDA  * $63      Load acc w/ OP2 lowest
B93F:  E9 01      SBC  # $01      Subtract 1 from it
B941:  85 63      STA  * $63      Write result of subtr. back
B943:  A5 64      LDA  * $64      Load acc w/ OP2 middle
B945:  E9 00      SBC  # $00      Note undeflow of lowest subtr.
B947:  85 64      STA  * $64      Write result of subtr. back
B949:  A5 65      LDA  * $65      Load acc w/ OP3 highest
B94B:  E9 00      SBC  # $00      Note underflow of middle subtr.
B94D:  85 65      STA  * $65      Write result of subtr. back
B94F:  60         RTS             Return from subroutine

******************************    Addition of acc contents to OP3

B950:  A9 01      LDA  # $01      Load acc w/ addition constant 1
B952:  18         CLC             Clear carry for addition
B953:  65 66      ADC  * $66      Add contents of OP3 lowest
B955:  85 66      STA  * $66      Write result of addition back
B957:  90 06      BCC  $B95F      If no overflow, then return
B959:  E6 67      INC  * $67      Incr.. OP3 middle if overflow
B95B:  D0 02      BNE  $B95F      If no overflow, then return
B95D:  E6 68      INC  * $68      Incr. OP3 highest for overflow
B95F:  60         RTS             Return from subroutine

******************************    Subtr. of constant 1 from OP3

B960:  38         SEC             Set carry for subtraction
B961:  A5 66      LDA  * $66      Load acc w/ OP3 lowest
B963:  E9 01      SBC  # $01      Subtract constant 1
B965:  85 66      STA  * $66      Write result
B967:  A5 67      LDA  * $67      Load acc w/ OP3 middle
B969:  E9 00      SBC  # $00      Take underflow into account
B96B:  85 67      STA  * $67      Write result
```

```
B96D:  A5 68      LDA  * $68      Load acc w/ OP3 highest
B96F:  E9 00      SBC  # $00      Account for underflow
B971:  85 68      STA  * $68      Write result
B973:  60         RTS             Return from subroutine

*****************************     Copy (for carry clear) the
                                  contents of OP1 into zero page
                                  memory for Bank-no, PC-hi,
                                  PC-lo

B974:  B0 0C      BCS  $B982      Return if carry set
B976:  A5 60      LDA  * $60      Load acc w/ OP1 lo (addr-lo)
B978:  A4 61      LDY  * $61      Load Y-reg w/OP1 mid (addr-hi)
B97A:  A6 62      LDX  * $62      Load X-reg w/OP1 hi (bnk-byte)
B97C:  85 04      STA  * $04      Bring in Z-P byte for PC-Lo
B97E:  84 03      STY  * $03      Bring in Z-P byte for PC-Hi
B980:  86 02      STX  * $02      Bring in Z-P byte for bank-no
B982:  60         RTS             Return from subroutine

*****************************     Put "from" operand in OP3
                                  Get "to" operand in OP1
                                  Copy "to" operand in OPH
                                  Form difference of OP1-OP3
                                  & store "step number" in OP1
                                  Copy "step number" in OP2

B983:  B0 2A      BCS  $B9AF      Exit if error in command param
B985:  20 01 B9   JSR  $B901      Copy contents of OP1 into OP3
B988:  20 A7 B7   JSR  $B7A7      Get "to" operand in OP1
B98B:  B0 22      BCS  $B9AF      "to" operand invalid, error exit
B98D:  A5 60      LDA  * $60      Copy the contents
B98F:  8D B7 0A   STA  $0AB7      of the 3-byte operand
B992:  A5 61      LDA  * $61      OP1 into the 3-byte
B994:  8D B8 0A   STA  $0AB8      temp operand in
B997:  A5 62      LDA  * $62      memory locations
B999:  8D B9 0A   STA  $0AB9      $0AB9-$0AB8-$0AB7
B99C:  20 0E B9   JSR  $B90E      Difference:OP1-OP3 in OP1
B99F:  A5 60      LDA  * $60      Copy the
B9A1:  85 63      STA  * $63      contents of
B9A3:  A5 61      LDA  * $61      3-byte OP1
```

```
B9A5:  85 64      STA  * $64      operand
B9A7:  A5 62      LDA  * $62      in the OP2
B9A9:  85 65      STA  * $65      operand
B9AB:  90 02      BCC  $B9AF      If OP1 > OP3, then error exit
B9AD:  18         CLC             Clear carry as marker for OK
B9AE:  24         .Byte $24       Skip to $B9B0 (RTS)

******************************    Routine exit for error
                                  encountered

B9AF:  38         SEC             Set carry = error-found marker
B9B0:  60         RTS             Return from the subroutine

******************************    Output for conversion command
                                  (&%+$)

B9B1:  20 A5 B7   JSR  $B7A5      Get the conversion value in OP1
B9B4:  20 B9 B8   JSR  $B8B9      output <Cr> <Esc-Q> <space>
B9B7:  A9 24      LDA  # $24      Load accu with <$>
B9B9:  20 D2 FF   JSR  $FFD2      Kernal BSOUT: output a char.
B9BC:  A5 62      LDA  * $62      Load hi of the 3-byte conv.value
B9BE:  F0 07      BEQ  $B9C7      If $00, suppress leading zeros
B9C0:  20 D2 B8   JSR  $B8D2      Acc in 2-byte ASCII: hi=A,lo=X
B9C3:  8A         TXA             ASCII for low nibble in acc
B9C4:  20 D2 FF   JSR  $FFD2      Kernal BSOUT: output a char
B9C7:  A5 60      LDA  * $60      Load lo of the 3-byte conv.value
B9C9:  A6 61      LDX  * $61      Load mid of 3-byte conv value
B9CB:  20 9F B8   JSR  $B89F      Output theses as 4 ASCII chars
B9CE:  20 B9 B8   JSR  $B8B9      output <Cr> <Esc-Q> <space>
B9D1:  A9 2B      LDA  # $2B      Load acc with <+>
B9D3:  20 D2 FF   JSR  $FFD2      Kernal BSOUT: output a char
B9D6:  20 07 BA   JSR  $BA07      Convert OP1 to decimal
B9D9:  A9 00      LDA  # $00      Marker for leading-zero suppres
B9DB:  A2 08      LDX  # $08      Output 8 characters
B9DD:  A0 03      LDY  # $03      Every 4 bits is an output digit
B9DF:  20 5D BA   JSR  $BA5D      Output AA0-AA3 as a decimal #
B9E2:  20 B9 B8   JSR  $B8B9      Output <Cr> <Esc-Q) <space>
B9E5:  A9 26      LDA  # $26      Load acc with <&>
B9E7:  20 D2 FF   JSR  $FFD2      Kernal BSOUT: output a char
B9EA:  A9 00      LDA  # $00      Marker for leading-zero suppres
```

```
B9EC:  A2 08      LDX  # $08     Output 8 characters
B9EE:  A0 02      LDY  # $02     Every 3 bits is an output digit
B9F0:  20 47 BA   JSR  $BA47     Output AA0-AA3 as an octal #
B9F3:  20 B9 B8   JSR  $B8B9     Output <Cr> <Esc-q> <space>
B9F6:  A9 25      LDA  # $25     Load accumulator with <%>
B9F8:  20 D2 FF   JSR  $FFD2     Kernal BSOUT: output a char
B9FB:  A9 00      LDA  # $00     Marker leading-zero suppression
B9FD:  A2 18      LDX  # $18     Output 18 characters
B9FF:  A0 00      LDY  # $00     Every bit is an output digit
BA01:  20 47 BA   JSR  $BA47     Output AA0-AA3 in binary
BA04:  4C 8B B0   JMP  $B08B     Jump to input wait loop

****************************     Convert contents of OP1 to an
                                 8-place decimal number in
                                 AA0-AA4

BA07:  20 01 B9   JSR  $B901     Copy contents of OP1 into OP3
BA0A:  A9 00      LDA  # $00     Clear AA0-AA3 for
BA0C:  A2 07      LDX  # $07     decimal number
BA0E:  9D A0 0A   STA  $0AA0,X   Clear AA4-AA7 as temp counter
BA11:  CA         DEX            for decimal conversion
BA12:  10 FA      BPL  $BA0E     Init. one's place of temp counter
BA14:  EE A7 0A   INC  $0AA7     with <1>
BA17:  A0 17      LDY  # $17     Loop cntr for conversion steps
BA19:  08         PHP            Store dec. and interrupt status
BA1A:  78         SEI            Disable all system interrupts
BA1B:  F8         SED            Decimal mode ON
BA1C:  46 68      LSR  * $68     Divide 3-byte value
BA1E:  66 67      ROR  # $67     in OP3
BA20:  66 66      ROR  # $66     by <2>
BA22:  90 0F      BCC  $BA33     NO REMAINDER-skip dec. add
BA24:  18         CLC            Clear carry for decimal addition
BA25:  A2 03      LDX  # $03     If a remainder is left from the
BA27:  BD A4 0A   LDA  $0AA4,X   division, add the contents
BA2A:  7D A0 0A   ADC  $0AA0,X   of the four-byte temp counter
BA2D:  9D A0 0A   STA  $0AA0,X   which is held (as power of 2)
BA30:  CA         DEX            in output memory
BA31:  10 F4      BPL  $BA27     (4 bytes=8 digits)
BA33:  18         CLC            Clear carry for decimal addition
BA34:  A2 03      LDX  # $03     Multiply contents of 4-byte
```

```
BA36:  BD A4 0A   LDA  $0AA4,X    counter by <2>
BA39:  7D A4 0A   ADC  $0AA4,X    The contents of the temp counter
BA3C:  9D A4 0A   STA  $0AA4,X    are always the power-of-two of
BA3F:  CA         DEX             the bit being processed in OP3
BA40:  10 F4      BPL  $BA36
BA42:  88         DEY             Decrement loop counter by 1
BA43:  10 D7      BPL  $BA1C      until all steps are processed
BA45:  28         PLP             amounts to an SED &CLI cmd
BA46:  60         RTS             Return from subroutine

******************************    Convert 3-byte OP1 operand to
                                  4-byte output operand OPA

BA47:  48         PHA             Put acc contents on stack
BA48:  A5 60      LDA  * $60      Copy OP1 (low-byte) into
BA4A:  8D A2 0A   STA  $0AA2      OPA (middle-low-byte)
BA4D:  A5 61      LDA  * $61      Copy OP1 (middle) into
BA4F:  8D A1 0A   STA  $0AA1      OPA (middle-high)
BA52:  A5 62      LDA  * $62      Copy OP1 (high) into
BA54:  8D A0 0A   STA  $0AA0      OPA (high)
BA57:  A9 00      LDA  # $00      Load acc with 00 and
BA59:  8D A3 0A   STA  $0AA3      copy into OPA (low)
BA5C:  68         PLA             Restore acc contents from stack

******************************    Output of the OPA operand
                                  corresponds to X & Y registers

BA5D:  8D B4 0A   STA  $0AB4      Set flag for zero-suppression
BA60:  8C B6 0A   STY  $0AB6      Store bit # for 1 output digit
BA63:  AC B6 0A   LDY  $0AB6      Get bit # for 1 output digit
BA66:  A9 00      LDA  # $00      Initialize acc as output storage
BA68:  0E A3 0A   ASL  $0AA3      Shift contents of
BA6B:  2E A2 0A   ROL  $0AA2      4-byte output operand
BA6E:  2E A1 0A   ROL  $0AA1      one bit position to
BA71:  2E A0 0A   ROL  $0AA0      the left. Store
BA74:  2A         ROL  A          MSB in accu
BA75:  88         DEY             Bit counter for 1 output digit - 1
BA76:  10 F0      BPL  $BA68      Loop until a digit is in acc
BA78:  A8         TAY             Secure output digit in Y
BA79:  D0 09      BNE  $BA84      If not equal to 0, then output
```

```
BA7B:  E0 01      CPX  # $01     Test for 1st place
BA7D:  F0 05      BEQ  $BA84     Yes, output digit in any case
BA7F:  AC B4 0A   LDY  $0AB4     Load zero-suppression flag
BA82:  F0 08      BEQ  $BA8C     Still active, don't output zero
BA84:  EE B4 0A   INC  $0AB4     Turn off zero suppression
BA87:  09 30      ORA  # $30     Load acc with <space> char.
BA89:  20 D2 FF   JSR  $FFD2     Kernal BSOUT: output a char
BA8C:  CA         DEX            Loop counter for num. of digits
BA8D:  D0 D4      BNE  $BA63     Not equal 0--output next digit
BA8F:  60         RTS            Return from subroutine

******************************   Monitor command: @
                                 (Disk command)

BA90:  D0 03      BNE  $BA95     Device addree identifier present
BA92:  A2 08      LDX  # $08     Set standard device address (8)
BA94:  2C         .Byte $2C      skip to $BA97

******************************   Disk command routine with
                                 parameter for device address

BA95:  A6 60      LDX  * $60     Get device # from OP1 (low)
BA97:  E0 04      CPX  # $04     Device number <4 is invalid
BA99:  90 65      BCC  $BB00     Display <?>--go input wait loop
BA9B:  E0 1F      CPX  # $1F     Device address >30 is invalid
BA9D:  B0 61      BCS  $BB00     Display <?>--go input wait loop
BA9F:  86 60      STX  * $60     Store device # in OP1 (low)
BAA1:  A9 00      LDA  # $00     Load bank # for LSV & filename
BAA3:  85 62      STA  * $62     Store in OP1 bank byte
BAA5:  85 B7      STA  * $B7     Set filename length to 0
BAA7:  AA         TAX            Clear acc + X-reg for SETBNK
BAA8:  20 68 FF   JSR  $FF68     Kernal SETBNK: Bank # for
                                 LSV+filename
BAAB:  20 E9 B8   JSR  $B8E9     Read a char. from input buffer
BAAE:  C6 7A      DEC  * $7A     Displ. pointer input buf -1 (like
                                 CHRGOT)
BAB0:  C9 24      CMP  # $24     Is char. read a <$> ?
BAB2:  F0 4F      BEQ  $BB03     Yes, then output directory
BAB4:  A9 00      LDA  # $00     Logical file number (0) in acc
BAB6:  A6 60      LDX  * $60     Get device # from OP1 (low)
```

```
BAB8:  A0 0F      LDY  # $0F     Set secondary addr. (15)
BABA:  20 BA FF   JSR  $FFBA     Kernal SETLFS: Set file param.
BABD:  20 C0 FF   JSR  $FFC0     Kernal OPEN: Open file
BAC0:  B0 32      BCS  $BAF4     OPEN error--CLRCH & exit
BAC2:  A2 00      LDX  # $00     Logical file (0) set as output
BAC4:  20 C9 FF   JSR  $FFC9     Kernal CKOUT: Set out channel
BAC7:  B0 2B      BCS  $BAF4     If error occurs, then exit
BAC9:  A6 7A      LDX  * $7A     Set display ptr. to input buffer
BACB:  E6 7A      INC  * $7A     and set to next char.
BACD:  BD 00 02   LDA  $0200,X   Read char. -input buffer, display
BAD0:  F0 05      BEQ  $BAD7     Cmd-end--close cmd channel
BAD2:  20 D2 FF   JSR  $FFD2     Kernal BSOUT: output a char
BAD5:  90 F2      BCC  $BAC9     OK, output next character
BAD7:  20 CC FF   JSR  $FFCC     Kernal CLRCH: I/O chnl reset
BADA:  20 B4 B8   JSR  $B8B4     <C/R> + clear rest of line
BADD:  A2 00      LDX  # $00     Set logical file (0) as input
BADF:  20 C6 FF   JSR  $FFC6     Kernal CHKIN: Set input chnl
BAE2:  B0 10      BCS  $BAF4     If error occurs, then exit
BAE4:  20 CF FF   JSR  $FFCF     Kernal BASIN: read a character
BAE7:  20 D2 FF   JSR  $FFD2     Kernal BSOUT: output a char
BAEA:  C9 0D      CMP  # $0D     Has <CR> been printed ?
BAEC:  F0 06      BEQ  $BAF4     Yes, CLRCH and exit routine
BAEE:  A5 90      LDA  * $90     Load system status in acc
BAF0:  29 BF      AND  # $BF     Mask out bit 6 (= end-of-file)
BAF2:  F0 F0      BEQ  $BAE4     No error? Continue...
BAF4:  20 CC FF   JSR  $FFCC     Kernal CLRCH: I/O chnl reset
BAF7:  A9 00      LDA  # $00     Completely close logical file (0)
BAF9:  38         SEC            Set carry for CLOSE routine
BAFA:  20 C3 FF   JSR  $FFC3     Kernal CLOSE: Close file
BAFD:  4C 8B B0   JMP  $B08B     Jump to input wait loop
BB00:  4C BC B0   JMP  $B0BC     Display <?>--go input wait loop

****************************     Routine for disk directory

BB03:  A0 FF      LDY  # $FF     Set filename length counter to -1
BB05:  A6 7A      LDX  * $7A     Get display pntr to input buffer
BB07:  CA         DEX            and set to preceding char.
BB08:  C8         INY            Increment filename counter
BB09:  E8         INX            Display pointer to next char.
BB0A:  BD 00 02   LDA  $0200,X   Read char. -input buf., display
```

```
BB0D:  D0 F9      BNE  $BB08   No cmd-end, then next char.
BB0F:  98         TYA          Copy filename length into A
BB10:  A6 7A      LDX  * $7A   Load filename addr.(low) X-reg
BB12:  A0 02      LDY  # $02   Load filename addr.(hi) Y-reg
BB14:  20 BD FF   JSR  $FFBD   Kernal SETNAM: Set filename
BB17:  A9 00      LDA  # $00   Logical file (0) in acc
BB19:  A6 60      LDX  * $60   Get dvc # from OP1 (low)
BB1B:  A0 60      LDY  # $60   Get secondary address (96)
BB1D:  20 BA FF   JSR  $FFBA   Kernal SETLFS: Set file param.
BB20:  20 C0 FF   JSR  $FFC0   Kernal OPEN: Open file
BB23:  B0 CF      BCS  $BAF4   If error occurs, then exit
BB25:  A2 00      LDX  # $00   Set log. file (0) as input
BB27:  20 C6 FF   JSR  $FFC6   Kernal CHKIN: Set input chnl
BB2A:  20 B4 B8   JSR  $B8B4   <C/R>+clear rest of line
BB2D:  A0 03      LDY  # $03   Counter reads first
BB2F:  84 63      STY  * $63   six directory bytes
BB31:  20 CF FF   JSR  $FFCF   Kernal BASIN: read a character
BB34:  85 60      STA  * $60   Store dir char. in OP1 (low)
BB36:  A5 90      LDA  * $90   Load system status in acc
BB38:  D0 BA      BNE  $BAF4   If error occurs, then exit
BB3A:  20 CF FF   JSR  $FFCF   Kernal BASIN: read a character
BB3D:  85 61      STA  * $61   Store directory char. in OP1 (hi)
BB3F:  A5 90      LDA  * $90   Load system status in acc
BB41:  D0 B1      BNE  $BAF4   If error occurs, then exit
BB43:  C6 63      DEC  * $63   Decrement dir. bytes skip cntr
BB45:  D0 EA      BNE  $BB31   Not equal to 0, read more bytes
BB47:  20 07 BA   JSR  $BA07   Prep. & display OP1 contents
BB4A:  A9 00      LDA  # $00   in decimal form: Output the
BB4C:  A2 08      LDX  # $08   length of a directory entry
BB4E:  A0 03      LDY  # $03   and number
BB50:  20 5D BA   JSR  $BA5D   of blocks free
BB53:  A9 20      LDA  # $20   Load acc with a <space> char.
BB55:  20 D2 FF   JSR  $FFD2   Kernal BSOUT: Char. output
BB58:  20 CF FF   JSR  $FFCF   Kernal BASIN: Read a character
BB5B:  F0 09      BEQ  $BB66   $0 is signal - end of 1st dir. line
BB5D:  A6 90      LDX  * $90   Load system STATUS in X-reg
BB5F:  D0 93      BNE  $BAF4   If error occurs, then exit
BB61:  20 D2 FF   JSR  $FFD2   Kernal BSOUT: print a character
BB64:  90 F2      BCC  $BB58   Output next char. in dir. line
BB66:  20 B4 B8   JSR  $B8B4   <C/R>+clear rest of line
```

```
BB69:  20 E1 FF   JSR  $FFE1      Kernal STOP: test for STOP key
BB6C:  F0 86      BEQ  $BAF4      If STOP, goto exit routine
BB6E:  A0 02      LDY  # $02      Read counter for 4 dir. bytes
BB70:  D0 BD      BNE  $BB2F      Unconditional jump to dir. read

******************************    END OF ROM monitor


BB72:  FF FF FF . . .             Fill characters

BFFB:  . . . FF FF FF

BFFE:  00 3A

******************************
```

```
*****************************   Jump table for editor routines

C000:  4C 7B C0    JMP  $C07B   CINT initializes editor & screen
C003:  4C 34 CC    JMP  $CC34   DISPLAY char in A, color in X
C006:  4C 34 C2    JMP  $C234   LP2 gets a char from IRQ buffer
C009:  4C 9B C2    JMP  $C29B   LOOP5 a char from the screen
C00C:  4C 2D C7    JMP  $C72D   PRINT vector for screen output
C00F:  4C 5B CC    JMP  $CC5B   SCRORG returns screen width
C012:  4C 5D C5    JMP  $C55D   KEY read key
       4C 87 FC    JMP  $FC87   (International versions only)
C015:  4C 51 C6    JMP  $C651   REPEAT the keyboard logic
C018:  4C 6A CC    JMP  $CC6A   PLOT sets/reads cursor position
C01B:  4C 57 CD    JMP  $CD57   CURSOR moves 80-cln cursor
C01E:  4C C1 C9    JMP  $C9C1   ESCAPE outputs ESC sequence
C021:  4C A2 CC    JMP  $CCA2   PFKEY defines a function key
C024:  4C 94 C1    JMP  $C194   IRQ jumps to editor IRQ routine
C027:  4C 0C CE    JMP  $CE0C   INIT80 initializes 80-column
C02A:  4C 2E CD    JMP  $CD2E   SWAPPER exch. 40/80 column
C02D:  4C 1B CA    JMP  $CA1B   WINDOW sets left/top or
                                right/lower corner of window
C030:  FF FF FF                 Free for future extensions

*****************************   Line starts, low bytes

C033:  00 28 50 78 A0 C8 F0 18  $0400, $0428, $0450
C03B:  40 68 90 B8 E0 08 30 58  $0478, $04A0, $04C8
C043:  80 A8 D0 F8 20 48 70 98  $04F0, $0518, $0540
C04B:  C0                       $0568, $0590, $05B8

*****************************   Line starts, high bytes

C04C:  04 04 04 04 04 04 04 05  $05E0, $0608, $0630
C054:  05 05 05 05 05 06 06 06  $0658, $0680, $06A8
C05C:  06 06 06 06 07 07 07 07  $06D0, $06F8, $0720
C064:  07                       $0748, $0770, $0798, $07C0

*****************************   Character output and keyboard
                                vectors
C065:  B9 C7       ($C7B9)      Entry: char output with CTRL
C067:  05 C8       ($C805)      Entry: char output with SHIFT
```

```
C069:  C1 C9       ($C9C1)      Entry: character output with ESC
C06B:  E1 C5       ($C5E1)      Entry: evaluate keyboard
C06D:  AD C6       ($C6AD)      Entry: Store keypress

******************************  Pointer to keyboard decoder
                                 table

C06F:  80 FA       ($FA80)      Keyboard decoder table 1a
C071:  D9 FA       ($FAD9)      Keyboard decoder table 2a
C073:  32 FB       ($FB32)      Keyboard decoder table 3a
C075:  8B FB       ($FB8B)      Keyboard decoder table 4a
C077:  80 FA       ($FA80)      Keyboard decoder table 1a
C079:  E4 FB       ($FBE4)      Keyboard decoder table 5a

******************************  Kernal routine: CINT
                                 Initialize editor and screen

C07B:  A9 03       LDA  # $03   Two highest-order bits of base
C07D:  0D 00 DD    ORA  $DD00   Set video because active-low
C080:  8D 00 DD    STA  $DD00   And save again
C083:  A9 FB       LDA  # $FB   Clear bit 2 of the data-direction
C085:  25 01       AND  * $01   Register and then set bit 1 of the
C087:  09 02       ORA  # $02   Data direction register and
C089:  85 01       STA  * $01   Save again
C08B:  20 CC FF    JSR  $FFCC   Kernal CLRCH: reset I/O chnls
C08E:  A9 00       LDA  # $00   Reset filter, volume, and entry in
C090:  20 80 FC    JSR  $FC80   Table for logged in cards
C093:  85 D8       STA  * $D8   Set text screen flag to "text"
C095:  85 D7       STA  * $D7   Set 40/80 column flag to "40"
C097:  85 D0       STA  * $D0   Clear keyboard buffer queue
C099:  85 D1       STA  * $D1   Clear function key flag
C09B:  85 D6       STA  * $D6   Reset keyboard input/get flag
C09D:  8D 21 0A    STA  $0A21   Reset pause (Ctrl-S) flag
C0A0:  8D 26 0A    STA  $0A26   Reset cursor-flash flag
C0A3:  85 D9       STA  * $D9   Pointer - char set in RAM/ROM
C0A5:  8D 2E 0A    STA  $0A2E   Base address - screen text RAM
C0A8:  A9 14       LDA  # $14   Init. value for base pointer
C0AA:  8D 2C 0A    STA  $0A2C   Text screen/char base pointer
C0AD:  A9 78       LDA  # $78   Initialization value bit-map base
C0AF:  8D 2D 0A    STA  $0A2D   Initialize bit-map base
```

```
C0B2:  A9 08     LDA  # $08     Initialization value attribute RAM
C0B4:  8D 2F 0A  STA  $0A2F     Initialize attribute RAM base
C0B7:  AD 4C C0  LDA  $C04C     Load initialization value ($04)
C0BA:  8D 3B 0A  STA  $0A3B     Initialize PAL system pointer
C0BD:  A9 0A     LDA  # $0A     Start value-keyboard buffer size
C0BF:  8D 20 0A  STA  $0A20     Init. flag - keyboard buffer size
C0C2:  8D 28 0A  STA  $0A28     Count pointer for flashing cursor
C0C5:  8D 27 0A  STA  $0A27     Flag for cursor flash mode
C0C8:  8D 24 0A  STA  $0A24     Flag: keyboard repeat delay
C0CB:  A9 04     LDA  # $04     Start value for count speed
C0CD:  8D 23 0A  STA  $0A23     Flag: repeat speed
C0D0:  20 83 C9  JSR  $C983     Initialize TAB positions
C0D3:  8D 22 0A  STA  $0A22     Flag for keyboard repeat pointer
C0D6:  0D 05 D5  ORA  $D505     Set the fast serial control bit in
C0D9:  8D 05 D5  STA  $D505     the MCR of the MMU
C0DC:  A9 60     LDA  # $60     Start value current cursor mode
C0DE:  8D 2B 0A  STA  $0A2B     Flag for current cursor mode
C0E1:  A9 D0     LDA  # $D0     Initialization value for the system
C0E3:  8D 34 0A  STA  $0A34     pointers: clear/move line
C0E6:  A2 1A     LDX  # $1A     Loop counter for z-page init.
C0E8:  BD 74 CE  LDA  $CE74,X   ROM copy of the 40-clm screen
C0EB:  95 E0     STA  * $E0,X   Copy start values in zero page
C0ED:  BD 8E CE  LDA  $CE8E,X   ROM copy of the 80-clm screen
C0F0:  9D 40 0A  STA  $0A40,X   Copy start values into RAM
C0F3:  CA        DEX            Decrement loop counter by 1
C0F4:  10 F2     BPL  $C0E8     Loop until all values transferred
C0F6:  A2 09     LDX  # $09     Loop counter for page 3 init.
C0F8:  BD 65 C0  LDA  $C065,X   ROM copy of the character and
C0FB:  9D 34 03  STA  $0334,X   Keyboard vectors into RAM area
C0FE:  CA        DEX            Decrement loop counter by 1
C0FF:  10 F7     BPL  $C0F8     Loop until all values transferred
C101:  2C 04 0A  BIT  $0A04     Check bit 6 of the init. flag
C104:  70 1E     BVS  $C124     Bit set, then skip
C106:  A2 0B     LDX  # $0B     Loop counter for page 3 init.
C108:  BD 6F C0  LDA  $C06F,X   ROM copy of the keyboard de-
C10B:  9D 3E 03  STA  $033E,X   coder. Table vectors RAM area
C10E:  CA        DEX            Decrement loop counter by 1
C10F:  10 F7     BPL  $C108     Loop until all values transferred
C111:  A2 4C     LDX  # $4C     Loop cntr for function key init.
C113:  BD A8 CE  LDA  $CEA8,X   Copy ROM copy of the f-key
```

```
C116:  9D 00 10   STA  $1000,X   lengths and strings into RAM
C119:  CA         DEX            Decrement loop counter by 1
C11A:  10 F7      BPL  $C113     Loop until all values transferred
C11C:  A9 40      LDA  # $40     Set bit 6 to "ON" and combine
C11E:  0D 04 0A   ORA  $0A04     with initialization flag
C121:  8D 04 0A   STA  $0A04     Place result in init. flag
C124:  20 2E CD   JSR  $CD2E     Switch 40/80 column mode
C127:  20 83 C9   JSR  $C983     Reset the tabs
C12A:  20 24 CA   JSR  $CA24     Window=whole screen
C12D:  20 42 C1   JSR  $C142     CLR/HOME
C130:  20 2E CD   JSR  $CD2E     Switch 40/80-column mode
C133:  20 24 CA   JSR  $CA24     Window=whole screen
C136:  20 42 C1   JSR  $C142     CLR/HOME
C139:  2C 05 D5   BIT  $D505     Test if 40/80-column mode
C13C:  30 03      BMI  $C141     Jump if 80
C13E:  20 2E CD   JSR  $CD2E     Switch 40/80-column mode
C141:  60         RTS            Return from subtroutine

******************************   Clear window (CLR/HOME)

C142:  20 50 C1   JSR  $C150     Cursor home
C145:  20 5E C1   JSR  $C15E     Calculate start address of line X
C148:  20 A5 C4   JSR  $C4A5     Clear line X
C14B:  E4 E4      CPX  * $E4     Compare  lower window border
C14D:  E8         INX            Increment line pointer
C14E:  90 F5      BCC  $C145     If lower border not reached

******************************   Cursor home in window

C150:  A6 E5      LDX  * $E5     Load upper window border into
C152:  86 EB      STX  * $EB     X-reg. Write curent cursor line
C154:  86 E8      STX  * $E8     Store as start input line
C156:  A4 E6      LDY  * $E6     Load left window border Y-reg
C158:  84 EC      STY  * $EC     Store the current cursor column
C15A:  84 E9      STY  * $E9     And as start input column
```

```
******************************      Set address of current line

C15C:  A6 EB      LDX  * $EB      Get current cursor line in X-reg
C15E:  BD 33 C0   LDA  $C033,X    Get low-byte of start line
C161:  24 D7      BIT  * $D7      Test 40/80-column mode
C163:  10 01      BPL  $C166      Jump if 40-column mode
C165:  0A         ASL  A          Otherwise address times two
C166:  85 E0      STA  * $E0      Store low byte
C168:  BD 4C C0   LDA  $C04C,X    Get high byte of the start line
C16B:  29 03      AND  # $03      Mask out bits 2-7=X MOD 4
C16D:  24 D7      BIT  * $D7      Test 40/80-column mode
C16F:  10 06      BPL  $C177      Jump if 40-column mode
C171:  2A         ROL  A          Else shift carry into high byte
C172:  0D 2E 0A   ORA  $0A2E      And add to video start address
C175:  90 03      BCC  $C17A      Unconditional jump to $C17A
C177:  0D 3B 0A   ORA  $0A3B      Video start address  40-column
C17A:  85 E1      STA  * $E1      Store high byte

******************************    Adapt attribute RAM address

C17C:  A5 E0      LDA  * $E0      Current screen line, low byte
C17E:  85 E2      STA  * $E2      To low byte of attrbute address
C180:  A5 E1      LDA  * $E1      Get high byte current screen line
C182:  24 D7      BIT  * $D7      Test for 40/80-column mode
C184:  10 07      BPL  $C18D      40-column mode is active
C186:  29 07      AND  # $07      Mask out bits 3-7
C188:  0D 2F 0A   ORA  $0A2F      Add attrbute RAM base
C18B:  D0 04      BNE  $C191      Unconditional jump
C18D:  29 03      AND  # $03      Mask out bits 2-7
C18F:  09 D8      ORA  # $D8      Add base of color RAM
C191:  85 E3      STA  * $E3      Store the attribute high byte
C193:  60         RTS             Return from the subroutine

******************************    IRQ routine

C194:  38         SEC             Set carry flag as FLAG
C195:  AD 19 D0   LDA  $D019      Load IRR from VIC
C198:  29 01      AND  # $01      Test raster-line interrupt bit
C19A:  F0 07      BEQ  $C1A3      If not set then jump
C19C:  8D 19 D0   STA  $D019      Clear the register
```

```
C19F:  A5 D8      LDA  * $D8      Test text/graphics
C1A1:  C9 FF      CMP  # $FF      If graphics screen enabled
C1A3:  F0 6F      BEQ  $C214      Then to appropriate routine
C1A5:  2C 11 D0   BIT  $D011      Test VIC control register 1
C1A8:  30 04      BMI  $C1AE      High byte of rester line is set
C1AA:  29 40      AND  # $40      Test extended-color mode
C1AC:  D0 31      BNE  $C1DF      Is set
C1AE:  38         SEC             Setset carry as FLAG
C1AF:  A5 D8      LDA  * $D8      Get text/graphic mode
C1B1:  F0 2C      BEQ  $C1DF      Text mode - jump
C1B3:  24 D8      BIT  * $D8      Test text/graphic mode
C1B5:  50 06      BVC  $C1BD      Bit 6=0 means no raster line
C1B7:  AD 34 0A   LDA  $0A34      IRQ. Else get raster line
C1BA:  8D 12 D0   STA  $D012      and refresh storage
C1BD:  A5 01      LDA  * $01      Get data-direction register and
C1BF:  29 FD      AND  # $FD      Mask out bits 0-1
C1C1:  09 04      ORA  # $04      Set bit 2 of the register
C1C3:  48         PHA             And save configuration on stack
C1C4:  AD 2D 0A   LDA  $0A2D      Base address of the graphics
C1C7:  48         PHA             Save base address on stack
C1C8:  AD 11 D0   LDA  $D011      Get control register 1 of the VIC
C1CB:  29 7F      AND  # $7F      Clear raster line 1 carry and
C1CD:  09 20      ORA  # $20      Set standard bit-map mode
C1CF:  A8         TAY             Control register to Y
C1D0:  AD 16 D0   LDA  $D016      Get VIC control register 2
C1D3:  24 D8      BIT  * $D8      Test text/graphic register
C1D5:  30 03      BMI  $C1DA      Multi-color mode set
C1D7:  29 EF      AND  # $EF      Clear multi-color bit
C1D9:  2C         .Byte  $2C      Skip to $C1DC
C1DA:  09 10      ORA  # $10      Set multi-color bit
C1DC:  AA         TAX             Control register 2 to X
C1DD:  D0 28      BNE  $C207      Unconditional jump

******************************    Text mode

C1DF:  A9 FF      LDA  # $FF      Raster line is last line
C1E1:  8D 12 D0   STA  $D012      Store as raster line
C1E4:  A5 01      LDA  * $01      Get data direction register
C1E6:  09 02      ORA  # $02      Set bit 1 of the register
C1E8:  29 FB      AND  # $FB      And clear bit 2
```

```
C1EA:  05 D9      ORA  * $D9      Bit 2 is then cleared
C1EC:  48         PHA             If CHARROM in RAM. Also
C1ED:  AD 2C 0A   LDA  $0A2C      storebase address of text/graphic
C1F0:  48         PHA             On the stack
C1F1:  AD 11 D0   LDA  $D011      Get VIC control register
C1F4:  29 5F      AND  # $5F      Clear carry and graphics
C1F6:  A8         TAY             Control register 1 to Y
C1F7:  AD 16 D0   LDA  $D016      Get VIC control register 2
C1FA:  29 EF      AND  # $EF      Clear multi-color bit
C1FC:  AA         TAX             Control register 2 to X
C1FD:  B0 08      BCS  $C207      Carry set=don't wait
C1FF:  A2 07      LDX  # $07      X is counter for delay loop
C201:  CA         DEX             Decrement the counter
C202:  D0 FD      BNE  $C201      And jump if not done
C204:  EA         NOP             Two NOPs in the delay loop
C205:  EA         NOP             To perfect it
C206:  AA         TAX             Control register 2 back to X

****************************      Set the IRQ register

C207:  68         PLA             Get base address back
C208:  8D 18 D0   STA  $D018      And base address to VIC
C20B:  68         PLA             Get data direction register from
C20C:  85 01      STA  * $01      Stack and save
C20E:  8C 11 D0   STY  $D011      Control register 1 to VIC
C211:  8E 16 D0   STX  $D016      And control register 2 to VIC
C214:  B0 13      BCS  $C229      If carry set then skip
C216:  AD 30 D0   LDA  $D030      Get 1/2 MHz clock register
C219:  29 01      AND  # $01      Mask out relevant bit
C21B:  F0 0C      BEQ  $C229      Jump if 1 MHz
C21D:  A5 D8      LDA  * $D8      Get text/graphic mode
C21F:  29 40      AND  # $40      Test raster-line interrupt bit
C221:  F0 06      BEQ  $C229      No raster-line interrupt
C223:  AD 11 D0   LDA  $D011      Get control register 1
C226:  10 01      BPL  $C229      No carry - jump
C228:  38         SEC             Set carry as FLAG
C229:  58         CLI             Enable all system interrupts
C22A:  90 07      BCC  $C233      Done if FLAG not set
C22C:  20 87 FC   JSR  $FC87      Call the kernal routine KEY
C22F:  20 E7 C6   JSR  $C6E7      Let VIC cursor flash
```

```
C232:  38          SEC              Set carry for OK
C233:  60          RTS              Return from subroutine

******************************      Get character from KEY

C234:  A6 D1       LDX  * $D1       Must characters be fetched from
C236:  F0 0C       BEQ  $C244       keyboard buffer? NO
C238:  A4 D2       LDY  * $D2       Get pointer to KEY buffer
C23A:  B9 0A 10    LDA  $100A,Y     Get character from KEY table
C23D:  C6 D1       DEC  * $D1       Decrement the character counter
C23F:  E6 D2       INC  * $D2       Increment the pointer
C241:  58          CLI              Enable all system interrupts
C242:  18          CLC              Clear carry for "char. fetched"
C243:  60          RTS              Return from subroutine

******************************      Get character from buffer

C244:  AC 4A 03    LDY  $034A       How many chars in the queue?
C247:  BD 4B 03    LDA  $034B,X     Get character from queue
C24A:  9D 4A 03    STA  $034A,X     And shift forward
C24D:  E8          INX              Increment the counter and move
C24E:  E4 D0       CPX  * $D0       Characters until all characters in
C250:  D0 F5       BNE  $C247       the queue are moved forward
C252:  C6 D0       DEC  * $D0       Offset of the keyboard queue -1
C254:  98          TYA              Character to acc.
C255:  58          CLI              Enable all system interrupts
C256:  18          CLC              Clear carry for "char. fetched"
C257:  60          RTS              Return from subroutine

******************************      Get input line (w/<CR>) LOOP4

C258:  20 2D C7    JSR  $C72D       Output character
C25B:  20 6F CD    JSR  $CD6F       Move cursor
C25E:  A5 D0       LDA  * $D0       # of chars in the keyboard buffer
C260:  05 D1       ORA  * $D1       Plus # of chars in  KEY buffer
C262:  F0 FA       BEQ  $C25E       If empty then wait
C264:  20 9F CD    JSR  $CD9F       Set cursor
C267:  20 34 C2    JSR  $C234       Get character from buffer
C26A:  C9 0D       CMP  # $0D       Is character <CR>
C26C:  D0 EA       BNE  $C258       No, then get next char.
```

```
C26E:  85 D6      STA  * $D6     Set input flag
C270:  A9 00      LDA  # $00     Clear cursor mode flag
C272:  85 F4      STA  * $F4
C274:  20 C3 CB   JSR  $CBC3     Determine end of input line
C277:  8E 30 0A   STX  $0A30     Save last column position
C27A:  20 B5 CB   JSR  $CBB5     Set line start
C27D:  A4 E6      LDY  * $E6     Load left window-border, Y-reg
C27F:  A5 E8      LDA  * $E8     Start of running input line
C281:  30 13      BMI  $C296     Input line is following line
C283:  C5 EB      CMP  * $EB     Compare with current cursor line
C285:  90 0F      BCC  $C296     Border not reached
C287:  A4 E9      LDY  * $E9     Start of  running input column
C289:  CD 30 0A   CMP  $0A30     Compare with last input column
C28C:  D0 04      BNE  $C292     Is not the same column
C28E:  C4 EA      CPY  * $EA     Compare with end of running in
C290:  F0 02      BEQ  $C294     line is reached
C292:  B0 11      BCS  $C2A5     Set input/get flag to get
C294:  85 EB      STA  * $EB     Write current cursor line
C296:  84 EC      STY  * $EC     Store the current cursor column
C298:  4C BC C2   JMP  $C2BC     Get character at cursor pos./

****************************     Get character from screen

C29B:  98         TYA            Y-register (column) via acc
C29C:  48         PHA            Save on stack
C29D:  8A         TXA            X-register (line) via
C29E:  48         PHA            Store on stack
C29F:  A5 D6      LDA  * $D6     Get input/get flag
C2A1:  F0 B8      BEQ  $C25B     To delay loop for GET
C2A3:  10 17      BPL  $C2BC     No <CR> necessary yet
C2A5:  A9 00      LDA  # $00     The input/get flag is set via
C2A7:  85 D6      STA  * $D6     The accumulator
C2A9:  A9 0D      LDA  # $0D     ASCII code for <CR>
C2AB:  A2 03      LDX  # $03     Compare code for screen with
C2AD:  E4 99      CPX  * $99     Standard input device
C2AF:  F0 04      BEQ  $C2B5     Input device is screen
C2B1:  E4 9A      CPX  * $9A     compare with standard output d
C2B3:  F0 03      BEQ  $C2B8     device. Output to screen
C2B5:  20 2D C7   JSR  $C72D     BSOUT entry screen
C2B8:  A9 0D      LDA  # $0D     ASCII code for <CR>
```

```
C2BA:  D0 39      BNE  $C2F5      Unconditional jump to end

*****************************     Character at cursor pos in ASCII

C2BC:  20 5C C1   JSR  $C15C      Get address of current line
C2BF:  20 58 CB   JSR  $CB58      Character and color at cursor pos
C2C2:  85 EF      STA  * $EF      Temp storage for print character
C2C4:  29 3F      AND  # $3F      Mask out bits 6/7
C2C6:  06 EF      ASL  * $EF      The character is then converted
C2C8:  24 EF      BIT  * $EF      to ASCII
C2CA:  10 02      BPL  $C2CE      Not a reverse character
C2CC:  09 80      ORA  # $80      Set bit 7
C2CE:  90 04      BCC  $C2D4      Test former bit 7
C2D0:  A6 F4      LDX  * $F4      Flag for quote mode active
C2D2:  D0 04      BNE  $C2D8      Is active, then jump
C2D4:  70 02      BVS  $C2D8      Test former bit 6
C2D6:  09 40      ORA  # $40      Set bit 6 for ASCII
C2D8:  20 FF C2   JSR  $C2FF      Test for " and set flags
C2DB:  A4 EB      LDY  * $EB      Get current cursor line in Y-reg
C2DD:  CC 30 0A   CPY  $0A30      Last column already reached?
C2E0:  90 0A      BCC  $C2EC      No, not yet
C2E2:  A4 EC      LDY  * $EC      Get current cursor column X-reg
C2E4:  C4 EA      CPY  * $EA      Compare with end
C2E6:  90 04      BCC  $C2EC      End line not yet reached
C2E8:  66 D6      ROR  # $D6      Shift carry into bit 7 of $D6
C2EA:  30 03      BMI  $C2EF      If set then new line
C2EC:  20 ED CB   JSR  $CBED      Cursor one position right
C2EF:  C9 DE      CMP  # $DE      Compare to ASCII "PI"
C2F1:  D0 02      BNE  $C2F5      Is not pi
C2F3:  A9 FF      LDA  # $FF      Else load adapted pi code
C2F5:  85 EF      STA  * $EF      Store as print character
C2F7:  68         PLA             Get X-register (line) via
C2F8:  AA         TAX             Acc from stack
C2F9:  68         PLA             Get Y-register (column)
C2FA:  A8         TAY             Via ac from stack
C2FB:  A5 EF      LDA  * $EF      Print char from temp storage
C2FD:  18         CLC             Flag for OK
C2FE:  60         RTS             Return from the subroutine
```

```
******************************    Test for (") and set flags

C2FF:  C9 22      CMP  # $22      Compare to quote
C301:  D0 08      BNE  $C30B      Other character, then end
C303:  A5 F4      LDA  * $F4      Get current quote mode
C305:  49 01      EOR  # $01      Reverse mode
C307:  85 F4      STA  * $F4      And store again
C309:  A9 22      LDA  # $22      Reload acc with ASCII value
C30B:  60         RTS             Return from subroutine

******************************    BSOUT continuation

C30C:  A5 EF      LDA  * $EF      Save current print character as
C30E:  85 F0      STA  * $F0      Last-printed character
C310:  20 57 CD   JSR  $CD57      Set cursor to current column
C313:  A5 F5      LDA  * $F5      Get insert mode flag
C315:  F0 02      BEQ  $C319      Insert mode is not active
C317:  46 F4      LSR  * $F4      Shift quote mode flag
C319:  68         PLA             Get first value from stack
C31A:  A8         TAY             And into Y-register
C31B:  68         PLA             Get second value from stack
C31C:  AA         TAX             And into X-register
C31D:  68         PLA             Get acc from stack
C31E:  18         CLC             Clear carry for OK
C31F:  60         RTS             Return from subroutine

******************************    Convert from ASCII to
                                  POKE-Code

C320:  09 40      ORA  # $40      Set bit 2 of the acc
C322:  A6 F3      LDX  * $F3      Get flag for RVS mode active
C324:  F0 02      BEQ  $C328      on/off. Not reverse character
C326:  09 80      ORA  # $80      Set high-rder bit (reverse)
C328:  A6 F5      LDX  * $F5      Insert-mode flag
C32A:  F0 02      BEQ  $C32E      No insert mode
C32C:  C6 F5      DEC  * $F5      Decrement the counter
C32E:  24 F6      BIT  * $F6      Test auto-insert flag
C330:  10 09      BPL  $C33B      Jump if not active
C332:  48         PHA             Save acc on the stack
C333:  20 E3 C8   JSR  $C8E3      Screen mode behind cursor
```

```
C336:  A2 00      LDX  # $00      Set insert-mode flag
C338:  86 F5      STX  * $F5      Back to zero
C33A:  68         PLA             Get acc from stack again
C33B:  20 2F CC   JSR  $CC2F      Output character at current pos

*****************************     Cursor at line end

C33E:  C4 E7      CPY  * $E7      Compare, right window-border
C340:  90 0A      BCC  $C34C      Right edge not yet reached
C342:  A6 EB      LDX  * $EB      Get current cursor line in X-reg
C344:  E4 E4      CPX  * $E4      Compare, lower window border
C346:  90 04      BCC  $C34C      Lower border not yet reached
C348:  24 F8      BIT  * $F8      Test scroll flag
C34A:  30 16      BMI  $C362      No scrolling then end
C34C:  20 5C C1   JSR  $C15C      Determine start addr of curnt line
C34F:  20 ED CB   JSR  $CBED      Cursor one character to the right
C352:  90 0E      BCC  $C362      No new line
C354:  20 74 CB   JSR  $CB74      Test line overflow bit
C357:  B0 08      BCS  $C361      Line overflow bit is set
C359:  38         SEC             Set carry bit for no scrolling
C35A:  24 F8      BIT  * $F8      Test scroll bit
C35C:  70 04      BVS  $C362      Jump if no scrolling
C35E:  20 7C C3   JSR  $C37C      Insert line at X
C361:  18         CLC             Clear carry for scrolled
C362:  60         RTS             Return from the subroutine

*****************************     Perform linefeed


C363:  A6 EB      LDX  * $EB      Get current cursor line in X-reg
C365:  E4 E4      CPX  * $E4      Compare, lower window border
C367:  90 0E      BCC  $C377      Lower border not yet reached
C369:  24 F8      BIT  * $F8      Test scroll bit
C36B:  10 06      BPL  $C373      Scrolling possible
C36D:  A5 E5      LDA  * $E5      Load upper window border, acc
C36F:  85 EB      STA  * $EB      Write current cursor line
C371:  B0 06      BCS  $C379      Unconditional jump to $C379
C373:  20 A6 C3   JSR  $C3A6      Scrolling
C376:  18         CLC             Carry clear for OK, scrolled
C377:  E6 EB      INC  * $EB      Increment curent cursor line by 1
```

```
C379:  4C 5C C1   JMP  $C15C      Determ. start addr of current line

******************************    Insert line (at line X)

C37C:  A6 E8      LDX  * $E8      Start of the running input line
C37E:  30 06      BMI  $C386      Line is a following-line
C380:  E4 EB      CPX  * $EB      Compare with current cursor line
C382:  90 02      BCC  $C386      Cursorline reached?
C384:  E6 E8      INC  * $E8      Incr the start of running inp line
C386:  A6 E4      LDX  * $E4      Load low window-border X-reg
C388:  20 5E C1   JSR  $C15E      Set address of the current line
C38B:  A4 E6      LDY  * $E6      Load left window-border, Y-reg
C38D:  E4 EB      CPX  * $EB      Compare with current cursor line
C38F:  F0 0F      BEQ  $C3A0      Cursor line is lower border
C391:  CA         DEX             Decrement line by 1 and then
C392:  20 76 CB   JSR  $CB76      Test the line overflow bit
C395:  E8         INX             Back to the current line
C396:  20 83 CB   JSR  $CB83      Set/clear line overflow bit
C399:  CA         DEX             Back to previous line
C39A:  20 0D C4   JSR  $C40D      MOVLIN: copy a window line
C39D:  4C 88 C3   JMP  $C388      Back to the loop
C3A0:  20 A5 C4   JSR  $C4A5      Clear line X
C3A3:  4C 93 CB   JMP  $CB93      Set the line carry bit

******************************    Scroll up

C3A6:  A6 E5      LDX  * $E5      Load upper window-border in
C3A8:  E8         INX             X-reg & increment by 1 line
C3A9:  20 76 CB   JSR  $CB76      Test the line overflow bit
C3AC:  90 0A      BCC  $C3B8      No overflow in line
C3AE:  E4 E4      CPX  * $E4      Compare lower window-border
C3B0:  90 F6      BCC  $C3A8      Border not yet reached
C3B2:  A6 E5      LDX  * $E5      Load upper window-border in
C3B4:  E8         INX             X-reg &increment by 1
C3B5:  20 85 CB   JSR  $CB85      Set line overflow bit
C3B8:  C6 EB      DEC  * $EB      Decrement crnt cursor line by 1
C3BA:  24 E8      BIT  * $E8      Test bit 7 of the input start line
C3BC:  30 02      BMI  $C3C0      And jump if set
C3BE:  C6 E8      DEC  * $E8      Else decrement the input line
C3C0:  A6 E5      LDX  * $E5      Load upper window-border in
```

```
C3C2:  E4 DF      CPX  * $DF     X-reg. Compare with cursor line
C3C4:  B0 02      BCS  $C3C8     If >= upper border, then jump
C3C6:  C6 DF      DEC  * $DF     Decrement cursor line
C3C8:  20 DC C3   JSR  $C3DC     Move remaining screen
C3CB:  A6 E5      LDX  * $E5     Load upper window-border into
C3CD:  20 76 CB   JSR  $CB76     X-reg. Test line overflow bit
C3D0:  08         PHP            Save flags on stack
C3D1:  20 85 CB   JSR  $CB85     Clear overflow bit of current line
C3D4:  28         PLP            And get flags back
C3D5:  90 04      BCC  $C3DB     If carry clear then end
C3D7:  24 F8      BIT  * $F8     Else test scroll flag
C3D9:  30 CB      BMI  $C3A6     Bit 7 set then scroll
C3DB:  60         RTS            Return from subroutine

******************************   Clear line X (with move)

C3DC:  20 5E C1   JSR  $C15E     Announce line X
C3DF:  A4 E6      LDY  * $E6     Load left window-border, Y-reg
C3E1:  E4 E4      CPX  * $E4     Compare lower window border
C3E3:  B0 0F      BCS  $C3F4     Border is reached
C3E5:  E8         INX            Pointer points to following-line
C3E6:  20 76 CB   JSR  $CB76     Test line overflow bit
C3E9:  CA         DEX            Point to current line again
C3EA:  20 83 CB   JSR  $CB83     Set/clear line overflow bit
C3ED:  E8         INX            Point back to following-line
C3EE:  20 0D C4   JSR  $C40D     MOVLIN: copy window line
C3F1:  4C DC C3   JMP  $C3DC     Copy next line

******************************   Poll Commodore key - wait

C3F4:  20 A5 C4   JSR  $C4A5     Clear line X
C3F7:  A9 7F      LDA  * $7F     Flag or run/direct mode
C3F9:  8D 00 DC   STA  $DC00     In PRA inCIA for keyboard read
C3FC:  AD 01 DC   LDA  $DC01     Get keybaord matrix
C3FF:  C9 DF      CMP  # $DF     Commodore key pressed?
C401:  D0 09      BNE  $C40C     If not pressed then end
C403:  A0 00      LDY  # $00     Commodore key is pressed
C405:  EA         NOP            A delay loop is executed when
C406:  CA         DEX            Scrolling in order to delay the
C407:  D0 FC      BNE  $C405     Output somewhat
```

```
C409:  88          DEY            The loop counts from 0 to
C40A:  D0 F9       BNE  $C405     65536 and then stops
C40C:  60          RTS            Return from the subroutine

******************************    MOVLIN: Copy a window line

C40D:  24 D7       BIT  * $D7     Test 40/80-column mode
C40F:  30 25       BMI  $C436     Jump if 80-column mode
C411:  BD 33 C0    LDA  $C033,X   Get low byte of the current line
C414:  85 DC       STA  * $DC     Store  low byte in $DA & $DC
C416:  85 DA       STA  * $DA
C418:  BD 4C C0    LDA  $C04C,X   Get high byte of the current addr
C41B:  29 03       AND  # $03     Mask out bits 2-7
C41D:  0D 3B 0A    ORA  $0A3B     And OR with video base address
C420:  85 DB       STA  * $DB     And save
C422:  29 03       AND  # $03     Combine bits 0 & 1 with  base
C424:  09 D8       ORA  # $D8     Address of the color RAM
C426:  85 DD       STA  * $DD     And store as high byte
C428:  B1 DA       LDA  ($DA),Y   Get source character and save it
C42A:  91 E0       STA  ($E0),Y   at the destination address. Then
C42C:  B1 DC       LDA  ($DC),Y   Get the source color &store it at
C42E:  91 E2       STA  ($E2),Y   The source address too
C430:  C4 E7       CPY  * $E7     Compare, right window-border
C432:  C8          INY            Increment the column pointer
C433:  90 F3       BCC  $C428     Jump if not the end
C435:  60          RTS            Return from the subroutine

******************************    Copy a line in 80-column

C436:  8E 31 0A    STX  $0A31     Store line number temporarily
C439:  8C 32 0A    STY  $0A32     Store column
C43C:  A2 18       LDX  # $18     Register 24 contains COPY bit
C43E:  20 DA CD    JSR  $CDDA     And get register value
C441:  09 80       ORA  # $80     Set COPY bit and store
C443:  20 CC CD    JSR  $CDCC     Register back in VDC
C446:  20 E6 CD    JSR  $CDE6     Set update address to current pos
C449:  AE 31 0A    LDX  $0A31     Get the line to copy
C44C:  BD 33 C0    LDA  $C033,X   Low byte of the line to copy
C44F:  0A          ASL  A         Times two because 80-column
C450:  85 DA       STA  * $DA     And store low byte
```

```
C452:  BD 4C C0   LDA  $C04C,X   Get high byte of the line to copy
C455:  29 03      AND  # $03     And mask out bits 3-7
C457:  2A         ROL  A         Get carry in (*2)
C458:  0D 2E 0A   ORA  $0A2E     Add video RAM base
C45B:  85 DB      STA  * $DB     And save as high byte
C45D:  A2 20      LDX  # $20     Block-start address high
C45F:  18         CLC            Clear carry for addition
C460:  98         TYA            Get column in acc and
C461:  65 DA      ADC  * $DA     Add low byte
C463:  85 DA      STA  * $DA     Start addr.+column to low byte
C465:  A9 00      LDA  # $00     Load acc with zero in order to
C467:  65 DB      ADC  * $DB     Add to the high byte
C469:  85 DB      STA  * $DB     And save as the new high byte
C46B:  20 CC CD   JSR  $CDCC     And as the block-start address
C46E:  E8         INX            Pointer to block-start addr low
C46F:  A5 DA      LDA  * $DA     Get the low byte of the dest
C471:  20 CC CD   JSR  $CDCC     address and inform VDC
C474:  38         SEC            Set carry for subtraction
C475:  A6 E7      LDX  * $E7     Load right window-border into
C477:  E8         INX            X-reg. Plus one
C478:  8A         TXA            And then into acc
C479:  ED 32 0A   SBC  $0A32     Subtract the current column
C47C:  8D 32 0A   STA  $0A32     And save as number
C47F:  A2 1E      LDX  # $1E     VDC word-count register
C481:  20 CC CD   JSR  $CDCC     Start copying
C484:  A2 20      LDX  # $20     Block-start address high
C486:  A5 DB      LDA  * $DB     Get high byte of source address
C488:  29 07      AND  # $07     Mask bits 3-7 out
C48A:  0D 2F 0A   ORA  $0A2F     And add attribute RAM
C48D:  20 CC CD   JSR  $CDCC     Set the registers
C490:  E8         INX            Pointer to block-start address
C491:  A5 DA      LDA  * $DA     Low. Get source address low
C493:  20 CC CD   JSR  $CDCC     And set
C496:  20 F9 CD   JSR  $CDF9     Set update address for attribute
C499:  AD 32 0A   LDA  $0A32     Get number of chars to copy
C49C:  A2 1E      LDX  # $1E     Reg. 31 is word-count register
C49E:  20 CC CD   JSR  $CDCC     Copy
C4A1:  AE 31 0A   LDX  $0A31     Get current line back
C4A4:  60         RTS            Return from the subroutine
```

```
******************************   Clear (line X) 40 column

C4A5:  A4 E6      LDY  * $E6     Load left window-border, Y-reg
C4A7:  20 85 CB   JSR  $CB85     Clear line overflow bit
C4AA:  20 5E C1   JSR  $C15E     Get start address of line X
C4AD:  24 D7      BIT  * $D7     Test 40/80 column mode
C4AF:  30 0F      BMI  $C4C0     Jump if 80-column mode
C4B1:  88         DEY            Dummy decrement, is
                                 incremented again
C4B2:  C8         INY            Increment column pointer
C4B3:  A9 20      LDA  # $20     Load acc with <space>
C4B5:  91 E0      STA  ($E0),Y   Store space in video RAM
C4B7:  A5 F1      LDA  * $F1     Color code for char output in acc
C4B9:  91 E2      STA  ($E2),Y   Store color in color RAM
C4BB:  C4 E7      CPY  * $E7     Compare right window-border
C4BD:  D0 F3      BNE  $C4B2     Jump if not done
C4BF:  60         RTS            Return from the subroutine

******************************   Clear line - 80 column

C4C0:  8E 31 0A   STX  $0A31     Save X-register
C4C3:  8C 32 0A   STY  $0A32     Save Y-register
C4C6:  A2 18      LDX  # $18     Select register 24
C4C8:  20 DA CD   JSR  $CDDA     Get current value
C4CB:  29 7F      AND  # $7F     Clear copy bit
C4CD:  20 CC CD   JSR  $CDCC     And save new value
C4D0:  A2 12      LDX  # $12     Update address high
C4D2:  18         CLC            Clear carry for addition
C4D3:  98         TYA            Get column in acc
C4D4:  65 E0      ADC  * $E0     Add start address low
C4D6:  48         PHA            Store low address on stack
C4D7:  8D 3C 0A   STA  $0A3C     Store the low byte
C4DA:  A9 00      LDA  # $00     Load acc with zero in order to
C4DC:  65 E1      ADC  * $E1     Add the carry to the high byte
C4DE:  8D 3D 0A   STA  $0A3D     Store the high byte
C4E1:  20 CC CD   JSR  $CDCC     And put in the register
C4E4:  E8         INX            Update address low
C4E5:  68         PLA            Get low byte from stack
C4E6:  20 CC CD   JSR  $CDCC     Low byte to VDC
C4E9:  A9 20      LDA  # $20     Load acc with space
```

```
C4EB:  20 CA CD   JSR  $CDCA    And into VDC data register
C4EE:  38         SEC           Set carry for subtraction
C4EF:  A5 E7      LDA  * $E7    Load right window-border in acc
C4F1:  ED 32 0A   SBC  $0A32    Subtract start column
C4F4:  48         PHA           Save number on stack
C4F5:  F0 14      BEQ  $C50B    Start column = righ border
C4F7:  AA         TAX           Get number in X
C4F8:  38         SEC           Set carry for addition
C4F9:  6D 3C 0A   ADC  $0A3C    Add low byte
C4FC:  8D 3C 0A   STA  $0A3C    And save again
C4FF:  A9 00      LDA  # $00    Load acc with zero in order to
C501:  6D 3D 0A   ADC  $0A3D    Add the carry to the high byte
C504:  8D 3D 0A   STA  $0A3D    Save high byte
C507:  8A         TXA           Get number of characters in acc
C508:  20 3E C5   JSR  $C53E    Acc in word-count register
C50B:  A2 12      LDX  # $12    Update address high
C50D:  18         CLC           Clear carry for addition
C50E:  98         TYA           Get column in acc
C50F:  65 E2      ADC  * $E2    And add low byte attribute
C511:  48         PHA           Save low byte on stack
C512:  A9 00      LDA  # $00    Load acc with zero in order to
C514:  65 E3      ADC  * $E3    Add the carry
C516:  20 CC CD   JSR  $CDCC    And write the high byte into the
C519:  E8         INX           Register. Update address low
C51A:  68         PLA           Get low byte from stack
C51B:  20 CC CD   JSR  $CDCC    And write in register
C51E:  AD 3D 0A   LDA  $0A3D    Get high byte of dest address
C521:  29 07      AND  # $07    Mask out bits 4-7
C523:  0D 2F 0A   ORA  $0A2F    And combine with dest address
C526:  8D 3D 0A   STA  $0A3D    And save
C529:  A5 F1      LDA  * $F1    Color code for char output in acc
C52B:  29 8F      AND  # $8F    Only color & ALT bit relevant
C52D:  20 CA CD   JSR  $CDCA    Get reg contents from DATA reg
C530:  68         PLA           Get number from stack
C531:  F0 03      BEQ  $C536    If zero then jump
C533:  20 3E C5   JSR  $C53E    Output color
C536:  AE 31 0A   LDX  $0A31    Get X-register back
C539:  A4 E7      LDY  * $E7    Load right window-border Y-reg
C53B:  60         RTS           Return from subroutine
```

```
C53C:  A9 01      LDA  # $01     Load counter with one
C53E:  A2 1E      LDX  # $1E     Select word-count register
C540:  20 CC CD   JSR  $CDCC     And determine value
C543:  2C 00 D6   BIT  $D600     Test status bit
C546:  10 FB      BPL  $C543     And wait until done
C548:  A2 12      LDX  # $12     Update address high
C54A:  20 DA CD   JSR  $CDDA     Get current value
C54D:  CD 3D 0A   CMP  $0A3D     Compare w/ high byte dest addr.
C550:  90 EA      BCC  $C53C     Doesn't match--correct error
C552:  A2 13      LDX  # $13     Update address low
C554:  20 DA CD   JSR  $CDDA     Get current value
C557:  CD 3C 0A   CMP  $0A3C     Compare with dest address low
C55A:  90 E0      BCC  $C53C     Doesn't match-correct error
C55C:  60         RTS            Return from the subroutine
```

```
C55D:  A5 01      LDA  * $01     Get bit 6 from zero-page data reg
C55F:  29 40      AND  # $40     Processor port. Bit 6 indicates if
C561:  49 40      EOR  # $40     the 40 or 80 char set is selected
C563:  4A         LSR  A         Invert bit 6 and bring to bit
C564:  4A         LSR  A         Position 4. Reset shift flag
C565:  85 D3      STA  * $D3     And store 40/80 mode
C567:  A0 58      LDY  # $58     Code for "no key" in zero page
C569:  84 D4      STY  * $D4     Store pointer for pressed key
C56B:  A9 00      LDA  # $00     Check value for matrix lines
C56D:  8D 00 DC   STA  $DC00     Responsible for matrix lines 1-8
C570:  8D 2F D0   STA  $D02F     Responsible for matrix line 9-11
C573:  AE 01 DC   LDX  $DC01     Port B=input of matrix columns
C576:  E0 FF      CPX  # $FF     Check if a key is pressed
C578:  D0 03      BNE  $C57D     Check which key is pressed
C57A:  4C 97 C6   JMP  $C697     No key, then continue
C57D:  A8         TAY            Displ cntr start of keyboard table
C57E:  AD 3E 03   LDA  $033E     Copy address low of  keyboard
C581:  85 CC      STA  * $CC     decoding table 1a in zero page
C583:  AD 3F 03   LDA  $033F     Copy address high of keyboard
C586:  85 CD      STA  * $CD     decoding table 1a in zero page
```

```
C588:  A9 FF      LDA  # $FF      Test value for keyboard matrix
C58A:  8D 2F D0   STA  $D02F      Set test lines 9-11 to high
C58D:  2A         ROL  A          Bit position of the test line to 0
C58E:  24 D3      BIT  * $D3      Pointer if testing 1-8 or 9-11
C590:  30 05      BMI  $C597      If testing lines 9-11 then skip
C592:  8D 00 DC   STA  $DC00      Test value in Port A
                                  (matrix line 1-8)
C595:  10 03      BPL  $C59A      Skip test of matrix lines 9-11
C597:  8D 2F D0   STA  $D02F      Test port A* (matrix lines 9-11)
C59A:  A2 08      LDX  # $08      Set counter for 8 matrix columns
C59C:  48         PHA             Store line test value in acc
C59D:  AD 01 DC   LDA  $DC01      Compare port B (output the
C5A0:  CD 01 DC   CMP  $DC01      matrix columns) with port B
C5A3:  D0 F8      BNE  $C59D      And wait
C5A5:  4A         LSR  A          Test the output value of matrix
C5A6:  B0 17      BCS  $C5BF      Columns bit by bit. C=1 -no key
C5A8:  48         PHA             Store matrix clmns output value
C5A9:  B1 CC      LDA  ($CC),Y    Get key code from keybrd table
C5AB:  C9 08      CMP  # $08      Key code 8 is the ALT key
C5AD:  F0 08      BEQ  $C5B7      To corresponding evaluation
C5AF:  C9 05      CMP  # $05      Check if code for SHIFT, C=,
C5B1:  B0 09      BCS  $C5BC      or Ctrl. No, then continue
C5B3:  C9 03      CMP  # $03      Is it code for the BREAK key?
C5B5:  F0 05      BEQ  $C5BC      Yes then continue for break key
C5B7:  05 D3      ORA  * $D3      Zero-page pointer - shift pattern
C5B9:  85 D3      STA  * $D3      Combine with the acc
C5BB:  2C         .Byte  $2C      Skip to $C5BE
C5BC:  84 D4      STY  * $D4      Place in zero-page for key code
C5BE:  68         PLA             Get matrix columns text value
C5BF:  C8         INY             Keyboard table disp. counter + 1
C5C0:  CA         DEX             Matrix column loop counter - 1
C5C1:  D0 E2      BNE  $C5A5      Loop until all columns tested
C5C3:  C0 59      CPY  # $59      Are all lines and columns tested?
C5C5:  B0 10      BCS  $C5D7      Yes, then evaluate key press
C5C7:  68         PLA             Get line test value from stack
C5C8:  38         SEC             Set carry flag for shifting the
C5C9:  2A         ROL  A          Line test value
C5CA:  B0 C2      BCS  $C58E      Continue test matrix lines 1-8
C5CC:  8D 00 DC   STA  $DC00      Set port A test value high ($FF)
C5CF:  26 D3      ROL  * $D3      Merge bit 7 in shift pattern flag
```

```
C5D1:  38          SEC                Because remaining matrix lines
C5D2:  66 D3       ROR  # $D3         9-11 are tested via port A*
C5D4:  2A          ROL  A             Clear bit for matrix line test 9-11
C5D5:  D0 B7       BNE  $C58E         Jump: test next matrix line

****************************          Evaluate the keyboard result

C5D7:  06 D3       ASL  * $D3         Eliminate the set bit 7 in the shift
C5D9:  46 D3       LSR  * $D3         pattern flag (marker port A* test)
C5DB:  68          PLA                Clear line test value from stack
C5DC:  A5 D4       LDA  * $D4         Code fro pressed key in acc
C5DE:  6C 3A 03    JMP  ($033A)       Vector - keyboard read ($C5E1)

****************************          Routine: evaluate keybaord

C5E1:  C9 57       CMP  # $57         Was it the "No Scroll" key?
C5E3:  D0 13       BNE  $C5F8         No, then skip
C5E5:  24 F7       BIT  * $F7         Z-P pause flag bit 6: 1=disable
C5E7:  70 5A       BVS  $C643         If pause not allowed then RTS
C5E9:  AD 25 0A    LDA  $0A25         Load acc with last shift pattern
C5EC:  D0 55       BNE  $C643         Not 0, then exit via RTS
C5EE:  A9 0D       LDA  # $0D         Invert bits 0,1, and 3 of the
C5F0:  4D 21 0A    EOR  $0A21         Z-P pause pointer and put in the
C5F3:  8D 21 0A    STA  $0A21         Zero-page pause pointer
C5F6:  50 30       BVC  $C628         Keyboard repeat routine
C5F8:  A5 D3       LDA  * $D3         Get current shift pattern in acc
C5FA:  F0 55       BEQ  $C651         No shift pattern, evaluate normal
C5FC:  C9 10       CMP  # $10         Was the 40 character set chosen
C5FE:  F0 44       BEQ  $C644         Yes, then to 40 evaluation
C600:  C9 08       CMP  # $08         Was ALT keypress indicated?
C602:  F0 42       BEQ  $C646         Yes, then to ALT evaluation
C604:  29 07       AND  # $07         Mask bits 3-7 from shift pattern
C606:  C9 03       CMP  # $03         Was C=-SHIFT switch selected?
C608:  D0 25       BNE  $C62F         No, re-evaluate shift pattern

****************************          C=/Shift character set switch

C60A:  A5 F7       LDA  * $F7         Check flag for C= shift switch
C60C:  30 43       BMI  $C651         Switch prohibit, to repeat routine
C60E:  AD 25 0A    LDA  $0A25         Get last-saved shift pattern
```

```
C611:  D0 3E      BNE  $C651     Not zero, then to repeat routine
C613:  24 D7      BIT  * $D7     Check for 40/80 column screen
C615:  10 09      BPL  $C620     Positive = 40 column screen
C617:  A5 F1      LDA  * $F1     Color code for char output in acc
C619:  49 80      EOR  # $80     Invert bit 7 of the color code
C61B:  85 F1      STA  * $F1     Store color code for char code
C61D:  4C 28 C6   JMP  $C628     Jump over VIC character switch
C620:  AD 2C 0A   LDA  $0A2C     System pointer for text/screen
C623:  49 02      EOR  # $02     Get base and invert bit 2 of this
C625:  8D 2C 0A   STA  $0A2C     Pointer
C628:  A9 08      LDA  # $08     Initialize the system pointer with
C62A:  8D 25 0A   STA  $0A25     8 for the last shift pattern
C62D:  D0 22      BNE  $C651     Jump to repeat routine

******************************   Load and evaluate decoder table
                                 corresponding to the shift pattern

C62F:  0A         ASL  A         Multiply shift pattern for disp *2
C630:  C9 08      CMP  # $08     If shift pattern for shift or C=
C632:  90 12      BCC  $C646     Found, then load decoder table
C634:  A9 06      LDA  # $06     Default value CTRL pattern, acc
C636:  A6 D4      LDX  * $D4     Check offset of the decoder table
C638:  E0 0D      CPX  # $0D     If it was the 13th key (S-key)
C63A:  D0 0A      BNE  $C646     Then set the pause flag, else skip
C63C:  24 F7      BIT  * $F7     Check if pause/Ctrl-s is allowed
C63E:  70 06      BVS  $C646     Not allow, evaluate decod. table
C640:  8E 21 0A   STX  $0A21     Get pause flag with key value 13
C643:  60         RTS            Return from the subroutine

******************************   Set the start address of the
                                 decoder table corresponding to
                                 the shift pattern

C644:  A9 0A      LDA  # $0A     Set default value to table 5a
C646:  AA         TAX            # of the decoder table in X-reg
C647:  BD 3E 03   LDA  $033E,X   Copy address low of decoder
C64A:  85 CC      STA  * $CC     table in zero-page memory
C64C:  BD 3F 03   LDA  $033F,X   Copy address high of decoder
C64F:  85 CD      STA  * $CD     table in zero-page memory
```

```
******************************   Routine REPEAT
                                 Repeat the keybaord logic

C651:  A4 D4      LDY  * $D4     Displ. to table start in Y-reg
C653:  B1 CC      LDA  ($CC),Y   Load acc with char code from
C655:  AA         TAX            Table and store char in X-reg
C656:  C4 D5      CPY  * $D5     Compare with pointer for current
C658:  F0 07      BEQ  $C661     key.  If equal, to repeat check
C65A:  A0 10      LDY  # $10     Counter for key repeat delay
C65C:  8C 24 0A   STY  $0A24     Initialize with $10
C65F:  D0 36      BNE  $C697     Jump to keypress evaluation
C661:  29 7F      AND  # $7F     Mask out bit 7, not a RVS char
C663:  2C 22 0A   BIT  $0A22     Check pointer for key repeat
C666:  30 16      BMI  $C67E      Allow all keys ($80), skip
C668:  70 5A      BVS  $C6C4      Now key allowed ($40), skip
C66A:  C9 7F      CMP  # $7F     Check if "character invalid"
C66C:  F0 29      BEQ  $C697     Yes, the default read and RTS
C66E:  C9 14      CMP  # $14     Was it the DEL key>
C670:  F0 0C      BEQ  $C67E     Yes, then repeat evaluation
C672:  C9 20      CMP  # $20     Was it the space bar?
C674:  F0 08      BEQ  $C67E     Yes, then repeat evaluation
C676:  C9 1D      CMP  # $1D     Was it the <CRSR-right> key?
C678:  F0 04      BEQ  $C67E     Yes, then repeat evaluation
C67A:  C9 11      CMP  # $11     Was it the <CRSR-down> key?
C67C:  D0 46      BNE  $C6C4     No, skip repeat evaluation

******************************   Key repeat evaluation

C67E:  AC 24 0A   LDY  $0A24     Get counter for repeat delay
C681:  F0 05      BEQ  $C688      Counter=0, then skip
C683:  CE 24 0A   DEC  $0A24     Repeat delay counter -1
C686:  D0 3C      BNE  $C6C4     Not zero, default read and RTS
C688:  CE 23 0A   DEC  $0A23     Count speed for repeat -1
C68B:  D0 37      BNE  $C6C4     Not zero, default read and RTS
C68D:  A0 04      LDY  # $04     Count speed for key repeat
C68F:  8C 23 0A   STY  $0A23     Reinitialize with $04
C692:  A4 D0      LDY  * $D0     Offset of  key buffer queue in Y
C694:  88         DEY            If more than 1 character in buffer
C695:  10 2D      BPL  $C6C4     Then default read and RTS
```

```
*****************************   Entry: No key pressed

C697:  4E 25 0A   LSR  $0A25     Divide last shift pattern by 2
C69A:  A4 D4      LDY  * $D4     Copy Displ to decoder table start
C69C:  84 D5      STY  * $D5     In pointer for current key
C69E:  E0 FF      CPX  # $FF     Was it code for "no character"?
C6A0:  F0 22      BEQ  $C6C4     Yes, then default read and RTS
C6A2:  A9 00      LDA  # $00     Reset the pause/Ctrl-S pointer
C6A4:  8D 21 0A   STA  $0A21     for  valid character
C6A7:  8A         TXA            Copy character code in acc
C6A8:  A6 D3      LDX  * $D3     Get current shift pattern in X-reg
C6AA:  4C C6 FC   JMP  $FCC6     Back to kernal routine: KEY

*****************************   Evaluate and store keypress

C6AD:  A2 09      LDX  # $09     Loop counter - 10 function keys
C6AF:  DD DD C6   CMP  $C6DD,X   Compare acc with key code table
C6B2:  F0 16      BEQ  $C6CA     Function key found, evaluate
C6B4:  CA         DEX            Decrement loop counter by 1
C6B5:  10 F8      BPL  $C6AF     Loop until all comparisons done
C6B7:  A6 D0      LDX  * $D0     Index: Keyboard buffer queue
C6B9:  EC 20 0A   CPX  $0A20     Compare with maximum size
C6BC:  B0 06      BCS  $C6C4     Max size reached, then skip
C6BE:  9D 4A 03   STA  $034A,X   Place char in keyboard buffer
C6C1:  E8         INX            Increment keyboard buff. queue
C6C2:  86 D0      STX  * $D0     Index by 1 character
C6C4:  A9 7F      LDA  # $7F     Check keyboard matrix
C6C6:  8D 00 DC   STA  $DC00     For default
C6C9:  60         RTS            Return from the subroutine

*****************************   Prepare keyboard buffer for
                                 KEY

C6CA:  BD 00 10   LDA  $1000,X   Get length from KEY X
C6CD:  85 D1      STA  * $D1     And in KEY character counter
C6CF:  A9 00      LDA  # $00     The position of the KEY in the
C6D1:  CA         DEX            Entire table is detremined
C6D2:  30 06      BMI  $C6DA     When all lengths added, end
C6D4:  18         CLC            Else clear carry for addition
C6D5:  7D 00 10   ADC  $1000,X   Add length of KEY X
```

```
C6D8:  90 F7      BCC  $C6D1     If no overflow, then continue
C6DA:  85 D2      STA  * $D2     Else store pointer
C6DC:  60         RTS            Return from subroutine

****************************     Key codes of 10 function keys

C6DD:  85 89                     F1  F2
C6DF:  86 8A                     F3  F4
C6E1:  87 8B                     F5  F6
C6E3:  88 8C                     F7  F8
C6E5:  83                        F9  (Shift-Run)
C6E6:  84                        F10 (Help-key)

****************************     Flash VIC cursor

C6E7:  24 D7      BIT  * $D7     Test for 40/80 column
C6E9:  30 41      BMI  $C72C     If 80 column then end
C6EB:  AD 27 0A   LDA  $0A27     Get VIC cursor mode
C6EE:  D0 3C      BNE  $C72C     Is turned off then end
C6F0:  CE 28 0A   DEC  $0A28     Else decrement the flash counter
C6F3:  D0 37      BNE  $C72C     If not zero, then end
C6F5:  AD 26 0A   LDA  $0A26     Get VIC cursor
C6F8:  29 C0      AND  # $C0     Mask out bits 0-5
C6FA:  C9 C0      CMP  # $C0     Cursor steady or turned off?
C6FC:  F0 2E      BEQ  $C72C     If so then end
C6FE:  A9 14      LDA  # $14     Set the VIC cursor flash counter
C700:  8D 28 0A   STA  $0A28     To $14=20
C703:  A4 EC      LDY  * $EC     Get current cursor columnY-Reg
C705:  AE 2A 0A   LDX  $0A2A     Get color at cursor pos. for flash
C708:  B1 E0      LDA  ($E0),Y   Get character at current column
C70A:  2C 26 0A   BIT  $0A26     Test VIC cursor mode
C70D:  30 10      BMI  $C71F     Character normal again
C70F:  8D 29 0A   STA  $0A29     Char at cursor pos before flash
C712:  20 7C C1   JSR  $C17C     Set color RAM address
C715:  B1 E2      LDA  ($E2),Y   Get color at cursor position
C717:  8D 2A 0A   STA  $0A2A     Save as color before flash
C71A:  A6 F1      LDX  * $F1     Color code-char output in X-Reg
C71C:  AD 29 0A   LDA  $0A29     Char at cursor pos before flash
C71F:  49 80      EOR  # $80     Invert the negative bit
C721:  20 40 CC   JSR  $CC40     Save character and color
```

```
C724:  AD 26 0A   LDA  $0A26      Get VIC cursor mode
C727:  49 80      EOR  # $80      Negate the flash condition
C729:  8D 26 0A   STA  $0A26      And save again
C72C:  60         RTS             Return from the subroutine

*****************************     BSOUT entry for screen output

C72D:  85 EF      STA  * $EF      Save character to print in z-page
C72F:  48         PHA             Save acc contents on stack
C730:  8A         TXA             Save X-reg contents on stack
C731:  48         PHA             Via acc
C732:  98         TYA             Save Y-reg contents on stack
C733:  48         PHA             Via acc
C734:  AD 21 0A   LDA  $0A21      Check contents of z-p pause flag
C737:  D0 FB      BNE  $C734      Wait until flag value is 0
C739:  85 D6      STA  * $D6      Clear input/get flag via keyboard
C73B:  A9 C3      LDA  # $C3      High byte of continuation on
C73D:  48         PHA             stack, to jump to rouitine via
C73E:  A9 0B      LDA  # $0B      RTS now the byte of the
C740:  48         PHA             continuation on the stack as well
C741:  A4 EC      LDY  * $EC      Get current cursor columnY-Reg
C743:  A5 EF      LDA  * $EF      Get char to print - temp storage
C745:  C9 0D      CMP  # $0D      Is it a carriage return <Cr> ?
C747:  F0 26      BEQ  $C76F      Yes, then output <CR>
C749:  C9 8D      CMP  # $8D      Is it a shift-CR?
C74B:  F0 22      BEQ  $C76F      Yes, then output <shift/CR>
C74D:  A6 F0      LDX  * $F0      Get value of previous character
C74F:  E0 1B      CPX  # $1B      Was it <ESC>, then handle char
C751:  D0 03      BNE  $C756      as <ESC> sequence, else to
C753:  4C BE C9   JMP  $C9BE      $C756 - evaluate ESC sequences
C756:  AA         TAX             Character to output to X-Reg
C757:  10 03      BPL  $C75C      Is it a character from 0 - 127 ?
C759:  4C 02 C8   JMP  $C802      No, evalute: exteneded ASCII
C75C:  C9 20      CMP  # $20      Is characetr to output < Blank ?
C75E:  90 56      BCC  $C7B6      Yes, then evaluate control codes
C760:  C9 60      CMP  # $60      Is it a letter?
C762:  90 03      BCC  $C767      Yes, then output letter
C764:  29 DF      AND  # $DF      Mask out bit 5
C766:  2C         .Byte  $2C      Skip to $C769
```

```
******************************    Output letter

C767:  29 3F      AND  # $3F      Mask out bits 6/7 of the char
C769:  20 FF C2   JSR  $C2FF      Test for quote
C76C:  4C 22 C3   JMP  $C322      Output character

******************************    <Carriage Return> - New line

C76F:  20 C3 CB   JSR  $CBC3      Search end of input line
C772:  E8         INX             Clear the line overflow bit
C773:  20 85 CB   JSR  $CB85      Of the following-line
C776:  A4 E6      LDY  * $E6      Load left window-border Y-reg
C778:  84 EC      STY  * $EC      Store the current cursor position
C77A:  20 63 C3   JSR  $C363      Execute linefeed

******************************    Reset Quote/Insert/RVS

C77D:  A5 F1      LDA  * $F1      Color code for char output in acc
C77F:  29 CF      AND  # $CF      Reverse and flash off for VDC
C781:  85 F1      STA  * $F1      Store color code for char output
C783:  A9 00      LDA  # $00      Load acc with zero for off
C785:  85 F5      STA  * $F5      And clear the bits: insert mode
C787:  85 F3      STA  * $F3      RVS flag
C789:  85 F4      STA  * $F4      Quote-mode flag
C78B:  60         RTS             Return from the subroutine

******************************    Control codes

C78C:  02         .Byte  $02      2=underline on
C78D:  07         .Byte  $07      7=bell
C78E:  09         .Byte  $09      9=tab
C78F:  0A         .Byte  $0A      A=linefeed
C790:  0B         .Byte  $0B      B=lock <Shift>/<Commodore>
C791:  0C         .Byte  $0C      C=unlock <Sh>/<C=>
C792:  0E         .Byte  $0E      E=lower case
C793:  0F         .Byte  $0f      F=flash on
C794:  11         .Byte  $11      11=cursor up
C795:  12         .Byte  $12      12=reverse on
C796:  13         .Byte  $13      13=home
C797:  14         .Byte  $14      14=delete
```

```
C798:  18          .Byte  $18      18=set/clear tab
C799:  1D          .Byte  $1d      1D=Cursor Right

*****************************      Addresses of the routines which
                                   execute control codes (-1)
                                   Accessed via RTS.

C79A:  C6 C8       $C8C6           Underline on
C79C:  8D C9       $C98D           Tab
C79E:  4E C9       $C94E           Bell
C7A0:  B0 C9       $C9B0           Linefeed
C7A2:  A5 C8       $C8A5           Disable <Sh>/<C=>
C7A4:  AB C8       $C8AB           Enable <Sh>/<C=>
C7A6:  7F C8       $C87F           Lower case
C7A8:  D4 C8       $C8D4           Flash on
C7AA:  59 C8       $C859           Cursor up
C7AC:  C1 C8       $C8C1           Reverse on
C7AE:  B2 C8       $C8B2           Home
C7B0:  1A C9       $C91A           Delete
C7B2:  60 C9       $C960           Set/clear tab
C7B4:  53 C8       $C853           Cursor right

*****************************      Execute control code

C7B6:  6C 34 03    JMP   ($0334)   Vector character output with Ctrl
C7B9:  C9 1B       CMP   # $1B     Is character <ESC>?
C7BB:  F0 38       BEQ   $C7F5     Yes, then end
C7BD:  A6 F5       LDX   * $F5     Insert mode set?
C7BF:  D0 08       BNE   $C7C9     Yes, then output char in reverse
C7C1:  C9 14       CMP   # $14     Is the character <Delete>?
C7C3:  F0 0B       BEQ   $C7D0     Then execute
C7C5:  A6 F4       LDX   * $F4     Is the quote-mode flag set?
C7C7:  F0 07       BEQ   $C7D0     If so, then reverse character
C7C9:  A2 00       LDX   # $00     Clear the last-printed character
C7CB:  86 EF       STX   * $EF     In the zero-page
C7CD:  4C 26 C3    JMP   $C326     And output character in reverse
```

```
C7D0:  A2 0D      LDX  # $0D      X is the counter for ctrl codes
C7D2:  DD 8C C7   CMP  $C78C,X    Compare with the table
C7D5:  F0 1F      BEQ  $C7F6      Found? Then jump to execution
C7D7:  CA         DEX             Else decrement the counter and
C7D8:  10 F8      BPL  $C7D2      Compare with next value
C7DA:  A2 0F      LDX  # $0F      Compare with the 16 possible
C7DC:  DD 4C CE   CMP  $CE4C,X    Codes for changing the color
C7DF:  F0 04      BEQ  $C7E5      Jump if found
C7E1:  CA         DEX             Else decrement counter and
C7E2:  10 F8      BPL  $C7DC      Compare with next value
C7E4:  60         RTS             Returns from the subroutine

******************************    Set color - 40-column

C7E5:  24 D7      BIT  * $D7      Test 40/80-column mode
C7E7:  30 03      BMI  $C7EC      Jump if 80-column mode
C7E9:  86 F1      STX  * $F1      Store color code for char outout
C7EB:  60         RTS             Return from subroutine

******************************    Set color - 80-column mode

C7EC:  A5 F1      LDA  * $F1      Color code for char output in acc
C7EE:  29 F0      AND  # $F0      Mask out lower nibble (bits 0-3)
C7F0:  1D 5C CE   ORA  $CE5C,X    OR with color code table
C7F3:  85 F1      STA  * $F1      Store color code for char output
C7F5:  60         RTS             Return from subroutine

******************************    Execute control codes

C7F6:  8A         TXA             Pointer to acc and then
C7F7:  0A         ASL             Multiply by two because a
C7F8:  AA         TAX             16-bit value is being fetched
C7F9:  BD 9B C7   LDA  $C79B,X    Get low byte of the start address
C7FC:  48         PHA             In acc and get
C7FD:  BD 9A C7   LDA  $C79A,X    High byte of the start address
C800:  48         PHA             In acc. Accessed via
C801:  60         RTS             RTS
```

```
******************************      Analyze extended ASCII

C802:  6C 36 03    JMP   ($0336)    Vector char output with shift
C805:  29 7F       AND   # $7F      Mask out bit 7, not shifted
C807:  C9 20       CMP   # $20      Compare with <space>
C809:  90 09       BCC   $C814      Less than 32
C80B:  C9 7F       CMP   # $7F      Is it ASCII code 127?
C80D:  D0 02       BNE   $C811      If not then jump
C80F:  A9 5E       LDA   # $5E      ASCII code for up-arrow
C811:  4C 20 C3    JMP   $C320      And output
C814:  A6 F4       LDX   * $F4      Get quote-mode flag
C816:  F0 05       BEQ   $C81D      Jump if not set
C818:  09 40       ORA   # $40      Else set bit 6
C81A:  4C 26 C3    JMP   $C326      Output as reverse character
C81D:  C9 14       CMP   # $14      Is the character <INSERT>?
C81F:  D0 03       BNE   $C824      Jump if not <INSERT>
C821:  4C E3 C8    JMP   $C8E3      Else execute <INSERT>
C824:  A6 F5       LDX   * $F5      Get insert-mode flag
C826:  D0 F0       BNE   $C818      If set, then as with quote
C828:  C9 11       CMP   # $11      Compare to cursor up
C82A:  F0 3B       BEQ   $C867      Jump if cursor-up
C82C:  C9 1D       CMP   # $1D      Cursor-left?
C82E:  F0 45       BEQ   $C875      If yes, then execute
C830:  C9 0E       CMP   # $0E      Compare if upper case
C832:  F0 5E       BEQ   $C892      Jump to execution
C834:  C9 12       CMP   # $12      Reverse off?
C836:  D0 03       BNE   $C83B      No, then skip
C838:  4C BF C8    JMP   $C8BF      Else clear RVS mode
C83B:  C9 02       CMP   # $02      Underline on?
C83D:  D0 03       BNE   $C842      If not then jump
C83F:  4C CE C8    JMP   $C8CE      Else set underline mode
C842:  C9 0F       CMP   # $0F      Flash mode off?
C844:  D0 03       BNE   $C849      Skip if not
C846:  4C DC C8    JMP   $C8DC      Else clear flash mode
C849:  C9 13       CMP   # $13      Is it <CLR/HOME>?
C84B:  D0 03       BNE   $C850      Skip if not
C84D:  4C 42 C1    JMP   $C142      Else clear window
C850:  09 80       ORA   # $80      Clear bit 7 -- it must be a color
C852:  D0 86       BNE   $C7DA      And jump to evaluation
```

```
******************************      Cursor right in window

C854:  20 ED CB   JSR  $CBED      Cursor one position to the right
C857:  B0 04      BCS  $C85D      New line begun
C859:  60         RTS             Return from subroutine

******************************      Cursor down

C85A:  20 63 C3   JSR  $C363      Perform linefeed
C85D:  20 74 CB   JSR  $CB74      Test line-overflow bit
C860:  B0 03      BCS  $C865      Line too long
C862:  38         SEC             Set carry and rotate
C863:  66 E8      ROR  # $E8      It in the start input line
C865:  18         CLC             Clear carry for OK
C866:  60         RTS             Return from the subroutine

******************************      Cursor up

C867:  A6 E5      LDX  * $E5      Load upper window-border in X
C869:  E4 EB      CPX  * $EB      Compare with current cursor line
C86B:  B0 F9      BCS  $C866      Is less than or equal
C86D:  20 5D C8   JSR  $C85D      Set line status
C870:  C6 EB      DEC  * $EB      Dec. current cursor line by 1
C872:  4C 5C C1   JMP  $C15C      Determine start addr current line

******************************      Cursor left in window

C875:  20 00 CC   JSR  $CC00      Cursor left
C878:  B0 EC      BCS  $C866      Cursor not moved
C87A:  D0 E9      BNE  $C865      Cursor moved, no new line
C87C:  E6 EB      INC  * $EB      Incr. current cursor line by 1
C87E:  D0 ED      BNE  $C86D      Unconditional jump

******************************      2nd character set

C880:  24 D7      BIT  * $D7      Test 40/80-column mode
C882:  30 07      BMI  $C88B      Jump if 80-column mode
C884:  AD 2C 0A   LDA  $0A2C      Get CHARROM base address
C887:  09 02      ORA  # $02      Set bits 0 and 1
C889:  D0 10      BNE  $C89B      Unconditional jump
```

```
C88B:  A5 F1      LDA  * $F1      Color code for char output in acc
C88D:  09 80      ORA  # $80      Select alternate character set
C88F:  85 F1      STA  * $F1      Store color code for char output
C891:  60         RTS             Return from subroutine

******************************    <Shift> <Commodore>

C892:  24 D7      BIT  * $D7      Test 40/80-column mode
C894:  30 09      BMI  $C89F      Jump if 80-column mode
C896:  AD 2C 0A   LDA  $0A2C      Get base address CHARROM
C899:  29 FD      AND  # $FD      Clear bits 0 and 1

C89B:  8D 2C 0A   STA  $0A2C      Store as new base address
C89E:  60         RTS             Return from the subroutine

******************************    <Shift> <Commodore>
                                  80-column

C89F:  A5 F1      LDA  * $F1      Color code for char output in acc
C8A1:  29 7F      AND  # $7F      Clear bit 7, first character set
C8A3:  85 F1      STA  * $F1      Set color code for char output
C8A5:  60         RTS             Return from subroutine

******************************    <Shift> <Commodore>
                                  enable/disable

C8A6:  A9 80      LDA  # $80      Set bit 7 to disable and OR
C8A8:  05 F7      ORA  * $F7      With flag register
C8AA:  30 04      BMI  $C8B0      Unconditional jump
C8AC:  A9 7F      LDA  # $7F      Clear bit 7 in order to
C8AE:  25 F7      AND  * $F7      Enable
C8B0:  85 F7      STA  * $F7      And save
C8B2:  60         RTS             Return from subroutine

******************************    Test for <Home>-<Home>
                                  combination

C8B3:  A5 F0      LDA  * $F0      Get last-printed character
C8B5:  C9 13      CMP  # $13      Was it HOME?
C8B7:  D0 03      BNE  $C8BC      If not, then end of the routine
```

```
C8B9:  20 24 CA   JSR  $CA24      Else cancel window
C8BC:  4C 50 C1   JMP  $C150      Jump to cursor home

******************************    Set/clear reverse mode

C8BF:  A9 00      LDA  # $00      Load acc with zero, clear RVS
C8C1:  2C         .Byte  $2C      Skip to $C8C4
C8C2:  A9 80      LDA  # $80      Set bit, turn RVS mode on
C8C4:  85 F3      STA  * $F3      And store flag
C8C6:  60         RTS             Return from subroutine

******************************    Turn underline on

C8C7:  A5 F1      LDA  * $F1      Color code for char output in acc
C8C9:  09 20      ORA  # $20      Set bit 6 for underline on
C8CB:  85 F1      STA  * $F1      store color code for char output
C8CD:  60         RTS             Return from subroutine

******************************    Turn underline off

C8CE:  A5 F1      LDA  * $F1      Color code for char output in acc
C8D0:  29 DF      AND  # $DF      Clear bit 5, underline off
C8D2:  85 F1      STA  * $F1      Store color code for char output
C8D4:  60         RTS             Return from subroutine

******************************    Set flash mode

C8D5:  A5 F1      LDA  * $F1      Color code for char output in acc
C8D7:  09 10      ORA  # $10      Set bit 4 for flash on
C8D9:  85 F1      STA  * $F1      Store color code for char output
C8DB:  60         RTS             Return from the subroutine

******************************    Turn flash mode off

C8DC:  A5 F1      LDA  * $F1      Color code for char output in acc
C8DE:  29 EF      AND  # $EF      Clear bit 4, no flash
C8E0:  85 F1      STA  * $F1      Store color code for char output
C8E2:  60         RTS             Return from the subroutine
```

```
******************************   Perform insert

C8E3:  20 1E CC   JSR  $CC1E     Copy cursor coordinates
C8E6:  20 C3 CB   JSR  $CBC3     Search for end of input line
C8E9:  E4 DF      CPX  * $DF     Compare line with cursor line
C8EB:  D0 02      BNE  $C8EF     If changed then jump
C8ED:  C4 DE      CPY  * $DE     Compare clmn with current clmn
C8EF:  90 21      BCC  $C912     Smaller
C8F1:  20 3E C3   JSR  $C33E     Cursor at line end
C8F4:  B0 22      BCS  $C918     Cannot be scrolled
C8F6:  20 00 CC   JSR  $CC00     Cursor one to the left
C8F9:  20 58 CB   JSR  $CB58     Get char and color cursor pos
C8FC:  20 ED CB   JSR  $CBED     Cursor one to the right again
C8FF:  20 32 CC   JSR  $CC32     Output character
C902:  20 00 CC   JSR  $CC00     Cursor one position to the left
C905:  A6 EB      LDX  * $EB     Get current cursor line in X-reg
C907:  E4 DF      CPX  * $DF     Compare w/ starting cursor line
C909:  D0 EB      BNE  $C8F6     Copy next character
C90B:  C4 DE      CPY  * $DE     Compare col. with starting col.
C90D:  D0 E7      BNE  $C8F6     If not reached, continue
C90F:  20 27 CC   JSR  $CC27     Space at current cursor position
C912:  E6 F5      INC  * $F5     Increment counter for insert
C914:  D0 02      BNE  $C918     If not zero then jump
C916:  C6 F5      DEC  * $F5     Else reset insert again
C918:  4C 32 C9   JMP  $C932     Reset old cursor position

******************************   Delete character to left of cursor

C91B:  20 75 C8   JSR  $C875     Cursor left with bit manipulation
C91E:  20 1E CC   JSR  $CC1E     Copy the cursor coordinate
C921:  B0 0F      BCS  $C932     Cursor left not possible
C923:  C4 E7      CPY  * $E7     Compare  right window-border
C925:  90 16      BCC  $C93D     Border not yet reached
C927:  A6 EB      LDX  * $EB     Get current cursor line in X
C929:  E8         INX            Increment the line by 1
C92A:  20 76 CB   JSR  $CB76     Test overflow bit
C92D:  B0 0E      BCS  $C93D     There is a following-line
C92F:  20 27 CC   JSR  $CC27     Else <space> at current position
```

```
****************************      Set old cursor address again

C932:  A5 DE      LDA  * $DE      Get column
C934:  85 EC      STA  * $EC      Store the current cursor column
C936:  A5 DF      LDA  * $DF      Get line
C938:  85 EB      STA  * $EB      Write current cursor line
C93A:  4C 5C C1   JMP  $C15C      Determine start address of line

****************************      Delete character under cursor

C93D:  20 ED CB   JSR  $CBED      Cursor one to the right
C940:  20 58 CB   JSR  $CB58      Get character and color at cursor
C943:  20 00 CC   JSR  $CC00      Cursor one to the left
C946:  20 32 CC   JSR  $CC32      Character at cursor position
C949:  20 ED CB   JSR  $CBED      Cursor back to the right
C94C:  4C 23 C9   JMP  $C923      Move line to cursor

****************************      Tab

C94F:  A4 EC      LDY  * $EC      Get current cursor col. in Y-reg
C951:  C8         INY             Increment the column pointer
C952:  C4 E7      CPY  * $E7      Compare  right window-border
C954:  B0 06      BCS  $C95C      No more tabs possible
C956:  20 6C C9   JSR  $C96C      Get next tab position
C959:  F0 F6      BEQ  $C951      Cursor is at tab pos, again
C95B:  2C         .Byte  $2C      Skip to $C95E
C95C:  A4 E7      LDY  * $E7      Right window-border to Y
C95E:  84 EC      STY  * $EC      Store the current cursor column
C960:  60         RTS             Return from subroutine

****************************      Set/clear tab

C961:  A4 EC      LDY  * $EC      Get current cursor col. in Y-reg
C963:  20 6C C9   JSR  $C96C      Get tab byte
C966:  45 DA      EOR  * $DA      Reverse the tab bit
C968:  9D 54 03   STA  $0354,X    And store again
C96B:  60         RTS             Return from subroutine
```

```
*****************************   Determine tab position

C96C:  98          TYA           Column to accumulator
C96D:  29 07       AND  # $07    Mask out bits 4-7=A MOD 7
C96F:  AA          TAX           And to X-register as pointer
C970:  BD 6C CE    LDA  $CE6C,X  Get power of 2
C973:  85 DA       STA  * $DA    And store in $DA
C975:  98          TYA           Column back to acc
C976:  4A          LSR  A        Shift acc right three times
C977:  4A          LSR  A        Amounting to INT(A/8)
C978:  4A          LSR  A
C979:  AA          TAX           Back into X-reg as pointer
C97A:  BD 54 03    LDA  $0354,X  Get tab byte
C97D:  24 DA       BIT  * $DA    Test if 8th tab is set
C97F:  60          RTS           Return from subroutine

*****************************   Clear the tabs (or reset)

C980:  A9 00       LDA  # $00    Load acc with zero to clear
C982:  2C          .Byte  $2C    Skip to $C985
C983:  A9 80       LDA  # $80    Every 8th position is a tab
C985:  A2 09       LDX  # $09    All 10 tab bytes
C987:  9D 54 03    STA  $0354,X  Are written with the value
C98A:  CA          DEX           Decrement the counter and
C98B:  10 FA       BPL  $C987    Jump if not yet done
C98D:  60          RTS           Return from subroutine

*****************************   CHR$(7) - Bell

C98E:  24 F9       BIT  * $F9    Test beep flag
C990:  30 FB       BMI  $C98D    No beep
C992:  A9 15       LDA  # $15    Set SID volume to
C994:  8D 18 D4    STA  $D418    15 (maximum)
C997:  A0 09       LDY  # $09    Attack/decay constant
C999:  A2 00       LDX  # $00    Sustain/release constant
C99B:  8C 05 D4    STY  $D405    Place in the corresponding reg.
C99E:  8E 06 D4    STX  $D406    (for voice 1)
C9A1:  A9 30       LDA  # $30    Define high byte of frequency
C9A3:  8D 01 D4    STA  $D401    For voice 1
C9A6:  A9 20       LDA  # $20    Select sawtooth
```

```
C9A8:  8D 04 D4   STA  $D404      And write to SID
C9AB:  A9 21      LDA  # $21      The tone is started
C9AD:  8D 04 D4   STA  $D404      By setting bit 0
C9B0:  60         RTS             Return from subroutine

*****************************     <LF> - cursor column remains

C9B1:  A5 EC      LDA  * $EC      Get current cursor column in acc
C9B3:  48         PHA             Save current column in acc
C9B4:  20 C3 CB   JSR  $CBC3      Search for end of line
C9B7:  20 63 C3   JSR  $C363      Perform linefeed
C9BA:  68         PLA             Get current column back
C9BB:  85 EC      STA  * $EC      Store the current cursor line
C9BD:  60         RTS             Return from subroutine

*****************************     Execute ESC sequences

C9BE:  6C 38 03   JMP  ($0338)    Vector char output with ESC
C9C1:  C9 1B      CMP  # $1B      Is character <ESC>?
C9C3:  D0 05      BNE  $C9CA      Jump if another character
C9C5:  46 EF      LSR  * $EF      Current character by 2
C9C7:  4C 7D C7   JMP  $C77D      Turn off all special functions
C9CA:  29 7F      AND  # $7F      Mask out bit 7, not reverse char
C9CC:  38         SEC             Set carry for subtraction
C9CD:  E9 40      SBC  # $40      Subtract 64 from ASCII value
C9CF:  C9 1B      CMP  # $1B      Compare with 27
C9D1:  B0 0A      BCS  $C9DD      Return if character greater than Z
C9D3:  0A         ASL  A          Acc * 2 -- 16-bit value fetched
C9D4:  AA         TAX             And then to X as pointer
C9D5:  BD DF C9   LDA  $C9DF,X    Get high byte of exec. routine
C9D8:  48         PHA             Save on stack
C9D9:  BD DE C9   LDA  $C9DE,X    Get low byte of routine on stack
C9DC:  48         PHA             Jump to routine via
C9DD:  60         RTS             RTS. Address is on the stack

*****************************     Addresses of the escape routine

C9DE:  9E CA      $CA9E           <ESC> @ - Clear cursor to end
C9E0:  EC CA      $CAEC           <ESC> A - Auto-insert on
C9E2:  15 CA      $CA15           <ESC> B - Set bottom - screen
```

```
C9E4:  E9 CA      $CAE9          <ESC> C - Auto-insert off
C9E6:  51 CA      $CA51          <ESC> D - Delete current line
C9E8:  0A CB      $CB0A          <ESC> E - Cursor flash off
C9EA:  20 CB      $CB20          <ESC> F - Cursor flash on
C9EC:  36 CB      $CB36          <ESC> G - Enable beep
C9EE:  39 CB      $CB39          <ESC> H - Disable beep
C9F0:  3C CA      $CA3C          <ESC> I - Insert line
C9F2:  B0 CB      $CBB0          <ESC> J - Cursor to start of line
C9F4:  51 CB      $CB51          <ESC> K - Cursor to end of line
C9F6:  E1 CA      $CAE1          <ESC> L - Enable scrolling
C9F8:  E4 CA      $CAE4          <ESC> M - Disable scrolling
C9FA:  47 CB      $CB47          <ESC> N - Reverse off (80-col)
C9FC:  7C C7      $C77C          <ESC> O - Inst, quote, RVS off
C9FE:  8A CA      $CA8A          <ESC> P - Clear to line start
CA00:  75 CA      $CA75          <ESC> Q - Clear to line end
CA02:  3E CB      $CB3E          <ESC> R - Reverse screen (80)
CA04:  F1 CA      $CAF1          <ESC> S - Block cursor (80)
CA06:  13 CA      $CA13          <ESC> T - Set top of screen
CA08:  FD CA      $CAFD          <ESC> U - Underline cursor 80
CA0A:  BB CA      $CABB          <ESC> V - Scroll up
CA0C:  C9 CA      $CAC9          <ESC> W - Scroll down
CA0E:  2B CD      $CD2B          <ESC> X - Switch 40/80-col.
CA10:  82 C9      $C982          <ESC> Y - Reset tabs to normal
CA12:  7F C9      $C97F          <ESC> Z - Clear all tabs

******************************   Definition of window borders

CA14:  18         CLC            Cursor position is top/left
CA15:  24         .Byte  $24     Skip to $CA17
CA16:  38         SEC            Cursor position is right/bottom
CA17:  A6 EC      LDX  * $EC     Get current cursor col in X-reg
CA19:  A5 EB      LDA  * $EB     Get current cursor line in acc
CA1B:  90 11      BCC  $CA2E     If carry cleared: left/top!
CA1D:  85 E4      STA  * $E4     Define bottom of screen window
CA1F:  86 E7      STX  * $E7     As well as right border
CA21:  4C 32 CA   JMP  $CA32     Execute remainder of routine
```

```
******************************   Define screen as window

CA24:  A5 ED      LDA  * $ED     Get max number of lines in A
CA26:  A6 EE      LDX  * $EE     Get max number of cols in X
CA28:  20 1D CA   JSR  $CA1D     Define as right/bottom
CA2B:  A9 00      LDA  # $00     Left/top with 0/0
CA2D:  AA         TAX
CA2E:  85 E5      STA  * $E5     And define as left
CA30:  86 E6      STX  * $E6     And top border
CA32:  A9 00      LDA  # $00     Load acc with zero and
CA34:  A2 04      LDX  # $04     The X-register with 4 in order to
CA36:  9D 5D 03   STA  $035D,X   Clear the line-overflow bit
CA39:  CA         DEX            Decrement counter and jump
CA3A:  D0 FA      BNE  $CA36     If not all bits cleared yet
CA3C:  60         RTS            Return from subroutine

******************************   Insert line

CA3D:  20 7C C3   JSR  $C37C     Move remainder of screen to X
CA40:  20 56 C1   JSR  $C156     Cursor left - determine start addr
CA43:  E8         INX            Increment the line
CA44:  20 76 CB   JSR  $CB76     Test line-overflow bit
CA47:  08         PHP            Save the carry
CA48:  20 81 CB   JSR  $CB81     Set/clear test-overflow bit
CA4B:  28         PLP            Get carry from stack
CA4C:  B0 03      BCS  $CA51     Cursor line is start line
CA4E:  38         SEC            Else mark old line
CA4F:  66 E8      ROR  # $E8     As following-line
CA51:  60         RTS            Return from subroutine

******************************   Delete current line

CA52:  20 B5 CB   JSR  $CBB5     Set line start address
CA55:  A5 E5      LDA  * $E5     Load top of window into acc
CA57:  48         PHA            Save on stack
CA58:  A5 EB      LDA  * $EB     Get current cursor line in acc
CA5A:  85 E5      STA  * $E5     Define as top of window
CA5C:  A5 F8      LDA  * $F8     Save scroll flag
CA5E:  48         PHA            On stack
CA5F:  A9 80      LDA  # $80     Don't scroll
```

```
CA61:  85 F8      STA  * $F8      Enable
CA63:  20 B8 C3   JSR  $C3B8      Scroll up
CA66:  68         PLA             Get scroll flag back
CA67:  85 F8      STA  * $F8      And reconstruct
CA69:  A5 E5      LDA  * $E5      Load top of window into acc
CA6B:  85 EB      STA  * $EB      Write current cursor line
CA6D:  68         PLA             Get top of window
CA6E:  85 E5      STA  * $E5      And write back
CA70:  38         SEC             Set carry in order to write to $E8
CA71:  66 E8      ROR  # $E8      Mark as following-line
CA73:  4C 56 C1   JMP  $C156      Cursor left window border

******************************    Delete from cursor to end of line

CA76:  20 1E CC   JSR  $CC1E      Save cursor coordinates
CA79:  20 AA C4   JSR  $C4AA      Clear current line at cursor
CA7C:  E6 EB      INC  * $EB      Incr. current cursor line by 1
CA7E:  20 5C C1   JSR  $C15C      Determine line start address
CA81:  A4 E6      LDY  * $E6      Load left window-border in Y
CA83:  20 74 CB   JSR  $CB74      Test line-overflow bit
CA86:  B0 F1      BCS  $CA79      Clear following-line too
CA88:  4C 32 C9   JMP  $C932      Set old cursor address

******************************    Delete from line start to cursor

CA8B:  20 1E CC   JSR  $CC1E      Save cursor coordinates
CA8E:  20 27 CC   JSR  $CC27      Space at current cursor position
CA91:  C4 E6      CPY  * $E6      Compare w/ left window-border
CA93:  D0 05      BNE  $CA9A      Not yet reached
CA95:  20 74 CB   JSR  $CB74      Test line-overflow bit
CA98:  90 EE      BCC  $CA88      No overflow, then end
CA9A:  20 00 CC   JSR  $CC00      Else cursor left
CA9D:  90 EF      BCC  $CA8E      If moved then clear line

******************************    Delete from cursor pos to end of
                                  line

CA9F:  20 1E CC   JSR  $CC1E      Save cursor coordinates
CAA2:  20 AA C4   JSR  $C4AA      Delete line
CAA5:  E6 EB      INC  * $EB      Incr. current cursor line by 1
```

```
CAA7:  20 5C C1   JSR  $C15C      Determine start addr. cursor line
CAAA:  A4 E6      LDY  * $E6      Load left window-border Y-reg
CAAC:  20 74 CB   JSR  $CB74      Test line overflow bit
CAAF:  B0 F1      BCS  $CAA2      Line not yet done
CAB1:  A5 EB      LDA  * $EB      Get current cursor line in acc
CAB3:  C5 E4      CMP  * $E4      Compare lower window border
CAB5:  90 EB      BCC  $CAA2      Lower border not yet reached
CAB7:  F0 E9      BEQ  $CAA2      Lower border reached
CAB9:  4C 32 C9   JMP  $C932      Reset old cursor address

******************************    Scroll up

CABC:  20 1E CC   JSR  $CC1E      Save cursor coordinates
CABF:  8A         TXA             Line to acc and
CAC0:  48         PHA             Then save on stack
CAC1:  20 A6 C3   JSR  $C3A6      Perform scroll-up
CAC4:  68         PLA             Get line back from stack
CAC5:  85 DF      STA  * $DF      And store
CAC7:  4C 32 C9   JMP  $C932      Old cursor coordinates back

******************************    Scroll down

CACA:  20 1E CC   JSR  $CC1E      Save cursor coordinates
CACD:  20 74 CB   JSR  $CB74      Test line-overflow bit
CAD0:  B0 03      BCS  $CAD5      Line is not overflow line
CAD2:  38         SEC             Mark that input line is not
CAD3:  66 E8      ROR  # $E8      Start line
CAD5:  A5 E5      LDA  * $E5      Load top of window in acc
CAD7:  85 EB      STA  * $EB      Write current cursor line
CAD9:  20 7C C3   JSR  $C37C      Scroll down
CADC:  20 85 CB   JSR  $CB85      Clear line-overflow bit
CADF:  4C 32 C9   JMP  $C932      Old cursor coordinates back

******************************    Enable/disable scrolling

CAE2:  A9 00      LDA  # $00      Enable scrolling
CAE4:  2C         .Byte  $2C      Skip to $CAE7
CAE5:  A9 80      LDA  # $80      Disable scrolling
CAE7:  85 F8      STA  * $F8      Store scroll flag
CAE9:  60         RTS             Return from subroutine
```

```
******************************    Set/clear flag for auto-insert

CAEA:  A9 00       LDA  # $00     Clear auto-insert flag
CAEC:  2C          .Byte  $2C     Skip to $CAEF
CAED:  A9 80       LDA  # $80     Set auto-insert flag
CAEF:  85 F6       STA  * $F6     And store flag
CAF1:  60          RTS            Return from subroutine

******************************    Turn on block cursor

CAF2:  24 D7       BIT  * $D7     Test 40/80-column mode
CAF4:  10 40       BPL  $CB36     For 40-column mode --> end
CAF6:  AD 2B 0A    LDA  $0A2B     Get VDC cursor mode
CAF9:  29 E0       AND  # $E0     Mask out bits 0-4 (start-scan)
CAFB:  4C 14 CB    JMP  $CB14     Save and VIC cursor off

******************************    Turn on underline cursor

CAFE:  24 D7       BIT  * $D7     Test 40/80-column flag
CB00:  10 34       BPL  $CB36     If 40-column, end
CB02:  AD 2B 0A    LDA  $0A2B     Get VDC cursor mode
CB05:  29 E0       AND  # $E0     Mask out start-scan
CB07:  09 07       ORA  # $07     Start-scan line is 7
CB09:  D0 09       BNE  $CB14     Unconditional jump to setting

******************************    Cursor flash off

CB0B:  24 D7       BIT  * $D7     Test 40/80-column mode
CB0D:  10 0B       BPL  $CB1A     If 40-column, then jump
CB0F:  AD 2B 0A    LDA  $0A2B     Get VDC cursor mode
CB12:  29 1F       AND  # $1F     Mask out flash
CB14:  8D 2B 0A    STA  $0A2B     And save again
CB17:  4C 91 CD    JMP  $CD91     Set mode and VIC off

******************************    for 40 column

CB1A:  AD 26 0A    LDA  $0A26     Get VIC cursor mode
CB1D:  09 40       ORA  # $40     Set bit 6 for steady
CB1F:  D0 12       BNE  $CB33     Unconditional jump to store
```

```
******************************    Cursor flash on

CB21:  24 D7      BIT  * $D7      Test 40/80-column mode
CB23:  10 09      BPL  $CB2E      Jump if 40 column
CB25:  AD 2B 0A   LDA  $0A2B      Get VDC cursor mode
CB28:  29 1F      AND  # $1F      Mask out flash
CB2A:  09 60      ORA  # $60      And define flash period
CB2C:  D0 E6      BNE  $CB14      Unconditional jump to store

******************************    for 40 column

CB2E:  AD 26 0A   LDA  $0A26      Get VIC cursor mode
CB31:  29 BF      AND  # $BF      Mask otu bit 6 (steady)
CB33:  8D 26 0A   STA  $0A26      And save again
CB36:  60         RTS             Return from subroutine

******************************    Set/clear flag for bell

CB37:  A9 00      LDA  # $00      Enable bell
CB39:  2C         .Byte  $2C      Skip to $CB3C
CB3A:  A9 80      LDA  # $80      Disable bell
CB3C:  85 F9      STA  * $F9      And store flag
CB3E:  60         RTS             Return from the subroutine

******************************    Reverse 80-column monitor

CB3F:  A2 18      LDX  # $18      Select register 24
CB41:  20 DA CD   JSR  $CDDA      And get current contents
CB44:  09 40      ORA  # $40      Set reverse flag
CB46:  D0 07      BNE  $CB4F      Unconditional jump to $CB4F

******************************    Switch 80-column monitor
                                  normal

CB48:  A2 18      LDX  # $18      Select register 24
CB4A:  20 DA CD   JSR  $CDDA      And get current contents
CB4D:  29 BF      AND  # $BF      Clear the reverse flag
CB4F:  4C CC CD   JMP  $CDCC      And store
```

```
******************************    Cursor to end of current line

CB52:  20 C3 CB   JSR  $CBC3      Determine start addr current line
CB55:  4C 3E C3   JMP  $C33E      Cursor to end of line

******************************    Get char and color at cursor pos

CB58:  A4 EC      LDY  * $EC      Get current cursor col in Y-reg
CB5A:  24 D7      BIT  * $D7      Test 40/80-column mode
CB5C:  30 07      BMI  $CB65      Jump if 80-column mode
CB5E:  B1 E2      LDA  ($E2),Y    Get color at cursor position
CB60:  85 F2      STA  * $F2      And save
CB62:  B1 E0      LDA  ($E0),Y    Get character at cursor position
CB64:  60         RTS             Return from subroutine

******************************    Get char. and color under cursor

CB65:  20 F9 CD   JSR  $CDF9      Set the update address to ARA
CB68:  20 D8 CD   JSR  $CDD8      Get current attribute
CB6B:  85 F2      STA  * $F2      Store attribute
CB6D:  20 E6 CD   JSR  $CDE6      Set the update address to video
CB70:  20 D8 CD   JSR  $CDD8      Get character from video RAM
CB73:  60         RTS             Return from subroutine

******************************    Routine to test line-overflow bit

CB74:  A6 EB      LDX  * $EB      Get current cursor line in X-reg
CB76:  20 9F CB   JSR  $CB9F      Determine power 2 & remainder
CB79:  3D 5E 03   AND  $035E,X    Clear line overflow bit
CB7C:  C9 01      CMP  # $01      No line set in the block?
CB7E:  4C 90 CB   JMP  $CB90      Jump to the end of the routine
CB81:  A6 EB      LDX  * $EB      Get current cursor line in X-reg
CB83:  B0 0E      BCS  $CB93      Jump if flag set
CB85:  20 9F CB   JSR  $CB9F      Determine power 2 & remainder
CB88:  49 FF      EOR  # $FF      One's complement of acc
CB8A:  3D 5E 03   AND  $035E,X    combine with line overflow table
CB8D:  9D 5E 03   STA  $035E,X    And store again
CB90:  A6 DA      LDX  * $DA      Get X from temp storage
CB92:  60         RTS             Return from subroutine
```

```
*****************************      Set the line-overflow bit

CB93:  24 F8      BIT  * $F8       Test scroll bit
CB95:  70 DF      BVS  $CB76       Jump if bit 6 set
CB97:  20 9F CB   JSR  $CB9F       Determine power 2 & remainder
CB9A:  1D 5E 03   ORA  $035E,X     Set the line-overflow bit
CB9D:  D0 EE      BNE  $CB8D       And update

*****************************      Routine finds 2^(X AND 7) and
                                   INT (X/8). Param in X-reg

CB9F:  86 DA      STX  * $DA       Save the accumulator
CBA1:  8A         TXA              X-register to acc
CBA2:  29 07      AND  # $07       Mask out bits 3-7=X MOD 8
CBA4:  AA         TAX              Acc back to X-reg
CBA5:  BD 6C CE   LDA  $CE6C,X     Get corresponding power of 2
CBA8:  48         PHA              Save acc on stack
CBA9:  A5 DA      LDA  * $DA       Get original value back
CBAB:  4A         LSR  A           This value is divided by 2
CBAC:  4A         LSR  A           Three times
CBAD:  4A         LSR  A           Which results in INT(X/8)
CBAE:  AA         TAX              Result to X-reg
CBAF:  68         PLA              Get power of 2 from stack
CBB0:  60         RTS              Return from subroutine

*****************************      Clear the overflow chain

CBB1:  A4 E6      LDY  * $E6       Put left window-bdr into Y-reg
CBB3:  84 EC      STY  * $EC       Save the current cursor column
CBB5:  20 74 CB   JSR  $CB74       Clr line-overfl. bit of cur. line
CBB8:  90 06      BCC  $CBC0       Carry cleared if all bits are 0
CBBA:  C6 EB      DEC  * $EB       Decrement current cursor line
CBBC:  10 F7      BPL  $CBB5       If not first line then jump
CBBE:  E6 EB      INC  * $EB       Increment current cursor line
CBC0:  4C 5C C1   JMP  $C15C       Find start addr of current line
```

```
*****************************   Search for end of input line

CBC3:  E6 EB      INC  * $EB    Increment current cursor line
CBC5:  20 74 CB   JSR  $CB74    Clear line-overflow bit
CBC8:  B0 F9      BCS  $CBC3    If not last line => Jump
CBCA:  C6 EB      DEC  * $EB    Decrement current cursor line
CBCC:  20 5C C1   JSR  $C15C    Find start addr of current line
CBCF:  A4 E7      LDY  * $E7    Load rt window-border, Y-reg
CBD1:  84 EC      STY  * $EC    Save the current cursor column
CBD3:  20 58 CB   JSR  $CB58    Get char. and color at cursor pos
CBD6:  A6 EB      LDX  * $EB    Get current cursor line in X-reg
CBD8:  C9 20      CMP  # $20    Is character <space>?
CBDA:  D0 0E      BNE  $CBEA    No, then jump
CBDC:  C4 E6      CPY  * $E6    Compare with lf window-border
CBDE:  D0 05      BNE  $CBE5    Not yet reached
CBE0:  20 74 CB   JSR  $CB74    Clear line-overflow bit
CBE3:  90 05      BCC  $CBEA    A line is still free
CBE5:  20 00 CC   JSR  $CC00    Cursor one position to the left
CBE8:  90 E9      BCC  $CBD3    Cursor can be moved
CBEA:  84 EA      STY  * $EA    Current input line: End
CBEC:  60         RTS           Return from subroutine

*****************************   Cursor 1 spc right in window

CBED:  48         PHA           Save acc on stack
CBEE:  A4 EC      LDY  * $EC    Get current cursor line in Y-reg
CBF0:  C4 E7      CPY  * $E7    Compare to rt window-border
CBF2:  90 07      BCC  $CBFB    Right window-border reached?
CBF4:  20 63 C3   JSR  $C363    No, then increment crsr column
CBF7:  A4 E6      LDY  * $E6    Load left window-border into Y
CBF9:  88         DEY           Decrement
CBFA:  38         SEC           Carry set means new line
CBFB:  C8         INY           Increment cursor column
CBFC:  84 EC      STY  * $EC    Store the current cursor column
CBFE:  68         PLA           Put acc back on stack
CBFF:  60         RTS           Return from the subroutine
```

```
******************************   Cursor 1 spc to left in window

CC00:  A4 EC      LDY  * $EC     Get current crsr column in Y-reg
CC02:  88         DEY            Decrement the column by 1
CC03:  30 04      BMI  $CC09     If negative, cursor in column 0
CC05:  C4 E6      CPY  * $E6     Compare with lf window-border
CC07:  B0 0F      BCS  $CC18     Left edge not reached, OK
CC09:  A4 E5      LDY  * $E5     Load top of window in Y-reg
CC0B:  C4 EB      CPY  * $EB     Compare with current cursor line
CC0D:  B0 0E      BCS  $CC1D     Cursor is in topmost line, end
CC0F:  C6 EB      DEC  * $EB     Decrement current cursor line
CC11:  48         PHA            Save acc on stack
CC12:  20 5C C1   JSR  $C15C     Find start address of the line
CC15:  68         PLA            Get acc back from stack
CC16:  A4 E7      LDY  * $E7     Load right window-bdr in Y-reg
CC18:  84 EC      STY  * $EC     Save the current cursor column
CC1A:  C4 E7      CPY  * $E7     Compare with right window-bdr
CC1C:  18         CLC            Clear carry for cursor moved
CC1D:  60         RTS            Return from the subroutine

******************************   Copy cursor (X/Y) to $DE/$DF

CC1E:  A4 EC      LDY  * $EC     Get current crsr column in Y-reg
CC20:  84 DE      STY  * $DE     Copy to $DE
CC22:  A6 EB      LDX  * $EB     Get current crsr column in X-reg
CC24:  86 DF      STX  * $DF     Copy to $DF
CC26:  60         RTS            Return from the subroutine

******************************   Space at current cursor position

CC27:  A5 F1      LDA  * $F1     Color code for char output in acc
CC29:  29 8F      AND  # $8F     Mask out bits 4-6 (attribute)
CC2B:  AA         TAX            And to X-register
CC2C:  A9 20      LDA  # $20     Load acc with space
CC2E:  2C         .Byte  $2C     Skip to $CC31
```

```
*****************************   Character (acc) at cursor position

CC2F:  A6 F1      LDX  * $F1    Load X-register with color
CC31:  2C         .Byte  $2C    Skip to $CC34
CC32:  A6 F2      LDX  * $F2    Color code reg. for insert/delete
CC34:  A8         TAY           Acc to Y-register
CC35:  A9 02      LDA  # $02    Place the value two in
CC37:  8D 28 0A   STA  $0A28    VIC cursor-flash counter
CC3A:  20 7C C1   JSR  $C17C    Adapt attribute address
CC3D:  98         TYA           And Y-register back to acc
CC3E:  A4 EC      LDY  * $EC    Get current crsr column in Y-reg
CC40:  24 D7      BIT  * $D7    Test 40/80 column mode
CC42:  30 06      BMI  $CC4A    Jump if 80-column mode
CC44:  91 E0      STA  ($E0),Y  Store character in 40-column
CC46:  8A         TXA           Put video RAM & X-reg. (color)
CC47:  91 E2      STA  ($E2),Y  In color memory
CC49:  60         RTS           Return from subroutine

*****************************   Character on 80-column screen
                                Acc: character, X: color, Y: col

CC4A:  48         PHA           Save acc on stack
CC4B:  8A         TXA           X-register (color) to acc
CC4C:  48         PHA           And store on stack
CC4D:  20 F9 CD   JSR  $CDF9    Set update register for attribute
CC50:  68         PLA           Get color from stack in acc
CC51:  20 CA CD   JSR  $CDCA    And store in attribute RAM
CC54:  20 E6 CD   JSR  $CDE6    Set update addr. for video RAM
CC57:  68         PLA           And get character from stack
CC58:  4C CA CD   JMP  $CDCA    Store character in video RAM

*****************************   Find chars/line & lines/window

CC5B:  38         SEC           Set carry
CC5C:  A5 E4      LDA  * $E4    Load bottom of window in acc
CC5E:  E5 E5      SBC  * $E5    Minus top yields lines
CC60:  A8         TAY           Of the window to Y-register
CC61:  38         SEC           Set the carry again
CC62:  A5 E7      LDA  * $E7    Load rt window-border into acc
CC64:  E5 E6      SBC  * $E6    Minus left window-border yields
```

```
CC66:  AA          TAX           Number of chars/line into X-reg
CC67:  A5 EE       LDA  * $EE    Max number of columns in acc
CC69:  60          RTS           Return from the subroutine

****************************     Get or set cursor position

CC6A:  B0 29       BCS  $CC95    If carry set - then get pos
CC6C:  8A          TXA           Line to acc
CC6D:  65 E5       ADC  * $E5    Add top of window
CC6F:  B0 14       BCS  $CC85    If overflow then end (Error!)
CC71:  C5 E4       CMP  * $E4    Compare to bottom of window
CC73:  F0 02       BEQ  $CC77    If reached, then OK
CC75:  B0 0E       BCS  $CC85    If oveflow then end (Error!)
CC77:  48          PHA           Save line on stack
CC78:  18          CLC           Clear carry for addition
CC79:  98          TYA           Get column in acc
CC7A:  65 E6       ADC  * $E6    And add left window-border
CC7C:  B0 06       BCS  $CC84    If overflow then end (Error!)
CC7E:  C5 E7       CMP  * $E7    Compare to rt window-border
CC80:  F0 04       BEQ  $CC86    If equal, then OK
CC82:  90 02       BCC  $CC86    If overflow then end (Error!)
CC84:  68          PLA           Get line from stack
CC85:  60          RTS           Return from subroutine

****************************     Make input line clear

CC86:  85 EC       STA  * $EC    Store the current cursor column
CC88:  85 E9       STA  * $E9    Store the start input line
CC8A:  68          PLA           Get line from stack
CC8B:  85 EB       STA  * $EB    Write current cursor line back
CC8D:  85 E8       STA  * $E8    Store as start input line
CC8F:  20 5C C1    JSR  $C15C    Determine addr of current line
CC92:  20 57 CD    JSR  $CD57    Set cursor to current column
CC95:  A5 EB       LDA  * $EB    Get current cursor line in acc
CC97:  E5 E5       SBC  * $E5    Subtract top of window
CC99:  AA          TAX           Result then to X
CC9A:  38          SEC           Set carry for subtraction
CC9B:  A5 EC       LDA  * $EC    Get current cursor column in acc
CC9D:  E5 E6       SBC  * $E6    Subtract left window-border
CC9F:  A8          TAY           Result to Y
```

```
CCA0:  18          CLC          Clear carry for OK
CCA1:  60          RTS          Return from subroutine

******************************  Kernal entry: PFKEY
                                 Program function key

CCA2:  CA          DEX          Dec the number of the ftn. key
CCA3:  86 DC       STX  * $DC   Number of (ftn key -1) in Z-P
CCA5:  84 DA       STY  * $DA   Store length of string in Z-P
CCA7:  8D AA 02    STA  $02AA   Z-P addr - string ptr in FETVEC
CCAA:  A8          TAY          Z-page address of string ptr in Y
CCAB:  B6 02       LDX  * $02,Y Get bank # of the ftn string in X
CCAD:  20 6B FF    JSR  $FF6B   Kernal: GETCFG get config
CCB0:  85 DE       STA  * $DE   Store in bank byte for ftn string
CCB2:  A2 0A       LDX  # $0A   Number of ftn keys (10) in acc
CCB4:  20 20 CD    JSR  $CD20   Add ftn str lengths up to (X -1)
CCB7:  85 DB       STA  * $DB   Store string length in zero page
CCB9:  A6 DC       LDX  * $DC   Get number of the (ftn key -1)
CCBB:  E8          INX          Create real ftn key number
CCBC:  20 20 CD    JSR  $CD20   Add ftn str lengths up to (X -1)
CCBF:  85 DD       STA  * $DD   Store string length
CCC1:  A6 DC       LDX  * $DC   Get number of (ftn key -1)
CCC3:  A5 DA       LDA  * $DA   Get string length of ftn key
CCC5:  38          SEC          Set carry for normal subtraction
CCC6:  FD 00 10    SBC  $1000,X Subtract length of the old ftn str
CCC9:  F0 2B       BEQ  $CCF6   No move necessary, continue
CCCB:  90 16       BCC  $CCE3   New string shorter than old
CCCD:  18          CLC          Clear carry for addition
CCCE:  65 DB       ADC  * $DB   Add total length + difference len
CCD0:  B0 4D       BCS  $CD1F   Length > 256 than RTS: error
CCD2:  AA          TAX          Put new maximum length in X
CCD3:  A4 DB       LDY  * $DB   Get old max length in Y
CCD5:  C4 DD       CPY  * $DD   If both are equal, than the last
CCD7:  F0 1D       BEQ  $CCF6   Ftn key was addressed
CCD9:  88          DEY          Decrement old max length by 1
CCDA:  CA          DEX          Decrement new max length by 1
CCDB:  B9 0A 10    LDA  $100A,Y Move ftn str's away from new
CCDE:  9D 0A 10    STA  $100A,X Insert position
CCE1:  B0 F2       BCS  $CCD5   And create space for the new str
CCE3:  65 DD       ADC  * $DD   Add difference length
```

```
CCE5:  AA          TAX             Copy new len in X
CCE6:  A4 DD       LDY  * $DD      Get old len in Y
CCE8:  C4 DB       CPY  * $DB      Compare with old max length
CCEA:  B0 0A       BCS  $CCF6      Equal, than space
CCEC:  B9 0A 10    LDA  $100A,Y    Insertion for new ftn string
CCEF:  9D 0A 10    STA  $100A,X    For ftn key is done
CCF2:  C8          INY             Increment old & new len
CCF3:  E8          INX             By 1 for move
CCF4:  90 F2       BCC  $CCE8      Until ftn strings shifted

******************************     Insert new function string

CCF6:  A6 DC       LDX  * $DC      Get number of the (ftn key -1)
CCF8:  20 20 CD    JSR  $CD20      Add ftn str lengths up to (X -1)
CCFB:  AA          TAX             Get str len up to the new ftn key
CCFC:  A4 DC       LDY  * $DC      Get # of the (ftn key -1)
CCFE:  A5 DA       LDA  * $DA      Length of the ftn string to insert
CD00:  99 00 10    STA  $1000,Y    Replace len entry in ftn str table
CD03:  A0 00       LDY  # $00      Initialize displacement pointer
CD05:  C6 DA       DEC  * $DA      Length of the ftn str = length -1
CD07:  30 15       BMI  $CD1E      All chars in table xferred, exit
CD09:  86 DF       STX  * $DF      Store the "to" string length
CD0B:  A6 DE       LDX  * $DE      Bank value where str is located
CD0D:  AD AA 02    LDA  $02AA      Load acc with FETVEC
CD10:  78          SEI             Disable all system interrupts
CD11:  20 A2 02    JSR  $02A2      FETCH: get ftn string character
CD14:  58          CLI             Enable all system interrupts
CD15:  A6 DF       LDX  * $DF      Position for ftn string in table
CD17:  9D 0A 10    STA  $100A,X    Enter character in ftn string table
CD1A:  E8          INX             Displ. to "to where" str buffer+1
CD1B:  C8          INY             Displ to "from where" str buff+1
CD1C:  D0 E7       BNE  $CD05      Jump in the string transfer loop
CD1E:  18          CLC             Marker for "OK" return
CD1F:  60          RTS             Return from the subroutine

******************************     Add lengths of ftn str's up to X

CD20:  A9 00       LDA  # $00      Load counter with zero
CD22:  18          CLC             Clear carry for addition
CD23:  CA          DEX             Previous key assignment
```

```
CD24:  30 05      BMI  $CD2B      If zero, then add all
CD26:  7D 00 10   ADC  $1000,X    Add length of key X
CD29:  90 F8      BCC  $CD23      Jump unconditionally to $CD23

CD2B:  60         RTS             Return from subroutine

******************************    Kernal routine: SWAPPER
                                  Switch 40/80-col modes

CD2C:  85 F0      STA  * $F0      Store acc as last-printed char
CD2E:  A2 1A      LDX  # $1A      Exchange the passive monitor
CD30:  BC 40 0A   LDY  $0A40,X    Storage with the active storage.
CD33:  B5 E0      LDA  * $E0,X    This is done 26 times because
CD35:  9D 40 0A   STA  $0A40,X    26 bytes must be copied
CD38:  98         TYA             The passive range lies from
CD39:  95 E0      STA  * $E0,X    $0A40 to $0A5B.
CD3B:  CA         DEX             Decrement the counter if
CD3C:  10 F2      BPL  $CD30      Not done exchanging
CD3E:  A2 0D      LDX  # $0D      Now the bit maps, the bit tables
CD40:  BC 60 0A   LDY  $0A60,X    Of active and passive screens
CD43:  BD 54 03   LDA  $0354,X    Must be exchanged.
CD46:  9D 60 0A   STA  $0A60,X    This is done 13 times
CD49:  98         TYA             The passive areas starts at
CD4A:  9D 54 03   STA  $0354,X    $0A60.
CD4D:  CA         DEX             Decrement counter and jump
CD4E:  10 F0      BPL  $CD40      If not done copying
CD50:  A5 D7      LDA  * $D7      Get status 40/80 column
CD52:  49 80      EOR  # $80      And invert flag bit
CD54:  85 D7      STA  * $D7      Save again
CD56:  60         RTS             Return from subroutine

******************************    Set cursor to current column

CD57:  24 D7      BIT  * $D7      Test for 40/80 column mode
CD59:  10 FB      BPL  $CD56      End if 40-column mode
CD5B:  A2 0E      LDX  # $0E      Cursor position high
CD5D:  18         CLC             Clear carry
CD5E:  A5 E0      LDA  * $E0      Low byte of current screen line
CD60:  65 EC      ADC  * $EC      Add cursor column
CD62:  48         PHA             Save low byte
```

```
CD63:  A5 E1      LDA  * $E1     High byte of current screen line
CD65:  69 00      ADC  # $00     Add the carry
CD67:  20 CC CD   JSR  $CDCC     And store the high byte
CD6A:  E8         INX            Increment register pointer to $0F
CD6B:  68         PLA            Get low byte from stack
CD6C:  4C CC CD   JMP  $CDCC     And save it too (return)

****************************     Set cursor color at cursor pos

CD6F:  24 D7      BIT  * $D7     Test for 40/80-column mode
CD71:  10 26      BPL  $CD99     Jump if 40 column mode
CD73:  20 7C C1   JSR  $C17C     Set attribute address
CD76:  A4 EC      LDY  * $EC     Get current crsr column in Y-reg
CD78:  20 F9 CD   JSR  $CDF9     Attribute addr in update register
CD7B:  20 D8 CD   JSR  $CDD8     Get current attribute
CD7E:  8D 33 0A   STA  $0A33     Store temporarily
CD81:  29 F0      AND  # $F0     Mask out bits 0-3 (color)
CD83:  85 DB      STA  * $DB     And store
CD85:  20 F9 CD   JSR  $CDF9     Attribute addr in update register
CD88:  A5 F1      LDA  * $F1     Color code for char output in acc
CD8A:  29 0F      AND  # $0F     Mask out bits 4-7 (attribute)
CD8C:  05 DB      ORA  * $DB     And combine with attribute
CD8E:  20 CA CD   JSR  $CDCA     Store at attribute address
CD91:  A2 0A      LDX  # $0A     Cursor mode and start-scan line
CD93:  AD 2B 0A   LDA  $0A2B     80-column cursor mode
CD96:  4C CC CD   JMP  $CDCC     And store
CD99:  A9 00      LDA  # $00     Acc equal zero and store
CD9B:  8D 27 0A   STA  $0A27     Means turn VIC cursor off
CD9E:  60         RTS            Return from subroutine

****************************     Turn cursor on (80-column)

CD9F:  24 D7      BIT  * $D7     Test 40/80-column mode
CDA1:  10 10      BPL  $CDB3     Jump if 40-columnmode
CDA3:  20 F9 CD   JSR  $CDF9     Set update to attribute address
CDA6:  AD 33 0A   LDA  $0A33     Temp storage for MOVLIN
CDA9:  20 CA CD   JSR  $CDCA     Store attribute
CDAC:  A2 0A      LDX  # $0A     Cursor mode and start-scan line
CDAE:  A9 20      LDA  # $20     Assigned value 32
CDB0:  4C CC CD   JMP  $CDCC     Place acc in VDC data register
```

```
******************************    Turn cursor on (40-column)

CDB3:  8D 27 0A   STA  $0A27      Turn VIC cursor on
CDB6:  AD 26 0A   LDA  $0A26      Steady or flashing cursor?
CDB9:  10 0E      BPL  $CDC9      Steady, then end
CDBB:  29 40      AND  # $40      Clear flash flag
CDBD:  8D 26 0A   STA  $0A26      And store again
CDC0:  AD 29 0A   LDA  $0A29      VIC character before flash
CDC3:  AE 2A 0A   LDX  $0A2A      VIC color before flash
CDC6:  20 34 CC   JSR  $CC34      Set old values
CDC9:  60         RTS             Return from subroutine

******************************    Acc in data register of VCR

CDCA:  A2 1F      LDX  # $1F      VCR data register
CDCC:  8E 00 D6   STX  $D600      Transmit register
CDCF:  2C 00 D6   BIT  $D600      Test status
CDD2:  10 FB      BPL  $CDCF      Not done yet, wait
CDD4:  8D 01 D6   STA  $D601      Store value in register
CDD7:  60         RTS             Return from subroutine

******************************    Get value of the data register

CDD8:  A2 1F      LDX  # $1F      VCR data register
CDDA:  8E 00 D6   STX  $D600      Transmit register
CDDD:  2C 00 D6   BIT  $D600      Test status
CDE0:  10 FB      BPL  $CDDD      Not done yet, wait
CDE2:  AD 01 D6   LDA  $D601      Get value of the register
CDE5:  60         RTS             Return from subroutine

******************************    Set update address to current
                                  screen position

CDE6:  A2 12      LDX  # $12      Update address high
CDE8:  18         CLC             Clear carry for addition
CDE9:  98         TYA             Y (column) to acc
CDEA:  65 E0      ADC  * $E0      Add low byte of current addr
CDEC:  48         PHA             Then on stack
CDED:  A9 00      LDA  # $00      Load acc with zero and then
CDEF:  65 E1      ADC  * $E1      Add the carry
```

```
CDF1:  20 CC CD   JSR  $CDCC      Store the high byte
CDF4:  68         PLA             Get low byte from stack
CDF5:  E8         INX             Increment register to $13
CDF6:  4C CC CD   JMP  $CDCC      And low byte in update register

******************************    Set update address for attribute

CDF9:  A2 12      LDX  # $12      Update register high byte
CDFB:  18         CLC             Clear carry for addition
CDFC:  98         TYA             Y (column) to acc
CDFD:  65 E2      ADC  * $E2      Add low byte of attribute addr
CDFF:  48         PHA             And then on stack
CE00:  A9 00      LDA  # $00      Load acc with zero and then
CE02:  65 E3      ADC  * $E3      Add carry
CE04:  20 CC CD   JSR  $CDCC      Store high byte
CE07:  68         PLA             Get low byte from stack and
CE08:  E8         INX             Increment register to $13
CE09:  4C CC CD   JMP  $CDCC      Store low byte

******************************    Copy character set in VDC RAM

CE0C:  A9 00      LDA  # $00      Load acc (low) & Y (high) with
CE0E:  A0 D0      LDY  # $D0      Start addr - CHARROM: $D000
CE10:  85 DA      STA  * $DA      Store these values in zero-page
CE12:  84 DB      STY  * $DB      Addresses $DA and $DB
CE14:  A2 12      LDX  # $12      Update register high
CE16:  A9 20      LDA  # $20      Start address of char generator
CE18:  20 CC CD   JSR  $CDCC      Define in VDC
CE1B:  E8         INX             Pointer to low byte
CE1C:  A9 00      LDA  # $00      $00 is low byte of start address
CE1E:  20 CC CD   JSR  $CDCC      Of the character generator
CE21:  A0 00      LDY  # $00      Index pointer to line/char
CE23:  A2 0E      LDX  # $0E      Select CHARROM
CE25:  A9 DA      LDA  # $DA      Zero-page address to access
CE27:  20 74 FF   JSR  $FF74      INDFET: LDA(XX),Y fr bank
CE2A:  20 CA CD   JSR  $CDCA      And store value in RAM
CE2D:  C8         INY             VDC. Increment index pointer
CE2E:  C0 08      CPY  # $08      All 8 character copied?
CE30:  90 F1      BCC  $CE23      No, then next line
CE32:  A9 00      LDA  # $00      Else load acc with zero
```

```
CE34:  20 CA CD   JSR  $CDCA      And store value in VDC-RAM
CE37:  88         DEY             Eight times
CE38:  D0 FA      BNE  $CE34      Jump if not yet done
CE3A:  18         CLC             Clear carry for addition
CE3B:  A5 DA      LDA  * $DA      Load acc with low byte
CE3D:  69 08      ADC  # $08      And add 8 to it
CE3F:  85 DA      STA  * $DA      Store again and
CE41:  90 E0      BCC  $CE23      If no carry than continue
CE43:  E6 DB      INC  * $DB      Else account for carry
CE45:  A5 DB      LDA  * $DB      And check if the high byte
CE47:  C9 E0      CMP  # $E0      points to end of CHARROM
CE49:  90 D8      BCC  $CE23      Else continue
CE4B:  60         RTS             Return from the subroutine

******************************    Table of the color codes (ASCII)

CE4C:  90 05 1C 9F 9C 1E 1F 9E
CE54:  81 95 96 97 98 99 9A 9B

******************************    Table of color codes for VDC

CE5C:  00 0F 08 07 0B 04 02 0D
CE64:  0A 0C 09 06 01 05 03 0E

******************************    Power of 2

CE6C:  80 40 20 10 08 04 02 01    128, 64, 32, 16, 8, 4, 2, 1

******************************    Init. values for 40-col screen

CE74:  00 04 00 D8 18 00 00 27    These values are copied to zero
CE7C:  00 00 00 00 00 18 27 00    page at $E0 during initialization
CE84:  00 0D 0D 00 00 00 00 00    They are explained in zero-page
CE8C:  00 00                      Comments

******************************    Init. values for 80-col screen

CE8E:  00 00 00 08 18 00 00 4F    These values are copied into
CE96:  00 00 00 00 00 18 4F 00    Page 3 at $0A40 during init.
CE9E:  00 07 07 00 00 00 00 00    They're explained in page-three
CEA6:  00 00                      Comments
```

```
*****************************      Init. assignment of function keys

CEA8: 07 06 0A 07 06 04 05 08      Length of function key strings
CEB0:  09 05                       (F1 - F8, Shift-Run, Help)

*****************************      Init. ftn key string assignments

CEB2:  47 52 41 50 48 49 43        GRAPHIC
CEB9:  44 4C 4F 41 44 22           DLOAD"
CEBF:  44 49 52 45 43 54 4F 52     DIRECTORY <Cr>
CEC7:  59 0D
CEC9:  53 43 4E 43 4C 52 0D        SCNCLR <Cr>
CED0:  44 53 41 56 45 22           DSAVE"
CED6:  52 55 4E 0D                 RUN <Cr>
CEDA:  4C 49 53 54 0D              LIST <Cr>
CEDF:  4D 4F 4E 49 54 4F 52 0D     MONITOR <Cr>
CEE7:  44 CC 22 2A 0D 52 55 4E     D <Shift - L> <Cr> RUN <Cr>
CEEF:  0D
CEF0:  48 45 4C 50 0D              HELP <Cr>

*****************************      Free area

CEF5:  FF FF FF . . .              Not used
CFFD:  . . . FF 00 FF              Not used
```

```
****************************** Reset routine

E000:  A2 FF      LDX  # $FF     Init. value for stack pointer
E002:  78         SEI              Disable all system interrupts
E003:  9A         TXS              Set system stack pointer to start
E004:  D8         CLD              Reset decimal mode
E005:  A9 00      LDA  # $00     Load acc with zero and
E007:  8D 00 FF   STA  $FF00     Enable all system ROMs
E00A:  A2 0A      LDX  # $0A     Set loop and displ. counter
E00C:  BD 4B E0   LDA  $E04B,X   Get byte from init. counter
E00F:  9D 00 D5   STA  $D500,X   Initialize MMU registers
E012:  CA         DEX              Loop and displ. counter -1
E013:  10 F7      BPL  $E00C     Transfer 11 values from table
E015:  8D 04 0A   STA  $0A04     Clear NMI/Reset status pointer
E018:  20 CD E0   JSR  $E0CD     NMI,IRQ+copy  z-page routines
E01B:  20 F0 E1   JSR  $E1F0     Check <CBM> code in RAM 1
E01E:  20 42 E2   JSR  $E242     Cartridge test for C-64 config
E021:  20 09 E1   JSR  $E109     Kernal IOINIT: Init I/O devices
E024:  20 3D F6   JSR  $F63D     Shift RUN/STOP keyboard test
Q027:  48         PHA              Save acc contents on stack
E028:  30 07      BMI  $E031     Bit 7 set, skip reset status test
E02A:  A9 A5      LDA  # $A5     System warm/cold start stat. ptr.
E02C:  CD 02 0A   CMP  $0A02     Test for warm-start status
E02F:  F0 03      BEQ  $E034     Warm-start status, then skip
E031:  20 93 E0   JSR  $E093     RAMTAS: Clear/test RAM
E034:  20 56 E0   JSR  $E056     RESTOR: Initialize I/O
E037:  20 00 C0   JSR  $C000     Routine CINT: Init. editor+scr.
E03A:  68         PLA              Get code for keyboard poll
E03B:  58         CLI              Enable all system interrupts
E03C:  30 03      BMI  $E041     Bit 7 set, skip monitor entry
E03E:  4C 00 B0   JMP  $B000     Kernal MONITOR entry
E041:  C9 DF      CMP  # $DF     Configure system as C-64?
E043:  F0 03      BEQ  $E048     Yes, then do it
E045:  6C 00 0A   JMP  ($0A00)   System restart vector ($4003)
E048:  4C 4B E2   JMP  $E24B     GO64MODE: configure C-64
```

```
*****************************      Initialization table for MMU

E04B:  00          .Byte  $00      $D500: Configuration Register
E04C:  00          .Byte  $00      $D501: Preconfig. Register A
E04D:  00          .Byte  $00      $D502: Preconfig. Register B
E04E:  00          .Byte  $00      $D503: Preconfig. Register C
E04F:  00          .Byte  $00      $D504: Preconfig. Register D
E050:  BF          .Byte  $BF      $D505: Mode Config. Register
E051:  04          .Byte  $04      $D506: RAM Config. Register
E052:  00          .Byte  $00      $D507: Page 0 Pointer Low
E053:  00          .Byte  $00      $D508: Page 0 Pointer High
E054:  01          .Byte  $01      $D509: Page 1 Pointer Low
E055:  00          .Byte  $00      $D50A: Page 1 Pointer High

*****************************      Kernal routine: RESTOR

E056:  A2 73       LDX  # $73      Low byte of kernal vector table
E058:  A0 E0       LDY  # $E0      High byte of kernal vector table
E05A:  18          CLC             Marker for dnload of vector table

*****************************      Kernal routine: VECTOR

E05B:  86 C3       STX  * $C3      Low byte of vector tbl in z-page
E05D:  84 C4       STY  * $C4      High byte of vector tbl ($E073)
E05F:  A0 1F       LDY  # $1F      Set loop counter to 32
E061:  B9 14 03    LDA  $0314,Y    Read byte from page 3 vector tbl
E064:  B0 02       BCS  $E068      If upload vector table, skip
E066:  B1 C3       LDA  ($C3),Y    Read a value from vector table
E068:  99 14 03    STA  $0314,Y    Store in page three vector table
E06B:  90 02       BCC  $E06F      If download vector table, skip
E06D:  91 C3       STA  ($C3),Y    Copy in indexed table
E06F:  88          DEY             Loop counter & displacement -1
E070:  10 EF       BPL  $E061      Loop until table transferred
E072:  60          RTS             Return from subroutine

*****************************      Vector table

E073:  65 FA       $FA65           Vector points to IRQ entry
E075:  03 B0       $B003           Vector to Monitor Break entry
E077:  40 FA       $FA40           Vector points to NMI entry
```

```
E079:  BD EF      $EFBD          Vctr pts to Kernal OPEN rout.
E07B:  88 F1      $F188          Vctr pts to Kernal CLOSE rout.
E07D:  06 F1      $F106          Vctr pts to Kernal CHKIN rout.
E07F:  4C F1      $F14C          Vctr pts to Kernal CKOUT rout.
E081:  26 F2      $F226          Vctr pts to Kernal CLRCH rout.
E083:  06 EF      $EF06          Vctr pts to Kernal BASIN rout.
E085:  79 EF      $EF79          Vctr pts to Kernal BSOUT rout.
E087:  6E F6      $F66E          Vctr pts to Kernal STOP rout.
E089:  EB EE      $EEEB          Vctr pts to Kernal GETIN rout.
F08B:  22 F2      $F222          Vctr pts to Kernal CLALL rout.
E08D:  06 B0      $B006          Vctr to Monitor Exmon entry
E08F:  6C F2      $F26C          Vector points to LOADSP entry
E091:  4E F5      $F54E          Vector points to SAVESP entry

******************************   Kernal routine: RAMTAS
                                 Clr z-page,set Memtop,Membot,
                                 RS-232 I/O buff's+cassette buff

E093:  A9 00      LDA  # $00     Init acc with $00, addr val low
E095:  A8         TAY            And copy to Y
E096:  99 02 00   STA  $0002,Y   Clear the entire zero page
E099:  C8         INY            Except for the 2 processor ports
E09A:  D0 FA      BNE  $E096     Registers $00 and $01
E09C:  A0 0B      LDY  # $0B     Set the zero-page cassette buffer
E09E:  84 B3      STY  * $B3     Pointer (z-page $B2-$B3) to the
E0A0:  85 B2      STA  * $B2     Start address $0B00
E0A2:  A0 0C      LDY  # $0C     Set the zero-page RS-232 input
E0A4:  84 C9      STY  * $C9     Buffer ptr (z-page $C8-$C9) to
E0A6:  85 C8      STA  * $C8     The start address $0C00
E0A8:  A0 0D      LDY  # $0D     Set the zero-page RS-232 output
E0AA:  84 CB      STY  * $CB     Buffer ptr (z-page $CA-$CB) to
E0AC:  85 CA      STA  * $CA     To the start address $0D00
E0AE:  18         CLC            Clear carry flag as marker
E0AF:  A0 FF      LDY  # $FF     Set top of memory
E0B1:  A2 00      LDX  # $00     In the system bank to $FF00
E0B3:  20 6B F7   JSR  $F76B     Jump to kernal rout. MEMTOP
E0B6:  A0 1C      LDY  # $1C     Set the memory bottom
E0B8:  A2 00      LDX  # $00     In the system bank to $1C00
E0BA:  20 7A F7   JSR  $F77A     Jump to kernal rout. MEMBOT
```

```
E0BD:  A0 40      LDY  # $40      Initialize the system
E0BF:  A2 00      LDX  # $00      RESTART vector at the address
E0C1:  8C 01 0A   STY  $0A01      $A00-$A01 w/ value $4000 for
E0C4:  8E 00 0A   STX  $0A00      The system cold-start entry
E0C7:  A9 A5      LDA  # $A5      Initialize the system cold-start/
E0C9:  8D 02 0A   STA  $0A02      Warm-start stat. ptr with $A5
E0CC:  60         RTS             Return from subroutine

****************************      Copy NMI, IRQ + z-pge rout's

E0CD:  A0 03      LDY  # $03      Init loop counter for four loops
E0CF:  B9 05 E1   LDA  $E105,Y    Get value from RAM bank table
E0D2:  8D 00 FF   STA  $FF00      Set corresponding configuration
E0D5:  A2 3F      LDX  # $3F      Transfer 64 bytes
E0D7:  BD 05 FF   LDA  $FF05,X    Read NMI+IRQ rout from ROM
E0DA:  9D 05 FF   STA  $FF05,X    Copy into underlying RAM
E0DD:  CA         DEX             Transfer loop counter -1
E0DE:  10 F7      BPL  $E0D7      Loop until 64 bytes transferred
E0E0:  A2 05      LDX  # $05      In the same manner the
E0E2:  BD FA FF   LDA  $FFFA,X    NMI, reset & IRQ vectors are
E0E5:  9D FA FF   STA  $FFFA,X    Copied from kernal ROM in the
E0E8:  CA         DEX             Underlying RAM, loop 'til all 3
E0E9:  10 F7      BPL  $E0E2      Vectors are transferred
E0EB:  88         DEY             Loop cntr for 4 RAM banks-1
E0EC:  10 E1      BPL  $E0CF      Copy rout.+vect. in 4 RAM bks
E0EE:  A2 59      LDX  # $59      90 bytes to transfer
E0F0:  BD 00 F8   LDA  $F800,X    Here the ROM originals of the
E0F3:  9D A2 02   STA  $02A2,X    FETCH,STASH,CMPARE,
E0F6:  CA         DEX             JSRFAR,JMPFAR routs copied
E0F7:  10 F7      BPL  $E0F0      In RAM at pages 2 and 3
E0F9:  A2 0C      LDX  # $0C      Transfer 13 bytes
E0FB:  BD 5A F8   LDA  $F85A,X    Here the original routine in ROM
E0FE:  9D F0 03   STA  $03F0,X    Is copied into the
E101:  CA         DEX             RAM area at address
E102:  10 F7      BPL  $E0FB      $03F0
E104:  60         RTS             Return from subroutine
```

```
*****************************      RAM bank table

E105:  00          .Byte  $00      RAM0,SysROM,Bs hi,Bs lo,i/o
E106:  40          .Byte  $40      RAM1,SysROM,Bs hi,Bs lo,i/o
E107:  80          .Byte  $80      RAM2,SysROM,Bs hi,Bs lo,i/o
E108:  C0          .Byte  $C0      RAM3,SysROM,Bs hi,Bs lo,i/o

*****************************      Kernal routine: IOINIT
                                   Initialization of the CIAs

E109:  A9 7F       LDA  # $7F      Load value for "clear interrupt"
E10B:  8D 0D DC    STA  $DC0D      Initialize ICR of CIA 1
E10E:  8D 0D DD    STA  $DD0D      Initialize ICR of CIA 2
E111:  8D 00 DC    STA  $DC00      Port A, CIA 1, matrix line 0
E114:  A9 08       LDA  # $08      "1 shot" initialization for timer
E116:  8D 0E DC    STA  $DC0E      CRA of CIA 1 tmr A to "1 shot"
E119:  8D 0E DD    STA  $DD0E      CRA of CIA 2 tmr A to "1 shot"
E11C:  8D 0F DC    STA  $DC0F      CRA of CIA 1 tmr B to "1 shot"
E11F:  8D 0F DD    STA  $DD0F      CRA of CIA 2 tmr B to "1 shot"
E122:  A2 00       LDX  # $00      CIA register to input mode
E124:  8E 03 DC    STX  $DC03      Data direction reg. B of CIA 1
E127:  8E 03 DD    STX  $DD03      Data direction reg. B of CIA 2
E12A:  CA          DEX             Xreg to value for "output mode"
E12B:  8E 02 DC    STX  $DC02      Data direction reg. A of CIA 1
E12E:  A9 07       LDA  # $07      Video controller to lower 16 K
E130:  8D 00 DD    STA  $DD00      ATN signal on port A, clr CIA 2
E133:  A9 3F       LDA  # $3F      Set bits 0 to 5 to output
E135:  8D 02 DD    STA  $DD02      Data direction reg A of CIA 2
E138:  A9 E3       LDA  # $E3      Initialize processor port data reg
E13A:  85 01       STA  * $01      With the default value $E3
E13C:  A9 2F       LDA  # $2F      Init. process port data dir reg
E13E:  85 00       STA  * $00      With default value $2F
E140:  A2 FF       LDX  # $FF      Initialize PAL/NTSC ptr (PAL)
E142:  AD 11 D0    LDA  $D011      Wait until MSB of the raster line
E145:  10 FB       BPL  $E142      Interrupt pointer is set
E147:  A9 08       LDA  # $08      Comp value PAL/NTSC version
E149:  CD 12 D0    CMP  $D012      Compare low byte raster intrpt
E14C:  90 06       BCC  $E154      Less than 8, then PAL version
E14E:  AD 11 D0    LDA  $D011      Wait until MSB of the raster line
E151:  30 F4       BMI  $E147      Interrupt is cleared
```

```
E153:  E8          INX            Set PAL/NTSC ptr to NTSC($0)
E154:  8E 03 0A    STX  $0A03     Store PAL/NTSC version ptr
E157:  A9 00       LDA  # $00     Init value for pointer
E159:  8D 37 0A    STA  $0A37     X-reg storage, bank operations
E15C:  8D 39 0A    STA  $0A39     80 column VDC temp storage
E15F:  8D 0A 0A    STA  $0A0A     Indirect IRQ  vector (cassette)
E162:  8D 3A 0A    STA  $0A3A     Initialize IRQ temp pointer
E165:  8D 36 0A    STA  $0A36     Raster line for raster interrupt
E168:  85 99       STA  * $99     Standard input device= keyboard
E16A:  A9 03       LDA  # $03     Set z-page storage for standard
E16C:  85 9A       STA  * $9A     Output device to 3 (=screen)
E16E:  A2 30       LDX  # $30     Transfer 49 bytes
E170:  BD C7 E2    LDA  $E2C7,X   Initialization table for VIC chip
E173:  9D 00 D0    STA  $D000,X   Copy into VIC control registers
E176:  CA          DEX            Loop/displacement counter -1
E177:  10 F7       BPL  $E170     Loop until 49 values transferred
E179:  A2 00       LDX  # $00     Set loop counter for VDC init
E17B:  20 DC E1    JSR  $E1DC     Initialize VDC registers
E17E:  AD 00 D6    LDA  $D600     Read VDC status
E181:  29 07       AND  # $07     Is bits 0-2 are cleared
E183:  F0 05       BEQ  $E18A     Yes, skip init. of VDC reg
E185:  A2 3B       LDX  # $3B     Displacement ptr to VDC table
E187:  20 DC E1    JSR  $E1DC     Initialize VDC registers
E18A:  2C 03 0A    BIT  $0A03     Check if PAL/NTSC version
E18D:  10 05       BPL  $E194     Skip, if NTSC version
E18F:  A2 3E       LDX  # $3E     Displacement ptr to VDC table
E191:  20 DC E1    JSR  $E1DC     Initialize VDC registers
E194:  AD 04 0A    LDA  $0A04     Check NMI/reset status pointer
E197:  30 15       BMI  $E1AE     VDC already init, then skip
E199:  20 27 C0    JSR  $C027     Routine INIT80: init. 80-column
E19C:  A9 80       LDA  # $80     Set bit 7 in acc,combine value
E19E:  0D 04 0A    ORA  $0A04     With the NMI/VDC status
E1A1:  8D 04 0A    STA  $0A04     And write in the status flag
E1A4:  A2 FF       LDX  # $FF     Loop counter high to high value
E1A6:  A0 FF       LDY  # $FF     Loop counter low to low value
E1A8:  88          DEY            Decrement loop counter low
E1A9:  D0 FD       BNE  $E1A8     Loop low code? No, continue
E1AB:  CA          DEX            Decrement loop counter high
E1AC:  D0 FA       BNE  $E1A8     Loop high done? No, continue
E1AE:  A9 00       LDA  # $00     Init. value for SID register
```

```
E1B0:  A2 18      LDX  # $18      SID displacement & loop pointer
E1B2:  9D 00 D4   STA  $D400,X    Clear SID register (low value)
E1B5:  CA         DEX             Loop and displ pointer -1
E1B6:  10 FA      BPL  $E1B2      Loop until 19 registers erased
E1B8:  A2 01      LDX  # $01      Load X-reg with #1
E1BA:  8E 1A D0   STX  $D01A      Set IRQ mask register
E1BD:  CA         DEX             Decrement X-reg to zero
E1BE:  8E 1C 0A   STX  $0A1C      Clear fast serial mode pointer
E1C1:  8E 0F 0A   STX  $0A0F      Clear RS-232 NMI status reg
E1C4:  CA         DEX             Set X-reg to high value ($FF)
E1C5:  8E 06 DC   STX  $DC06      Place value in timer B low
E1C8:  8E 07 DC   STX  $DC07      Place value in timer B high
E1CB:  A2 11      LDX  # $11      Code for "force load" & timer A
E1CD:  8E 0F DC   STX  $DC0F      Start in the CIA control register
E1D0:  20 C3 E5   JSR  $E5C3      Test routine, if fast serial mode
E1D3:  20 D6 E5   JSR  $E5D6      Is recognized by the disk drive
E1D6:  20 C3 E5   JSR  $E5C3      And responds to the
E1D9:  4C 4E E5   JMP  $E54E      Clock low signal and RTS

*****************************     Initialize the VDC register

E1DC:  BC F8 E2   LDY  $E2F8,X    Get register selection from table
E1DF:  30 0D      BMI  $E1EE      Check end criterium (Bit 7 = on)
E1E1:  E8         INX             Displacement to VDC table +1
E1E2:  BD F8 E2   LDA  $E2F8,X    Get register write value from tbl
E1E5:  E8         INX             Displacement to VDC table +1
E1E6:  8C 00 D6   STY  $D600      Set VDC register selection port
E1E9:  8D 01 D6   STA  $D601      Write VDC register data port
E1EC:  10 EE      BPL  $E1DC      Jump to loop start
E1EE:  E8         INX             Displacement to VDC table +1
E1EF:  60         RTS             Return from subroutine

*****************************     Check <CBM> code in RAM1

E1F0:  A2 F5      LDX  # $F5      Initialize the 2-byte zero-page
E1F2:  A0 FF      LDY  # $FF      Ptr addr $C3(lo) - $C4(hi) with
E1F4:  86 C3      STX  * $C3      The start address of the
E1F6:  84 C4      STY  * $C4      Kernal vector table ($FFF5)
E1F8:  A9 C3      LDA  # $C3      Set FETVEC for fetch routine to
E1FA:  8D AA 02   STA  $02AA      Start of the vector table
```

```
E1FD:  A0 02      LDY  # $02      Displacement for FETCH rout.
E1FF:  A2 7F      LDX  # $7F      Config. code (RAM 1 only)
E201:  20 A2 02   JSR  $02A2      FETCH rout: LDA from any bnk
E204:  D9 C4 E2   CMP  $E2C4,Y    Check for code <C> <B> <M>
E207:  D0 1B      BNE  $E224      Not equal, then exit
E209:  88         DEY             Loop until three letters checked
E20A:  10 F3      BPL  $E1FF
E20C:  A2 F8      LDX  # $F8      Initialize the 2-byte zero-page
E20E:  A0 FF      LDY  # $FF      Ptr at addrs $C3 (lo) - $C4 (hi)
E210:  86 C3      STX  * $C3      With the addr of kernal C-128
E212:  84 C4      STY  * $C4      Mode Vector ($FFF8)
E214:  A0 01      LDY  # $01      Displacement for FETCH rout
E216:  A2 7F      LDX  # $7F      Config. code (RAM1 only)
E218:  20 A2 02   JSR  $02A2      FETCH rout: LDA from any bnk
E21B:  99 02 00   STA  $0002,Y    Place entry address hi - lo in
E21E:  88         DEY             Zero-page $02-$03. Loop
E21F:  10 F5      BPL  $E216      Until both addresses transferred
E221:  6C 02 00   JMP  ($0002)    Indirect jump via zero page

******************************    Kernal vector: C128MODE

E224:  A9 40      LDA  # $40      RAM 1, enable all system ROMs
E226:  8D 00 FF   STA  $FF00      And set configuration
E229:  A9 24      LDA  # $24      Initialize the 2-byte kernal
E22B:  A0 E2      LDY  # $E2      Vector for the 128 mode with
E22D:  8D F8 FF   STA  $FFF8      The default value
E230:  8C F9 FF   STY  $FFF9      $E224
E233:  A2 03      LDX  # $03      Loop counter for 3 transfers
E235:  BD C3 E2   LDA  $E2C3,X    Load <C> <B> <M> from table
E238:  9D F4 FF   STA  $FFF4,X    And copy to the vector range of
E23B:  CA         DEX             RAM bank 1. Loop until the
E23C:  D0 F7      BNE  $E235      Three letters are transferred
E23E:  8E 00 FF   STX  $FF00      RAM 0, enable all system ROMs
E241:  60         RTS             Return from subroutine
```

```
*****************************
```
Check if EXROM input on MCR
Serves to switch modes if C64
cartridge inserted

```
E242:  AD 05 D5   LDA  $D505
E245:  29 30      AND  # $30
E247:  C9 30      CMP  # $30
E249:  F0 20      BEQ  $E26B
```
Read MCR register of the MMU
Check if bit 5 set for EXROM
input
Yes, then no 64 cartridge present

```
*****************************
```
Kernal routine: GO64MODE
Configure system as C64

```
E24B:  A9 E3      LDA  # $E3
E24D:  85 01      STA  * $01
E24F:  A9 2F      LDA  # $2F
E251:  85 00      STA  * $00
E253:  A2 08      LDX  # $08
E255:  BD 62 E2   LDA  $E262,X
E258:  95 01      STA  * $01,X
E25A:  CA         DEX
E25B:  D0 F8      BNE  $E255
E25D:  8E 30 D0   STX  $D030
E260:  4C 02 00   JMP  $0002
```
C-64 system values in
Data register processor port
C-64 system values in
Data direction reg processor port
8 bytes to be copied
Here the ROM original of the
Routine, which configures C-64
Is copied into zero page because
The routine can run only there
Set clock frequency to 1 MHz
To zero-page rout: config. C-64

```
*****************************
```
This routine configures C-128
as a C-64. It can run only in the
zero page because all other areas
are switched off.

```
E263:  A9 F7      LDA  # $F7
E265:  8D 05 D5   STA  $D505
E268:  6C FC FF   JMP  ($FFFC)
```
Write init value for C-64 system
In the MCR register of the MMU
Jump to RESET vector C-64

```
*****************************
```
Function ROM test C-128 mode

```
E26B:  A2 03      LDX  # $03
E26D:  8E C0 0A   STX  $0AC0
E270:  A9 00      LDA  # $00
E272:  9D C1 0A   STA  $0AC1,X
E275:  CA         DEX
```
Initialize loop and displ counter
For cartridge test
The first 4 bytes of the PAT
(Physical address table of the
Expansion card) are cleared here

```
E276:  10 FA      BPL  $E272      ($00 initialized)
E278:  85 9E      STA  * $9E      Low addr value for cartridge test
E27A:  A0 09      LDY  # $09      Displacement to cart code(CBM)
E27C:  AE C0 0A   LDX  $0AC0      Displacement cntr for cart check
E27F:  BD BC E2   LDA  $E2BC,X    Get high addr value from table
E282:  85 9F      STA  * $9F      And place it in zero page
E284:  BD C0 E2   LDA  $E2C0,X    Get bank val. for test from table
E287:  85 02      STA  * $02      And place it in  z-page bank byte
E289:  A6 02      LDX  * $02      Get bank code from zero page
E28B:  A9 9E      LDA  # $9E      Get addr $9E as VETVEC in acc
E28D:  20 D0 F7   JSR  $F7D0      INDFET:LDA(fetvec),Y any bk
E290:  D9 BD E2   CMP  $E2BD,Y    Test 1 character for "CBM" code
E293:  D0 21      BNE  $E2B6      Not equal, next bank/address
E295:  88         DEY             Continue test for "CBM" code
E296:  C0 07      CPY  # $07      If 3 code chars are recognized
E298:  B0 EF      BCS  $E289      Then continue, else in test loop
E29A:  A6 02      LDX  * $02      Get bank code of current test
E29C:  A9 9E      LDA  # $9E      Get addr $9E as FETVEC in acc
E29E:  20 D0 F7   JSR  $F7D0      INDFET: LDA(fetvec),Y any bk
E2A1:  AE C0 0A   LDX  $0AC0      Get F ROM displacement pointer
E2A4:  9D C1 0A   STA  $0AC1,X    ID table of expansion card
E2A7:  C9 01      CMP  # $01      Check expansion indicated
E2A9:  D0 0B      BNE  $E2B6      No, then skip to next test
E2AB:  A5 9E      LDA  * $9E      Low of entry address in acc
E2AD:  A4 9F      LDY  * $9F      High of entry address in Y-reg
E2AF:  85 04      STA  * $04      Low of entry address in PC low
E2B1:  84 03      STY  * $03      High of entry address in PC hi
E2B3:  20 CD 02   JSR  $02CD      JSRFAR: JSR to any bk+RTS
E2B6:  CE C0 0A   DEC  $0AC0      Loop/displacement counter -1
E2B9:  10 BF      BPL  $E27A      Not zero, then continue test
E2BB:  60         RTS             Return from subroutine

****************************      High addresses for cartridge test

E2BC:  C0 80 C0 80                $C000, $8000, $C000, $8000
```

```
*****************************      Bank numbers for cartridge test

E2C0:  04 04 08 08                 inROM,inROM,exROM,exROM

*****************************      Code for cartridge indication

E2C4:  43 42 4D                    <C> <B> <M>

*****************************      Intialization table for VIC regs

E2C7:  00 00 00 00 00 00 00 00
E2CF:  00 00 00 00 00 00 00 00
E2D7:  00 1B FF 00 00 00 08 00
E2DF:  14 FF 01 00 00 00 00 00
E2E7:  0D 0B 01 02 03 01 02 00
E2EF:  01 02 03 04 05 06 07 FF
E2F7:  FC

*****************************      Initialization table for VDC regs

E2F8:  00 7E 01 50 02 66 03 49     VDC tab 1
E300:  04 20 05 00 06 19 07 1D
E308:  08 00 09 07 0A 20 0B 07
E310:  0C 00 0D 00 0E 00 0F 00
E318:  14 08 15 00 17 08 18 20
E320:  19 40 1A F0 1B 00 1C 20
E328:  1D 07 22 7D 23 64 24 05
E330:  16 78
E332:  FF          .Byte  $FF      Separator
E333:  19 47                       VDC tab 2
E335:  FF          .Byte  $FF      Separator
E336:  04 27 07 20                 VDC tab 3
E33A:  FF          .Byte  $FF      Separator

*****************************      Kernal routine: TALK

E33B:  09 40       ORA  # $40      Set bit 6 for TALK
E33D:  2C          .Byte  $2C      Skip to $E340
E33E:  09 20       ORA  # $20      Set bit 5 for listen
E340:  20 EC E7    JSR  $E7EC      Wait for end of RS-232 transfer
```

```
*****************************
```

Kernal routine: LISTN

```
E343:  48          PHA
E344:  24 94       BIT  * $94
E346:  10 0A       BPL  $E352
E348:  38          SEC
E349:  66 A3       ROR  * $A3
E34B:  20 8C E3    JSR  $E38C
E34E:  46 94       LSR  * $94
E350:  46 A3       LSR  * $A3
E352:  68          PLA
E353:  85 95       STA  * $95
E355:  20 73 E5    JSR  $E573
E358:  20 57 E5    JSR  $E557
E35B:  AD 00 DD    LDA  $DD00
E35E:  29 08       AND  # $08
E360:  D0 12       BNE  $E374
E362:  20 D6 E5    JSR  $E5D6
E365:  A9 FF       LDA  # $FF
E367:  8D 0C DC    STA  $DC0C
E36A:  20 BC E5    JSR  $E5BC
E36D:  8A          TXA
E36E:  A2 14       LDX  # $14
E370:  CA          DEX
E371:  D0 FD       BNE  $E370
E373:  AA          TAX
E374:  AD 00 DD    LDA  $DD00
E377:  09 08       ORA  # $08
E379:  8D 00 DD    STA  $DD00
E37C:  20 73 E5    JSR  $E573
E37F:  20 4E E5    JSR  $E54E
E382:  20 57 E5    JSR  $E557
```

Save Talk/Listn marker on stack
Another byte to output?
No, then continue
Set carry for rotation
Set flag for EOI
Output byte to serial bus
Erase character in buffer marker
Clear flag for EOI again
Get old acc contents back
Store byte to output in zero page
SEI, 1 MHz, turn sprites off
Output data high
Check if the ATN signal is set
On data port A of CIA 2
Not set, then skip
Pulse for fast serial mode
I/O data buffer for serial
Set transfer to high value
Wait for response from bus
Store X-reg contents in acc
Set loop counter to 20
Decrement loop counter by 1
Wait until loop counted down
Recreate old X-reg contents
Read port A of CIA 2
Set ATN lo signal & write back
to Port A of CIA 2
Clk freq. 1MHz, turn sprites off
Output clock low
Output data high

```
*****************************   Delay loop about 1 millisecond

E385:  8A          TXA          Store X-reg contents in acc
E386:  A2 B8       LDX  # $B8   Set loop counter to 184
E388:  CA          DEX          Decrement loop counter by 1
E389:  D0 FD       BNE  $E388   Loop until counter = 0
E38B:  AA          TAX          Restore X-reg contents

*****************************   Byte on serial bus (prepare)

E38C:  20 73 E5    JSR  $E573   Clock freq. 1 MHZ, sprites off
E38F:  20 57 E5    JSR  $E557   Output data high
E392:  20 69 E5    JSR  $E569   Get bit from serial bus into carry
E395:  90 03       BCC  $E39A   Data not low, then OK and skip
E397:  4C 28 E4    JMP  $E428   "Device not present" - sys status
E39A:  2C 0D DC    BIT  $DC0D   Test CIA interrupt control reg.
E39D:  20 45 E5    JSR  $E545   Output clock high
E3A0:  24 A3       BIT  * $A3   Zero-page pointer for EOI set?
E3A2:  10 0A       BPL  $E3AE   No, then skip
E3A4:  20 69 E5    JSR  $E569   Get bit from serial bus into carry
E3A7:  90 FB       BCC  $E3A4   Wait for data low signal
E3A9:  20 69 E5    JSR  $E569   Get bit from serial bus into carry
E3AC:  B0 FB       BCS  $E3A9   Wait for data high signal
E3AE:  AD 00 DD    LDA  $DD00   Here data is read from port A
E3B1:  CD 00 DD    CMP  $DD00   Of CIA 2
E3B4:  D0 F8       BNE  $E3AE
E3B6:  48          PHA          Data read are stored on the stack
E3B7:  AD 0D DC    LDA  $DC0D   Check interrupt control register
E3BA:  29 08       AND  # $08   Is timer A on "one shot"?
E3BC:  F0 05       BEQ  $E3C3   Yes, then skip
E3BE:  A9 C0       LDA  # $C0   Set Control bits 6 and 7 in sys-
E3C0:  8D 1C 0A    STA  $0A1C   tem pointer for fast serial mode
E3C3:  68          PLA          Get data read back from stack
E3C4:  10 E8       BPL  $E3AE   Bit 7 cleared, then skip
E3C6:  09 10       ORA  # $10   Set bit 4 for clk output on serial
E3C8:  8D 00 DD    STA  $DD00   bus and write in port A
E3CB:  29 08       AND  # $08   Check if bit 3 is set
E3CD:  D0 13       BNE  $E3E2   No, then skip
E3CF:  2C 1C 0A    BIT  $0A1C   Check bit 7, serial mode pointer
E3D2:  10 0E       BPL  $E3E2   Bit 7 cleared, then skip
```

```
E3D4:  20 D6 E5   JSR  $E5D6     Impulse for fast serial mode
E3D7:  A5 95      LDA  * $95     Get stored byte and write in
E3D9:  8D 0C DC   STA  $DC0C     CIA input/output register
E3DC:  20 BC E5   JSR  $E5BC     Wait for response from bus
E3DF:  4C 12 E4   JMP  $E412     Byte output over serial bus

****************************     Byte on serial bus (output)

E3E2:  A9 08      LDA  # $08     Initialize counter for number of
E3E4:  85 A5      STA  * $A5     Bits to send with 8
E3E6:  AD 00 DD   LDA  $DD00     Here data is read from port A
E3E9:  CD 00 DD   CMP  $DD00     Of CIA 2
E3EC:  D0 F8      BNE  $E3E6
E3EE:  0A         ASL  A         Data shifted into the carry flag
E3EF:  90 34      BCC  $E425     Output data high,output timeout
E3F1:  66 95      ROR  * $95     Prepare bit for output
E3F3:  B0 05      BCS  $E3FA     Check if bit is set
E3F5:  20 60 E5   JSR  $E560     No, then output data low
E3F8:  D0 03      BNE  $E3FD     And jump to clock high output
E3FA:  20 57 E5   JSR  $E557     Output data high
E3FD:  20 45 E5   JSR  $E545     Output clock high
E400:  EA         NOP            No Operation
E401:  EA         NOP            No Operation
E402:  EA         NOP            No Operation
E403:  EA         NOP            No Operation
E404:  AD 00 DD   LDA  $DD00     Read port A of CIA 2
E407:  29 DF      AND  # $DF     Bit 5:Clear data output serial bus
E409:  09 10      ORA  # $10     Bit 4:Set clock output serial bus
E40B:  8D 00 DD   STA  $DD00     Write in data port A
E40E:  C6 A5      DEC  * $A5     Decrement bit counter by 1
E410:  D0 D4      BNE  $E3E6     Output additional bit, then loop
E412:  8A         TXA            Copy contents of X-reg to acc
E413:  48         PHA            And store X-reg on stack
E414:  A2 22      LDX  # $22     High impulse counter to #34
E416:  20 69 E5   JSR  $E569     Get 1 bit from serial bus to carry
E419:  B0 05      BCS  $E420     Data high, then skip
E41B:  68         PLA            Get old X-reg contents from
E41C:  AA         TAX            stack and restore
E41D:  4C 9F E5   JMP  $E59F     Reset clock freq. and sprites
E420:  CA         DEX            Decrement data high counter
```

```
E421:  D0 F3      BNE  $E416     Not yet 22 high pulses, continue
E423:  68         PLA            Get old X-reg contents f/ stack
E424:  AA         TAX            Restore X-reg contents
E425:  A9 03      LDA  # $03     Code for system status: Time out
E427:  2C         .Byte  $2C     Skip to $E42A
E428:  A9 80      LDA  # $80     Code status:Device not present
E42A:  48         PHA            Store status code on stack
E42B:  AD 1C 0A   LDA  $0A1C     Test the fast serial mode pointer
E42E:  29 7F      AND  # $7F     Mask out bit 7, only fast/slow
E430:  8D 1C 0A   STA  $0A1C     Write in fast-mode flag
E433:  68         PLA            Get status error code
E434:  20 57 F7   JSR  $F757     Set new system status
E437:  20 9F E5   JSR  $E59F     Reset clock freq. and sprites
E43A:  18         CLC            Set indicator for OK
E43B:  4C 35 E5   JMP  $E535     Turn off device with Unlsn

******************************   Kernal routine: ACPTR
                                 Get byte from serial bus

E43E:  20 73 E5   JSR  $E573     System clk 1MHz, sprites off
E441:  A9 00      LDA  # $00     Clear the zero-page ptr for the
E443:  85 A5      STA  * $A5     serial EOI indicator
E445:  2C 0D DC   BIT  $DC0D     Read bit 7 of the CIA ISR
E448:  8A         TXA            Store current cont of the X-reg
E449:  48         PHA            Via the acc on the stack
E44A:  20 45 E5   JSR  $E545     Clock signal on port A
E44D:  20 69 E5   JSR  $E569     Get bit from serial bus into carry
E450:  10 FB      BPL  $E44D     Wait for data high signal
E452:  A2 0D      LDX  # $0D     Initialize loop counter with #13
E454:  AD 00 DD   LDA  $DD00     Read data port A of CIA 2
E457:  29 DF      AND  # $DF     Bit 6: clear "serial bus pulse on"
E459:  8D 00 DD   STA  $DD00     And write in data port
E45C:  AD 00 DD   LDA  $DD00     Read data port A of CIA 2 and
E45F:  CD 00 DD   CMP  $DD00     A bit arrives over the bus
E462:  D0 F8      BNE  $E45C     On the port
E464:  0A         ASL  A         Shift data bit into the carry flag
E465:  10 1D      BPL  $E484     Get data byte from bus
E467:  CA         DEX            Decrement loop counter by 1
E468:  D0 F2      BNE  $E45C     Loop not zero, then skip
E46A:  A5 A5      LDA  * $A5     Test zero-page EOI pointer
```

```
E46C:  D0 0F     BNE  $E47D   For #0, EOI received, else skip
E46E:  20 60 E5  JSR  $E560   Data low signal on serial bus
E471:  20 45 E5  JSR  $E545   Clock high signal on serial bus
E474:  A9 40     LDA  # $40   Code for status: EOI line
E476:  20 57 F7  JSR  $F757   Reset system status
E479:  E6 A5     INC  * $A5   EOI pnter to time error if timeout
E47B:  D0 D5     BNE  $E452   Get data byte to EOI
E47D:  68        PLA          Restore stored X-reg contents
E47E:  AA        TAX          via the acc from the stack
E47F:  A9 02     LDA  # $02   Code status: timeout for reading
E481:  4C 2A E4  JMP  $E42A   Reset system status
E484:  A2 08     LDX  # $08   Set counter for 8 data bits
E486:  AD 0D DC  LDA  $DC0D   Read interrupt control register
E489:  29 08     AND  # $08   Test if timer, clock, or bus
E48B:  D0 28     BNE  $E4B5   Interrupt. Yes, then skip
E48D:  AD 00 DD  LDA  $DD00   Read data port A of CIA 2 and
E490:  CD 00 DD  CMP  $DD00   Wait until a bit arrives over
E493:  D0 F8     BNE  $E48D   The port
E495:  0A        ASL  A       Shift data bit into the carry
E496:  10 EE     BPL  $E486   No, wait until data are valid
E498:  66 A4     ROR  * $A4   Data bit in bit storage
E49A:  AD 00 DD  LDA  $DD00   Read data port of CIA 2 and
E49D:  CD 00 DD  CMP  $DD00   Wait until a bit arrives over
E4A0:  D0 F8     BNE  $E49A   The port
E4A2:  0A        ASL  A       Shift data bit into the carry flag
E4A3:  30 F5     BMI  $E49A   No, then wait
E4A5:  CA        DEX          Counter for data bit number -1
E4A6:  F0 17     BEQ  $E4BF   8 data bits arrived, then skip
E4A8:  AD 00 DD  LDA  $DD00   Read data port A of CIA 2 and
E4AB:  CD 00 DD  CMP  $DD00   Wait until a bit arrives over
E4AE:  D0 F8     BNE  $E4A8   The port
E4B0:  0A        ASL  A       Shift data bit into the carry flag
E4B1:  10 F5     BPL  $E4A8   Jump if bit received is "0"
E4B3:  30 E3     BMI  $E498   Jump if bit received is "1"
E4B5:  AD 0C DC  LDA  $DC0C   Store contents of I/O data buffer
E4B8:  85 A4     STA  * $A4   In the zero page
E4BA:  A9 C0     LDA  # $C0   Set bits 6 and 7 in the sys flag
E4BC:  8D 1C 0A  STA  $0A1C   For the fast serial mode
E4BF:  68        PLA          Restore old X-reg contents via
E4C0:  AA        TAX          The acc from the stack
```

```
E4C1:  20 60 E5   JSR  $E560      Data low signal on serial bus
E4C4:  24 90      BIT  * $90      Test STATUS for set EOI bit
E4C6:  50 03      BVC  $E4CB      No EOF found, then continue
E4C8:  20 38 E5   JSR  $E538      Shut device off - Unlsn routine
E4CB:  20 9F E5   JSR  $E59F      Reset clock freq and sprites
E4CE:  A5 A4      LDA  * $A4      Get data byte in the accumulator
E4D0:  18         CLC             Set indicator for OK
E4D1:  60         RTS             Return from subroutine

******************************    Kernal routine: SECND
                                  Send sec. address after LISTEN

E4D2:  85 95      STA  * $95      Store sec. address in zero page
E4D4:  20 7C E3   JSR  $E37C      Output with attention (ATN)
E4D7:  AD 00 DD   LDA  $DD00      Read data port A of CIA 2
E4DA:  29 F7      AND  # $F7      Mask out bit 3 and take the ATN
E4DC:  8D 00 DD   STA  $DD00      Signal back to the serial bus
E4DF:  60         RTS             Return from  subroutine

******************************    Kernal routine: TKSA

E4E0:  85 95      STA  * $95      Store secondary add in zero page
E4E2:  20 7C E3   JSR  $E37C      Output with attention (ATN)
E4E5:  24 90      BIT  * $90      Test STATUS for set EOF bit
E4E7:  30 4C      BMI  $E535      EOF encounter, to Unlsn routine
E4E9:  20 73 E5   JSR  $E573      Clock freq1MHz, sprites off
E4EC:  20 60 E5   JSR  $E560      Data low signal on serial bus
E4EF:  20 D7 E4   JSR  $E4D7      Entry in SECND routine
E4F2:  20 45 E5   JSR  $E545      Clock high signal on serial bus
E4F5:  AD 00 DD   LDA  $DD00      Read data port A of CIA 2 and
E4F8:  CD 00 DD   CMP  $DD00      Wait until a bit arrives over the
E4FB:  D0 F8      BNE  $E4F5      Port
E4FD:  0A         ASL  A          Shift data bit into the carry
E4FE:  30 F5      BMI  $E4F5      And wait for data high
E500:  4C 9F E5   JMP  $E59F      Reset clock freq and sprites
```

```
*****************************   Kernal routine: CIOUT

E503:  24 94      BIT  * $94    Output another byte?
E505:  30 05      BMI  $E50C    Yes, then to output loop
E507:  38         SEC           Set carry for rotation
E508:  66 94      ROR  * $94    Set flag for buffered byte
E50A:  D0 05      BNE  $E511    Skip output loop
E50C:  48         PHA           Save the byte on the stack
E50D:  20 8C E3   JSR  $E38C    Output buffered byte on stack
E510:  68         PLA           Get byte from the stack
E511:  85 95      STA  * $95    Place in zero-page output storage
E513:  18         CLC           Carry set for "OK" indicator
E514:  60         RTS           Return from subroutine

*****************************   Kernal routine: UNTLK

E515:  20 73 E5   JSR  $E573    Reset clock frequency
E518:  20 4E E5   JSR  $E54E    Clock low signal to port A
E51B:  AD 00 DD   LDA  $DD00    Read data port A of CIA 2
E51E:  09 08      ORA  # $08    Set bit 3 in this value and
E520:  8D 00 DD   STA  $DD00    Output ATN lo signal on the bus
E523:  A9 5F      LDA  # $5F    Load code for UNTLK in acc
E525:  2C         .Byte  $2C    Skip to $E528

*****************************   Kernal routine: UNLSN

E526:  A9 3F      LDA  # $3F    Load code for UNLSN in acc
E528:  48         PHA           And store on stack
E529:  AD 1C 0A   LDA  $0A1C    Status pointer for "fast serial"
E52C:  29 7F      AND  # $7F    Mask out bit 7
E52E:  8D 1C 0A   STA  $0A1C    And write back
E531:  68         PLA           Restore old acc contents
E532:  20 43 E3   JSR  $E343    Kernal routine: LISTN
E535:  20 D7 E4   JSR  $E4D7    Reset ATN, high
E538:  8A         TXA           Store X-reg contents in acc
E539:  A2 0A      LDX  # $0A    Time loop for 40 microseconds
E53B:  CA         DEX           Decrement loop counter by 1
E53C:  D0 FD      BNE  $E53B    Wait until loop processed
E53E:  AA         TAX           Restore old X-reg contents
E53F:  20 45 E5   JSR  $E545    Clock high signal on port A
```

```
E542:  4C 57 E5   JMP  $E557      Data high signal on port A

******************************    Clock high signal

E545:  AD 00 DD   LDA  $DD00      Read data port A of CIA 2
E548:  29 EF      AND  # $EF      Clear bit 4 for clock output on
E54A:  8D 00 DD   STA  $DD00      serial bus and write in port A
E54D:  60         RTS             Return from subroutine

******************************    Clock low signal

E54E:  AD 00 DD   LDA  $DD00      Read data port A of CIA 2
E551:  09 10      ORA  # $10      Set bit 4 for clock output on
E553:  8D 00 DD   STA  $DD00      serial bus and write in port A
E556:  60         RTS             Return from subroutine

******************************    Data high signal

E557:  AD 00 DD   LDA  $DD00      Read data port A of CIA 2
E55A:  29 DF      AND  # $DF      Clear bit 5 for data output on
E55C:  8D 00 DD   STA  $DD00      serial bus and write in port A
E55F:  60         RTS             Return from subroutine

******************************    Data Lo Signal

E560:  AD 00 DD   LDA  $DD00      Read data port A of CIA 2
E563:  09 20      ORA  # $20      Set bit 5 for data output on serial
E565:  8D 00 DD   STA  $DD00      Bus and write in port A
E568:  60         RTS             Return from subroutine

******************************    Get a bit from serial bus to carry

E569:  AD 00 DD   LDA  $DD00      Read data port A of CIA 2 and
E56C:  CD 00 DD   CMP  $DD00      Wait until a bit arrives over
E56F:  D0 F8      BNE  $E569      The port
E571:  0A         ASL  A          Bit received (bit 7) into carry
E572:  60         RTS             Return from subroutine
```

```
******************************   Set system clock freq. to 1MHz
                                 And turn all sprites off
                .

E573:  78          SEI           Disable all system interrupts
E574:  2C 3A 0A    BIT  $0A3A    Test interrupt storage
E577:  30 25       BMI  $E59E    Bit 7 set, then return
E579:  2C 37 0A    BIT  $0A37    Check clock frequency
E57C:  30 20       BMI  $E59E    Bit 7 set, then return
E57E:  AD 30 D0    LDA  $D030    VIC register for clock frequency
E581:  8D 37 0A    STA  $0A37    Save in system storage
E584:  AD 15 D0    LDA  $D015    Enable VIC registers for sprites
E587:  8D 38 0A    STA  $0A38    Save in system storage
E58A:  A9 00       LDA  # $00    Init. status for 1 MHz, no sprites
E58C:  8D 15 D0    STA  $D015    Turn all sprites off
E58F:  8D 30 D0    STA  $D030    Set clock frequency to 1 MHz
E592:  AD 38 0A    LDA  $0A38    Were sprites on?
E595:  F0 07       BEQ  $E59E    No, then return
E597:  8A          TXA           Store X-reg contents in acc
E598:  A2 00       LDX  # $00    Delay loop for 1.3 milliseconds
E59A:  CA          DEX           Decrement loop counter by 1
E59B:  D0 FD       BNE  $E59A    Process entire delay loop
E59D:  AA          TAX           Restore old X-reg contents
E59E:  60          RTS           Return from subroutine

******************************   Reset clock frequency and sprite
                                 pointers to their original status

E59F:  2C 3A 0A    BIT  $0A3A    Test interrupt storage
E5A2:  30 16       BMI  $E5BA    Bit 7 set, then return
E5A4:  2C 37 0A    BIT  $0A37    Check clock frequency storage
E5A7:  10 11       BPL  $E5BA    Frequency not changed, skip
E5A9:  AD 38 0A    LDA  $0A38    Write the stored value of sprite
E5AC:  8D 15 D0    STA  $D015    Enable register back
E5AF:  AD 37 0A    LDA  $0A37    Write the stored value of system
E5B2:  8D 30 D0    STA  $D030    Clock frequency back
E5B5:  A9 00       LDA  # $00    Clear temp storage for
E5B7:  8D 37 0A    STA  $0A37    System clock frequency
E5BA:  58          CLI           Enable all system interrupts
E5BB:  60          RTS           Return from subroutine
```

```
*****************************      Wait for response from bus

E5BC:  AD 0D DC   LDA  $DC0D       Get CIA interrupt control reg.
E5BF:  29 08      AND  # $08       Wait until bit 4 (SRQ input from
E5C1:  F0 F9      BEQ  $E5BC       Serial bus) is cleared
E5C3:  AD 0E DC   LDA  $DC0E       Read control register A of CIA
E5C6:  29 80      AND  # $80       Eliminate bit 7 for 50 Hz freq.
E5C8:  09 08      ORA  # $08       Set timer to mode toggle and
E5CA:  8D 0E DC   STA  $DC0E       "One shot" and start timer
E5CD:  AD 05 D5   LDA  $D505       Mask out the control bit for fast
E5D0:  29 F7      AND  # $F7       Serial mode in mode config. reg
E5D2:  8D 05 D5   STA  $D505       Of the MMU
E5D5:  60         RTS              Return from subroutine

*****************************      Fast pulse on serial bus

E5D6:  AD 05 D5   LDA  $D505       Set the control bit for the fast
E5D9:  09 08      ORA  # $08       Serial mode in mode config reg
E5DB:  8D 05 D5   STA  $D505       Of the MMU
E5DE:  A9 7F      LDA  # $7F       Clear code for interrupt
E5E0:  8D 0D DC   STA  $DC0D       To interrupt control register
E5E3:  A9 00      LDA  # $00       Load timer A high in CIA 2 with
E5E5:  8D 05 DC   STA  $DC05       the high value #0
E5E8:  A9 04      LDA  # $04       Load timer A low in CIA 2 with
E5EA:  8D 04 DC   STA  $DC04       the low value #4
E5ED:  AD 0E DC   LDA  $DC0E       Read control register A of CIA
E5F0:  29 80      AND  # $80       Eliminate bit 7 for 50 HZ freq
E5F2:  09 55      ORA  # $55       Set timer to force load, toggle
E5F4:  8D 0E DC   STA  $DC0E       Serial bus off and start timer A
E5F7:  2C 0D DC   BIT  $DC0D       Read interrupt control register
E5FA:  60         RTS              Return from subroutine

*****************************      Kernal routine: FSTMOD

E5FB:  90 C6      BCC  $E5C3       Wait - response from serial bus
E5FD:  B0 D7      BCS  $E5D6       Fast pulse on serial bus
```

```
*****************************   RS-232 output

E5FF:  A5 B4      LDA  * $B4    Number of bits to send
E601:  F0 47      BEQ  $E64A    Is byte completely transferred?
E603:  30 3F      BMI  $E644    Is stop bit required?
E605:  46 B6      LSR  * $B6    Shift next bit into carry
E607:  A2 00      LDX  # $00    Initialize X-reg as ind. with $00
E609:  90 01      BCC  $E60C    Bit cleared?
E60B:  CA         DEX           No, then set X-reg to $FF
E60C:  8A         TXA           Copy bit cleared indicator to acc
E60D:  45 BD      EOR  * $BD    Combine with parity status
E60F:  85 BD      STA  * $BD    Save again in zero-page parity
E611:  C6 B4      DEC  * $B4    Decrement bit counter by 1
E613:  F0 06      BEQ  $E61B    All bits transferred, continue
E615:  8A         TXA           Copy X-reg contents into acc
E616:  29 04      AND  # $04    Isolate bit 2
E618:  85 B5      STA  * $B5    And put in output register
E61A:  60         RTS           Return from subroutine

*****************************   Check transmit parity

E61B:  A9 20      LDA  # $20    Set bit 5 in acc for parity
E61D:  2C 11 0A   BIT  $0A11    Check RS-232 command reg.
E620:  F0 14      BEQ  $E636    Op. mode without parity, skip
E622:  30 1C      BMI  $E640    Op. mode with set parity?
E624:  70 14      BVS  $E63A    Op. mode for uneven parity?
E626:  A5 BD      LDA  * $BD    Parity equal one?
E628:  D0 01      BNE  $E62B    No, then skip
E62A:  CA         DEX           Set parity to $FF
E62B:  C6 B4      DEC  * $B4    Set bit counter to $FF
E62D:  AD 10 0A   LDA  $0A10    Get RS-232 control reg. in acc
E630:  10 E3      BPL  $E615    Are two stop bits required?
E632:  C6 B4      DEC  * $B4    Set bit counter to $FE
E634:  D0 DF      BNE  $E615    Not zero, calculate stop bits
E636:  E6 B4      INC  * $B4    Bit counter +1, no parity
E638:  D0 F0      BNE  $E62A    Not zero, calculate stop bits
E63A:  A5 BD      LDA  * $BD    Get parity value from zero page
E63C:  F0 ED      BEQ  $E62B    Output a zero bit for 0
E63E:  D0 EA      BNE  $E62A    Not zero, then output 1-bit
E640:  70 E9      BVS  $E62B    Routine: output 0-bit
```

```
E642:  50 E6       BVC  $E62A      Routine: output 1-bit-fixed parity
E644:  E6 B4       INC  * $B4      Increment bit counter by 1
E646:  A2 FF       LDX  # $FF      Put code value- stop bit in X-reg
E648:  D0 CB       BNE  $E615      Unconditional jump

******************************     3-line / X-line handshake test

E64A:  AD 11 0A    LDA  $0A11      Load acc with RS-232 cmd reg
E64D:  4A          LSR  A          Shift bit 0 into carry flag
E64E:  90 07       BCC  $E657      Skip 3-line handshake read
E650:  2C 01 DD    BIT  $DD01      Read port B of CIA 2
E653:  10 1D       BPL  $E672      Is DATA SET READY (DSR)
                                   signal missing
E655:  50 1E       BVC  $E675      Is CLEAR TO SEND (CTS)
                                   signal missing?
E657:  A9 00       LDA  # $00      Clear z-page buffer for RS-232
E659:  85 BD       STA  * $BD      Parity ($00) and the zero page
E65B:  85 B5       STA  * $B5      Storage for the start bit to send
E65D:  AE 15 0A    LDX  $0A15      Copy number of bits to transfer
E660:  86 B4       STX  * $B4      Into zero-page as counter
E662:  AC 1A 0A    LDY  $0A1A      Comp index to start of output
E665:  CC 1B 0A    CPY  $0A1B      buffer with end. If all bytes are
E668:  F0 13       BEQ  $E67D      transferred then done.
E66A:  B1 CA       LDA  ($CA),Y    Get data byte from RS-232
E66C:  85 B6       STA  * $B6      buffer and pass in  storage
E66E:  EE 1A 0A    INC  $0A1A      Index: incr start of output buffer
E671:  60          RTS             Return from subroutine

******************************     Set NMI status for RS-232

E672:  A9 40       LDA  # $40      Code for DATA SET READY
                                   (DSR) missing
E674:  2C          .Byte  $2C      Skip to $E677
E675:  A9 10       LDA  # $10      Code for CLEAR TO SEND
                                   (CTS) missing
E677:  0D 14 0A    ORA  $0A14      Combine with RS-232 status reg
E67A:  8D 14 0A    STA  $0A14      And put in status register
E67D:  A9 01       LDA  # $01      Load acc with $01 and clear the
E67F:  8D 0D DD    STA  $DD0D      NMI for timer A
E682:  4D 0F 0A    EOR  $0A0F      Combine w/ RS-232 NMI status
```

```
E685:  09 80      ORA  # $80      Reverse flag for RS-232 & place
E687:  8D 0F 0A   STA  $0A0F      value in the RS-232 NMI status
E68A:  8D 0D DD   STA  $DD0D      Allow all further NMIs
E68D:  60         RTS             Return from subroutine

****************************      Calculate num. RS-232 data bits

E68E:  A2 09      LDX  # $09      Default value to 8 data bits
E690:  A9 20      LDA  # $20      Check value for num of data bits
E692:  2C 10 0A   BIT  $0A10      Check RS-232 control register
E695:  F0 01      BEQ  $E698      Bit 5 cleared?
E697:  CA         DEX             Decrement number of data bits
E698:  50 02      BVC  $E69C      Bit 6 cleared?
E69A:  CA         DEX             Decrement number of data bits
E69B:  CA         DEX             Decrement number of data bits
E69C:  60         RTS             Return from subroutine

****************************      Process bit received

E69D:  A6 A9      LDX  * $A9      Check if it is a start bit
E69F:  D0 33      BNE  $E6D4      No, skip
E6A1:  C6 A8      DEC  * $A8      Decrement bit counter by 1
E6A3:  F0 3A      BEQ  $E6DF      All bits received, then continue
E6A5:  30 0D      BMI  $E6B4      If stop bits expected, then skip
E6A7:  A5 A7      LDA  * $A7      Get received bit in acc
E6A9:  45 AB      EOR  * $AB      And combine for parity
E6AB:  85 AB      STA  * $AB      Place parity value in zero page
E6AD:  46 A7      LSR  * $A7      Shift received bit into carry flag
E6AF:  66 AA      ROR  * $AA      And in input buffer
E6B1:  60         RTS             Return from subroutine

****************************      Set start bit pointer when all
                                  stop bits have been received

E6B2:  C6 A8      DEC  * $A8      Decrement bit counter by 1
E6B4:  A5 A7      LDA  * $A7      Get stop bit value in acc and
E6B6:  F0 6B      BEQ  $E723      Check if it is zero. Skip
E6B8:  AD 10 0A   LDA  $0A10      RS-232 control register in acc
E6BB:  0A         ASL  A          Number of stop bits into carry
E6BC:  A9 01      LDA  # $01      Addition value num of stop bits
```

```
E6BE:  65 A8      ADC  * $A8      Add data bits and stop bits
E6C0:  D0 EF      BNE  $E6B1      Not all stop bits received, skip
E6C2:  A9 90      LDA  # $90      RXD over flag received in acc
E6C4:  8D 0D DD   STA  $DD0D      And enable NMI
E6C7:  0D 0F 0A   ORA  $0A0F      Combine w/ RS-232 NMI status
E6CA:  8D 0F 0A   STA  $0A0F      And place in RS-232 NMI status
E6CD:  85 A9      STA  * $A9      Set flag for start bit
E6CF:  A9 02      LDA  # $02      Init. acc with 2 for transmission
E6D1:  4C 7F E6   JMP  $E67F      And clear the NMI for timer B

****************************      Check for RS-232 start bit

E6D4:  A5 A7      LDA  * $A7      Get start bit value in acc
E6D6:  D0 EA      BNE  $E6C2      Not zero, skip. Else reset the
E6D8:  85 A9      STA  * $A9      Zero-page start bit flag and reset
E6DA:  A9 01      LDA  # $01      the zero-page ptr for RS-232
E6DC:  85 AB      STA  * $AB      Reset input parity
E6DE:  60         RTS             Return from subroutine

****************************      Process received byte

E6DF:  AC 18 0A   LDY  $0A18      Index to the start of RS-232
E6E2:  C8         INY             Increment input buffer by 1
E6E3:  CC 19 0A   CPY  $0A19      Compare with end. If buffer,
E6E6:  F0 2A      BEQ  $E712      Then set appropriate status
E6E8:  8C 18 0A   STY  $0A18      Write buffer index
E6EB:  88         DEY             And decrement by 1 again
E6EC:  A5 AA      LDA  * $AA      Get received bit from zero page
E6EE:  AE 15 0A   LDX  $0A15      Number of data bits in X-reg
E6F1:  E0 09      CPX  # $09      8 bits, 1 stop bit received?
E6F3:  F0 04      BEQ  $E6F9      Yes, then everything OK
E6F5:  4A         LSR  A          Shift bits in correct position
E6F6:  E8         INX             Increment data bit counter by 1
E6F7:  D0 F8      BNE  $E6F1      Jump to byte adjustment
E6F9:  91 C8      STA  ($C8),Y    Write byte in input buffer
E6FB:  A9 20      LDA  # $20      Control value for parity check
E6FD:  2C 11 0A   BIT  $0A11      Test RS-232 command register
E700:  F0 B0      BEQ  $E6B2      Transfer is without parity
E702:  30 AD      BMI  $E6B1      Fixed bit value for parity
E704:  A5 A7      LDA  * $A7      Recevied parity bit in acc
```

```
E706:  45 AB      EOR  * $AB     Compare with calculated parity
E708:  F0 03      BEQ  $E70D     Equal, then continue with OK
E70A:  70 A5      BVS  $E6B1     Equal parity, continue with OK
E70C:  2C         .Byte  $2C     Skip to $E70F
E70D:  50 A2      BVC  $E6B1     Unequal parity,continue w/ OK
E70F:  A9 01      LDA  # $01     Code for parity error in acc
E711:  2C         .Byte  $2C     Skip to $E714
E712:  A9 04      LDA  # $04     Input buffer full of code in acc
E714:  2C         .Byte  $2C     Skip to $E717
E715:  A9 80      LDA  # $80     Break command received in acc
E717:  2C         .Byte  $2C     Skip to $E71A
E718:  A9 02      LDA  # $02     Load error code in Acc.
E71A:  0D 14 0A   ORA  $0A14     Combine code w/ RS-232 status
E71D:  8D 14 0A   STA  $0A14     And place in RS-232 status reg
E720:  4C C2 E6   JMP  $E6C2     Jump: receive the next byte

******************************   RS-232 CKOUT,output RS-232

E723:  A5 AA      LDA  * $AA     Get received byte in acc
E725:  D0 F1      BNE  $E718     Framing error
E727:  F0 EC      BEQ  $E715     Break command received
E729:  85 9A      STA  * $9A     Place device num in zero page
E72B:  AD 11 0A   LDA  $0A11     Load RS-232 command register
E72E:  4A         LSR  A         Shift bit 0 (handshake) into carry
E72F:  90 29      BCC  $E75A     Jump for 3-line handshake
E731:  A9 02      LDA  # $02     Code DATA SET READY test in
E733:  2C 01 DD   BIT  $DD01     acc. Read port B of CIA 2
E736:  10 1D      BPL  $E755     No DSR signal, then error
E738:  D0 20      BNE  $E75A     No Request To Send signal
E73A:  AD 0F 0A   LDA  $0A0F     Get RS-232 NMI status in acc
E73D:  29 02      AND  # $02     When data-receive is active, then
E73F:  D0 F9      BNE  $E73A     Wait, until reception is done
E741:  2C 01 DD   BIT  $DD01     Read port B of CIA 2
E744:  70 FB      BVS  $E741     Wait for Clear To Send signal
E746:  AD 01 DD   LDA  $DD01     Read port B CIA 2 and set bit 2
E749:  09 02      ORA  # $02     For Request To Send signal
E74B:  8D 01 DD   STA  $DD01     Write in port B
E74E:  2C 01 DD   BIT  $DD01     Read port B CIA2 and wait for
E751:  70 07      BVS  $E75A     Clear To Send signal
E753:  30 F9      BMI  $E74E     Poll Data Set Ready
```

```
E755:  A9 40      LDA  # $40      Code - missing Data Set Ready
E757:  8D 14 0A   STA  $0A14      Write signal in RS-233 status
E75A:  18         CLC             Set carry for OK indicator
E75B:  60         RTS             Return from subroutine

******************************    Output in RS-232 Buffer
                                  CTS = Clear to send
                                  DSR = Data set read

E75C:  20 70 E7   JSR  $E770      Start transfer is necessary
E75F:  AC 1B 0A   LDY  $0A1B      Index end RS-232 output buffer
E762:  C8         INY             Get in X-reg and increment by 1
E763:  CC 1A 0A   CPY  $0A1A      Comp with start of output buffer
E766:  F0 F4      BEQ  $E75C      Buffer full, then wait
E768:  8C 1B 0A   STY  $0A1B      Set new index to output buffer
E76B:  88         DEY             And decrement this pointer by 1
E76C:  A5 9E      LDA  * $9E      Get byte to output in acc
E76E:  91 CA      STA  ($CA),Y    And write in output buffer
E770:  AD 0F 0A   LDA  $0A0F      Copy RS-232 NMI flag into acc
E773:  4A         LSR  A          Test if bit 0 is set
E774:  B0 1E      BCS  $E794      Sending already?
E776:  A9 10      LDA  # $10      Initialize timer A with $10
E778:  8D 0E DD   STA  $DD0E      And then start it
E77B:  AD 16 0A   LDA  $0A16      Set the 2-byte timer for the
E77E:  8D 04 DD   STA  $DD04      Transmit baud rate in
E781:  AD 17 0A   LDA  $0A17      $DD04-$DD05
E784:  8D 05 DD   STA  $DD05
E787:  A9 81      LDA  # $81      Code  timer A underflow NMI
E789:  20 7F E6   JSR  $E67F      NMI on underflow of timer A
E78C:  20 4A E6   JSR  $E64A      Chk CTS+DSR, enable transfer
E78F:  A9 11      LDA  # $11      Initialize timer A with $11
E791:  8D 0E DD   STA  $DD0E      And start it
E794:  60         RTS             Return from subroutine

******************************    RS-232 CHKIN, Set RS-232
                                  input

E795:  85 99      STA  * $99      Place device num. in zero-page
E797:  AD 11 0A   LDA  $0A11      RS-232 command register in acc
E79A:  4A         LSR  A          Shift bit 0 (handshake) into carry
```

```
E79B:  90 28      BCC  $E7C5     3-line handshake, then continue
E79D:  29 08      AND  # $08     Test duplex operation
E79F:  F0 24      BEQ  $E7C5     Full duplex, then continue
E7A1:  A9 02      LDA  # $02     Code for DSR signal test
E7A3:  2C 01 DD   BIT  $DD01     Test port B of CIA 2 for DSR
E7A6:  10 AD      BPL  $E755     Missing, then set status and exit
E7A8:  F0 22      BEQ  $E7CC     Test Ready to Send signal
E7AA:  AD 0F 0A   LDA  $0A0F     RS-232 NMI status flag in acc
E7AD:  4A         LSR  A         Is send operation active, then
E7AE:  B0 FA      BCS  $E7AA     Wait until transfer finished
E7B0:  AD 01 DD   LDA  $DD01     Read port B of CIA and
E7B3:  29 FD      AND  # $FD     Eliminate bit 0 - Request to Send
E7B5:  8D 01 DD   STA  $DD01     Return signal on port B
E7B8:  AD 01 DD   LDA  $DD01     Read port B of CIA 2 and
E7BB:  29 04      AND  # $04     Check D T R signal
E7BD:  F0 F9      BEQ  $E7B8     Not present, then wait
E7BF:  A9 90      LDA  # $90     Get NMI mask for "flag" in acc
E7C1:  18         CLC            Clear carry as OK indicator
E7C2:  4C 7F E6   JMP  $E67F     Enable RS-232 NMI

******************************   RS-232 CHKIN for 3-line
                                 handshake

E7C5:  AD 0F 0A   LDA  $0A0F     Get RS-232 NMI status in acc
E7C8:  29 12      AND  # $12     If the RS-232 is not yet active
E7CA:  F0 F3      BEQ  $E7BF     Then start
E7CC:  18         CLC            Clear carry as OK indicator
E7CD:  60         RTS            Return from subroutine

******************************   GET from RS-232

E7CE:  AD 14 0A   LDA  $0A14     Get RS-232 status byte in acc
E7D1:  AC 19 0A   LDY  $0A19     Index end RS-232 input buffer
E7D4:  CC 18 0A   CPY  $0A18     Comp with  start of input buffer
E7D7:  F0 0B      BEQ  $E7E4     If equal, then buffer empty: Skip
E7D9:  29 F7      AND  # $F7     Mask out bit 3 (buffer empty)
E7DB:  8D 14 0A   STA  $0A14     And clear in RS-232 status
E7DE:  B1 C8      LDA  ($C8),Y   Read 1 byte from RS-232 buffer
E7E0:  EE 19 0A   INC  $0A19     Index RS-232 input buffer + 1
E7E3:  60         RTS            Return from subroutine
```

```
******************************   GET RS-232 if buffer empty

E7E4:  09 08      ORA  # $08     Set bit 3 (marker - buffer empty)
E7E6:  8D 14 0A   STA  $0A14     In RS-232 status
E7E9:  A9 00      LDA  # $00     Pass $00 as character read
E7EB:  60         RTS            Return from subroutine

******************************   Wait for end of RS-232

E7EC:  48         PHA            Save acc contents on stack
E7ED:  AD 0F 0A   LDA  $0A0F     Get RS-232 NMI flag
E7F0:  F0 11      BEQ  $E803     Not set, then OK and continue
E7F2:  AD 0F 0A   LDA  $0A0F     Read RS-232 NMI flag again
E7F5:  29 03      AND  # *03     Bit 0 = send, bit 1 = receive
E7F7:  D0 F9      BNE  $E7F2     Wait for end
E7F9:  A9 10      LDA  # $10     Load acc with $10
E7FB:  8D 0D DD   STA  $DD0D     Interrupt via "flag" line
E7FE:  A9 00      LDA  # $00     RS-232 NMI flag
E800:  8D 0F 0A   STA  $0A0F     Set status to "OK"
E803:  68         PLA            Restore acc contents
E804:  60         RTS            Return from subroutine

******************************   NMI routine for RS-232

E805:  98         TYA            Interrupt Control Register (ICR)
E806:  2D 0F 0A   AND  $0A0F     Combine with RS-232 NMI flag
E809:  AA         TAX            And store result in X-reg
E80A:  29 01      AND  # $01     Mask bits 1 - 7 and check if
E80C:  F0 28      BEQ  $E836     Send operation is active. no:Skip
E80E:  AD 00 DD   LDA  $DD00     Load acc with data port
E811:  29 FB      AND  # $FB     Clear bit 2 (TXD) and pass the
E813:  05 B5      ORA  * $B5     Bit to send
E815:  8D 00 DD   STA  $DD00     Store in data port
E818:  AD 0F 0A   LDA  $0A0F     Copy RS-232 NMI flag in acc
E81B:  8D 0D DD   STA  $DD0D     And write again into ICR
E81E:  8A         TXA            ICR/RS-232 NMI combine  acc
E81F:  29 12      AND  # $12     Isolate bits 1 and 4
E821:  F0 0D      BEQ  $E830     Not set, start byte reception
E823:  29 02      AND  # $02     Isloate bit 1, call of timer B
E825:  F0 06      BEQ  $E82D     Not set, the start bit
```

```
E827:  20 78 E8   JSR  $E878     Process received bit
E82A:  4C 30 E8   JMP  $E830     Start reception of byte
E82D:  20 A9 E8   JSR  $E8A9     Preparation for recept. next byte
E830:  20 FF E5   JSR  $E5FF     Start reception of byte
E833:  4C 49 E8   JMP  $E849     Return from interrupt
E836:  8A         TXA            Store X-reg contents in acc
E837:  29 02      AND  # $02     Data reception?
E839:  F0 06      BEQ  $E841     No, the skip processing
E83B:  20 78 E8   JSR  $E878     Process received bit
E83E:  4C 49 E8   JMP  $E849     Return from interrupt
E841:  8A         TXA            Restore old X-reg contents
E842:  29 10      AND  # $10     Check if a start bit expected
E844:  F0 03      BEQ  $E849     No, then continue
E846:  20 A9 E8   JSR  $E8A9     Prepare next bit reception
E849:  AD 0F 0A   LDA  $0A0F     Load RS-232 NMI flag
E84C:  8D 0D DD   STA  $DD0D     Copy in ICR of CIA 2
E84F:  60         RTS            Return from subroutine

******************************   Timer constants RS-232 baud
                                 rate.Table 1 for NTSC version

E850:  C1 27      (= 10177)      50  Baud
E852:  3E 1A      (=  6718)      75  Baud
E854:  C5 11      (=  4549)     110  Baud
E856:  74 0E      (=  3700)     134.5 Baud
E858:  ED 0C      (=  3309)     150  Baud
E85A:  45 06      (=  1605)     300  Baud
E85C:  F0 02      (=   752)     600  Baud
E85E:  46 01      (=   326)    1200  Baud
E860:  B8 00      (=   184)    1800  Baud
E862:  71 00      (=   113)    2400  Baud

******************************   Timer constant for RS-232 baud
                                 rate. Table 2 for PAL version

E864:  19 26      (=  9753)      50  Baud
E866:  44 19      (=  6468)      75  Baud
E868:  1A 11      (=  4378)     110  Baud
E86A:  E8 0D      (=  3560)     134.5 Baud
E86C:  70 0C      (=  3184)     150  Baud
```

```
E86E:  06 06       (=  1542)      300 Baud
E870:  D1 02       (=   736)      600 Baud
E872:  37 01       (=   311)     1200 Baud
E874:  AE 00       (=   174)     1800 Baud
E876:  69 00       (=   105)     2400 Baud

******************************    Input NMI routine for RS-232

E878:  AD 01 DD   LDA  $DD01       Read data port B of CIA 2
E87B:  29 01      AND  # $01       Isolate bit for receive data
E87D:  85 A7      STA  * $A7       And in Z-P RS-232 input bit flag
E87F:  AD 06 DD   LDA  $DD06       Get low value of CIA 2 timer B
E882:  E9 28      SBC  # $28       And subtract 28 from it
E884:  6D 16 0A   ADC  $0A16       Add full-bit time baud rate high
E887:  8D 06 DD   STA  $DD06       And reset timer B
E88A:  AD 07 DD   LDA  $DD07       Get high value of CIA 2 timer B
E88D:  6D 17 0A   ADC  $0A17       Add full-bit time baudrate high
E890:  8D 07 DD   STA  $DD07       And reset timer B high
E893:  A9 11      LDA  # $11       Write $11 in control register of
E895:  8D 0F DD   STA  $DD0F       CIA 2 = Start timer B
E898:  AD 0F 0A   LDA  $0A0F       Get RS-232 NMI status in acc
E89B:  8D 0D DD   STA  $DD0D       And set CIA interrupt control reg
E89E:  A9 FF      LDA  # $FF       Initialization value for timer B
E8A0:  8D 06 DD   STA  $DD06       Set timer B low to high value
E8A3:  8D 07 DD   STA  $DD07       Set timer B high to high value
E8A6:  4C 9D E6   JMP  $E69D       Process received bit

******************************    NMI routine for RS-232 output

E8A9:  AD 12 0A   LDA  $0A12       RS-232 user baud rate in acc
E8AC:  8D 06 DD   STA  $DD06       Ad in timer low of CIA 2
E8AF:  AD 13 0A   LDA  $0A13       RS-232 user baud rate in acc
E8B2:  8D 07 DD   STA  $DD07       And in timer B high of CIA 2
E8B5:  A9 11      LDA  # $11       Write $11 in control register of
E8B7:  8D 0F DD   STA  $DD0F       CIA 2 = start timer
E8BA:  A9 12      LDA  # $12       Invert bits 0, 1 and 4 of RS-232
E8BC:  4D 0F 0A   EOR  $0A0F       NMI flag. This value
E8BF:  8D 0F 0A   STA  $0A0F       Back in the NMI flag
E8C2:  A9 FF      LDA  # $FF       Initialization value for timer B
E8C4:  8D 06 DD   STA  $DD06       Set timer B low to high value
```

```
E8C7:  8D 07 DD   STA  $DD07      Set timer B high to high value
E8CA:  AE 15 0A   LDX  $0A15      Number of bits to send
E8CD:  86 A8      STX  * $A8      In z-page: Counter RS-232 Bits
E8CF:  60         RTS             Return from subroutine

****************************      Read program header from tape

E8D0:  A5 93      LDA  * $93      Save load/verify pointer on sys
E8D2:  48         PHA             Stack via the accumulator
E8D3:  20 F2 E9   JSR  $E9F2      Routine:read data block tape
E8D6:  68         PLA             Get load/verify flag from stack
E8D7:  85 93      STA  * $93      And back to zero page
E8D9:  B0 3D      BCS  $E918      If error occurred, return
E8DB:  A0 00      LDY  # $00      Set displacement to tape buffer
E8DD:  B1 B2      LDA  ($B2),Y    Get byte of read data block
E8DF:  C9 05      CMP  # $05      Was it and EOT marker?
E8E1:  F0 34      BEQ  $E917      Yes, then return
E8E3:  C9 01      CMP  # $01      Header type BASIC program?
E8E5:  F0 08      BEQ  $E8EF      Yes, evaluate correspondingly
E8E7:  C9 03      CMP  # $03      Header type machine lang prg?
E8E9:  F0 04      BEQ  $E8EF      Yes then evaluate appropriately
E8EB:  C9 04      CMP  # $04      Header type for data block?
E8ED:  D0 E1      BNE  $E8D0      No, then read in
E8EF:  AA         TAX             Store header type in X-reg
E8F0:  24 9D      BIT  * $9D      Check kernal status flag
E8F2:  10 22      BPL  $E916      Ctrl messages not allowed, skip
E8F4:  A0 63      LDY  # $63      Displace to "FOUND" message
E8F6:  20 22 F7   JSR  $F722      Output control message
E8F9:  A0 05      LDY  # $05      Set displace to start of filename
E8FB:  B1 B2      LDA  ($B2),Y    Read character from tape buffer
E8FD:  20 D2 FF   JSR  $FFD2      Kernal BSOUT: Output a char
E900:  C8         INY             Increment displace pointer by 1
E901:  C0 15      CPY  # $15      Max filename length = 16 char
E903:  D0 F6      BNE  $E8FB      Not yet reached, continue
E905:  A5 A1      LDA  * $A1      Middle-value time byte in acc
E907:  69 02      ADC  # $02      Delay loop for 8.5 seconds
E909:  A4 91      LDY  * $91      Check z-page stop / C= key flag
E90B:  C8         INY             Increment this value by 1
E90C:  D0 04      BNE  $E912      Key pressed, then continue
E90E:  C5 A1      CMP  * $A1      Check the 8.5 second delay loop
```

```
E910:  D0 F7      BNE  $E909     Time not up, continue waiting
E912:  C0 F0      CPY  # $F0     Was the space key pressed?
E914:  F0 BA      BEQ  $E8D0     Yes, then read header
E916:  18         CLC            Set indicator for OK
E917:  88         DEY            Old stop / C= key flag value
E918:  60         RTS            Return from subroutine

****************************     Write data block on tape
                                 Write header to tape, header type
                                  in acc: 3=mach. lang.,1=BASIC

E919:  85 9E      STA  * $9E     Put header type in zero page
E91B:  20 80 E9   JSR  $E980     Get tape buffer addr.- zero page
E91E:  90 5F      BCC  $E97F     Address invalid, then skip
E920:  A5 C2      LDA  * $C2     Put start address high in acc
E922:  48         PHA            And save on stack
E923:  A5 C1      LDA  * $C1     Put start address low in acc
E925:  48         PHA            And save on stack
E926:  A5 AF      LDA  * $AF     Put end address high into acc
E928:  48         PHA            And save on stack
E929:  A5 AE      LDA  * $AE     Put end address low in acc
E92B:  48         PHA            And save on stack
E92C:  A0 BF      LDY  # $BF     Get tape buffer length for loop
E92E:  A9 20      LDA  # $20     Load acc with char for space
E930:  91 B2      STA  ($B2),Y   Clear tape buffer
E932:  88         DEY            Loop until the entire length given
E933:  D0 FB      BNE  $E930     In Y is cleared
E935:  A5 9E      LDA  * $9E     Get the header type
E937:  91 B2      STA  ($B2),Y   At 1st position in tape buffer
E939:  C8         INY            Displacement to tape buffer +1
E93A:  A5 C1      LDA  * $C1     Get start add low from zero page
E93C:  91 B2      STA  ($B2),Y   And put it in the tape buffer
E93E:  C8         INY            Displacement to tape buffer + 1
E93F:  A5 C2      LDA  * $C2     Get start address high from Z-P
E941:  91 B2      STA  ($B2),Y   And put it in the tape buffer
E943:  C8         INY            Displacement to tape buffer + 1
E944:  A5 AE      LDA  * $AE     Get end address low from Z-P
E946:  91 B2      STA  ($B2),Y   And put it in the tape buffer
E948:  C8         INY            Displacement to tape buffer + 1
E949:  A5 AF      LDA  * $AF     Get end address high from Z-P
```

```
E94B:  91 B2      STA  ($B2),Y   And put it in the tape buffer
E94D:  C8         INY            Displacement to tape buffer + 1
E94E:  84 9F      STY  * $9F     Save displ. in tape buffer
E950:  A0 00      LDY  # $00     Clear cntr  for length of filename
E952:  84 9E      STY  * $9E     In zero page
E954:  A4 9E      LDY  * $9E     Get counter for filename length
E956:  C4 B7      CPY  * $B7     And compare with actual length
E958:  F0 0D      BEQ  $E967     All letters in buffer, then skip
E95A:  20 AE F7   JSR  $F7AE     Get letters from filename
E95D:  A4 9F      LDY  * $9F     Get displ. to tape buffer
E95F:  91 B2      STA  ($B2),Y   Letter of the filename in buffer
E961:  E6 9E      INC  * $9E     Counter for filename length + 1
E963:  E6 9F      INC  * $9F     Displacement to tape buffer + 1
E965:  D0 ED      BNE  $E954     Loop for next letter
E967:  20 87 E9   JSR  $E987     Start and address of tape buffer
E96A:  A9 69      LDA  # $69     Store check sum data and header
E96C:  85 AB      STA  * $AB     Block ($69) in zero page
E96E:  20 1C EA   JSR  $EA1C     Write block to tape
E971:  A8         TAY            Save current acc contents
E972:  68         PLA            Get end address high from stack
E973:  85 AE      STA  * $AE     And place in zero page again
E975:  68         PLA            Get end address low from stack
E976:  85 AF      STA  * $AF     And store in zero page again
E978:  68         PLA            Get start address high from stack
E979:  85 C1      STA  * $C1     And store in zero page again
E97B:  68         PLA            Get start address low from stack
E97C:  85 C2      STA  * $C2     And store in zero page again
E97E:  98         TYA            Get acc contents back
E97F:  60         RTS            Return from subroutine

****************************     Get tape buffer address and
                                 check  for validity

E980:  A6 B2      LDX  * $B2     Start of tape buffer in X-reg
E982:  A4 B3      LDY  * $B3     Start of tape buffer in Y-reg
E984:  C0 02      CPY  # $02     Zero page and stack not allowed
E986:  60         RTS            Return from subroutine
```

```
******************************      Tape end addr = start addr + 192

E987:  20 80 E9    JSR  $E980      Get tape buffer address
E98A:  8A          TXA             Start of tape buffer low in acc
E98B:  85 C1       STA  * $C1      And in Z-P I/O start address low
E98D:  18          CLC             Clear carry for addition
E98E:  69 C0       ADC  # $C0      End address=start address + 192
E990:  85 AE       STA  * $AE      New end address low in Z-P
E992:  98          TYA             Start of tape buffer high in acc
E993:  85 C2       STA  * $C2      and in Z-P I/O start address high
E995:  69 00       ADC  # $00      End addr high=start address hi +
E997:  85 AF       STA  * $AF      carry, end address high in Z-P
E999:  60          RTS             Return from subroutine

******************************      Seach tape header for name

E99A:  20 D0 E8    JSR  $E8D0      Search for next tape header
E99D:  B0 1E       BCS  $E9BD      IF EOT found, then return
E99F:  A0 05       LDY  # $05      Displace to name in tape buffer
E9A1:  84 9F       STY  * $9F      Store in zero page
E9A3:  A0 00       LDY  # $00      Init. the counter for the length
E9A5:  84 9E       STY  * $9E      Of the filename in the zero page
E9A7:  C4 B7       CPY  * $B7      Compare  length of target name
E9A9:  F0 11       BEQ  $E9BC      If equal, continue evaluation
E9AB:  20 AE F7    JSR  $F7AE      Get character of target name
E9AE:  A4 9F       LDY  * $9F      Displ. to filenames in tape buffer
E9B0:  D1 B2       CMP  ($B2),Y    Compare with target character
E9B2:  D0 E6       BNE  $E99A      Not equal, then not found
E9B4:  E6 9E       INC  * $9E      Filename legnth counter +1
E9B6:  E6 9F       INC  * $9F      Filename displ. to tape buffer +1
E9B8:  A4 9E       LDY  * $9E      Filename legnth counter in Y-reg
E9BA:  D0 EB       BNE  $E9A7      Next character comparison
E9BC:  18          CLC             Set indicator for OK
E9BD:  60          RTS             Return from subroutine
```

```
*****************************      Increment tape buffer pointer

E9BE:  20 80 E9   JSR  $E980       Get the tape buffer address
E9C1:  E6 A6      INC  * $A6       Z-P cassette buffer address +1
E9C3:  A4 A6      LDY  * $A6       And compare to
E9C5:  C0 C0      CPY  # $C0       Maximum value 192
E9C7:  60         RTS              Return from subroutine

*****************************      Wait for button on datasette

E9C8:  20 DF E9   JSR  $E9DF       Check if button pressed
E9CB:  F0 1A      BEQ  $E9E7       Button pressed, OK & continue
E9CD:  A0 1B      LDY  # $1B       Displ. to "Press Play on Tape"
E9CF:  20 22 F7   JSR  $F722       in Y. Output control message
E9D2:  20 8F EA   JSR  $EA8F       Test for stop-key interruption
E9D5:  20 DF E9   JSR  $E9DF       Check if key pressed
E9D8:  D0 F8      BNE  $E9D2       No, then to delay loop
E9DA:  A0 6A      LDY  # $6A       Displacement for "OK" message
E9DC:  4C 22 F7   JMP  $F722       Output control message

*****************************      Check if tape button pressed

E9DF:  A9 10      LDA  # $10       Set bit 4 for button test
E9E1:  24 01      BIT  * $01       Check data reg. processor port
E9E3:  D0 02      BNE  $E9E7       Not pressed,then exit
E9E5:  24 01      BIT  * $01       Check again
E9E7:  18         CLC              Yes: zero flag=1, no zero flag=0
E9E8:  60         RTS              Return from subroutine

*****************************      Wait for "record & play" keys

E9E9:  20 DF E9   JSR  $E9DF       Check if tape button is pressed
E9EC:  F0 F9      BEQ  $E9E7       Button pressed, OK & continue
E9EE:  A0 2E      LDY  # $2E       Displ. to "Press R & P on Tape"
E9F0:  D0 DD      BNE  $E9CF       Button delay loop/stop key chck
```

```
******************************      Read data block from tape

E9F2:  A9 00      LDA  # $00      System status with indicator
E9F4:  85 90      STA  * $90      Initialize for everything OK
E9F6:  85 93      STA  * $93      Clear Load/Verify pointer
E9F8:  20 87 E9   JSR  $E987      Get tape buffer addr/end address

******************************    Load program from tape

E9FB:  20 C8 E9   JSR  $E9C8      Wait for button on datasette
E9FE:  B0 1F      BCS  $EA1F      STOP key pressed, return
EA00:  78         SEI             Disbale all system interrupts
EA01:  A9 00      LDA  # $00      Init. value for IRQ storage
EA03:  85 AA      STA  * $AA      Tape-read mode input byte
EA05:  85 B4      STA  * $B4      storage. Tape temp pointer
EA07:  85 B0      STA  * $B0      Cassette time constant
EA09:  85 9E      STA  * $9E      Casettes error pass 1
EA0B:  85 9F      STA  * $9F      Cassette error pass 2
EA0D:  85 9C      STA  * $9C      Tape flag for byte recived
EA0F:  A9 90      LDA  # $90      IRQ on pin "flag"
EA11:  A2 0E      LDX  # $0E      Number of IRQ vector ($EAEB)
EA13:  D0 11      BNE  $EA26      Write data block to tape

******************************    Write tape buffer to tape

EA15:  20 87 E9   JSR  $E987      Load tape buffer address
EA18:  A9 14      LDA  # $14      Set length of the WRITE leader
EA1A:  85 AB      STA  * $AB      Store in zero page

******************************    Write data block to tape

EA1C:  20 E9 E9   JSR  $E9E9      Wait for record & play
EA1F:  B0 7A      BCS  $EA9B      STOP pressed, return
EA21:  78         SEI             Disable all system interrupts
EA22:  A9 82      LDA  # $82      IRQ on underflow of timer B
EA24:  A2 08      LDX  # $08      Number of IRQ vector ($EE2E)
EA26:  A0 00      LDY  # $00      Set interrupt mask register CIA
EA28:  8C 1A D0   STY  $D01A      To #0 (Interrupt disable)
EA2B:  88         DEY             Decrement Y-reg to $FF and set
EA2C:  8C 19 D0   STY  $D019      Interrupt Request Register
```

```
EA2F:  8D 0D DC   STA  $DC0D      Reset IRQ mask
EA32:  AD 0E DC   LDA  $DC0E      Load CIA control reg A, timer B
EA35:  09 19      ORA  # $19      "One shot" and start
EA37:  8D 0F DC   STA  $DC0F      Control reg.B, IRQ on timer B
EA3A:  29 91      AND  # $91      Set time compare pointer for tape
EA3C:  8D 0B 0A   STA  $0A0B      Operations
EA3F:  20 EC E7   JSR  $E7EC      Wait for end of R-232 transfer
EA42:  AD 11 D0   LDA  $D011      Copy VIC control reg. into acc
EA45:  A8         TAY             And into Y-reg
EA46:  29 10      AND  # $10      Set bit 4, screen on
EA48:  8D 39 0A   STA  $0A39      Store value in VDC temp storage
EA4B:  98         TYA             Old value back into acc
EA4C:  29 6F      AND  # $6F      Clear bit 8 of raster comparison
EA4E:  8D 11 D0   STA  $D011      And turn the screen off
EA51:  20 74 E5   JSR  $E574      Clock to 1 MHz and sprites off
EA54:  AD 14 03   LDA  $0314      IRQ vector low address in IRQ
EA57:  8D 09 0A   STA  $0A09      Temp storage for tape operations
EA5A:  AD 15 03   LDA  $0315      IRQ vector high address in IRQ
EA5D:  8D 0A 0A   STA  $0A0A      Temp storage for tape operations
EA60:  20 9B EE   JSR  $EE9B      Reset IRQ vector for tape operat.
EA63:  A9 02      LDA  # $02      Number of data blocks to reaed
EA65:  85 BE      STA  * $BE      Store in zero page
EA67:  20 5A ED   JSR  $ED5A      Initialize bit counter, serial I/O
EA6A:  A5 01      LDA  * $01      Turn cass. motor on by setting
EA6C:  29 1F      AND  # $1F      4th bit of the processor port data
EA6E:  85 01      STA  * $01      Register
EA70:  85 C0      STA  * $C0      Set pointer for tape motor
EA72:  A2 FF      LDX  # $FF      Counter for delay loop high
EA74:  A0 FF      LDY  # $FF      Counter for delay loop low
EA76:  88         DEY             X and Y regs are decremented
EA77:  D0 FD      BNE  $EA76      From 65535 to 0 to create the
EA79:  CA         DEX             Necessary delay
EA7A:  D0 F8      BNE  $EA74      For tape operations
EA7C:  58         CLI             Enable interrupt for tape I/O

*****************************     Wait for tape I/O end

EA7D:  AD 0A 0A   LDA  $0A0A      Compare with tape IRQ vector
EA80:  CD 15 03   CMP  $0315      with  normal IRQ pointer high
EA83:  18         CLC             Set indicator for OK
```

```
EA84:  F0 15      BEQ  $EA9B     IRQ vectors equal, then done
EA86:  20 8F EA   JSR  $EA8F     Check if STOP key pressed
EA89:  20 3D F6   JSR  $F63D     If pressed, set flag
EA8C:  4C 7D EA   JMP  $EA7D     Continue to wait for end

******************************   Test for STOP key

EA8F:  20 E1 FF   JSR  $FFE1     Kernal STOP: Test for stop key
EA92:  18         CLC            Set indicator for everything OK
EA93:  D0 0B      BNE  $EAA0     STOP not pressed, RTS exit
EA95:  20 57 EE   JSR  $EE57     Motor off, set normal IRQ
EA98:  38         SEC            Set carry for error
EA99:  68         PLA            Get return address form stack
EA9A:  68         PLA            And clear
EA9B:  A9 00      LDA  # $00     Load code for "interrupt" in acc
EA9D:  8D 0A 0A   STA  $0A0A     And set indicator for normal IRQ
EAA0:  60         RTS            Return from subroutine

******************************   Prepare cassette synchronization

EAA1:  86 B1      STX  * $B1     Store X-reg contents in Z-P
EAA3:  A5 B0      LDA  * $B0     Timing constant for tape in acc
EAA5:  0A         ASL  A         The timing constant is multiplied
EAA6:  0A         ASL  A         By the factor 4
EAA7:  18         CLC            Clear carry for addition
EAA8:  65 B0      ADC  * $B0     Add timing constant (corres. *5)
EAAA:  18         CLC            Clear carry for addition
EAAB:  65 B1      ADC  * $B1     Add old X-reg contents & place
EAAD:  85 B1      STA  * $B1     This value in the zero page
EAAF:  A9 00      LDA  # $00     Load low value for timer A
EAB1:  24 B0      BIT  * $B0     Check if timing constant >128
EAB3:  30 01      BMI  $EAB6     Yes, then skip alignment
EAB5:  2A         ROL  A         The inti value for timer A is
EAB6:  06 B1      ASL  * $B1     Multiplied by 4 by rotating the
EAB8:  2A         ROL  A         Contents of the acc in connection
EAB9:  06 B1      ASL  * $B1     With shifting of tape timing
EABB:  2A         ROL  A         constant
EABC:  AA         TAX            Store high of timer value in X
EABD:  AD 06 DC   LDA  $DC06     Low value CIA 1 timer B in acc
EAC0:  C9 16      CMP  # $16     Change timer B high to 63755
```

```
EAC2:  90 F9      BCC  $EABD     Yes, then loop to timer read
EAC4:  65 B1      ADC  * $B1     Add low for initialization
EAC6:  8D 04 DC   STA  $DC04     And set in timer A low
EAC9:  8A         TXA            Add high value of the init in acc
EACA:  6D 07 DC   ADC  $DC07     With carry to timer B high
EACD:  8D 05 DC   STA  $DC05     And set in timer A high
EAD0:  AD 0B 0A   LDA  $0A0B     Copy init. value from tape time
EAD3:  8D 0E DC   STA  $DC0E     Constant to start timer A
EAD6:  8D 0D 0A   STA  $0A0D     Reset timer A flag
EAD9:  AD 0D DC   LDA  $DC0D     Interrupt Control Register in acc
EADC:  29 10      AND  # $10     Check negative edge on FLAG
EADE:  F0 09      BEQ  $EAE9     No, wait for negative edge
EAE0:  A9 EA      LDA  # $EA     Place the contents of zero page
EAE2:  48         PHA            Locations $EA and $E9 on the
EAE3:  A9 E9      LDA  # $E9     Sys stack as quasi return address
EAE5:  48         PHA
EAE6:  4C C8 EE   JMP  $EEC8     Simulate the interrupt call
EAE9:  58         CLI            Enable all system interrupts
EAEA:  60         RTS            Return from subroutine

****************************     Interrupt routine for tape read

EAEB:  AE 07 DC   LDX  $DC07     CIA 1 timer B hi in X-reg
EAEE:  A0 FF      LDY  # $FF     Init Y-reg with with high value
EAF0:  98         TYA            And for subtraction in acc
EAF1:  ED 06 DC   SBC  $DC06     Subtract timer B low of #255
EAF4:  EC 07 DC   CPX  $DC07     Is timer B high decremented?
EAF7:  D0 F2      BNE  $EAEB     Yes, back to time comparison
EAF9:  86 B1      STX  * $B1     Place timer B high in zero page
EAFB:  AA         TAX            Time low since last signal in X
EAFC:  8C 06 DC   STY  $DC06     Timer B low to high value
EAFF:  8C 07 DC   STY  $DC07     Timer B high to high value
EB02:  A9 19      LDA  # $19     Set timer B mode
EB04:  8D 0F DC   STA  $DC0F     And start timer B
EB07:  AD 0D DC   LDA  $DC0D     Interrupt Control Register in acc
EB0A:  8D 0C 0A   STA  $0A0C     And in systetm storage for tape
EB0D:  98         TYA            Initialize acc with #255
EB0E:  E5 B1      SBC  * $B1     Subtract timer B high from #255
EB10:  86 B1      STX  * $B1     Store elapsed time in zero page
EB12:  4A         LSR  A         The value stored in the acc
```

```
EB13:  66 B1      ROR  # $B1     For the elapsed time
EB15:  4A         LSR  A         Is divided by the
EB16:  66 B1      ROR  # $B1     Factor 4
EB18:  A5 B0      LDA  * $B0     Get timing constant from z-page
EB1A:  18         CLC            Clear carry for addition
EB1B:  69 3C      ADC  # $3C     Add #60 to timing constant
EB1D:  C5 B1      CMP  * $B1     > time since last signal?
EB1F:  B0 4A      BCS  $EB6B     Yes, then no information, skip
EB21:  A6 9C      LDX  * $9C     Was a byte received
EB23:  F0 03      BEQ  $EB28     No, then skip
EB25:  4C 1F EC   JMP  $EC1F     Continue byte-receive routine
EB28:  A6 A3      LDX  * $A3     Was byte read entirely?
EB2A:  30 1B      BMI  $EB47     Yes, then evaluate
EB2C:  A2 00      LDX  # $00     Code for short pulse  X-reg (0)
EB2E:  69 30      ADC  # $30     Set acc for pulse read
EB30:  65 B0      ADC  * $B0     And add timing constant
EB32:  C5 B1      CMP  * $B1     Short time pulse received?
EB34:  B0 1C      BCS  $EB52     Yes, then skip long pulse
EB36:  E8         INX            Code for long pule in X-reg (1)
EB37:  69 26      ADC  # $26     Set acc for pulse read
EB39:  65 B0      ADC  * $B0     And add timing constant
EB3B:  C5 B1      CMP  * $B1     Long time pulse received?
EB3D:  B0 17      BCS  $EB56     Yes,  skip other pulse duration
EB3F:  69 2C      ADC  # $2C     Check if the previous time
EB41:  65 B0      ADC  * $B0     Pulse was stil longer. If so,
EB43:  C5 B1      CMP  * $B1     It is a byte header pulse
EB45:  90 03      BCC  $EB4A     No, then skip processing
EB47:  4C CF EB   JMP  $EBCF     Process received byte
EB4A:  A5 B4      LDA  * $B4     Check if timer A is enables
EB4C:  F0 1D      BEQ  $EB6B     No, then skip
EB4E:  85 A8      STA  * $A8     Set pointer for "READ ERROR"
EB50:  D0 19      BNE  $EB6B     Jump to timer interrupt read
EB52:  E6 A9      INC  * $A9     Pntr for pulse-length change +1
EB54:  B0 02      BCS  $EB58     Skip change decrement
EB56:  C6 A9      DEC  * $A9     Pntr for pulse length change -1
EB58:  38         SEC            Set carry for subtraction
EB59:  E9 13      SBC  # $13     From read value #19, as well as
EB5B:  E5 B1      SBC  * $B1     Subtract elapsed time
EB5D:  65 92      ADC  * $92     Add zero page storage for timing
EB5F:  85 92      STA  * $92     Correction flag and store
```

```
EB61:  A5 A4      LDA  * $A4     Invert the zero page flag for the
EB63:  49 01      EOR  # $01     Reception of both pulses
EB65:  85 A4      STA  * $A4     And store in zero page again
EB67:  F0 2B      BEQ  $EB94     Both pulses received, then skip
EB69:  86 C5      STX  * $C5     Store signal received in z-page
EB6B:  A5 B4      LDA  * $B4     Check if timer A is enabled
EB6D:  F0 22      BEQ  $EB91     No, then terminate interrupt
EB6F:  AD 0C 0A   LDA  $0A0C     Get contents of ICR in acc
EB72:  29 01      AND  # $01     Was it a timer A interrupt
EB74:  D0 05      BNE  $EB7B     Yes, then skip
EB76:  AD 0D 0A   LDA  $0A0D     Check if timer A is run down
EB79:  D0 16      BNE  $EB91     No, then terminate interrupt
EB7B:  A9 00      LDA  # $00     Clear the zero-page flag for
EB7D:  85 A4      STA  * $A4     Pulse count (low value)
EB7F:  8D 0D 0A   STA  $0A0D     Set pointer for timer A timeout
EB82:  A5 A3      LDA  * $A3     Check is byte is completely read
EB84:  10 30      BPL  $EBB6     No, then skip
EB86:  30 BF      BMI  $EB47     Yes, process correspondingly
EB88:  A2 A6      LDX  # $A6     Initialization value for timer A
EB8A:  20 A1 EA   JSR  $EAA1     Prepare tape for reading
EB8D:  A5 9B      LDA  * $9B     Zero-page parity byte in acc
EB8F:  D0 B9      BNE  $EB4A     Not zero, then parity error
EB91:  4C 33 FF   JMP  $FF33     Back to kernal interrupt
EB94:  A5 92      LDA  * $92     Timing correction pointer in acc
EB96:  F0 07      BEQ  $EB9F     Flag cleared, then skip
EB98:  30 03      BMI  $EB9D     Smaller then zero, skip dec
EB9A:  C6 B0      DEC  * $B0     Z-page timing constant -1
EB9C:  2C         .Byte  $2C     Skip to $EB9F
EB9D:  E6 B0      INC  * $D0     Z-page timing constant +1
EB9F:  A9 00      LDA  # $00     Z-page pointer timing constant
EBA1:  85 92      STA  * $92     Erase correction (low value)
EBA3:  E4 C5      CPX  * $C5     Compare pulse received with
EBA5:  D0 0F      BNE  $EBB6     previous  Not equal, OK & skip
EBA7:  8A         TXA            Check if short pulse received
EBA8:  D0 A0      BNE  $EB4A     No, then read error. Skip
EBAA:  A5 A9      LDA  * $A9     Pulse length change pntr in acc
EBAC:  30 BD      BMI  $EB6B     Negative value, then skip
EBAE:  C9 10      CMP  # $10     16 short pulses received?
EBB0:  90 B9      BCC  $EB6B     No, then for negative value
EBB2:  85 96      STA  * $96     Yes, EOB flag received
```

```
EBB4:  B0 B5      BCS  $EB6B    Unconditional jump
EBB6:  8A         TXA           Put received bit in acc
EBB7:  45 9B      EOR  * $9B    Compare witb tape parity
EBB9:  85 9B      STA  * $9B    Store in tape parity again
EBBB:  A5 B4      LDA  * $B4    Check if timer A is enabled
EBBD:  F0 D2      BEQ  $EB91    No, then end interrupt
EBBF:  C6 A3      DEC  * $A3    Zero-page storage for bit cntr -1
EBC1:  30 C5      BMI  $EB88    Parity bit received? Yes, skip
EBC3:  46 C5      LSR  * $C5    No, then bit read into
EBC5:  66 BF      ROR  # $BF    Zero-page storage for tape data
EBC7:  A2 DA      LDX  # $DA    Initialization value for timer A
EBC9:  20 A1 EA   JSR  $EAA1    Prepare cassette synchronization
EBCC:  4C 33 FF   JMP  $FF33    Back to IRQ routine
EBCF:  A5 96      LDA  * $96    Check if EOB received
EBD1:  F0 04      BEQ  $EBD7    No, skip timer read
EBD3:  A5 B4      LDA  * $B4    Check if timer A enabled
EBD5:  F0 07      BEQ  $EBDE    No, skip bit counter test
EBD7:  A5 A3      LDA  * $A3    Check if Z-P bit cntr is negative
EBD9:  30 03      BMI  $EBDE    Yes, wait for byte header
EBDB:  4C 56 EB   JMP  $EB56    Process long pulse,no header
EBDE:  46 B1      LSR  * $B1    byte. Halve the elapsed time
EBE0:  A9 93      LDA  # $93    since the last negativce edge and
EBE2:  38         SEC           Subtract this value
EBE3:  E5 B1      SBC  * $B1    From the constant #147
EBE5:  65 B0      ADC  * $B0    Add zero-page timing constant
EBE7:  0A         ASL  A        And double this value
EBE8:  AA         TAX           To X-reg, init value for timer A
EBE9:  20 A1 EA   JSR  $EAA1    Prepare cassette synchronization
EBEC:  E6 9C      INC  * $9C    Set Z-P pointer:"byte received"
EBEE:  A5 B4      LDA  * $B4    Check if timer A enabled
EBF0:  D0 11      BNE  $EC03    Yes, then skip
EBF2:  A5 96      LDA  * $96    Check if EOB received
EBF4:  F0 26      BEQ  $EC1C    No, to normal IRQ routine
EBF6:  85 A8      STA  * $A8    Set z-page display for read error
EBF8:  A9 00      LDA  # $00    Clear z-page storage for EOB
EBFA:  85 96      STA  * $96    marker.  (low value)
EBFC:  A9 81      LDA  # $81    Code value for timer A enable
EBFE:  8D 0D DC   STA  $DC0D    Enable interrupt for timer A
EC01:  85 B4      STA  * $B4    Set z-page flag, timer A possible
EC03:  A5 96      LDA  * $96    Copy  z-page for received EOB
```

```
EC05:  85 B5      STA  * $B5     In flag for valid EOB
EC07:  F0 09      BEQ  $EC12     No EOB marker, then skip
EC09:  A9 00      LDA  # $00     Control code for timer A disable
EC0B:  85 B4      STA  * $B4     Put in appropriate z-page pointer
EC0D:  A9 01      LDA  # $01     Control code, disabling timer A
EC0F:  8D 0D DC   STA  $DC0D     Interrupts in CIA control register
EC12:  A5 BF      LDA  * $BF     Z-page shift register, READ in
EC14:  85 BD      STA  * $BD     Z-page storage for read byte
EC16:  A5 A8      LDA  * $A8     Combine Z-P pointer for read
EC18:  05 A9      ORA  * $A9     error with pulse change pointer
EC1A:  85 B6      STA  * $B6     Place in error code of byte
EC1C:  4C 33 FF   JMP  $FF33     Back to normal IRQ call
EC1F:  20 5A ED   JSR  $ED5A     Set bit counter for serial output
EC22:  85 9C      STA  * $9C     Pointer: reset "byte received"
EC24:  A2 DA      LDX  # $DA     Initialization value for timer A
EC26:  20 A1 EA   JSR  $EAA1     Prepare cassette synchronization
EC29:  A5 BE      LDA  * $BE     Check if number of remaining
EC2B:  F0 02      BEQ  $EC2F     blocks is zero. If so, skip
EC2D:  85 A7      STA  * $A7     Reset number of blocks to read
EC2F:  A9 0F      LDA  # $0F     Mask value for count before read
EC31:  24 AA      BIT  * $AA     Test pointer, reading from tape
EC33:  10 17      BPL  $EC4C     If all characters received, end
EC35:  A5 B5      LDA  * $B5     Test if valid EOB received
EC37:  D0 0C      BNE  $EC45     Yes, then skip
EC39:  A6 BE      LDX  * $BE     Is the number of blocks
EC3B:  CA         DEX            remaining to be read = 1?
EC3C:  D0 0B      BNE  $EC49     No, to normal IRQ call
EC3E:  A9 08      LDA  # $08     Set bit 3 in A for "long block"
EC40:  20 57 F7   JSR  $F757     Reset system status pointer
EC43:  D0 04      BNE  $EC49     Uncond. jump normal IRQ rout
EC45:  A9 00      LDA  # $00     Z-P pointer, "reading from tape"
EC47:  85 AA      STA  * $AA     Set to "scan" (low value)
EC49:  4C 33 FF   JMP  $FF33     Back to normal IRQ routine
EC4C:  70 31      BVS  $EC7F     Skip for tape read pointer "read"
EC4E:  D0 18      BNE  $EC68     Skip for tape read pointer"count"
EC50:  A5 B5      LDA  * $B5     Check if EOB received
EC52:  D0 F5      BNE  $EC49     Yes, back to normal IRQ routine
EC54:  A5 B6      LDA  * $B6     Test if byte-read error occurred
EC56:  D0 F1      BNE  $EC49     Yes, back to normal IRQ routine
EC58:  A5 A7      LDA  * $A7     Get number of blocks to read yet
```

```
EC5A:  4A        LSR  A          And shift bit 0 into carry flag
EC5B:  A5 BD     LDA  * $BD      Get read byte from zero page
EC5D:  30 03     BMI  $EC62      If it is a count byte, then skip
EC5F:  90 18     BCC  $EC79      More than one block read, skip
EC61:  18        CLC             Reset carry flag pointer
EC62:  B0 15     BCS  $EC79      Skip if only one block read
EC64:  29 0F     AND  # $0F      Mask out upper nibble (bits 4-7)
EC66:  85 AA     STA  * $AA      Store as count value, counter -1
EC68:  C6 AA     DEC  * $AA      And check if all sync bytes
EC6A:  D0 DD     BNE  $EC49      received .No, to normal IRQ
EC6C:  A9 40     LDA  # $40      Setbit 6 in the acc and the z-page
EC6E:  85 AA     STA  * $AA      Tape read pointer to: "read"
EC70:  20 51 ED  JSR  $ED51      Copy input/output start address
EC73:  A9 00     LDA  # $00      Clear zero page pointer for read
EC75:  85 AB     STA  * $AB      Checksum (set to low value)
EC77:  F0 D0     BEQ  $EC49      Back to normal IRQ routine
EC79:  A9 80     LDA  # $80      Set bit 7 in acc and the zero page
EC7B:  85 AA     STA  * $AA      Tape read pointer to: "end"
EC7D:  D0 CA     BNE  $EC49      Back to normal IRQ routine
EC7F:  A5 B5     LDA  * $B5      Check if EOB marker set
EC81:  F0 0A     BEQ  $EC8D      No, then skip
EC83:  A9 04     LDA  # $04      Set bit 2 in A for short block
EC85:  20 57 F7  JSR  $F757      Reset system status pointer
EC88:  A9 00     LDA  # $00      Code for read pointer to "scan"
EC8A:  4C 0C ED  JMP  $ED0C      Set and jump absolute
EC8D:  20 B7 EE  JSR  $EEB7      Check if end reached
EC90:  90 03     BCC  $EC95      No, then continue as normal
EC92:  4C 0A ED  JMP  $ED0A      To read end for a block
EC95:  A6 A7     LDX  * $A7      Is the number of blocks left to
EC97:  CA        DEX             Read = 1?
EC98:  F0 2E     BEQ  $ECC8      Yes, pass 2 (correction pass)
EC9A:  A5 93     LDA  * $93      Test if verify marker set
EC9C:  F0 0D     BEQ  $ECAB      No, then skip
EC9E:  A0 00     LDY  # $00      Set displacment comparison, #0
ECA0:  20 CC F7  JSR  $F7CC      Fetch routine for LSV calls
ECA3:  C5 BD     CMP  * $BD      Compare with byte read
ECA5:  F0 04     BEQ  $ECAB      Both equal, then OK and skip
ECA7:  A9 01     LDA  # $01      Code for character read error
ECA9:  85 B6     STA  * $B6      In zero page tape temp pointer
ECAB:  A5 B6     LDA  * $B6      Test tape temp pointer for error
```

```
ECAD:  F0 4C      BEQ  $ECFB     No error occurred, then skip
ECAF:  A2 3D      LDX  # $3D     Check if 31 errors encountered
ECB1:  E4 9E      CPX  * $9E     While reading
ECB3:  90 3F      BCC  $ECF4     Yes, then not correctable
ECB5:  A6 9E      LDX  * $9E     Displ. for add read error in stack
ECB7:  A5 AD      LDA  * $AD     Get address byte of error low
ECB9:  9D 01 01   STA  $0101,X   And store error address on stack
ECBC:  A5 AC      LDA  * $AC     Get address byte of error high
ECBE:  9D 00 01   STA  $0100,X   And store error address on stack
ECC1:  E8         INX            Increment error addr-displ. ptr +
ECC2:  E8         INX            Error number-counter by 2
ECC3:  86 9E      STX  * $9E     And place in error counter
ECC5:  4C FB EC   JMP  $ECFB     Continue as if no error occurred
ECC8:  A6 9F      LDX  * $9F     Check if all read errors
ECCA:  E4 9E      CPX  * $9E     Corrected
ECCC:  F0 37      BEQ  $ED05     Yes, then continue
ECCE:  A5 AC      LDA  * $AC     Get current addr. byte low value
ECD0:  DD 00 01   CMP  $0100,X   Compare w/ error addr byte low
ECD3:  D0 30      BNE  $ED05     Not equal, then skip
ECD5:  A5 AD      LDA  * $AD     Get current addr byte high value
ECD7:  DD 01 01   CMP  $0101,X   Compare with address byte high
ECDA:  D0 29      BNE  $ED05     Not equal, then skip
ECDC:  E6 9F      INC  * $9F     Increment the z-page correction
ECDE:  E6 9F      INC  * $9F     counter for pass 2 by 2
ECE0:  A5 93      LDA  * $93     Check if verify marker set
ECE2:  F0 0C      BEQ  $ECF0     No, then set
ECE4:  A0 00      LDY  # $00     Displacement for fetch routine
ECE6:  20 CC F7   JSR  $F7CC     Fetch routien for LSV calls
ECE9:  C5 BD      CMP  * $BD     Read byte equal memory byte?
ECEB:  F0 18      BEQ  $ED05     Yes, then skip
ECED:  C8         INY            Incremern displacement pointer
ECEE:  84 B6      STY  * $B6     And put in z-page error pointer
ECF0:  A5 B6      LDA  * $B6     Check if error occurred
ECF2:  F0 07      BEQ  $ECFB     No, then skip
ECF4:  A9 10      LDA  # $10     Set bit 4 -read error not corrected
ECF6:  20 57 F7   JSR  $F757     Reset system status pointer
ECF9:  D0 0A      BNE  $ED05     Unconditional jump
ECFB:  A5 93      LDA  * $93     Check if verify marker set
ECFD:  D0 06      BNE  $ED05     Yes, then skip
ECFF:  A8         TAY            Set displacement pointer to #0
```

```
ED00:  A5 BD      LDA  * $BD     Get byte into acc
ED02:  20 BC F7   JSR  $F7BC     STASH rout. for LSV routines
ED05:  20 C1 EE   JSR  $EEC1     Incr input/output start address
ED08:  D0 44      BNE  $ED4E     Back to normal IRQ routine
ED0A:  A9 80      LDA  # $80     Code for read pointer to "end"
ED0C:  85 AA      STA  * $AA     Set tape read pntr according, acc
ED0E:  78         SEI            Disable all system interrupts
ED0F:  A2 01      LDX  # $01     Code, value for int. of timer A
ED11:  8E 0D DC   STX  $DC0D     Disable in ICR
ED14:  AE 0D DC   LDX  $DC0D     Reset interrupt pointer
ED17:  A6 BE      LDX  * $BE     Test if number of blocks
ED19:  CA         DEX            remaining to process is zero
ED1A:  30 02      BMI  $ED1E     Yes, then skip
ED1C:  86 BE      STX  * $BE     Store new number in zero page
ED1E:  C6 A7      DEC  * $A7     Decrement z-page block counter
ED20:  F0 08      BEQ  $ED2A     Block counter = 0, then skip
ED22:  A5 9E      LDA  * $9E     Check if error encountered in
ED24:  D0 28      BNE  $ED4E     pass 1. Yes, then skip
ED26:  85 BE      STA  * $BE     Number of blocks to process: 0
ED28:  F0 24      BEQ  $ED4E     Back to normal IRQ routine
ED2A:  20 57 EE   JSR  $EE57     Routine: end tape I/O
ED2D:  20 51 ED   JSR  $ED51     Copy start addr in load pointer
ED30:  A0 00      LDY  # $00     Clear the z-page ptr for chksum
ED32:  84 AB      STY  * $AB     Set displacement to zero
ED34:  20 CC F7   JSR  $F7CC     FETCH routine for LSV operat.
ED37:  45 AB      EOR  * $AB     Combine memory byte with
ED39:  85 AB      STA  * $AB     chksum & store in chksum pntr
ED3B:  20 C1 EE   JSR  $EEC1     Increment input/output start addr
ED3E:  20 B7 EE   JSR  $EEB7     Check if end address reached
ED41:  90 F1      BCC  $ED34     Not end address, then continue
ED43:  A5 AB      LDA  * $AB     Compare the generate checksum
ED45:  45 BD      EOR  * $BD     With the checksum read
ED47:  F0 05      BEQ  $ED4E     Equal, then OK and continue
ED49:  A9 20      LDA  # $20     Set bit 5 (checksum error)
ED4B:  20 57 F7   JSR  $F757     Reset system status pointer
ED4E:  4C 33 FF   JMP  $FF33     Back to normal IRQ routine
```

```
*****************************      Copy input/output start address

ED51:  A5 C2      LDA  * $C2       Get input/output
ED53:  85 AD      STA  * $AD       Store high value in z-page $AD
ED55:  A5 C1      LDA  * $C1       Get input/output start addr low
ED57:  85 AC      STA  * $AC       Store low value in z-page $AC
ED59:  60         RTS              Return from subroutine

*****************************      Set bit counter for serial output

ED5A:  A9 08      LDA  # $08       Counter for 8 bits to transfer
ED5C:  85 A3      STA  * $A3       Initialize in zero page
ED5E:  A9 00      LDA  # $00       Set the high byte of the 2 byte
ED60:  85 A4      STA  * $A4       Zero page counter to $00
ED62:  85 A8      STA  * $A8       Clear tape read error flag
ED64:  85 9B      STA  * $9B       Initialize parity for tape
ED66:  85 A9      STA  * $A9       Initialize tape zero read flag
ED68:  60         RTS              Return from subroutine

*****************************      Write a bit to tape

ED69:  A5 BD      LDA  * $BD       Bit to output from z-page to acc
ED6B:  4A         LSR  A           And bit to output (0) in carry
ED6C:  A9 60      LDA  # $60       Set time for "0-bit"
ED6E:  90 02      BCC  $ED72       Set timer and output
ED70:  A9 B0      LDA  # $B0       Set time for "1-bit"
ED72:  A2 00      LDX  # $00       Low value for timer high byte
ED74:  8D 06 DC   STA  $DC06       CIA1 timer B low byte -bit time
ED77:  8E 07 DC   STX  $DC07       CIA1 timer B hi-byte low value
ED7A:  AD 0D DC   LDA  $DC0D       Clear interrupt flag
ED7D:  A9 19      LDA  # $19       Load timer B, "one shot" & start
ED7F:  8D 0F DC   STA  $DC0F       CIA control reg. IRQ at timer
ED82:  A5 01      LDA  * $01       Inverse value for output bit
ED84:  49 08      EOR  # $08       Invert in processor port and
ED86:  85 01      STA  * $01       Put back in processor port
ED88:  29 08      AND  # $08       Save current signal
ED8A:  60         RTS              Return from subroutine
```

```
ED8B:  38          SEC              Set carry for rotation
ED8C:  66 B6       ROR  * $B6       Negate block written flag
ED8E:  30 3C       BMI  $EDCC       Interrupt return

******************************      Interrupt routine for tape write

ED90:  A5 A8       LDA  * $A8       Check if byte pulse written
ED92:  D0 12       BNE  $EDA6       Yes then skip byte pulse write
ED94:  A9 10       LDA  # $10       Low value for byte freq in acc
ED96:  A2 01       LDX  # $01       High value for byte freq in X
ED98:  20 74 ED    JSR  $ED74       Write "byte" pulse to tape
ED9B:  D0 2F       BNE  $EDCC       If first half wave, to normal IRQ
ED9D:  E6 A8       INC  * $A8       Set pointer for pulse written
ED9F:  A5 B6       LDA  * $B6       Test "block written" pointer
EDA1:  10 29       BPL  $EDCC       Yes, then back to normal IRQ
EDA3:  4C 1B EE    JMP  $EE1B       Block finished, continue write
EDA6:  A5 A9       LDA  * $A9       Check if longer pulse written
EDA8:  D0 09       BNE  $EDB3       Yes, then skip long pulse
EDAA:  20 70 ED    JSR  $ED70       Write long pulse to tape
EDAD:  D0 1D       BNE  $EDCC       If first half wave, to normal IRQ
EDAF:  E6 A9       INC  * $A9       Set pointer for pulse written
EDB1:  D0 19       BNE  $EDCC       Back to normal IRQ routine
EDB3:  20 69 ED    JSR  $ED69       Write one bit to tape
EDB6:  D0 14       BNE  $EDCC       If first half wave, to normal IRQ
EDB8:  A5 A4       LDA  * $A4       Invert the zero-page bit pulse
EDBA:  49 01       EOR  # $01       Pointer and
EDBC:  85 A4       STA  * $A4       Save it again
EDBE:  F0 0F       BEQ  $EDCF       If #0, write both pulses
EDC0:  A5 BD       LDA  * $BD       Invert bit 0 of the zero-page bit
EDC2:  49 01       EOR  # $01       Shift storage
EDC4:  85 BD       STA  * $BD       And save again
EDC6:  29 01       AND  # $01       Eliminate current bit & combine
EDC8:  45 9B       EOR  * $9B       With parity bit of the byte
EDCA:  85 9B       STA  * $9B       And store in parity flag
EDCC:  4C 33 FF    JMP  $FF33       Back to normal IRQ routine
EDCF:  46 BD       LSR  * $BD       Shift bit out and decrement the
EDD1:  C6 A3       DEC  * $A3       Zero-page bit counter by 1
EDD3:  A5 A3       LDA  * $A3       Is end reached already?
```

```
EDD5:  F0 3B      BEQ  $EE12   Yes, then generate parity. Skip
EDD7:  10 F3      BPL  $EDCC   No, then back to normal IRQ
EDD9:  20 5A ED   JSR  $ED5A   Set bit counter for serial output
EDDC:  58         CLI          Enable all system interrupts
EDDD:  A5 A5      LDA  * $A5   Check if sync bytes written
EDDF:  F0 12      BEQ  $EDF3   Yes, then skip
EDE1:  A2 00      LDX  # $00   Clear the checksum storage for
EDE3:  86 C5      STX  * $C5   the read buffer (low value)
EDE5:  C6 A5      DEC  * $A5   Decrememt sync counter by 1
EDE7:  A6 BE      LDX  * $BE   Check if the first block
EDE9:  E0 02      CPX  # $02   Is already written
EDEB:  D0 02      BNE  $EDEF   No, then skip
EDED:  09 80      ORA  # $80   Set bit 7 in sync byte
EDEF:  85 BD      STA  * $BD   And in zero page bit shift storage
EDF1:  D0 D9      BNE  $EDCC   Back to normal IRQ routine
EDF3:  20 B7 EE   JSR  $EEB7   Check if end address reached
EDF6:  90 0A      BCC  $EE02   Not reached, continue write
EDF8:  D0 91      BNE  $ED8B   Set "block written" pointer
EDFA:  E6 AD      INC  * $AD   Current address byte +1
EDFC:  A5 C5      LDA  * $C5   Get buffer checksum from Z-P
EDFE:  85 BD      STA  * $BD   Store value in bit shift storage
EE00:  B0 CA      BCS  $EDCC   Back to normal IRQ routine
EE02:  A0 00      LDY  # $00   Set displacement pointer to #0
EE04:  20 CC F7   JSR  $F7CC   FETCH routine for LSV operat.
EE07:  85 BD      STA  * $BD   Bring char in bit shift storage
EE09:  45 C5      EOR  * $C5   Combine with checksum storage
EE0B:  85 C5      STA  * $C5   And store again
EE0D:  20 C1 EE   JSR  $EEC1   Incr input/output start address
EE10:  D0 BA      BNE  $EDCC   Back to normal IRQ routine
EE12:  A5 9B      LDA  * $9B   Invert parity bit of byte from
EE14:  49 01      EOR  # $01   Z-P and copy into the bit-shift
EE16:  85 BD      STA  * $BD   Storage
EE18:  4C 33 FF   JMP  $FF33   Back to the normal IRQ routine
EE1B:  C6 BE      DEC  * $BE   Check if all bits written
EE1D:  D0 03      BNE  $EE22   No, then skip
EE1F:  20 B0 EE   JSR  $EEB0   Turn recorder motor off
EE22:  A9 50      LDA  # $50   Initialize zero-page counter for
EE24:  85 A7      STA  * $A7   The "shorts"
EE26:  A2 08      LDX  # $08   Displacement for IRQ #1 (write)
EE28:  78         SEI          Disable all system interrupts
```

```
EE29:  20 9B EE   JSR  $EE9B      Set the IRQ vectors
EE2C:  D0 EA      BNE  $EE18      Back to the normal IRQ routine

****************************      Write the header (IRQ #1)

EE2E:  A9 78      LDA  # $78      Code for "header pulse" in acc
EE30:  20 72 ED   JSR  $ED72      And write header pulse
EE33:  D0 E3      BNE  $EE18      If first half wave, to normal IRQ
EE35:  C6 A7      DEC  * $A7      Decrement header counter by 1
EE37:  D0 DF      BNE  $EE18      No end, to normal IRQ routine
EE39:  20 5A ED   JSR  $ED5A      Set bit counter for serial output
EE3C:  C6 AB      DEC  * $AB      Dur. of short before & after data
EE3E:  10 D8      BPL  $EE18      No end, to normal IRQ routine
EE40:  A2 0A      LDX  # $0A      Displacement for IRQ #2 (write)
EE42:  20 9B EE   JSR  $EE9B      Set the IRQ vector
EE45:  58         CLI             Enable all system interrupts
EE46:  E6 AB      INC  * $AB      Decrement duration of shorts
EE48:  A5 BE      LDA  * $BE      Check if all blocks written
EE4A:  F0 49      BEQ  $EE95      Yes, then skip
EE4C:  20 51 ED   JSR  $ED51      Copy input/output end address
EE4F:  A2 09      LDX  # $09      Reset the zero-page counter for
EE51:  86 A5      STX  * $A5      the Sync with #9 and reset the
EE53:  86 B6      STX  * $B6      "block written" pointer
EE55:  D0 82      BNE  $EDD9      Unconditional jump

****************************      End recorder operation

EE57:  08         PHP             Save processor status on stack
EE58:  78         SEI             Disable all system interrupts
EE59:  AD 11 D0   LDA  $D011      Contents of VIC control reg in A
EE5C:  0D 39 0A   ORA  $0A39      Combine with VDC temp pointer
EE5F:  29 7F      AND  # $7F      Turn screen off
EE61:  8D 11 D0   STA  $D011      And write value in VIC reg
EE64:  2C 3A 0A   BIT  $0A3A      Check IRQ storage
EE67:  30 16      BMI  $EE7F      Bit 7 set, then skip
EE69:  2C 37 0A   BIT  $0A37      Check clock frequency storage
EE6C:  10 11      BPL  $EE7F      Bit 7 cleared, then no update
EE6E:  AD 38 0A   LDA  $0A38      Get status for sprites
EE71:  8D 15 D0   STA  $D015      And set sprite display register
EE74:  AD 37 0A   LDA  $0A37      Get saved clock frequency and
```

```
EE77:  8D 30 D0   STA  $D030      Set system back to old value
EE7A:  A9 00      LDA  # $00      Clear storage for
EE7C:  8D 37 0A   STA  $0A37      System clock frequency
EE7F:  20 B0 EE   JSR  $EEB0      Turn cassette motor off
EE82:  20 B8 E1   JSR  $E1B8      Set timing and CIAs to standard
EE85:  AD 0A 0A   LDA  $0A0A      Is interrupt vector to standard?
EE88:  F0 09      BEQ  $EE93      Yes, then exit
EE8A:  8D 15 03   STA  $0315      Sys IRQ vector high to standard
EE8D:  AD 09 0A   LDA  $0A09      Get IRQ address low
EE90:  8D 14 03   STA  $0314      Sys IRQ vector low to standard
EE93:  28         PLP             Get processor status back
EE94:  60         RTS             Return from subroutine

******************************    Terminate tape operation

EE95:  20 57 EE   JSR  $EE57      End recorder operation
EE98:  4C 33 FF   JMP  $FF33      Back to normal IRQ routine

******************************    Set the IRQ vector

EE9B:  BD A0 EE   LDA  $EEA0,X    X-indexed IRQ lo-addr f/ table
EE9E:  8D 14 03   STA  $0314      Copy into sys IRQ vector low
EEA1:  BD A1 EE   LDA  $EEA1,X    X-indexed IRQ high addr f/ table
EEA4:  8D 15 03   STA  $0315      Copy into sys IRQ vector high
EEA7:  60         RTS             Return from subroutine

******************************    Table of IRQ vectors

EEA8:  2E EE      ($EE2E)         IRQ #1: Write to tape (header)
EEAA:  90 ED      ($ED90)         IRQ #2: Write to tape (buffer)
EEAC:  65 FA      ($FA65)         Normal IRQ for keyboard read
EEAE:  EB EA      ($EAEB)         IRQ for reading from tape

******************************    Turn recorder motor off

EEB0:  A5 01      LDA  * $01      Status of processor port data reg
EEB2:  09 20      ORA  # $20      In acc, set bit 5 and
EEB4:  85 01      STA  * $01      Turn the recorder motor off
EEB6:  60         RTS             Return from subroutine
```

```
****************************        Check if end address reached
                                    If end address > start addr. C=0

EEB7:  38          SEC              Set carry for subtraction
EEB8:  A5 AC       LDA  * $AC       Low of I/O start address in acc
EEBA:  E5 AE       SBC  * $AE       Subtract low of I/O end address
EEBC:  A5 AD       LDA  * $AD       High of I/O start address in acc
EEBE:  E5 AF       SBC  * $AF       Subtract high of I/O end address
EEC0:  60          RTS              Return from subroutine

****************************        Incr. input/output start address

EEC1:  E6 AC       INC  * $AC       Low value of I/O start addr.+ 1
EEC3:  D0 02       BNE  $EEC7       No overflow in low value, exit
EEC5:  E6 AD       INC  * $AD       High value of I/O address + 1
EEC7:  60          RTS              Return from subroutine

****************************        Clear break flag in processor
                                    status

EEC8:  08          PHP              Put processor status on stack
EEC9:  68          PLA              And copy back into acc
EECA:  29 EF       AND  # $EF       Clear break flag
EECC:  48          PHA              And put status back on stack
EECD:  4C 17 FF    JMP  $FF17       Jump to kernal IRQ routine

****************************        Check cassette recorder keys
                                    (IRQ)

EED0:  A5 01       LDA  * $01       Get processor port data register
EED2:  29 10       AND  # $10       And test if key pressed
EED4:  F0 0A       BEQ  $EEE0       No key pressed, then exit
EED6:  A0 00       LDY  # $00       Indicator for cassette recorder
EED8:  84 C0       STY  * $C0       Reset OFF in zero-page tape flag
EEDA:  A5 01       LDA  * $01       Get processor port data register
EEDC:  09 20       ORA  # $20       And set bit for motor off
EEDE:  D0 08       BNE  $EEE8       Unconditional jump
EEE0:  A5 C0       LDA  * $C0       Check z-page tape flag for motor
EEE2:  D0 06       BNE  $EEEA       If motor on, then skip
EEE4:  A5 01       LDA  * $01       Get processor port data register
```

```
EEE6:  29 DF      AND  # $DF      And clear bit for motor on
EEE8:  85 01      STA  * $01      Write back into processor port
EEEA:  60         RTS             Return from subroutine

*****************************     Kernal routine: GETIN
                                  Read a character

EEEB:  A5 99      LDA  * $99      Load acc with current input dev.
EEED:  D0 0A      BNE  $EEF9      Not keyboard, then continue
EEEF:  A5 D0      LDA  * $D0      Num. of char in keyboard buffer
EEF1:  05 D1      ORA  * $D1      Combine with function key pntr
EEF3:  F0 0F      BEQ  $EF04      No char there, then "OK" exit
EEF5:  78         SEI             Disable all system interrupts
EEF6:  4C 06 C0   JMP  $C006      Get char from keyboard buffer

*****************************     GETIN evaluation not RS-232

EEF9:  C9 02      CMP  # $02      Check if RS-232 is the input
device
EEFB:  D0 18      BNE  $EF15      Not RS-232, to BASIN routine
EEFD:  84 97      STY  * $97      Store current contents of Y-reg
EEFF:  20 CE E7   JSR  $E7CE      GETIN routine of RS-232
EF02:  A4 97      LDY  * $97      Get old contents of Y-reg back
EF04:  18         CLC             Set marker for everything OK
EF05:  60         RTS             Return from subroutine

*****************************     Kernal routine: BASIN
                                  Read character

EF06:  A5 99      LDA  * $99      Load acc with current input dev.
EF08:  D0 0B      BNE  $EF15      Not keyboard, then continue
EF0A:  A5 EC      LDA  * $EC      Get current cursor column in acc
EF0C:  85 E9      STA  * $E9      In z-page start of input column
EF0E:  A5 EB      LDA  * $EB      Get current cursor line in acc
EF10:  85 E8      STA  * $E8      In zero page start of input line
EF12:  4C 09 C0   JMP  $C009      Get character from screen
EF15:  C9 03      CMP  # $03      Check if input device is screen
EF17:  D0 09      BNE  $EF22      Not screen, then continue
EF19:  85 D6      STA  * $D6      In zero-page pointer for input/get
EF1B:  A5 E7      LDA  * $E7      Load right window-border in acc
```

```
EF1D:  85 EA      STA  * $EA      In zero page for end of input line
EF1F:  4C 09 C0   JMP  $C009      Get character from screen
EF22:  B0 38      BCS  $EF5C      Dev>3, read char from serial bus
EF24:  C9 02      CMP  # $02      Input device 2 (RS-232) set?
EF26:  F0 3F      BEQ  $EF67      Yes, then get char from RS-232
EF28:  86 97      STX  * $97      Save current contents of X-reg
EF2A:  20 48 EF   JSR  $EF48      Read a character from cassette
EF2D:  B0 16      BCS  $EF45      Exit from routine: Read cassette
EF2F:  48         PHA             Save acc contents on stack
EF30:  20 48 EF   JSR  $EF48      Read a character from cassette
EF33:  B0 0D      BCS  $EF42      Error occurred, then skip
EF35:  D0 05      BNE  $EF3C      Last character read from tape?
EF37:  A9 40      LDA  # $40      Put EOF marker in acc
EF39:  20 57 F7   JSR  $F757      And set STATUS accordingly
EF3C:  C6 A6      DEC  * $A6      Decrement tape buffer pointer
EF3E:  A6 97      LDX  * $97      Get X-reg contents back
EF40:  68         PLA             Get acc contents back from stack
EF41:  60         RTS             Return from subroutine

****************************      Error occurred reading from tape

EF42:  AA         TAX             Put error number in X-reg
EF43:  68         PLA             Get character
EF44:  8A         TXA             Put error number in acc
EF45:  A6 97      LDX  * $97      Restore x-reg contents
EF47:  60         RTS             Return from subroutine

****************************      Read a character from cassette

EF48:  20 BE E9   JSR  $E9BE      Increment tape buffer pointer
EF4B:  D0 0B      BNE  $EF58      Still chars in buffer, then read
EF4D:  20 F2 E9   JSR  $E9F2      Read next block from cassette
EF50:  B0 09      BCS  $EF5B      STOP key pressed, then stop
EF52:  A9 00      LDA  # $00      Load acc with $00 & in z-page
EF54:  85 A6      STA  * $A6      Storage for cassette buffer pntr
EF56:  F0 F0      BEQ  $EF48      Get next character
EF58:  B1 B2      LDA  ($B2),Y    Read a character from the buffer
EF5A:  18         CLC             Set indicator for "OK"
EF5B:  60         RTS             Return from subroutine
```

```
******************************    Get character from serial bus

EF5C:  A5 90      LDA  * $90      Load system status in acc
EF5E:  D0 03      BNE  $EF63      Status not OK, then exit
EF60:  4C 3E E4   JMP  $E43E      Kernal ACPTR: get byte from
                                  serial bus
EF63:  A9 0D      LDA  # $0D      Load code for <CR> in acc
EF65:  18         CLC             Set indicator for OK
EF66:  60         RTS             Return from subroutine

******************************    Get character from RS-232

EF67:  20 FD EE   JSR  $EEFD      Read a byte from RS-232
EF6A:  B0 F9      BCS  $EF65      Error occurred, then exit
EF6C:  C9 00      CMP  # $00      Was character read a zero-byte?
EF6E:  D0 F6      BNE  $EF66      No, then OK exit
EF70:  AD 14 0A   LDA  $0A14      Load RS-232 status in acc
EF73:  29 60      AND  # $60      Data set ready (DSR) missing?
EF75:  D0 EC      BNE  $EF63      Yes, then return <CR> code
EF77:  F0 EE      BEQ  $EF67      No, then new read attempt

******************************    Kernal routine: BSOUT
                                  (character out)

EF79:  48         PHA             Store character to output
EF7A:  A5 9A      LDA  * $9A      Get curent output device
EF7C:  C9 03      CMP  # $03      Is it the screen (3)?
EF7E:  D0 04      BNE  $EF84      No, then skip screen output
EF80:  68         PLA             Get character to output
EF81:  4C 0C C0   JMP  $C00C      In routine: Char output screen

******************************    BSOUT output not to screen

EF84:  90 04      BCC  $EF8A      Output to RS-232 / Datassette
EF86:  68         PLA             Get character
EF87:  4C 03 E5   JMP  $E503      BSOUT output to serial (DA> 3)
EF8A:  4A         LSR  A          Test if RS-232 or datasette
EF8B:  68         PLA             Get character to output
EF8C:  85 9E      STA  * $9E      And store in zero page
EF8E:  8A         TXA             Save current contents of X-reg
```

```
EF8F:  48          PHA             On stack via acc
EF90:  98          TYA             Save current contents of Y-reg
EF91:  48          PHA             On stack via acc
EF92:  90 23       BCC  $EFB7      Jump to the RS-232 output
EF94:  20 BE E9    JSR  $E9BE      Increment tape buffer pointer
EF97:  D0 0E       BNE  $EFA7      Buffer not full, char in buffer
EF99:  20 15 EA    JSR  $EA15      Write buffer to tape
EF9C:  B0 0E       BCS  $EFAC      If STOP key pressed, stop
EF9E:  A9 02       LDA  # $02      Set control byte for data block
EFA0:  A0 00       LDY  # $00      Set displacement to tape buffer
EFA2:  91 B2       STA  ($B2),Y    And write control byte to buffer
EFA4:  C8          INY             Increment the displacement to
EFA5:  84 A6       STY  * $A6      the tape buffer and store in Z-P
EFA7:  A5 9E       LDA  * $9E      Character to output from Z-P
EFA9:  91 B2       STA  ($B2),Y    Write in output buffer
EFAB:  18          CLC             Set indicator for OK
EFAC:  68          PLA             Restore old values from stack
EFAD:  A8          TAY             Restore Y-reg contents
EFAE:  68          PLA             Restore
EFAF:  AA          TAX             X-reg contents
EFB0:  A5 9E       LDA  * $9E      Get character to output
EFB2:  90 02       BCC  $EFB6      Everything OK, then return
EFB4:  A9 00       LDA  # $00      Flag for "STOP" key pressed
EFB6:  60          RTS             Return from subroutine

******************************     Output RS-232 character

EFB7:  20 5F E7    JSR  $E75F      Write character in RS-232 buffer
EFBA:  4C AB EF    JMP  $EFAB      Clean up stack and return

******************************     Kernal routine: OPEN
                                   Open a logical file

EFBD:  A6 B8       LDX  * $B8      Get logical file number in X-reg
EFBF:  20 02 F2    JSR  $F202      Find LFN in LFN table
EFC2:  F0 2F       BEQ  $EFF3      Found, then output error
EFC4:  A6 98       LDX  * $98      Get number of open files
EFC6:  E0 0A       CPX  # $0A      Max of 10 open are possible
EFC8:  B0 26       BCS  $EFF0      More than 10 open, then error
EFCA:  E6 98       INC  * $98      Number of open files +1
```

```
EFCC:  A5 B8      LDA  * $B8      Get logical file number in acc
EFCE:  9D 62 03   STA  $0362,X    Enter LFN in LFN table
EFD1:  A5 B9      LDA  * $B9      Get secondary address in acc
EFD3:  09 60      ORA  # $60      Set Print, Input, Get in SA
EFD5:  85 B9      STA  * $B9      And store in SA mem again
EFD7:  9D 76 03   STA  $0376,X    Enter SA in SA table
EFDA:  A5 BA      LDA  * $BA      Load device address in acc
EFDC:  9D 6C 03   STA  $036C,X    GA in GA-Table
EFDF:  F0 0D      BEQ  $EFEE      Was it the keyboard (0), skip
EFE1:  C9 02      CMP  # $02      Check if RS-232 selected as dev
EFE3:  F0 5B      BEQ  $F040      Yes, then skip to RS-232
EFE5:  90 0F      BCC  $EFF6      Less than 2, it is tape OPEN
EFE7:  C9 03      CMP  # $03      Check if screen selected as dev
EFE9:  F0 03      BEQ  $EFEE      Yes, then skip
EFEB:  20 CB F0   JSR  $F0CB      Open file on serial bus
EFEE:  18         CLC             Set marker for everything OK
EFEF:  60         RTS             Return from subroutine
```

```
*****************************   Open routine for tape operation

EFF0:  4C 7C F6   JMP  $F67C    I/O error #1 (Too many files)
EFF3:  4C 7F F6   JMP  $F67F    I/O error #2 (File open)
EFF6:  20 80 E9   JSR  $E980    Get tape buffer start address
EFF9:  B0 03      BCS  $EFFE    Carry set, then valid address
EFFB:  4C 94 F6   JMP  $F694    I/O error #9 (Illegal device num)
EFFE:  A5 B9      LDA  * $B9    Get secondary address in acc
F000:  29 0F      AND  # $0F    Mask out upper nibble (4-7)
F002:  D0 1F      BNE  $F023    Not zero, wait for record & play
F004:  20 C8 E9   JSR  $E9C8    Wait for key on datasette
F007:  B0 36      BCS  $F03F    Invaid, then carry = 1, RTS
F009:  20 0F F5   JSR  $F50F    Message "SEARCHING FOR"
F00C:  A5 B7      LDA  * $B7    Length of filename in acc
F00E:  F0 0A      BEQ  $F01A    No filename present, then skip
F010:  20 9A E9   JSR  $E99A    Find corresponding tape header
F013:  90 18      BCC  $F02D    Not found, then continue
F015:  F0 28      BEQ  $F03F    Return with carry set
F017:  4C 85 F6   JMP  $F685    I/O error #4 (File not found)
F01A:  20 D0 E8   JSR  $E8D0    Find next header on cassette
F01D:  90 0E      BCC  $F02D    If found then continue
F01F:  F0 1E      BEQ  $F03F    Return w/ carry on because EOT
F021:  B0 F4      BCS  $F017    Cont search because a PRG file
F023:  20 E9 E9   JSR  $E9E9    Wait for record & play buttons
F026:  B0 17      BCS  $F03F    STOP key pressed, then stop
F028:  A9 04      LDA  # $04    Control code-data header in acc
F02A:  20 19 E9   JSR  $E919    Write tape header to cassette
F02D:  A9 BF      LDA  # $BF    Pointer to end of tape buffer in A
F02F:  A4 B9      LDY  * $B9    Get secondary address in Y-reg
F031:  C0 60      CPY  # $60    SA code for print, input, or get?
F033:  F0 07      BEQ  $F03C    Yes, then set pointer and RTS
F035:  A0 00      LDY  # $00    Set displacement for tape buffer
F037:  A9 02      LDA  # $02    Control byte for data block
F039:  91 B2      STA  ($B2),Y  Write into cassette buffer
F03B:  98         TYA           Copy displacement from Y to A
F03C:  85 A6      STA  * $A6    And set zero page tape buffer
F03E:  18         CLC           Set indicator for OK
F03F:  60         RTS           Return from subroutine
```

```
****************************** RS-232 Open

F040:  20 B0 F0   JSR  $F0B0     Reset CIAs
F043:  8C 14 0A   STY  $0A14     Clear Z-P RS-232 status byte
F046:  C4 B7      CPY  * $B7     Compare with lengh of filename
F048:  F0 0B      BEQ  $F055     Equal zero, calculate data bits
F04A:  20 AE F7   JSR  $F7AE     Get 1 byte for RS-232 register
F04D:  99 10 0A   STA  $0A10,Y   Init. RS-232 control register,
F050:  C8         INY            Command register, and the
F051:  C0 04      CPY  # $04     User baud rate
F053:  D0 F1      BNE  $F046     Loop until 4 values transferred
F055:  20 8E E6   JSR  $E68E     Calculate number of data bits
F058:  8E 15 0A   STX  $0A15     Storage number of bits to send
F05B:  AD 10 0A   LDA  $0A10     Load RS-232 control register
F05E:  29 0F      AND  # $0F     Isolate bits for baud rate
F060:  F0 1C      BEQ  $F07E     Determine code value - baud rate
F062:  0A         ASL  A         Multiply by 2 for table displace
F063:  AA         TAX            Copy to X-reg for index
F064:  AD 03 0A   LDA  $0A03     Get PAL/NTSC pointer
F067:  D0 09      BNE  $F072     Not NTSC version, then skip
F069:  BC 4F E8   LDY  $E84F,X   Timer constant RS-232 b-rate
                                 NTSC Hi
F06C:  BD 4E E8   LDA  $E84E,X   Timer constant RS-232 b-rate
                                 NTSC Lo
F06F:  4C 78 F0   JMP  $F078     Skip to save baud rate
F072:  BC 63 E8   LDY  $E863,X   Timer constant RS-232 b-rate
                                 PAL Hi
F075:  BD 62 E8   LDA  $E862,X   Timer constant RS-232 b-rate
                                 PAL Lo
F078:  8C 13 0A   STY  $0A13     Store high value of baud rate
F07B:  8D 12 0A   STA  $0A12     Store low value of baud rate
F07E:  AD 12 0A   LDA  $0A12     Get low value baud rate
F081:  0A         ASL  A         And multiply by 2
F082:  AA         TAX            Store value in X-reg
F083:  AD 13 0A   LDA  $0A13     Get high value of baudrate
F086:  2A         ROL  A         And multiply by 2
F087:  A8         TAY            Store value in Y-reg
F088:  8A         TXA            Low val code determine in acc
F089:  69 C8      ADC  # $C8     Add decimal 200
F08B:  8D 16 0A   STA  $0A16     Store timer val transmit baud rate
```

```
F08E:  98          TYA          High val code determine in acc
F08F:  69 00       ADC  # $00   Add decimal 000
F091:  8D 17 0A    STA  $0A17   Store timer value - transmit rate
F094:  AD 11 0A    LDA  $0A11   Get RS-232 command register
F097:  4A          LSR  A       Check for 3-line handshake
F098:  90 09       BCC  $F0A3   Yes, then skip DSR test
F09A:  AD 01 DD    LDA  $DD01   Check if DATA SET READY
F09D:  0A          ASL  A       (DSR) signal missing
F09E:  B0 03       BCS  $F0A3   No, then skip
F0A0:  20 55 E7    JSR  $E755   Set status for DSR
F0A3:  AD 18 0A    LDA  $0A18   Set start of RS-232 input buffer
F0A6:  8D 19 0A    STA  $0A19   equal to end of input buffer
F0A9:  AD 1B 0A    LDA  $0A1B   Set start of RS-232 out. buffer
F0AC:  8D 1A 0A    STA  $0A1A   equal to end of output buffer
F0AF:  60          RTS          Return from subroutine

******************************  Reset CIAs to RS-232

F0B0:  A9 7F       LDA  # $7F   Value for "clr interrupts" in acc
F0B2:  8D 0D DD    STA  $DD0D   Reset IRQs
F0B5:  A9 06       LDA  # $06   Set bits 1 and 2 to output
F0B7:  8D 03 DD    STA  $DD03   Data direction register port B
F0BA:  8D 01 DD    STA  $DD01   Port register port B
F0BD:  A9 04       LDA  # $04   Set bit 2 of data port A (CIA 2)
F0BF:  0D 00 DD    ORA  $DD00   For the RS-232 data output
F0C2:  8D 00 DD    STA  $DD00   (TXD Signal)
F0C5:  A0 00       LDY  # $00   Load Y with $00 and clear the
F0C7:  8C 0F 0A    STY  $0A0F   RS-232 NMI flag
F0CA:  60          RTS          Return from subroutine

******************************  Open file on serial bus

F0CB:  A5 B9       LDA  * $B9   Load secondary address in acc
F0CD:  30 04       BMI  $F0D3   If bit 7 set for "CLOSE", exit
F0CF:  A4 B7       LDY  * $B7   Get length of filename
F0D1:  D0 02       BNE  $F0D5   Not zero, then continue
F0D3:  18          CLC          Clear carry for OK indicator
F0D4:  60          RTS          Return from subroutine
```

```
*****************************      Send filename on serial bus

F0D5:  A9 00      LDA  # $00       Set the status byte to the
F0D7:  85 90      STA  * $90       Marker $00 (= everything OK)
F0D9:  A5 BA      LDA  * $BA       Load device address in acc
F0DB:  20 3E E3   JSR  $E33E       Wait for end of RS-232 transfer
F0DE:  24 90      BIT  * $90       Test STATUS for set EOF bit
F0E0:  30 0B      BMI  $F0ED       If EOF, then output error
F0E2:  A5 B9      LDA  * $B9       Load secondary address in acc
F0E4:  09 F0      ORA  # $F0       Set control nibble in SA
F0E6:  20 D2 E4   JSR  $E4D2       Rout. SECND: SA for LISTEN
F0E9:  A5 90      LDA  * $90       Load system STATUS in acc
F0EB:  10 05      BPL  $F0F2       If OK, continue as normal
F0ED:  68         PLA              Remove RTS address from stack
F0EE:  68         PLA              Remove RTS address from stack
F0EF:  4C 88 F6   JMP  $F688       I/O error #5 (Device not present)
F0F2:  A5 B7      LDA  * $B7       Get length of filename
F0F4:  F0 0D      BEQ  $F103       No name given, then skip
F0F6:  A0 00      LDY  # $00       Displ. to first char of filename
F0F8:  20 AE F7   JSR  $F7AE       Read 1 character of filename
F0FB:  20 03 E5   JSR  $E503       Krnal CIOUT: byte to serial bus
F0FE:  C8         INY              Increment displacement pointer
F0FF:  C4 B7      CPY  * $B7       Displacement = filename length?
F101:  D0 F5      BNE  $F0F8       No, then continue to output
F103:  4C B0 F5   JMP  $F5B0       UNLSN on serial bus and RTS

*****************************      Kernal routine: CHKIN
                                   Set input channel

F106:  20 02 F2   JSR  $F202       Search for LFN in LFN table
F109:  D0 3E      BNE  $F149       I/O error #3 (File not found)
F10B:  20 12 F2   JSR  $F212       Reset LFN,DA,SA
F10E:  F0 13      BEQ  $F123       DA = 0, then set standard
F110:  C9 03      CMP  # $03       Is it the DA 3 (= screen )?
F112:  F0 0F      BEQ  $F123       Yes, then set screen for standard
F114:  B0 11      BCS  $F127       Greater than 3, then serial eval.
F116:  C9 02      CMP  # $02       Check if RS-232 selected
F118:  D0 03      BNE  $F11D       No, then it was the datasette
F11A:  4C 95 E7   JMP  $E795       To RS-233 input
F11D:  A6 B9      LDX  * $B9       Get secondary address in X-reg
```

```
F11F:  E0 60      CPX  # $60      Is the secondary address = 0?
F121:  D0 20      BNE  $F143      I/O error #6 (Not input file)
F123:  85 99      STA  * $99      In Z-P for standard input device
F125:  18         CLC             Set indicator for OK
F126:  60         RTS             Return from subroutine

******************************    Evaluation for CHKIN on serial

F127:  AA         TAX             Store device address in X
F128:  20 3B E3   JSR  $E33B      Rout. TALK: cmd to serial bus
F12B:  24 90      BIT  * $90      Test STATUS for set EOF bit
F12D:  30 11      BMI  $F140      Bit 7 set = "Device not present"
F12F:  A5 B9      LDA  * $B9      Load secondary address in acc
F131:  10 05      BPL  $F138      Send secondary addr. for TALK
F133:  20 E9 E4   JSR  $E4E9      Wait for clock signal
F136:  10 03      BPL  $F13B      Skip output of TALK sec. addr.
F138:  20 E0 E4   JSR  $E4E0      Routine TKSA: sec addr for talk
F13B:  8A         TXA             Get device addr. back from acc
F13C:  24 90      BIT  * $90      Test STATUS for set EOF bit
F13E:  10 E3      BPL  $F123      Everthing OK,  set input device
F140:  4C 88 F6   JMP  $F688      I/O error #5 (Device not present)
F143:  4C 8B F6   JMP  $F68B      I/O error #6 (Not input file)
F146:  4C 8E F6   JMP  $F68E      I/O error #7 (Not output file)
F149:  4C 82 F6   JMP  $F682      I/O error #3 (File not open)

******************************    Kernal routine: CKOUT
                                  Set output channel

F14C:  20 02 F2   JSR  $F202      Search for LFN in LFN table
F14F:  D0 F8      BNE  $F149      I/O Eerror #3 (File not open)
F151:  20 12 F2   JSR  $F212      Reset LFN, DA, SA
F154:  F0 F0      BEQ  $F146      I/O error #7 (Not output file)
F156:  C9 03      CMP  # $03      Compare with DAA 3 (= screen)
F158:  F0 0F      BEQ  $F169      Yes, then set as standard output
F15A:  B0 11      BCS  $F16D      DA > 3, then serial evaluation
F15C:  C9 02      CMP  # $02      Check if RS-232 selected
F15E:  D0 03      BNE  $F163      No, then skip
F160:  4C 29 E7   JMP  $E729      To RS-232 output
F163:  A6 B9      LDX  * $B9      Get secondary address in X-reg
F165:  E0 60      CPX  # $60      Is the secondary address = 0?
```

```
F167:  F0 DD      BEQ  $F146     I/O error #7 (Not output file)
F169:  85 9A      STA  * $9A     In Z-P for standard out device
F16B:  18         CLC            Set indicator for OK
F16C:  60         RTS            Return from subroutine

****************************     Evaluation for CKOUT on serial

F16D:  AA         TAX            Dev addr. for LISTN in X-reg
F16E:  20 3E E3   JSR  $E33E     Rout LISTN: cmd to serial
F171:  24 90      BIT  * $90     Test STATUS for set EOF bit
F173:  30 CB      BMI  $F140     I/O error #5 (Device not present)
F175:  A5 B9      LDA  * $B9     Load secondary address in acc
F177:  10 05      BPL  $F17E     OPEN/CLOSE bit clr, then skip
F179:  20 D7 E4   JSR  $E4D7     Reset ATN signal
F17C:  D0 03      BNE  $F181     Skip output of listen sec. addr
F17E:  20 D2 E4   JSR  $E4D2     Rout SECND: sec. addr for listn
F181:  8A         TXA            Device address back in acc
F182:  24 90      BIT  * $90     Test status for set EOF bit
F184:  10 E3      BPL  $F169     Everything OK, then RTS
F186:  30 B8      BMI  $F140     I/O error #5 (Device not present)

****************************     Kernal routine: CLOSE
                                 Close a file

F188:  66 92      ROR  * $92     Rotate carry as marker, Z-P flag
F18A:  20 07 F2   JSR  $F207     Search for LFN in LFN table
F18D:  D0 DC      BNE  $F16B     Not found, then OK return
F18F:  20 12 F2   JSR  $F212     LFN,DA,SA renew corr. tables
F192:  8A         TXA            Table displacement pointer
F193:  48         PHA            Save on stack
F194:  A5 BA      LDA  * $BA     Load device address in acc
F196:  F0 4C      BEQ  $F1E4     Addressed device the keyboard?
F198:  C9 03      CMP  # $03     Check if device addressed was
F19A:  F0 48      BEQ  $F1E4     Screen (3)Yes, then skip
F19C:  B0 31      BCS  $F1CF     Was it a device on the serial bus?
F19E:  C9 02      CMP  # $02     Was it the RS-232?
F1A0:  D0 07      BNE  $F1A9     No, then close on cassette
F1A2:  68         PLA            Get the displacement to the table
F1A3:  20 E5 F1   JSR  $F1E5     Delete file entry from table
F1A6:  4C B0 F0   JMP  $F0B0     Reset CIAs and RTS
```

```
******************************      Close a tape file

F1A9:  A5 B9      LDA  * $B9      Load secondary address in acc
F1AB:  29 0F      AND  # $0F      Mask out upper nibble (4-7)
F1AD:  F0 35      BEQ  $F1E4      Delete file entry from table
F1AF:  20 80 E9   JSR  $E980      Get tape buffer address & check
F1B2:  A9 00      LDA  # $00      Set marker for close and
F1B4:  38         SEC             Set control marker carry
F1B5:  20 8C EF   JSR  $EF8C      Write character in buffer
F1B8:  20 15 EA   JSR  $EA15      Write buffer to tape
F1BB:  90 04      BCC  $F1C1      All OK, continue with tape close
F1BD:  68         PLA             Get character output back
F1BE:  A9 00      LDA  # $00      Replace with CHR$(0)
F1C0:  60         RTS             Return from subroutine

******************************      Delete file entry

F1C1:  A5 B9      LDA  * $B9      Load secondary address in acc
F1C3:  C9 62      CMP  # $62      Lower nibble of the SA = 2?
F1C5:  D0 1D      BNE  $F1E4      Delete file entry from table
F1C7:  A9 05      LDA  # $05      Set control byte for EOT header
F1C9:  20 19 E9   JSR  $E919      Write data block to tape
F1CC:  4C E4 F1   JMP  $F1E4      Delete file entry from table
F1CF:  24 92      BIT  * $92      Check tape time constant
F1D1:  10 0E      BPL  $F1E1      Less than 128, then send close
F1D3:  A5 BA      LDA  * $BA      Load device address into acc
F1D5:  C9 08      CMP  # $08      Was it a disk drive (8-15)
F1D7:  90 08      BCC  $F1E1      No, then skip disk close
F1D9:  A5 B9      LDA  * $B9      Load secondary address into acc
F1DB:  29 0F      AND  # $0F      Mask out upper nibble (bits 4-7)
F1DD:  C9 0F      CMP  # $0F      Was cmd channel (15) opened
F1DF:  F0 03      BEQ  $F1E4      then delete file entry from table
F1E1:  20 9E F5   JSR  $F59E      Send CLOSE cmd to device

******************************      Delete file entry from table

F1E4:  68         PLA             Get displacement to table
F1E5:  AA         TAX             Copy displacement from A to X
F1E6:  C6 98      DEC  * $98      Number of open files - 1
```

```
F1E8:  E4 98      CPX  * $98      Was the table entry found the
F1EA:  F0 14      BEQ  $F200      Last table entry? Then exit
F1EC:  A4 98      LDY  * $98      Get num. of open files for displ.
F1EE:  B9 62 03   LDA  $0362,Y    Get last entry from LFN table
F1F1:  9D 62 03   STA  $0362,X    And copy to free position
F1F4:  B9 6C 03   LDA  $036C,Y    Get last entry in DA table
F1F7:  9D 6C 03   STA  $036C,X    And copy to free position
F1FA:  B9 76 03   LDA  $0376,Y    Get last entry from SA table
F1FD:  9D 76 03   STA  $0376,X    And copy to free position
F200:  18         CLC            Set indicator for OK
F201:  60         RTS            Return from subroutine

******************************    Search for LFX in X in LFN
                                  table

F202:  A9 00      LDA  # $00      Clear status byte and
F204:  85 90      STA  * $90      Set indicator for everything OK
F206:  8A         TXA            Copy target for LFN in acc
F207:  A6 98      LDX  * $98      Get number of open files
F209:  CA         DEX            Dec by 1, because used as index
F20A:  30 05      BMI  $F211      All comparisons negative, exit
F20C:  DD 62 03   CMP  $0362,X    Cmp with byte from LFN table
F20F:  D0 F8      BNE  $F209      Not equal, then next comparison
F211:  60         RTS            Return from subroutine

******************************    LFN,DA,SA corresponding to
                                  the X-reg
                                  Get displacement to tables

F212:  BD 62 03   LDA  $0362,X    The logical file number specified
F215:  85 B8      STA  * $B8      by X-reg in z-page byte for LFN
F217:  BD 76 03   LDA  $0376,X    The secondary address specified
F21A:  85 B9      STA  * $B9      by X-reg in  z-page byte for SA
F21C:  BD 6C 03   LDA  $036C,X    The device address specified by t
F21F:  85 BA      STA  * $BA      X-reg in zero-page byte for DA
F221:  60         RTS            Return from subroutine
```

```
*****************************      Kernal routine: CLALL
                                   Reset all open files

F222:  A9 00      LDA  # $00       Load acc with 0 and in zero-page
F224:  85 98      STA  * $98       Storage for number of open files

*****************************      Kernal routine: CLRCH
                                   Reset input/output channel

F226:  A2 03      LDX  # $03       Load code for device screen (3)
F228:  E4 9A      CPX  * $9A       Cmp with current output dev in
F22A:  B0 03      BCS  $F22F       CLRCH rout - dev on serial bus
F22C:  20 26 E5   JSR  $E526       Rout UNLSN:cmd to serial bus
F22F:  E4 99      CPX  * $99       Cmp with current input device in
F231:  B0 03      BCS  $F236       CLRCH rout dev on serial bus
F233:  20 15 E5   JSR  $E515       Rout UNTLK:cmd to serial bus
F236:  86 9A      STX  * $9A       Set  screen as output device and
F238:  A9 00      LDA  # $00       The keyboard as the standard
F23A:  85 99      STA  * $99       Input device
F23C:  60         RTS              Return from subroutine

*****************************      Set standard I/O devices

F23D:  85 BA      STA  * $BA       In Z-P byte for current dev addr
F23F:  C5 9A      CMP  * $9A       Cmp with current output device
F241:  D0 05      BNE  $F248       Not equal, cmp with input dev
F243:  A9 03      LDA  # $03       Load acc with dev addr for
F245:  85 9A      STA  * $9A       Screen (3) & set as output device
F247:  2C         .Byte  $2C       Skip to $F24A
F24(:  C5 99      CMP  # $99       Cmp with current input device
F24A:  D0 04      BNE  $F250       Not equal, search in DA table
F24C:  A9 00      LDA  # $00       Load acc with code for keybaord
F24E:  85 99      STA  * $99       (0) and set the keyboard as input
F250:  A5 BA      LDA  * $BA       Load device address in acc
F252:  A6 98      LDX  * $98       Number of open files in X-reg
F254:  CA         DEX              Decremnt by 1,  used as index
F255:  30 0D      BMI  $F264       All comparions negative,  exit
F257:  DD 6C 03   CMP  $036C,X     Cmp with table for dev addr.
F25A:  D0 F8      BNE  $F254       Not found, then next compare
F25C:  BD 62 03   LDA  $0362,X     Get LFN for corresponding DA
```

```
F25F:  20 C3 FF   JSR  $FFC3      Kernal CLOSE: close file
F262:  90 EC      BCC  $F250      If carry clear, next close
F264:  60         RTS             Return from subroutine

****************************      Kernal routine: LOAD
                                  Load file in a memory range

F265:  86 C3      STX  * $C3      Place start address low in z-page
F267:  84 C4      STY  * $C4      Place start addr. high in z-page
F269:  6C 30 03   JMP  ($0330)    Vector points LOADSP ($F26C)
F26C:  85 93      STA  * $93      Zero-page flag, LOAD/VERIFY
F26E:  A9 00      LDA  # $00      Load acc with $00 and
F270:  85 90      STA  * $90      Set status to everything OK
F272:  A5 BA      LDA  * $BA      Load device address in acc
F274:  C9 04      CMP  # $04      Check for valid device address
F276:  B0 03      BCS  $F27B      Dev addr greater than 4 is valid
F278:  4C 26 F3   JMP  $F326      Check for datasette, else invalid

****************************      Load routine from serial bus

F27B:  AD 1C 0A   LDA  $0A1C      Read sys pointer for fast serial
F27E:  29 BE      AND  # $BE      Mode & eliminate bit 6 (1 = fast,
F280:  8D 1C 0A   STA  $0A1C      0 = slow)
F283:  A6 B9      LDX  * $B9      Get secondary address in X-reg
F285:  86 9E      STX  * $9E      And store in zero page $9F
F287:  A4 B7      LDY  * $B7      Get length of filename
F289:  D0 03      BNE  $F28E      Not zero, skip error message
F28B:  4C 1A F3   JMP  $F31A      I/O error #8 (Missing filename)
F28E:  84 9F      STY  * $9F      Store length of filenames
F290:  20 0F F5   JSR  $F50F      Output "Searching for" message
F293:  20 A1 F3   JSR  $F3A1      Chk filenames & fast serialmode
F296:  B0 03      BCS  $F29B      Carry set, then OK. Skip
F298:  4C 9B F3   JMP  $F39B      Set load end address and RTS
F29B:  A4 9F      LDY  * $9F      Length of filename in Y-reg and
F29D:  84 B7      STY  * $B7      In z-page for length of filename
F29F:  A9 60      LDA  # $60      SA 0, high nibble for Input/Get
F2A1:  85 B9      STA  * $B9      In zero-page for sec. address
F2A3:  20 CB F0   JSR  $F0CB      Send talk command to serial bus
F2A6:  A5 BA      LDA  * $BA      Load device address in acc
F2A8:  20 3B E3   JSR  $E33B      Rout TALK: cmd to serial bus
```

```
F2AB:  A5 B9      LDA  * $B9     Load secondary address into acc
F2AD:  20 E0 E4   JSR  $E4E0     Rout TKSA: Sec addr for TALK
F2B0:  20 3E E4   JSR  $E43E     Get a byte from serial bus
F2B3:  85 AE      STA  * $AE     Place start address in zero page
F2B5:  20 3E E4   JSR  $E43E     Get a byte from serial bus
F2B8:  85 AF      STA  * $AF     Store start addr high in z-page
F2BA:  A5 90      LDA  * $90     Load system status in acc
F2BC:  4A         LSR  A         Shift timeout bit right
F2BD:  4A         LSR  A         Shift timeout bit into carry
F2BE:  B0 57      BCS  $F317     Timeout for read, File not found
F2C0:  A5 9E      LDA  * $9E     Get stored secondary address
F2C2:  D0 08      BNE  $F2CC     Not equal to 0, then skip
F2C4:  A5 C3      LDA  * $C3     Copy the start address given by
F2C6:  85 AE      STA  * $AE     The X and Y registers for the
F2C8:  A5 C4      LDA  * $C4     Load command from $C3,$C4
F2CA:  85 AF      STA  * $AF     To $AE,$AF
F2CC:  20 33 F5   JSR  $F533     Disp. control message on screen
F2CF:  A9 FD      LDA  # $FD     Mask out read timeout bit from
F2D1:  25 90      AND  * $90     Status and write back
F2D3:  85 90      STA  * $90     To status
F2D5:  20 E1 FF   JSR  $FFE1     Kernal STOP: test for STOP key
F2D8:  F0 49      BEQ  $F323     To interruption of load routine
F2DA:  20 3E E4   JSR  $E43E     Kernal routine: ACPTR
F2DD:  AA         TAX            Store acc contents in X
F2DE:  A5 90      LDA  * $90     Load system STATUS in acc
F2E0:  4A         LSR  A         Eliminate the "read timeout" bit
F2E1:  4A         LSR  A         From the status byte
F2E2:  B0 EB      BCS  $F2CF     If timeout, then new read attempt
F2E4:  8A         TXA            Restore old acc contents
F2E5:  A4 93      LDY  * $93     Test z-page load/verify pointer
F2E7:  F0 12      BEQ  $F2FB     If zero, then it's load
F2E9:  85 BD      STA  * $BD     Store in zero page parity buffer
F2EB:  A0 00      LDY  # $00     Displac pointer for FETCH rout
F2ED:  20 C9 F7   JSR  $F7C9     FETCH rout for LSV operations
F2F0:  C5 BD      CMP  * $BD     Compare with Z-P parity buffer
F2F2:  F0 0A      BEQ  $F2FE     If equal, then OK and skip
F2F4:  A9 10      LDA  # $10     Not equal, then OK and skip
F2F6:  20 57 F7   JSR  $F757     Kernal STATUS: Set sys status
F2F9:  D0 03      BNE  $F2FE     Not OK, then skip store
F2FB:  20 BF F7   JSR  $F7BF     Indsta routine via Z-P $AE-$AF
```

```
F2FE:  E6 AE      INC  * $AE     Low byte of memory pointer +1
F300:  D0 08      BNE  $F30A     No overflow, then skip
F302:  E6 AF      INC  * $AF     High byte of memory pointer +1
F304:  A5 AF      LDA  * $AF     Check if high byte points in $
F306:  C9 FF      CMP  # $FF     $FF00 range. If yes, then jump
F308:  F0 16      BEQ  $F320     To error output
F30A:  24 90      BIT  * $90     Test STATUS for set EOF bit
F30C:  50 C1      BVC  $F2CF     No EOF yet, then continue
F30E:  20 15 E5   JSR  $E515     Rout UNTLK: cmd to serial bus
F311:  20 9E F5   JSR  $F59E     Send Unlistn- Close to serial bus
F314:  4C 9B F3   JMP  $F39B     Clear carry and return
F317:  4C 85 F6   JMP  $F685     I/O error #4 (File not found)
F31A:  4C 91 F6   JMP  $F691     I/O error #8 (Missing filename)
F31D:  4C 94 F6   JMP  $F694     I/O error #9 (Illegal device num)
F320:  4C 97 F6   JMP  $F697     I/O error #10
F323:  4C B5 F5   JMP  $F5B5     Jump, LOAD routine interrupted
F326:  C9 01      CMP  # $01     Is it a load from Datassette?
F328:  D0 F3      BNE  $F31D     No, then I/O error #9
F32A:  20 80 E9   JSR  $E980     Get and check tape buffer addr.
F32D:  90 EE      BCC  $F31D     Tape buffer address illegal, error
F32F:  20 C8 E9   JSR  $E9C8     Wait for button on recorder
F332:  B0 6C      BCS  $F3A0     Interrupt STOP key, then RTS
F334:  20 0F F5   JSR  $F50F     Output SEARCH FOR filename
F337:  A5 B7      LDA  * $B7     Z-P storage for filename length
F339:  F0 09      BEQ  $F344     Length =0, skip name search
F33B:  20 9A E9   JSR  $E99A     Seach for tape header after name
F33E:  90 0B      BCC  $F34B     OK, then continue
F340:  F0 5E      BEQ  $F3A0     Interrupt STOP key, then RTS
F342:  B0 D3      BCS  $F317     I/O error #4 (File not found)
F344:  20 D0 E8   JSR  $E8D0     Read program header from tape
F347:  F0 57      BEQ  $F3A0     Interrupt STOP key, then RTS
F349:  B0 CC      BCS  $F317     I/O error #4 (File not found)
F34B:  38         SEC            Marker: Set error found
F34C:  A5 90      LDA  * $90     Load system status in acc
F34E:  29 10      AND  # $10     Eliminate bit 4 for read error
F350:  D0 4E      BNE  $F3A0     Bit 4 set (read error), then RTS
F352:  E0 01      CPX  # $01     Code, header type#1 BASIC prg
F354:  F0 11      BEQ  $F367     If it is a BASIC program, skip
F356:  E0 03      CPX  # $03     Code, header type #3 (ML prg)
F358:  D0 DD      BNE  $F337     If not #1 or #3, continue search
```

```
F35A:  A0 01      LDY  # $01      Displacement to cassette buffer
F35C:  B1 B2      LDA  ($B2),Y    Get start addr low from buffer
F35E:  85 C3      STA  * $C3      And copy it to load addr ptr low
F360:  C8         INY             Displacement in cass buffer +1
F361:  B1 B2      LDA  ($B2),Y    Get start addr high from buffer
F363:  85 C4      STA  * $C4      & copy it in load addr ptr high
F365:  B0 04      BCS  $F36B      Unconditional jump for ML prg
F367:  A5 B9      LDA  * $B9      Load secondary address in acc
F369:  D0 EF      BNE  $F35A      Is it 0 (append)? No, then skip
F36B:  A0 03      LDY  # $03      Displacement to cassette buffer
F36D:  B1 B2      LDA  ($B2),Y    Get end address low from buffer
F36F:  A0 01      LDY  # $01      Displacement to cassette buffer
F371:  F1 B2      SBC  ($B2),Y    Subtract start addr low from end
F373:  AA         TAX             Addr & store low value in X reg
F374:  A0 04      LDY  # $04      Displacement to cassette buffer
F376:  B1 B2      LDA  ($B2),Y    Get end addr high from buffer
F378:  A0 02      LDY  # $02      Displacement to cassette buffer
F37A:  F1 B2      SBC  ($B2),Y    Subtract start addr high from end
F37C:  A8         TAY             Addr & store high value in Y-reg
F37D:  18         CLC             Clear carry for addition
F37E:  8A         TXA             Program length low back in acc
F37F:  65 C3      ADC  * $C3      Memory start addr + prg length
F381:  85 AE      STA  * $AE      Place in pointer for end addr low
F383:  98         TYA             Program length high back in acc
F384:  65 C4      ADC  * $C4      Memory start addr + prg length
F386:  85 AF      STA  * $AF      Place in pntr for end addr high
F388:  C9 FF      CMP  # $FF      Does end addr extend into
F38A:  F0 94      BEQ  $F320      $FF00. Yes, then I/O error #0
F38C:  A5 C3      LDA  * $C3      Copy the memory start address
F38E:  85 C1      STA  * $C1      low into z-page load pointer low
F390:  A5 C4      LDA  * $C4      Copy the memory start addr high
F392:  85 C2      STA  * $C2      Into the z-page load pointer high
F394:  20 33 F5   JSR  $F533      Output LOADING/VERIFYING
F397:  20 FB E9   JSR  $E9FB      Load program from tape
F39A:  24         .Byte  $24      Skip to $F39C
```

```
*****************************   Set prg end address after LOAD

F39B:  18          CLC          Set carry for OK indicator
F39C:  A6 AE       LDX  * $AE   Program end addr low in X-reg
F39E:  A4 AF       LDY  * $AF   Program end addr high in Y-reg
F3A0:  60          RTS          Return from subroutine

*****************************   Check filenames and the
                                 "fast serial mode"

F3A1:  A0 00       LDY  # $00   Set displace for FETCH routine
F3A3:  20 AE F7    JSR  $F7AE   Get byte of filename
F3A6:  C9 24       CMP  # $24   Is first character a <$>?
F3A8:  F0 F6       BEQ  $F3A0   Yes, then return: RTS
F3AA:  A6 BA       LDX  * $BA   Load device address in X-reg
F3AC:  A0 0F       LDY  # $0F   Set secondary address to (15)
F3AE:  A9 00       LDA  # $00   Set logical file number to 0
F3B0:  20 38 F7    JSR  $F738   Rout SETLFS: Set file params.
F3B3:  85 B7       STA  * $B7   Set length of the filename to 0
F3B5:  20 C0 FF    JSR  $FFC0   Kernal OPEN: Open file
F3B8:  A6 B8       LDX  * $B8   Get logical file number in X
F3BA:  20 C9 FF    JSR  $FFC9   Krnl CKOUT: Set output chnl
F3BD:  90 08       BCC  $F3C7   No error, then continue
F3BF:  20 8C F4    JSR  $F48C   Close logical file again
F3C2:  68          PLA          Remove  RTS addr from stack
F3C3:  68          PLA          Remove RTS addr from stack
F3C4:  4C 88 F6    JMP  $F688   I/O error #5 (Device not present)
F3C7:  A0 03       LDY  # $03   Loop and displacement counter
F3C9:  B9 0B F5    LDA  $F50B,Y Cmd sequence string to disk
F3CC:  20 D2 FF    JSR  $FFD2   Kernal BSOUT: Output a char
F3CF:  88          DEY          Decrement loop and displ. by 1
F3D0:  D0 F7       BNE  $F3C9   Loop to U0; CHR$(31) to disk
F3D2:  20 AE F7    JSR  $F7AE   Get character fo filename
F3D5:  20 D2 FF    JSR  $FFD2   Kernal BSOUT: Output a char
F3D8:  C8          INY          Incr. displacement to filename
F3D9:  C4 9F       CPY  * $9F   Compare with length of filename
F3DB:  D0 F5       BNE  $F3D2   Not reached, next character
F3DD:  20 CC FF    JSR  $FFCC   Krnl CLRCH: Reset I/O channel
F3E0:  2C 1C 0A    BIT  $0A1C   Check "fast serial mode" pointer
F3E3:  70 05       BVS  $F3EA   Fast transfer possible, skip
```

```
F3E5:  20 8C F4   JSR  $F48C      Close logical file again
F3E8:  38         SEC             Set OK indicator
F3E9:  60         RTS             Return from subroutine

****************************      LOAD / VERIFY routines in
                                  burst mode

F3EA:  A5 9F      LDA  * $9F      Temp storage for filename length
F3EC:  85 B7      STA  * $B7      In Z-P ptr for filename length
F3EE:  78         SEI             Disable system interrupts
F3EF:  20 45 E5   JSR  $E545      Clock high signal to serial bus
F3F2:  20 C3 E5   JSR  $E5C3      Wait for response from bus
F3F5:  2C 0D DC   BIT  $DC0D      Clear the CIA IRQ flag
F3F8:  20 03 F5   JSR  $F503      Invert clock low/high signal
F3FB:  20 BA F4   JSR  $F4BA      Get byte from bus (trans status)
F3FE:  C9 02      CMP  # $02      Check if transfer status "File not
F400:  D0 08      BNE  $F40A      Found" displayed. No, then skip
F402:  20 8C F4   JSR  $F48C      Clock hi to ser. bus & close file
F405:  68         PLA             Remove 2-byte RTS return addr
F406:  68         PLA             From stack
F407:  4C 85 F6   JMP  $F685      I/O error #4 (File not found)
F40A:  48         PHA             Transfer status to stack
F40B:  C9 1F      CMP  # $1F      Is it indicator for last block?
F40D:  D0 0B      BNE  $F41A      No, then skip
F40F:  20 03 F5   JSR  $F503      Invert clock low/high signal
F412:  20 BA F4   JSR  $F4BA      Get byte from bus (block-byte #)
F415:  85 A5      STA  * $A5      In Z-P byte number loop counter
F417:  4C 21 F4   JMP  $F421      Set load address
F41A:  C9 02      CMP  # $02      Check transfer status
F41C:  90 03      BCC  $F421      Code $01 indicates OK.OK,skip
F41E:  68         PLA             Erase stored transfer status
F41F:  B0 77      BCS  $F498      Jump to "Load error" exit
F421:  20 33 F5   JSR  $F533      Output LOADING/VERIFYING
F424:  20 03 F5   JSR  $F503      Invert clock low/high signal
F427:  20 BA F4   JSR  $F4BA      Get byte from bus,load addr low
F42A:  85 AE      STA  * $AE      Save in Z-P address pointer low
F42C:  20 03 F5   JSR  $F503      Invert clock low/high signal
F42F:  20 BA F4   JSR  $F4BA      Get byte from bus,load addr hi
F432:  85 AF      STA  * $AF      Store in Z-P address pointer
F434:  A6 9E      LDX  * $9E      Load & check stored sec address
```

```
F436:  D0 08      BNE  $F440   Not zero, then load prg absolute
F438:  A5 C3      LDA  * $C3   Get LOAD address low in acc
F43A:  A6 C4      LDX  * $C4   Get LOAD load addr hi in X-reg
F43C:  85 AE      STA  * $AE   Load addr low in addr pntr low
F43E:  86 AF      STX  * $AF   Load addr hi in addr pointer high
F440:  A5 AE      LDA  * $AE   Get address pointer low
F442:  A6 AF      LDX  * $AF   Get address pointer hi in X-reg
F444:  85 AC      STA  * $AC   Set acc as load address pointer
F446:  86 AD      STX  * $AD   Set X-reg as load addr pointer
F448:  68         PLA          Get transfer status from stack
F449:  C9 1F      CMP  # $1F   Status point to last prg block?
F44B:  F0 32      BEQ  $F47F   Yes, skip standard block length
F44D:  20 03 F5   JSR  $F503   Invert clock low/high signal
F450:  A9 FC      LDA  # $FC   Set the data byte counter for the
F452:  85 A5      STA  * $A5   First block of file to read to 252
F454:  20 3D F6   JSR  $F63D   Test shift RUN/STOP
F457:  20 E1 FF   JSR  $FFE1   Kernal STOP: Test STOP key
F45A:  F0 4A      BEQ  $F4A6   If zero exit through STOP key
F45C:  20 C5 F4   JSR  $F4C5   Read block from disk &process
F45F:  B0 51      BCS  $F4B2   Error in memory addr, then RTS
F461:  20 BA F4   JSR  $F4BA   Get byte from bus (xfer status)
F464:  C9 02      CMP  # $02   Check transfer status
F466:  90 06      BCC  $F46E   Code $01 indicates OK.OK,skip
F468:  C9 1F      CMP  # $1F   Was it the status for last block?
F46A:  F0 0B      BEQ  $F477   Yes, then read last block
F46C:  D0 2A      BNE  $F498   Jump to "load error" exit
F46E:  20 03 F5   JSR  $F503   Invert clock low/high signal
F471:  A9 FE      LDA  # $FE   Set data byte counter for normal
F473:  85 A5      STA  * $A5   Block to 254 bytes
F475:  D0 DD      BNE  $F454   Unconditional jump to read rout
```

```
******************************    Read last block in burst mode

F477:  20 03 F5   JSR  $F503      Invert clock low/high signal
F47A:  20 BA F4   JSR  $F4BA      Get byte from bus (block-byte #)
F47D:  85 A5      STA  * $A5      In Z-P byte number loop counter
F47F:  20 03 F5   JSR  $F503      Invert clock low/high signal
F482:  20 C5 F4   JSR  $F4C5      Read block from disk &process
F485:  B0 2B      BCS  $F4B2      Error in memory addr, then RTS
F487:  A9 40      LDA  # $40      Put EOF marker code in acc
F489:  20 57 F7   JSR  $F757      Kernal SETMSG: Set sys status

******************************    Clock high on bus and close file

F48C:  20 45 E5   JSR  $E545      Clock high signal on serial bus
F48F:  58         CLI             Enable all system interrupts
F490:  A5 B8      LDA  * $B8      Get logical file number in acc
F492:  38         SEC             Set carry flag for CLOSE routine
F493:  20 C3 FF   JSR  $FFC3      Kernal CLOSE: Close file
F496:  18         CLC             Set marker for OK
F497:  60         RTS             Return from subroutine

******************************    General "Load error" exit

F498:  A9 02      LDA  # $02      Err code for timeout during read
F49A:  20 57 F7   JSR  $F757      Kernal SETMSG: Set sys status
F49D:  20 8C F4   JSR  $F48C      Clock high on bus and close file
F4A0:  68         PLA             Delete the RTS return address
F4A1:  68         PLA             From the stack
F4A2:  A9 29      LDA  # $29      Error # for BASIC error: LOAD
F4A4:  38         SEC             Set marker for error found
F4A5:  60         RTS             Return from subroutine

******************************    Exit for STOP key interruption

F4A6:  20 8C F4   JSR  $F48C      Clock high on bus and close file
F4A9:  A9 00      LDA  # $00      Set zero-page pointer for current
F4AB:  85 B9      STA  * $B9      Secondary address to #0
F4AD:  68         PLA             Delete the RTS return addr from
F4AE:  68         PLA             The stack
F4AF:  4C B5 F5   JMP  $F5B5      Routine: Exit via break key
```

```
******************************    Exit for error in memory address

F4B2:  20 8C F4   JSR  $F48C      Clock high on bus and close file
F4B5:  68         PLA             Delete the RTS return addr from
F4B6:  68         PLA             The stack
F4B7:  4C 97 F6   JMP  $F697      To output of I/O error #10

******************************    Read a data byte in burst mode

F4BA:  A9 08      LDA  # $08      Set control bit for bus interrupt
F4BC:  2C 0D DC   BIT  $DC0D      Read interrupt control register
F4BF:  F0 FB      BEQ  $F4BC      And wait for serial bus interrupt
F4C1:  AD 0C DC   LDA  $DC0C      Read CIA data buff from ser bus
F4C4:  60         RTS             Return from subroutine

******************************    Read data block in burst mode

F4C5:  A9 08      LDA  # $08      Set conrol bit for bus interrupt
F4C7:  2C 0D DC   BIT  $DC0D      Read interrupt control register
F4CA:  F0 FB      BEQ  $F4C7      And wait for serial bus interrupt
F4CC:  AC 0C DC   LDY  $DC0C      Read CIA data buff from ser bus
F4CF:  AD 00 DD   LDA  $DD00      Read data port A of CIA 2,
F4D2:  49 10      EOR  # $10      invert the clock signal
F4D4:  8D 00 DD   STA  $DD00      accordingly, & write to port A
F4D7:  98         TYA             Copy data buffer into acc
F4D8:  A4 93      LDY  * $93      Test Z-P LOAD/VERIFY pointer
F4DA:  F0 12      BEQ  $F4EE      For $00 it's a LOAD routine
F4DC:  85 BD      STA  * $BD      Store data byte for verify operat.
F4DE:  A0 00      LDY  # $00      Displace pointer for FETCH rout
F4E0:  20 C9 F7   JSR  $F7C9      FETCH rout for LSV operations
F4E3:  C5 BD      CMP  * $BD      Compare data byte with memory
F4E5:  F0 0A      BEQ  $F4F1      Both equal, then OK &continue
F4E7:  A9 10      LDA  # $10      Not equal, then set error marker
F4E9:  20 57 F7   JSR  $F757      Kernal status: Set system status
F4EC:  D0 03      BNE  $F4F1      Skip STASH rout (for LOAD)
F4EE:  20 BF F7   JSR  $F7BF      STASH rout for LSV operations
F4F1:  E6 AE      INC  * $AE      Inc low value for I/O operations
F4F3:  D0 08      BNE  $F4FD      No overflow occurred, skip
F4F5:  E6 AF      INC  * $AF      Increment high value of I/O addr
```

```
F4F7:  A5 AF      LDA  * $AF      Check if the high value of I/O
F4F9:  C9 FF      CMP  # $FF      Addr points to sys vector table
F4FB:  F0 05      BEQ  $F502      Yes, then invalid & skip to RTS
F4FD:  C6 A5      DEC  * $A5      Decrement data byte counter
F4FF:  D0 C4      BNE  $F4C5      Loop until all bytes read
F501:  18         CLC             Set marker for "everything OK"
F502:  60         RTS             Return from subroutine

******************************    Invert clock signal on port A

F503:  AD 00 DD   LDA  $DD00      Read data port A of CIA 2,
F506:  49 10      EOR  # $10      invert clock signal and
F508:  8D 00 DD   STA  $DD00      Write back to port A
F50B:  60         RTS             Return from subroutine

******************************    Control sequence to disk in
                                  reverse order. Send
                                  U0;CHR$(31)

F50C:  1F 30 55                   <CHR$(31)> <0> <U>

******************************    Output control msg
                                  SEARCHING FOR <filename>

F50F:  A5 9D      LDA  * $9D      Pointer if ctrl messages allowed
F511:  10 1F      BPL  $F532      Not allowed, then return
F513:  A0 0C      LDY  # $0C      Displacement to SEARCHING
F515:  20 22 F7   JSR  $F722      Output system/control message
F518:  A5 B7      LDA  * $B7      Get length of filename in acc
F51A:  F0 16      BEQ  $F532      Length equal 0, then return
F51C:  A0 17      LDY  # $17      Displacement to "FOR" text
F51E:  20 22 F7   JSR  $F722      Output system/control message
```

```
*****************************   Output filenames

F521:  A4 B7      LDY  * $B7    Get length of current filename
F523:  F0 0D      BEQ  $F532    Length = 0, then skip
F525:  A0 00      LDY  # $00    Init displacement to filenames
F527:  20 AE F7   JSR  $F7AE    Get 1 byte of filename
F52A:  20 D2 FF   JSR  $FFD2    Kernal BSOUT: Output a char
F52D:  C8         INY           Incr. displ. to start of filename
F52E:  C4 B7      CPY  * $B7    Compare with length of filename
F530:  D0 F5      BNE  $F527    Not equal, then next character
F532:  60         RTS           Return from subroutine

*****************************   Output LOADING/VERIFYING

F533:  A0 49      LDY  # $49    Displacement to LOADING text
F535:  A5 93      LDA  * $93    Get Load-Verify mark from Z-P
F537:  F0 02      BEQ  $F53B    If load (0), then output
F539:  A0 59      LDY  # $59    Displacement to "Verify" text
F53B:  4C 1E F7   JMP  $F71E    Output system/control message

*****************************   Kernal routine: SAVESP
                                Save a memory range

F53E:  86 AE      STX  * $AE    Store low addr of "store to"
F540:  84 AF      STY  * $AF    Store high addr of "store to"
F542:  AA         TAX           Z-P addr of "store from" in X
F543:  B5 00      LDA  * $00,X  Get Z-P addr "from" low value
F545:  85 C1      STA  * $C1    and store in "store from" low
F547:  B5 01      LDA  * $01,X  Get Z-P addr "from" high value
F549:  85 C2      STA  * $C2    And store in "store from" high
F54B:  6C 32 03   JMP  ($0332)  vector to SAVESP ($F54E)
F54E:  A5 BA      LDA  * $BA    Load device address in acc
F550:  C9 01      CMP  # $01    Is output device the Datassette?
F552:  F0 74      BEQ  $F5C8    Yes, then in cassette save routine
F554:  C9 04      CMP  # $04    Device address less than 4?
F556:  B0 09      BCS  $F561    No, then skip error message
F558:  4C 94 F6   JMP  $F694    I/O error #9 (Illegal device #)
F55B:  4C 91 F6   JMP  $F691    I/O error #8 (Missing filename)
F55E:  4C 85 F6   JMP  $F685    I/O error #4 (File not found)
F561:  A4 B7      LDY  * $B7    Length of filename in Y-reg
```

```
F563:  F0 F6      BEQ  $F55B     Length=0, output I/O error 8
F565:  A9 61      LDA  # $61     Secondary address to Print/Write
F567:  85 B9      STA  * $B9     In z-page storage for sec. addr
F569:  20 CB F0   JSR  $F0CB     Test length and sec. address
F56C:  20 BC F5   JSR  $F5BC     If allowed, output SAVING
F56F:  A5 BA      LDA  * $BA     Load device address in acc
F571:  20 3E E3   JSR  $E33E     Rout LISTN: cmd to serial bus
F574:  A5 B9      LDA  * $B9     Load secondary address in acc
F576:  20 D2 E4   JSR  $E4D2     Rout SECND:sec addr for Listn
F579:  A0 00      LDY  # $00     Set Y-reg to 0 as displacement
F57B:  20 51 ED   JSR  $ED51     Copy start addr from C1,C2 to
                                 AD,AC
F57E:  20 03 E5   JSR  $E503     Rout CIOUT: Byte to serial bus
F581:  A5 AD      LDA  * $AD     Store start address high value
F583:  20 03 E5   JSR  $E503     Rout. CIOUT: Byte to serial bus
F586:  20 B7 EE   JSR  $EEB7     Subtr.: Start address - End addr
F589:  B0 10      BCS  $F59B     End address reached, then exit
F58B:  20 CC F7   JSR  $F7CC     Place start address in FETVEC
F58E:  20 03 E5   JSR  $E503     Rout. CIOUT: Byte to serial bus
F591:  20 E1 FF   JSR  $FFE1     Kernal STOP: Test STOP key
F594:  F0 1F      BEQ  $F5B5     If pressed, interrupt SAVESP
F596:  20 C1 EE   JSR  $EEC1     Incr. start addr ($AC,$AD) by 1
F599:  D0 EB      BNE  $F586     Overflow in high byte, then exit
F59B:  20 26 E5   JSR  $E526     Rout UNLSN: cmd to serial bus
F59E:  24 B9      BIT  * $B9     Test bit 7 of secondary address
F5A0:  30 11      BMI  $F5B3     If bit 7 is set, then skip
F5A2:  A5 BA      LDA  * $BA     Load device address in acc
F5A4:  20 3E E3   JSR  $E33E     Rout LISTN: cmd to serial bus
F5A7:  A5 B9      LDA  * $B9     Load secondary address in acc
F5A9:  29 EF      AND  # $EF     Get lower nibble of SA
F5AB:  09 E0      ORA  # $E0     Send via above CLOSE to dev.
F5AD:  20 D2 E4   JSR  $E4D2     Rout SECND:sec. addr for Listn
F5B0:  20 26 E5   JSR  $E526     Rout UNLSN:cmd to serial bus
F5B3:  18         CLC            Set indicator for OK
F5B4:  60         RTS            Return from subroutine

****************************     SAVESP exit via break

F5B5:  20 9E F5   JSR  $F59E     Close write channel to device
F5B8:  A9 00      LDA  # $00     Load acc with $00 as marker
```

```
F5BA:  38          SEC          Set carry for break/error ind.
F5BB:  60          RTS          Return from subroutine

****************************    Check if SAVING control
                                message can be printed

F5BC:  A5 9D       LDA  * $9D   Test if control message allowed
F5BE:  10 37       BPL  $F5F7   Not allowed, then return
F5C0:  A0 51       LDY  # $51   Displ. to SAVING in Y-reg
F5C2:  20 22 F7    JSR  $F722   Output "SAVING" message
F5C5:  4C 21 F5    JMP  $F521   And output filename: RTS

****************************    Save routine for datasette

F5C8:  20 80 E9    JSR  $E980   Cass buffer pointer in X+Y reg
F5CB:  90 8B       BCC  $F558   Page 0,1 not allowed: I/O err #9
F5CD:  20 E9 E9    JSR  $E9E9   Wait for "record & play" keys
F5D0:  B0 25       BCS  $F5F7   STOP, then interrupt
F5D2:  20 BC F5    JSR  $F5BC   If allowed, output "SAVING"
F5D5:  A2 03       LDX  # $03   Header type3=ML prg (absolute)
F5D7:  A5 B9       LDA  * $B9   Load secondary address in acc
F5D9:  29 01       AND  # $01   Test if bit 0 set
F5DB:  D0 02       BNE  $F5DF   Yes, then machine language prg
F5DD:  A2 01       LDX  # $01   Header type 1= BASIC program
F5DF:  8A          TXA          Copy header type in acc
F5E0:  20 19 E9    JSR  $E919   And write header to tape
F5E3:  B0 12       BCS  $F5F7   Exit, if stop key pressed
F5E5:  20 18 EA    JSR  $EA18   Save program to cassette
F5E8:  B0 0D       BCS  $F5F7   Exit, if stop key pressed
F5EA:  A5 B9       LDA  * $B9   Load secondary address in acc
F5EC:  29 02       AND  # $02   Check if bit 1 set
F5EE:  F0 06       BEQ  $F5F6   Not set, then "OK" exit
F5F0:  A9 05       LDA  # $05   Code for EOT control byte in acc
F5F2:  20 19 E9    JSR  $E919   And write block to tape
F5F5:  24          .Byte  $24   Skip to $F5F7
F5F6:  18          CLC          Set indicator for "OK"
F5F7:  60          RTS          Return from subroutine
```

```
******************************
```

Kernal routine: UDTIM
Update the internal 24-hour clock

```
F5F8:  E6 A2      INC  * $A2     Low byte of 24 hr sys clock +1
F5FA:  D0 06      BNE  $F602     No overflow, skip correction
F5FC:  E6 A1      INC  * $A1     Middle byte of 24 hr sys clk +1
F5FE:  D0 02      BNE  $F602     No overflow, skip correction
F600:  E6 A0      INC  * $A0     High byte of 24 hr sys clock +1
F602:  38         SEC            Set carry for subtraction
F603:  A5 A2      LDA  * $A2     The appropriate values are
F605:  E9 01      SBC  # $01     checked by subtraction to see if
F607:  A5 A1      LDA  * $A1     Internal 24-hr system clock is set
F609:  E9 1A      SBC  # $1A     To the clock time 24.00.00 in
F60B:  A5 A0      LDA  * $A0     the bytes $A0-$A1-$A2
F60D:  E9 4F      SBC  # $4F     In this case the 3 bytes must be
F60F:  90 08      BCC  $F619     Reinitialized
F611:  A2 00      LDX  # $00     24-hour sys clock to 00.00.00
F613:  86 A0      STX  * $A0     Z-P byte for system clock High
F615:  86 A1      STX  * $A1     Z-P byte for sys clock Middle
F617:  86 A2      STX  * $A2     Z-P byte for system clock Low
F619:  AD 1D 0A   LDA  $0A1D     Check temp storage 24hr clk low
F61C:  D0 0B      BNE  $F629     Not zero, then only low value -1
F61E:  AD 1E 0A   LDA  $0A1E     Check temp storage 24hr clk mid
F621:  D0 03      BNE  $F626     Not zero, only low and mid -1
F623:  CE 1F 0A   DEC  $0A1F     Temp storage 24hr clk high -1
F626:  CE 1E 0A   DEC  $0A1E     Temp storage 24hr clk mid -1
F629:  CE 1D 0A   DEC  $0A1D     Temp storage 24hr clock low -1
F62C:  2C 03 0A   BIT  $0A03     Test PAL / NTSC pointer
F62F:  10 0C      BPL  $F63D     NTSC system if "plus"
F631:  CE 36 0A   DEC  $0A36     Raster line line-pointer -1
F634:  10 07      BPL  $F63D     Not yet zero, then skip init.
F636:  A9 05      LDA  # $05     Sys ptr for raster line at which
F638:  8D 36 0A   STA  $0A36     Int. is generated is init. w/ 5
F63B:  D0 BB      BNE  $F5F8     Uncond. jump to new UDTIM
```

```
*****************************      Keyboard row selection to
                                   For RUN/STOP & SHIFT keys

F63D:  AD 01 DC   LDA  $DC01       Read port B for keyboard matrix
F640:  CD 01 DC   CMP  $DC01       And wait
F643:  D0 F8      BNE  $F63D
F645:  AA         TAX              Keyboard code to X-reg and
F646:  30 13      BMI  $F65B       Skip if RUN/STOP pressed
F648:  A2 BD      LDX  # $BD       Bit map for SHIFT row select
F64A:  8E 00 DC   STX  $DC00       In port A for matrix line select
F64D:  AE 01 DC   LDX  $DC01       Port B for keyboard matrix cols
F650:  EC 01 DC   CPX  $DC01       Read and wait
F653:  D0 F8      BNE  $F64D
F655:  8D 00 DC   STA  $DC00       In port A for matrix line select
F658:  E8         INX              Increment value by 1
F659:  D0 02      BNE  $F65D       Neither shift key, skip
F65B:  85 91      STA  * $91       Z-P STOP/reset RVS pointer
F65D:  60         RTS              Return from subroutine

*****************************      Kernal routine: RDTIM
                                   Read 24-hour system clock

F65E:  78         SEI              Disable all system interrupts
F65F:  A5 A2      LDA  * $A2       Zero-page byte for sys clock low
F661:  A6 A1      LDX  * $A1       Zero-page byte for sys clock mid
F663:  A4 A0      LDY  * $A0       Z-P byte for system clock high

*****************************      Kernal routine: SETTIM
                                   Set 24-hr system clock

F665:  78         SEI              Disable system interrupts
F666:  85 A2      STA  * $A2       Zero-page byte for sys clock low
F668:  86 A1      STX  * $A1       Zero-page byte for sys clock mid
F66A:  84 A0      STY  * $A0       Z-P byte for system clock high
F66C:  58         CLI              Enable system interrupts
F66D:  60         RTS              Return from subroutine
```

```
******************************    Kernal routine: STOP
                                  Test for pressed STOP key

F66E:  A5 91      LDA  * $91      Get Z-P storage for stop flag
F670:  C9 7F      CMP  # $7F      Was STOP key pressed?
F672:  D0 07      BNE  $F67B      No, return with equal flag 0
F674:  08         PHP             Save status equal flag
F675:  20 CC FF   JSR  $FFCC      Krnl CLRCH: Reset I/O chnls
F678:  85 D0      STA  * $D0      Clear Z-P keyboard buffer pntr
F67A:  28         PLP             Get status of equal flag
F67B:  60         RTS             Return from subroutine

******************************    Output I/O error message

F67C:  A9 01      LDA  # $01      I/O error #1 (Too many files)
F67E:  2C         .Byte  $2C       Skip to $F681
F67F:  A9 02      LDA  # $02      I/O error #2 (File open)
F681:  2C         .Byte  $2C       Skip to $F684
F682:  A9 03      LDA  # $03      I/O error #3 (File not open)
F684:  2C         .Byte  $2C       Skip to $F687
F685:  A9 04      LDA  # $04      I/O error #4 (File not found)
F687:  2C         .Byte  $2C       Skip to $F68A
F688:  A9 05      LDA  # $05      I/O error #5 (Device not present)
F68A:  2C         .Byte  $2C       Skip to $F68D
F68B:  A9 06      LDA  # $06      I/O error #6 (Not input file)
F68D:  2C         .Byte  $2C       Skip to $F690
F68E:  A9 07      LDA  # $07      I/O error #7 (Not output file)
F690:  2C         .Byte  $2C       Skip to $F693
F691:  A9 08      LDA  # $08      I/O error #8 (Missing filename)
F693:  2C         .Byte  $2C       Skip to $F696
F694:  A9 09      LDA  # $09      I/O error #9 (Illegal device #)
F696:  2C         .Byte  $2C       Skip to $F699
F697:  A9 10      LDA  # $10      I/O error #10
F699:  48         PHA             Store I/O error code on stack
F69A:  20 CC FF   JSR  $FFCC      Krnl CLRCH: Reset I/O chnls
F69D:  A0 00      LDY  # $00      Displacement to I/O err message
F69F:  24 9D      BIT  * $9D      Check if sys messages allowed
F6A1:  50 0A      BVC  $F6AD      Not allowed, then exit
F6A3:  20 22 F7   JSR  $F722      Rout: output sys/ctrl messages
F6A6:  68         PLA             Get error code number in acc
```

```
F6A7:  48          PHA            And store on stack
F6A8:  09 30       ORA  # $30     Create ASCII value of error code
F6AA:  20 D2 FF    JSR  $FFD2     Kernal BSOUT: Output a char
F6AD:  68          PLA            Delete error code from stack
F6AE:  38          SEC            Set carry flag as marker
F6AF:  60          RTS            Return from subroutine

*****************************     Table of sys & control messages
                                  Offset to start in parentheses

F6B0:  0D 49 2F 4F 20 45 52 52    <Cr> I/O ERROR #($00)
F6B8:  4F 52 20 A3
FCBC:  0D 53 45 41 52 43 48 49    <Cr> SEARCHING($0C)
F6C4:  4E 47 A0
F6C7:  46 4F 52 A0                FOR ($17)
F6CB:  0D 50 52 45 53 53 20 50    <Cr>PRESS PLAY ON TAPE
F6D3:  4C 41 59 20 4F 4E 20 54    ($1B)
F6DB:  41 50 C5
F6DE:  50 52 45 53 53 20 52 45    PRESS RECORD & PLAY ON
F6E6:  43 4F 52 44 20 26 20 50    TAPE  ($2E)
F6EE:  4C 41 59 20 4F 4E 20 54
F6F6:  41 50 C5
F6F9:  0D 4C 4F 41 44 49 4E C7    <Cr> LOADING ($49)
F701:  0D 53 41 56 49 4E 47 A0    <Cr> SAVING  ($51)
F709:  0D 56 45 52 49 46 59 49    <Cr> VERIFYING  ($59)
F711:  4E C7
F713:  0D 46 4F 55 4E 44 A0       <Cr> FOUND ($63)
F71A:  0D 4F 4B 8D                <Cr> OK <Cr> ($6A)

*****************************     Output system/control messages

F71E:  24 9D       BIT  # $9D     Check if output allowed
F720:  10 0D       BPL  $F72F     No, then exit
F722:  B9 B0 F6    LDA  $F6B0,Y   Read byte from message table
F725:  08          PHP            And store on stack
F726:  29 7F       AND  # $7F     Mask out bit 7, no RVS chara
F728:  20 D2 FF    JSR  $FFD2     Kernal BSOUT: Output a char
F72B:  C8          INY            Set displ. to next character
F72C:  28          PLP            Get character from stack
F72D:  10 F3       BPL  $F722     Bit 7 set is end marker
F72F:  18          CLC            Clear carry as "output" marker
```

```
F730:  60          RTS              Return from subroutine

*******************************     Kernal routine: SETNAM
                                    Set parameters for filename

F731:  85 B7       STA  * $B7       Z-P byte for length of filename
F733:  86 BB       STX  * $BB       Z-P byte for filename addr low
F735:  84 BC       STY  * $BC       Z-P byte for filename addr high
F737:  60          RTS              Return from subroutine

*******************************     Kernal routine: SETLFS
                                    Set the logical file parameters

F738:  85 B8       STA  * $B8       Z-P byte for logical file number
F73A:  86 BA       STX  * $BA       Z-P byte for device address
F73C:  84 B9       STY  * $B9       Z-P byte for secondary address
F73E:  60          RTS              Return from subroutine

*******************************     Kernal routine: SETBNK

F73F:  85 C6       STA  * $C6       Bank num for current LSV call
F741:  86 C7       STX  * $C7       Bank num for current filename
F743:  60          RTS              Return from subroutine

*******************************     Kernal routine: READST
                                    Read system status word

F744:  A5 BA       LDA  * $BA       Load device address in acc
F746:  C9 02       CMP  # $02       RS-232 addressed
F748:  D0 0B       BNE  $F755       No, then get normal status
F74A:  AD 14 0A    LDA  $0A14       Get RS-232 status
F74D:  48          PHA              And store on stack
F74E:  A9 00       LDA  # $00       Load acc with $00 in RS-232
F750:  8D 14 0A    STA  $0A14       Bring status as evrything OK
F753:  68          PLA              Get RS-232 status from stack
F754:  60          RTS              Return from subroutine
```

```
******************************   Match status to system status

F755:  A5 90      LDA  * $90     Get system STATUS in acc
F757:  05 90      ORA  * $90     Combine acc with system status
F759:  85 90      STA  * $90     Put in zero page for status
F75B:  60         RTS            Return from subroutine

******************************   Kernal routine: SETMSG
                                 Allow system/control messages

F75C:  85 9D      STA  * $9D     Z-P byte for system/control msg
F75E:  60         RTS            Return from subroutine

******************************   Kernal routine: SETTMO
                                 In order to allow timeout in
                                 IEEE bit 7 in acc to 1

F75F:  8D 0E 0A   STA  $0A0E     Acc contents in IEEE timeout
F762:  60         RTS            flag. Return from subroutine

******************************   Kernal routine: MEMTOP
                                 Set the upper memory end
                                 pointer

F763:  90 06      BCC  $F76B     Carry 0 = set / Carry 1 = read
F765:  AE 07 0A   LDX  $0A07     Low addr RAM end in sys bank
F768:  AC 08 0A   LDY  $0A08     High addr RAM end in sys bank
F76B:  8E 07 0A   STX  $0A07     Low addr RAM end in sys bank
F76E:  8C 08 0A   STY  $0A08     High addr RAM end in sys bank
F771:  60         RTS            Return from subroutine

******************************   Kernal routine: MEMBOT
                                 Set the lower memory end
                                 pointer

F772:  90 06      BCC  $F77A     Carry 0 = set / Carry 1 = read
F774:  AE 05 0A   LDX  $0A05     Low addr RAM start in sys bank
F777:  AC 06 0A   LDY  $0A06     Hi addr RAM start in sys bank
F77A:  8E 05 0A   STX  $0A05     Low addr RAM start in sys bank
F77D:  8C 06 0A   STY  $0A06     Hi addr RAM start in sys bank
```

```
F780:  60          RTS              Return from subroutine

*******************************     Kernal routine: IOBASE

F781:  A2 00       LDX  # $00       Pass address low of I/O range
F783:  A0 D0       LDY  # $D0       Pass address high of I/O range
F785:  60          RTS              Return from subroutine

*******************************     Kernal routine: LKUPSA
                                    Search in SA table for SA

F786:  98          TYA              Put the SA to search for in acc
F787:  A6 98       LDX  * $98       Get number of open files
F789:  CA          DEX              Decrement by 1, used as index
F78A:  30 0F       BMI  $F79B       All comparisons negative, exit
F78C:  DD 76 03    CMP  $0376,X     Cmp with hi byte from SA table
F78F:  D0 F8       BNE  $F789       Not found, next comparison
F791:  20 12 F2    JSR  $F212       Get LFN,DA,SA from table
                                    correspomding to X
F794:  AA          TAX              Copy found DA into X
F795:  A5 B8       LDA  * $B8       Get logical file number in acc
F797:  A4 B9       LDY  * $B9       Get secondary address in Y
F799:  18          CLC              Carry clear = marker for found
F79A:  60          RTS              Return from subroutine

*******************************     Exit from LKUPSA if not found

F79B:  38          SEC              Carry set = marker for not found
F79C:  60          RTS              Return from subroutine

*******************************     Kernal routine: LKUPLA
                                    Search in LFN table for LFN

F79D:  AA          TAX              Store LFN value to search in X
F79E:  20 02 F2    JSR  $F202       Set status OK, search LFN table
F7A1:  F0 EE       BEQ  $F791       Found, update the z-page, exit
F7A3:  D0 F6       BNE  $F79B       Not found, exit with err marker
```

```
*****************************   Kernal routine: DMA-CALL

F7A5:  BD F0 F7   LDA  $F7F0,X   Get x indexed value-config table
F7A8:  29 FE      AND  # $FE     Mask out bit 0 -I/O, D000-DFFF
F7AA:  AA         TAX            Copy config value to X-reg
F7AB:  4C F0 03   JMP  $03F0     Jump to low bank DMA routine

*****************************   FETCH for chars from filename

F7AE:  8E 35 0A   STX  $0A35     Store contents of X-reg
F7B1:  A6 C7      LDX  * $C7     Bank # for current filename
                                 (BB,BC)
F7B3:  A9 BB      LDA  # $BB     Put in acc $BB for FETVEC
F7B5:  20 D0 F7   JSR  $F7D0     Rout. INDFET:
                                 LDA(fetvec),Y any bank
F7B8:  AE 35 0A   LDX  $0A35     Get old contents of X-reg back
F7BB:  60         RTS            Return from subroutine

*****************************   STASH routine for LSV
                                 operations

F7BC:  A2 AC      LDX  # $AC     Pointer to LSV I/O addr 1 (lo)
F7BE:  2C         .Byte  $2C     Skip to $F7C1
F7BF:  A2 AE      LDX  # $AE     Pointer to LSV I/O addr 2 (lo)
F7C1:  8E B9 02   STX  $02B9     Put contents X-reg in STATVEC
F7C4:  A6 C6      LDX  * $C6     Bank # of the current LSV call
F7C6:  4C DA F7   JMP  $F7DA     Rout. INDSTA:
                                 STA(stavec),Y any bank

*****************************   FETCH routine for LSV
                                 operations

F7C9:  A9 AE      LDA  # $AE     Pointer for LSV I/O addr 1 (lo)
F7CB:  2C         .Byte  $2C     Skip to $F7CE
F7CC:  A9 AC      LDA  # $AC     Pointer to LSV I/O addr 2 (lo)
F7CE:  A6 C6      LDX  * $C6     Bank # of current LSV calls
```

```
*****************************   Preparation for FETCH routine

F7D0:  8D AA 02   STA  $02AA    Place acc contents in FETVEC
F7D3:  BD F0 F7   LDA  $F7F0,X  Load  config value determined
F7D6:  AA         TAX           By X from table and to X-reg
F7D7:  4C A2 02   JMP  $02A2    FETCH Rout.: LDA any bank

*****************************   Preparation for STASH routine

F7DA:  48         PHA           Store acc contents for STA cmd
F7DB:  BD F0 F7   LDA  $F7F0,X  Load config value determined
F7DE:  AA         TAX           By X from  table and to X-reg
F7DF:  68         PLA           Load acc contents for STA cmd
F7E0:  4C AF 02   JMP  $02AF    STASH rout. :STA in any bank

*****************************   Preparation of CMPFAR routine

F7E3:  48         PHA           Store acc contents for compare
F7E4:  BD F0 F7   LDA  $F7F0,X  Get the config value determined
F7E7:  AA         TAX           by X from the table
F7E8:  68         PLA           Get acc contents for compare
F7E9:  4C BE 02   JMP  $02BE    CMPARE routine: CMP with
                                any bank

*****************************   Kernal routine: GETCFG
                                Load X with defined config
                                value

F7EC:  BD F0 F7   LDA  $F7F0,X  Load X with defined config
F7EF:  60         RTS           value. Return from subroutine
```

```
*****************************
```

Configuration table for all "far" operations

```
F7F0:  3F     (% 0011 1111)
F7F1:  7F     (% 0111 1111)
F7F2:  BF     (% 1011 1111)
F7F3:  FF     (% 1111 1111)
F7F4:  16     (% 0001 0110)

F7F5:  56     (% 0101 0110)
F7F6:  96     (% 1001 0110)
F7F7:  D6     (% 1101 0110)
F7F8:  2A     (% 0010 1010)

F7F9:  6A     (% 0110 1010)
F7FA:  AA     (% 1010 1010)
F7FB:  EA     (% 1110 1010)
F7FC:  06     (% 0000 0110)
F7FD:  0A     (% 0000 1010)
F7FE:  01     (% 0000 0001)
F7FF:  00     (% 0000 0000)
```

Bit 0: 0= I/O area $D000-$DFFF
1 = RAM/ROM area
Bit 1: 0=ROM in $4000 - $7FFF
1 = RAM in $4000 - $7FFF
Bit 3,2: 00 = System ROM $8000-$BFFF
01 = Internal function ROM
10 = External function ROM
11 = RAM area
Bit 5,4: 00 = System ROM $C000-$FFFF
01 = Internal function ROM
10 = External function ROM
11 = RAM area
Bit 7,6: 00 = RAM bank 0
01 = RAM bank 1
10 = RAM bank 2 (bank 0)
11 = RAM bank 3 (bank 1)

```
*****************************
```

ROM copy of FETCH routine ($02A2)

```
F800:  AD 00 FF   LDA  $FF00
F803:  8E 00 FF   STX  $FF00
F806:  AA         TAX
F807:  B1 FF      LDA  ($FF),Y
F809:  8E 00 FF   STX  $FF00
F80C:  60         RTS
```

Save current config value in A
Set new config value via X
Transfer old value to X
!!! LDA (Fetvec) ,Y
Restore old configuration
Return from subroutine

```
*****************************
```

ROM copy of STASH routine ($02AF)

```
F80D:  48         PHA
F80E:  AD 00 FF   LDA  $FF00
F811:  8E 00 FF   STX  $FF00
F814:  AA         TAX
F815:  68         PLA
```

Save acc contents for STA
Save current config value in A
Set new config value via X
Transfer old value to X
Get the STA value back

```
F816:  91 FF      STA  ($FF),Y    !!! STA (Stavec) ,Y
F818:  8E 00 FF   STX  $FF00      Restore old config value
F81B:  60         RTS             Return from subroutine

*****************************     ROM copy of CMPARE routine
                                  ($02BE)

F81C:  48         PHA             Save comparison value for CMP
F81D:  AD 00 FF   LDA  $FF00      Save current config value in A
F820:  8E 00 FF   STX  $FF00      Set new config value via X
F823:  AA         TAX             Transfer old value to X
F824:  68         PLA             Get CMP comparison value back
F825:  D1 FF      CMP  ($FF),Y    !!! CMP (Cmpvec) ,Y
F827:  8E 00 FF   STX  $FF00      Restore old config
F82A:  60         RTS             Return from subroutine

*****************************     ROM copy of JSRFAR routine
                                  ($02CD)

F82B:  20 E3 02   JSR  $02E3      JMPFAR rout.:JMP to any bank
F82E:  85 06      STA  * $06      Save acc in Z-P acc storage
F830:  86 07      STX  * $07      Save X-reg in Z-P X-reg storage
F832:  84 08      STY  * $08      Save Y-reg in Z-P Y-reg storage
F834:  08         PHP             Save processor status on stack
F835:  68         PLA             Get status in acc
F836:  85 05      STA  * $05      And save in Z-P status storage
F838:  BA         TSX             Stack pointer via X
F839:  86 09      STX  * $09      Save in Z-P stack ptr storage
F83B:  A9 00      LDA  # $00      Load configuration reg with $00
F83D:  8D 00 FF   STA  $FF00      And enable all system ROMs
F840:  60         RTS             Return from subroutine

*****************************     ROM copy of JMPFAR routine
                                  ($02E3)

F841:  A2 00      LDX  # $00      In this loop, the values placed in
F843:  B5 03      LDA  * $03,X    Zero page (bytes $03-$04-$05)
F845:  48         PHA             for the program counter and
F846:  E8         INX             processor status are placed on t
F847:  E0 03      CPX  # $03      the stack. They are required for
```

```
F849:  90 F8      BCC  $F843      the RTI at the end of the routine
F84B:  A6 02      LDX  * $02      Load bank pntr for config displ.
F84D:  20 6B FF   JSR  $FF6B      Kernal GETCFG: config value
F850:  8D 00 FF   STA  $FF00      from table. Set config register
F853:  A5 06      LDA  * $06      Get zero-page acc storage
F855:  A6 07      LDX  * $07      Get zero-page X-reg storage
F857:  A4 08      LDY  * $08      Get zero-page Y-reg storage
F859:  40         RTI             Jump to prg counter address

******************************    Copy of routine in ($03F0)

F85A:  AE 00 FF   LDX  $FF00      Get configuration regi in X-reg
F85D:  8C 01 DF   STY  $DF01      Set DMA controller ctrl register
F860:  8D 00 FF   STA  $FF00      Load config register with acc
F863:  8E 00 FF   STX  $FF00      Load config register with X-reg
F866:  60         RTS             Return from subroutine

******************************    Kernal routine: PHOENIX
                                  Old cold-start routines

F867:  78         SEI             Disable system interrupts
F868:  A2 03      LDX  # $03      Initialize bank and displ. pntrs
F86A:  8E C0 0A   STX  $0AC0      For external card to #3
F86D:  AE C0 0A   LDX  $0AC0      Get displacement pntr in X-reg
F870:  BD C1 0A   LDA  $0AC1,X    Check ID table for cart. spaces
F873:  F0 11      BEQ  $F886      Table entry = 0: not "logged in"
F875:  A0 00      LDY  # $00      Set entry address low to $00
F877:  BD BC E2   LDA  $E2BC,X    Get entry addr high from table
F87A:  85 03      STA  * $03      Store entry addr high in PC hi
F87C:  84 04      STY  * $04      Store entry address low in PC lo
F87E:  BD C0 E2   LDA  $E2C0,X    Get bank value from bank table
F881:  85 02      STA  * $02      Store it in Z-P bank storage
F883:  20 CD 02   JSR  $02CD      JSRFAR rout.: JSR to any bank
                                  +RTS
F886:  CE C0 0A   DEC  $0AC0      Dec. displacement pointer by 1
F889:  10 E2      BPL  $F86D      Check all 4 cartridge areas
F88B:  58         CLI             Enable system interrupts
F88C:  A2 08      LDX  # $08      Device addr for boot-load (8)
F88E:  A9 30      LDA  # $30      Load acc with character <0>
F890:  85 BF      STA  * $BF      Zero-page byte for serial buffer
```

```
F892:  86 BA      STX  * $BA      Set device address for disk 8
F894:  8A         TXA               Copy device addr (8) into acc
F895:  20 3D F2   JSR  $F23D        Set standard I/O devices
F898:  A2 00      LDX  # $00        Init. length cntr for boot-load
F89A:  86 9F      STX  * $9F        Filename with #0
F89C:  86 C2      STX  * $C2        Set sector # for boot load ($00)
F89E:  E8         INX               Increment init. counter by 1
F89F:  86 C1      STX  * $C1        Set track # for boot load ($01)
F8A1:  C8         INY               Increment Y loop register by 1
F8A2:  D0 FD      BNE  $F8A1        Loop 256 times, until reg is zero
F8A4:  E8         INX               Increment X loop register by 1
F8A5:  D0 FA      BNE  $F8A1        Loop 256 times, until reg is zero
F8A7:  A2 0C      LDX  # $0C        Displace pointer for DOS buffer
F8A9:  BD 08 FA   LDA  $FA08,X      Get char of DOS BOOT cmd
F8AC:  9D 00 01   STA  $0100,X      And copy into DOS string buffer
F8AF:  CA         DEX               Dec. displacement pointer by 1
F8B0:  10 F7      BPL  $F8A9        Loop until 13 chars transferred
F8B2:  A5 BF      LDA  * $BF        Get drive # from Z-P storage
F8B4:  8D 06 01   STA  $0106        And put in DOS buffer
F8B7:  A9 00      LDA  # $00        Bank # for current LSV call
F8B9:  A2 0F      LDX  # $0F        Bank # for current filename
F8BB:  20 3F F7   JSR  $F73F        Routine SETBNK: Bank for
                                    LSV+filename
F8BE:  A9 01      LDA  # $01        Set length of filename to 1
F8C0:  A2 15      LDX  # $15        Addr low of filename  (=FA15)
F8C2:  A0 FA      LDY  # $FA        Addr of high filename  ("I")
F8C4:  20 31 F7   JSR  $F731        Routine SETNAM: Set filename
F8C7:  A9 00      LDA  # $00        Logical file number in acc (0)
F8C9:  A0 0F      LDY  # $0F        Secondary addr in Y-reg
F8CB:  A6 BA      LDX  * $BA        Set device addr in X-reg
F8CD:  20 38 F7   JSR  $F738        Routine SETLFS: Set file param
F8D0:  20 C0 FF   JSR  $FFC0        Kernal OPEN: Open file
                                    0,8,15,"I"
F8D3:  B0 16      BCS  $F8EB        Error encoutnered, end boot load
F8D5:  A9 01      LDA  # $01        Set length of filename to 1
F8D7:  A2 16      LDX  # $16        Addr low of filename  (=FA16)
F8D9:  A0 FA      LDY  # $FA        Add high of filename  ("#")
F8DB:  20 31 F7   JSR  $F731        Routine SETNAM: Set filename
F8DE:  A9 0D      LDA  # $0D        Logical file number in acc (13)
F8E0:  A8         TAY               And set as sec. address (13)
```

```
F8E1:  A6 BA      LDX  * $BA      Get device address in X-reg
F8E3:  20 38 F7   JSR  $F738      Routine SETLFS: Set file param
F8E6:  20 C0 FF   JSR  $FFC0      Kernal OPEN: open file
                                  13,8,13,"#"
F8E9:  90 03      BCC  $F8EE      All clear, then continue boot load
F8EB:  4C 8B F9   JMP  $F98B      Initialize disk, then RTS
F8EE:  A9 00      LDA  # $00      Initialize the 2-byte zero-page
F8F0:  A0 0B      LDY  # $0B      Pointer ($AC-$AD) with the
F8F2:  85 AC      STA  * $AC      Start address of the
F8F4:  84 AD      STY  * $AD      System cassette buffer ($0B00)
F8F6:  20 D5 F9   JSR  $F9D5      Load start sctr 01 00 in cass buff
F8F9:  A2 00      LDX  # $00      Clear loop and displ. pointer
F8FB:  BD 00 0B   LDA  $0B00,X    Check the first 3 bytes of the
F8FE:  DD C4 E2   CMP  $E2C4,X    Start sector read from the disk
F901:  D0 E8      BNE  $F8EB      Into the cassette buffer for the
F903:  E8         INX             Auto-start code (<C><B><M>).
F904:  E0 03      CPX  # $03      If found, then it is a boot prgm
F906:  90 F3      BCC  $F8FB
F908:  20 17 FA   JSR  $FA17      Kernal PRIMM: Output string

******************************    Kernal constant for BOOTING
                                  message

F90B:  0D 42 4F 4F 54 49 4E 47    <CR> <B> <O> <O> <T> <I> <N> <G>
F913:  20 00                      <space>

******************************    Set pointer and boot status

F915:  BD 00 0B   LDA  $0B00,X    Get 4 address load pointers from
F918:  95 A9      STA  * $A9,X    BOOT sector at address $0B03
F91A:  E8         INX             and init. the 2 zero-page address
F91B:  E0 07      CPX  # $07      Pointers in $AC-$AD/ $AE-$AF
F91D:  90 F6      BCC  $F915      Loop until pointers are loaded
F91F:  BD 00 0B   LDA  $0B00,X    Get output char from cass buffer
F922:  F0 06      BEQ  $F92A      The value $00 is the end marker
F924:  20 D2 FF   JSR  $FFD2      Kernal BSOUT: Output a char
F927:  E8         INX             Incr displ. to cassette buffer by 1
F928:  D0 F5      BNE  $F91F      Uncond. jump to char output
F92A:  86 9E      STX  * $9E      Store displ. to cassette buffer
F92C:  20 17 FA   JSR  $FA17      Kernal PRIMM: Output string
```

```
*****************************      BOOTING message constants

F92F:  2E 2E 2E 0D 00              <.> <.> <.> <CR>

*****************************      BOOT routine

F932:  0D 00 A5    ORA  $A500      Bank pntr  BOOT sector in bank
F935:  AE 85 C6    LDX  $C685      Copy pointer for STASH routine
F938:  A5 AF       LDA  * $AF      Get cntr for #of BOOT blocks
F93A:  F0 09       BEQ  $F945      All BOOT blocks read, then exit
F93C:  C6 AF       DEC  * $AF      Decr. boot block counter by 1
F93E:  20 B3 F9    JSR  $F9B3      Load next track/sector from disk
F941:  E6 AD       INC  * $AD      Increment load addr high by 1
F943:  D0 F3       BNE  $F938      Jump to read next block
F945:  20 8B F9    JSR  $F98B      Initialize disk to BOOT
F948:  A6 9E       LDX  * $9E      Displacement to cassette buffer
F94A:  2C          .Byte  $2C      Skip to $F94D
F94B:  E6 9F       INX  * $9F      Incr. filename length counter
F94D:  E8          INX             Set displ. to char after 0 code
F94E:  BD 00 0B    LDA  $0B00,X    Get char after 0 code (filename)
F951:  D0 F8       BNE  $F94B      Not zero, continue read
F953:  E8          INX             Set displ. to char after 0 code
F954:  86 04       STX  * $04      And place in PC lo pointer
F956:  A6 9E       LDX  * $9E      Displ. to char before filename
F958:  A9 3A       LDA  # $3A      Replace 0 with <:>
F95A:  9D 00 0B    STA  $0B00,X    And put in front of filename
F95D:  CA          DEX             Set displ. to character before <:>
F95E:  A5 BF       LDA  * $BF      ASCII character of drive spec
F960:  9D 00 0B    STA  $0B00,X    Put <0:xxxx> front of filename
F963:  86 9E       STX  * $9E      Save low address of filename
F965:  A6 9F       LDX  * $9F      Get length of filename
F967:  F0 15       BEQ  $F97E      No filename present, then skip
F969:  E8          INX             Incr. filename length ptr by 2
F96A:  E8          INX             Because <0:> included in count
F96B:  8A          TXA             Copy length of filename to A
F96C:  A6 9E       LDX  * $9E      Get address low of filename
F96E:  A0 0B       LDY  # $0B      Set address high of filename
F970:  20 31 F7    JSR  $F731      Routine SETNAM: Set filename
F973:  A9 00       LDA  # $00      Initialize acc & X-reg with $00
F975:  AA          TAX             for the SETBNK routine
```

```
F976:  20 3F F7   JSR  $F73F      Routine SETBNK: bank for
                                  LSV+filename
F979:  A9 00      LDA  # $00      Set acc as "LOAD" marker
F97B:  20 69 F2   JSR  $F269      Jump to kernal LOAD vector
F97E:  A9 0B      LDA  # $0B      Set the Z-P storage for PC hi
F980:  85 03      STA  * $03      To $0B (cassette buffer)
F982:  A9 0F      LDA  # $0F      Set the Z-P pointer to the value
F984:  85 02      STA  * $02      $0F (system ROM)
F986:  20 CD 02   JSR  $02CD      JSRFAR rout.: JSR bank +RTS
F989:  18         CLC             Clear carry for OK indicator
F98A:  60         RTS             Return from subroutine

****************************      Floppy init. for BOOTING

F98B:  08         PHP             Save processor status on stack
F98C:  48         PHA             Save acc contents on stack
F98D:  20 CC FF   JSR  $FFCC      Kernal CLRCH: Reset I/O chnls
F990:  A9 0D      LDA  # $0D      Close logical file number (13)
F992:  18         CLC             Set carry to "everything OK"
F993:  20 C3 FF   JSR  $FFC3      Kernal CLOSE: Close file
F996:  A2 00      LDX  # $00      Set logical file (0) to output
F998:  20 C9 FF   JSR  $FFC9      Kernal CKOUT: Set output chnl
F99B:  B0 0A      BCS  $F9A7      If error, then close again
F99D:  A9 55      LDA  # $55      Load acc with character <U>
F99F:  20 D2 FF   JSR  $FFD2      Kernal BSOUT: Output a char
F9A2:  A9 49      LDA  # $49      Load acc with character <I>
F9A4:  20 D2 FF   JSR  $FFD2      Kernal BSOUT: Output a char
F9A7:  20 CC FF   JSR  $FFCC      Kernal CLRCH: Reset I/O chnls
F9AA:  A9 00      LDA  # $00      Close logical file number (0)
F9AC:  38         SEC             Set carry to "everything OK"
F9AD:  20 C3 FF   JSR  $FFC3      Kernal CLOSE: Close file
F9B0:  68         PLA             Get acc contents from stack
F9B1:  28         PLP             Get old processor status
F9B2:  60         RTS             Return from subroutine
```

```
*****************************
```
Reset track and sector in DOS output buffer and load sectpr

```
F9B3:  A6 C2      LDX  * $C2
F9B5:  E8         INX
F9B6:  E0 15      CPX  # $15
F9B8:  90 04      BCC  $F9BE
F9BA:  A2 00      LDX  # $00
F9BC:  E6 C1      INC  * $C1
F9BE:  86 C2      STX  * $C2
F9C0:  8A         TXA
F9C1:  20 FB F9   JSR  $F9FB
F9C4:  8D 00 01   STA  $0100
F9C7:  8E 01 01   STX  $0101
F9CA:  A5 C1      LDA  * $C1
F9CC:  20 FB F9   JSR  $F9FB
F9CF:  8D 03 01   STA  $0103
F9D2:  8E 04 01   STX  $0104
F9D5:  A2 00      LDX  # $00
F9D7:  20 C9 FF   JSR  $FFC9
F9DA:  A2 0C      LDX  # $0C
F9DC:  BD 00 01   LDA  $0100,X
F9DF:  20 D2 FF   JSR  $FFD2
F9E2:  CA         DEX
F9E3:  10 F7      BPL  $F9DC
F9E5:  20 CC FF   JSR  $FFCC
F9E8:  A2 0D      LDX  # $0D
F9EA:  20 C6 FF   JSR  $FFC6
F9ED:  A0 00      LDY  # $00
F9EF:  20 CF FF   JSR  $FFCF
F9F2:  20 BC F7   JSR  $F7BC
F9F5:  C8         INY
F9F6:  D0 F7      BNE  $F9EF
F9F8:  4C CC FF   JMP  $FFCC
```

Get sector # from z-page storage
Increment sector by 1
Check for valid sector number
Sector # less than 21, then OK
Load value for sector number 0
Increment track number by 1
Reset zero-page sector number
Copy sector number in acc an
Convert sector to 2-byte ASCII
Put sector # low in DOS buffer
Put sector # high in DOS buffer
Load acc with track # from Z-P
Convert track to 2-byte ASCII
Put track # low in DOS buffer
Put track # high in DOS buffer
Set logical file #0 fro CKOUT
Kernal CKOUT: Set output chnl
Output 13 char from DOS buffer
Get 1 char from DOS output buf
Kernal BSOUT: Output a char
Loop counter to DOS buffer -1
Loop until 13 characters output
Kernal CLRCH: Reset I/O chnls
Set logical file (13) to input
Kernal CHKIN: Set input chnl
Displ. for STASH routine to #0
Kernal BASIN: Read a character
STASH routine for LSV operat.
Incr. STASH displ. pointer by 1
Loop until 256 bytes read
Kernal CLRCH: Reset I/O chnls

```
*****************************      Process acc contents as 2-byte
                                   ACSII(X=hi,A=lo) (only to#99)

F9FB:  A2 30      LDX  # $30       ASCII value for char <0> to X
F9FD:  38         SEC              Set carry for subtraction
F9FE:  E9 0A      SBC  # $0A       Subtract dec 10 from acc
FA00:  90 03      BCC  $FA05       Carry clear, then underflow, exit
FA02:  E8         INX              Increment ASCII hi char by 1
FA03:  B0 F9      BCS  $F9FE       Unconditional jump
FA05:  69 3A      ADC  # $3A       Underflow, create ASCII lo
FA07:  60         RTS              Return from subroutine

*****************************      Kernal constant for
                                   BOOT-LOAD

FA08:  30 30 20 31 30 20 30 20     <0> <0> < > <1> <0> < > <0> < >
FA10:  33 31 3A 31 55 49 23        <3> <1> <:> <1> <U> <I> <#>

*****************************      Kernal routine: PRIMM
                                   Output the test following JSR

FA17:  48         PHA              Store acc contents on stack
FA18:  8A         TXA              Save current X-reg contents on
FA19:  48         PHA              Stack via acc
FA1A:  98         TYA              Save current Y-reg contents on
FA1B:  48         PHA              Stack via acc
FA1C:  A0 00      LDY  # $00       Load displacement pntr with $00
FA1E:  BA         TSX              Load stack pointer into X
FA1F:  FE 04 01   INC  $0104,X     Lo byte of RTS addr in stack+1
FA22:  D0 03      BNE  $FA27       No overflow, skip
FA24:  FE 05 01   INC  $0105,X     Hi byte of RTS addr in stack +1
FA27:  BD 04 01   LDA  $0104,X     Put lo byte of RTS addr in stack
FA2A:  85 CE      STA  * $CE       In Z-P (for post-indexed addr)
FA2C:  BD 05 01   LDA  $0105,X     Put hi byte of RTS addr in stack
FA2F:  85 CF      STA  * $CF       In Z-P (for post-indexed addr)
FA31:  B1 CE      LDA  ($CE),Y     Get byte from RTS addr + Y-reg
FA33:  F0 05      BEQ  $FA3A       $00 = end marker, then exit rout
FA35:  20 D2 FF   JSR  $FFD2       Kernal BSOUT: Output char
FA38:  90 E4      BCC  $FA1E       No error, then next character
FA3A:  68         PLA              Get a byte from the stack and
```

```
FA3B:  A8          TAY          Restore old contents of Y-reg
FA3C:  68          PLA          Get a byte from the stack and
FA3D:  AA          TAX          Restore old contents of X-reg
FA3E:  68          PLA          Restore old acc contents
FA3F:  60          RTS          Return from subroutine

******************************  NMI routine

FA40:  D8          CLD          Reset decimal mode
FA41:  A9 7F       LDA  # $7F   Set NMI marker
FA43:  8D 0D DD    STA  $DD0D   Clear NMI possibility
FA46:  AC 0D DD    LDY  $DD0D   Read and clear flags
FA49:  30 14       BMI  $FA5F   Check if RS-23 is active
FA4B:  20 3D F6    JSR  $F63D   Read shift RUN/STOP
FA4E:  20 E1 FF    JSR  $FFE1   Kernal STOP: Test STOP key
FA51:  D0 0C       BNE  $FA5F   Not pressed, then skip I/O init
FA53:  20 56 E0    JSR  $E056   Stand.vctrs for I/O+ interrupt
FA56:  20 09 E1    JSR  $E109   Initialize I/O
FA59:  20 00 C0    JSR  $C000   Init I/O and clear screen
FA5C:  6C 00 0A    JMP  ($0A00) BASIC warm-start-entry($4003)
FA5F:  20 05 E8    JSR  $E805   Jump to NMI rout. for RS-232
FA62:  4C 33 FF    JMP  $FF33   Return to the IRQ calling routine

******************************  IRQ routine

FA65:  D8          CLD          Reset decimal mode
FA66:  20 24 C0    JSR  $C024   Entry to editor IRQ routine
FA69:  90 12       BCC  $FA7D   Exit IRQ for raster interrupt
FA6B:  20 F8 F5    JSR  $F5F8   Rout. UDTIM:Set 24hr clk
FA6E:  20 D0 EE    JSR  $EED0   Check recorder-keyboard
FA71:  AD 0D DC    LDA  $DC0D   Get CIA interrupt control reg.
FA74:  AD 04 0A    LDA  $0A04   Get sy NMI/reset status pointer
FA77:  4A          LSR  A       Check if bit 0 is cleared
FA78:  90 03       BCC  $FA7D   Yes, then back to IRQ routine
FA7A:  20 06 40    JSR  $4006   BASIC IRQ entry
FA7D:  4C 33 FF    JMP  $FF33   Return to the IRQ calling routine
```

```
*****************************
```

Keybaord decoder table 1a
ASCII character set normal

```
FA80:  14 0D 1D 88 85 86 87 11
FA88:  33 57 41 34 5A 53 45 01
FA90:  35 52 44 36 43 46 54 58
FA98:  37 59 47 38 42 48 55 56
FAA0:  39 49 4A 30 4D 4B 4F 4E
FAA8:  2B 50 4C 2D 2E 3A 40 2C
FAB0:  5C 2A 3B 13 01 3D 5E 2F
FAB8:  31 5F 04 32 20 02 51 03
FAC0:  84 38 35 09 32 34 37 31
FAC8:  1B 2B 2D 0A 0D 36 39 33
FAD0:  08 30 2E 91 11 9D 1D FF
FAD8:  FF
```

```
*****************************
```

Keyboard decoder table 2a
ASCII character set with shift

```
FAD9:  94 8D 9D 8C 89 8A 8B 91
FAE1:  23 D7 C1 24 DA D3 C5 01
FAE9:  25 D2 C4 26 C3 C6 D4 D8
FAF1:  27 D9 C7 28 C2 C8 D5 D6
FAF9:  29 C9 CA 30 CD CB CF CE
FB01:  DB D0 CC DD 3E 5B BA 3C
FB09:  A9 C0 5D 93 01 3D DE 3F
FB11:  21 5F 04 22 A0 02 D1 83
FB19:  84 38 35 18 32 34 37 31
FB21:  1B 2B 2D 0A 8D 36 39 33
FB29:  08 30 2E 91 11 9D 1D FF
FB31:  FF
```

```
*****************************
```

Keyboard decoder table 3a
ASCII character set with C=

```
FB32:  94 8D 9D 8C 89 8A 8B 91
FB3A:  96 B3 B0 97 AD AE B1 01
FB42:  98 B2 AC 99 BC BB A3 BD
FB4A:  9A B7 A5 9B BF B4 B8 BE
FB52:  29 A2 B5 30 A7 A1 B9 AA
FB5A:  A6 AF B6 DC 3E 5B A4 3C
FB62:  A8 DF 5D 93 01 3D DE 3F
FB6A:  81 5F 04 95 A0 02 AB 03
```

```
FB72:  84 38 35 18 32 34 37 31
FB7A:  1B 2B 2D 0A 8D 36 39 33
FB82:  08 30 2E 91 11 9D 1D FF
FB8A:  FF
```

```
******************************
```

Keyboard decoder table 4a
ASCII character set with CTRL

```
FB8B:  FF FF FF FF FF FF FF FF
FB93:  1C 17 01 9F 1A 13 05 FF
FB9B:  9C 12 04 1E 03 06 14 18
FBA3:  1F 19 07 9E 02 08 15 16
FBAB:  12 09 0A 92 0D 0B 0F 0E
FBB3:  FF 10 0C FF FF 1B 00 FF
FBBB:  1C FF 1D FF FF 1F 1E FF
FBC3:  90 06 FF 05 FF FF 11 FF
FBCB:  84 38 35 18 32 34 37 31
FBD3:  1B 2B 2D 0A 8D 36 39 33
FBDB:  08 30 2E 91 11 9D 1D FF
FBE3:  FF
```

```
******************************
```

Keyboard decoder table 5a
ASCII character set with ALT

```
FBE4:  14 0D 1D 88 85 86 87 11
FBEC:  33 D7 C1 34 DA D3 C5 01
FBF4:  35 D2 C4 36 C3 C6 D4 D8
FBFC:  37 D9 C7 38 C2 C8 D5 D6
FC04:  39 C9 CA 30 CD CB CF CE
FC0C:  2B D0 CC 2D 2E 3A 40 2C
FC14:  5C 2A 3B 13 01 3D 5E 2F
FC1C:  31 5F 04 32 20 02 51 03
FC24:  84 38 35 09 32 34 37 31
FC2C:  1B 2B 2D 0A 0D 36 39 33
FC34:  08 30 2E 91 11 9D 1D FF
FC3C:  FF
```

```
******************************
```

Free area

```
FC3D:  FF FF FF . . .
FC7D:  . . . FF FF FF
FFEF:  . .  .FF FF FF
```

Free area U.S. Versions

```
******************************
```
International models only
used to load International character sets

```
******************************
```
Clear SID registers and edit pointers

```
FC80:  8D C5 0A   STA  $0AC5
FC83:  8D 18 D4   STA  $D418
FC86:  60         RTS
```
Clear sys accent mode flag(A=0)
Clear SID volume register
Return from subroutine

```
******************************
```
Entry to kernal routine: KEY
International models only

```
FC87:  2C C5 0A   BIT  $0AC5
FC8A:  30 37      BMI  $FCC3
FC8C:  A5 D3      LDA  * $D3
FC8E:  29 10      AND  # $10
FC90:  F0 0D      BEQ  $FC9F
FC92:  AD 3F 03   LDA  $033F
FC95:  C9 FD      CMP  # $FD
FC97:  F0 2A      BEQ  $FCC3
FC99:  A9 34      LDA  # $34
FC9B:  A0 FE      LDY  # $FE
FC9D:  D0 0B      BNE  $FCAA
FC9F:  AD 3F 03   LDA  $033F
FCA2:  C9 FA      CMP  # $FA
FCA4:  F0 1D      BEQ  $FCC3
FCA6:  A9 6F      LDA  # $6F
FCA8:  A0 C0      LDY  # $C0
```
Test bit 7 of accent-mode flag
Bit 7 set, construct accent
Get current SHIFT pattern in acc
Test bit 4 for ASCII-DIN switch
If ASCII char set selected, skip
Test, if the high addr of the first
Decodertable points to DIN set
Yes, then OK and skip
Load X and A as pointers for the
Vctr table to DIN decoder table
Uncond. jump to load routine
Test if the high addr of the first
Decoder table points to ASCII
set. Yes, then OK and skip
Load X and A as pointers for the
Vect table - ASCII decoder table

```
******************************
```
Reset table set vector
International models only

```
FCAA:  85 CC      STA  * $CC
FCAC:  84 CD      STY  * $CD
FCAE:  A0 0B      LDY  # $0B
FCB0:  B1 CC      LDA  ($CC),Y
FCB2:  99 3E 03   STA  $033E,Y
FCB5:  88         DEY
```
Save pointer to vector table low
Save pointer to vector table high
Loop counter for 6 vectors
Get byte from ROM vector table
Store it in the system vector table
Decrement vector loop counter

```
FCB6:  10 F8      BPL  $FCB0      Loop until 6 vectors copied
FCB8:  C8         INY             Count Y-reg back up to zero
FCB9:  8C C5 0A   STY  $0AC5      And clear the accent-mode flag
FCBC:  08         PHP             Store processor status on stck
FCBD:  78         SEI             Disable system interrupts
FCBE:  20 0C CE   JSR  $CE0C      Kernal routine: DLCHR
FCC1:  28         PLP             Get processor status back: CLI
FCC2:  60         RTS             Return from subroutine

****************************      Check the accent keys and
                                  generate combine accent
                                  International models only

FCC3:  4C 5D C5   JMP  $C55D      Routine: read keyboard matrix
FCC6:  AE 3F 03   LDX  $033F      Check if the high addres of first
FCC9:  E0 FD      CPX  # $FD      Decoder table points to DIN set
FCCB:  D0 55      BNE  $FD22      No, skip: Store keypress
FCCD:  AE C5 0A   LDX  $0AC5      Check system accent-mode flag
FCD0:  30 50      BMI  $FD22      Bit 7 set, store keypress
FCD2:  F0 1D      BEQ  $FCF1      No accent set, then skip
FCD4:  BC 45 FE   LDY  $FE45,X    Get value from combin. table
FCD7:  CA         DEX             Decrement table value by 1
FCD8:  88         DEY             Displacement table -1
FCD9:  48         PHA             Save character code on stack
FCDA:  98         TYA             Get displace from table in acc
FCDB:  DD 45 FE   CMP  $FE45,X    Compare with combination table
FCDE:  68         PLA             Get character code from stack
FCDF:  90 08      BCC  $FCE9      Combin. table searched, skip
FCE1:  D9 4A FE   CMP  $FE4A,Y    Is it a combination character?
FCE4:  D0 F2      BNE  $FCD8      Continue searching comb. table
FCE6:  B9 65 FE   LDA  $FE65,Y    Get character from table
FCE9:  48         PHA             Store character code from stack
FCEA:  29 7F      AND  # $7F      Mask out bit 7, not RVS char
FCEC:  C9 20      CMP  # $20      Compare to space/shift space
FCEE:  68         PLA             Get char code back from stack
FCEF:  90 23      BCC  $FD14      Compare < $20: disable ctrl char
FCF1:  A2 05      LDX  # $05      Loop counter for accent table
FCF3:  DD 3F FE   CMP  $FE3F,X    Compare char with accent table
FCF6:  F0 03      BEQ  $FCFB      Character found in table, exit
FCF8:  CA         DEX             Decrement loop counter by 1
```

```
FCF9:  D0 F8      BNE  $FCF3     Loop until all comp. performed
FCFB:  8E C5 0A   STX  $0AC5     Store displ.
FCFE:  E0 00      CPX  # $00     Displ. = #0, no accent present
FD00:  F0 20      BEQ  $FD22     If zero, then store keypress
FD02:  A8         TAY            Copy character code in Y-reg
FD03:  24 F6      BIT  * $F6     Check if the auto-insert mode is
FD05:  30 0D      BMI  $FD14     Enabled. No, then RTS
FD07:  24 D7      BIT  * $D7     Check 40/80-column pointer
FD09:  10 0A      BPL  $FD15     40-column screen active, skip
FD0B:  A2 0A      LDX  # $0A     Load X-reg with # of VDC reg
FD0D:  20 DA CD   JSR  $CDDA     Read corresponding VDC reg
FD10:  29 40      AND  # $40     Check current cursor mode
FD12:  D0 06      BNE  $FD1A     If flash mode, output character
FD14:  60         RTS            Return from subroutine

******************************   Output constructed accent
                                 International models only

FD15:  AD 27 0A   LDA  $0A27     Check cursor on/off pointer
FD18:  D0 FA      BNE  $FD14     Value not=0: Cursordisable;RTS
FD1A:  98         TYA            Get character code back in acc
FD1B:  09 40      ORA  # $40     Set bit 6 in character code
FD1D:  29 7F      AND  # $7F     Mask bit 7, not RVS character
FD1F:  4C 2F CC   JMP  $CC2F     Output char at cursor position
FD22:  A6 D3      LDX  * $D3     Get SHIFT pattern in X-reg
FD24:  A4 D5      LDY  * $D5     Flag for pressed key in Y-reg
FD26:  6C 3C 03   JMP  ($033C)   Vector: Store keypress ($C6AD)

******************************   Keyboard decoder table 1b
                                 DIN charr set normal, Ctrl, Alt
                                 International models only

FD29:  14 0D 1D 88 85 86 87 11
FD31:  33 57 41 34 59 53 45 01
FD39:  35 52 44 36 43 46 54 58
FD41:  37 5A 47 38 42 48 55 56
FD49:  39 49 4A 30 4D 4B 4F 4E
FD51:  BE 50 4C AF 2E BC BD 2C
FD59:  5B 2B BB 13 01 23 5D 2D
FD61:  31 3C 04 32 20 02 51 03
FD69:  84 38 35 09 32 34 37 31
```

```
FD71:  1B 2B 2D 0A 0D 36 39 33
FD79:  08 30 2E 91 11 9D 1D FF
FD81:  FF
```

```
******************************
```

Keyboard decoder table 2b
DIN character set with shift
International models only

```
FD82:  94 8D 9D 8C 89 8A 8B 91
FD8A:  40 D7 C1 24 D9 D3 C5 01
FD92:  25 D2 C4 26 C3 C6 D4 D8
FD9A:  2F DA C7 28 C2 C8 D5 D6
FDA2:  29 C9 CA 3D CD CB CF CE
FDAA:  3F D0 CC C0 3A DC DD 3B
FDB2:  5E 2A DB 93 01 27 5C 5F
FDBA:  21 3E 04 22 A0 02 D1 83
FDC2:  84 38 35 18 32 34 37 31
FDCA:  1B 2B 2D 0A 8D 36 39 33
FDD2:  08 30 2E 91 11 9D 1D FF
FDDA:  FF
```

```
******************************
```

Keyboard decoder table 3b
DIN character set with C=
International models only

```
FDDB:  94 8D 9D 8C 89 8A 8B 91
FDE3:  96 A7 A8 97 A2 AA A3 01
FDEB:  98 A9 C4 99 C5 D3 CE A4
FDF3:  9A C2 DF 9B A1 C9 D6 D7
FDFB:  D1 C3 D5 C1 CB DA D8 CD
FE03:  AB D9 C8 BF BA CA B0 AC
FE0B:  AD A6 DB 93 01 DD DE B9
FE13:  81 B1 04 95 A0 02 A5 03
FE1B:  84 38 35 18 32 34 37 31
FE23:  1B 2B 2D 0A 8D 36 39 33
FE2B:  08 30 2E 91 11 9D 1D FF
FE33:  FF
```

```
******************************
```

Pointers to keyboard decoder tables
International models only

```
FE34:  29 FD    ($FD29)
FE36:  82 FD    ($FD82)

FE38:  DB FD    ($FDDB)
FE3A:  8B FB    ($FB8B)
FE3C:  29 FD    ($FD29)
FE3E:  29 FD    ($FD29)
```

Keyboard decoder table 1b
Keyboard decoder table 2b

Keyboard decoder table 3b
Keyboard decoder table 4a
Keyboard decoder table 1b
Keyboard decoder table 1b

```
******************************
```

Table of three accent characters
International models only

```
FE40:  AF C0 BF 00 00
```

<´ > <‘ > <^> (last via C=/< >)

```
******************************
```

Offset table to combinations of combined characters
International models only

```
FE45:  01 03 07 0C 0C 0C
```

```
******************************
```

Table of possible characters for a Combined accent character
International models only

```
FE4B:  45 C0 41 45 55 AF 41 45
FE53:  49 4F 55
```

<E> <‘> <A> <E> <U> <´> <A> <E> <I> <O> <U>

```
FE56:  FF FF FF FF FF FF FF FF
FE5E:  FF FF FF FF FF FF FF FF
```

Fill values; not used

```
******************************
```

Table of combined accent characters
International models only

```
FE64:  AC BF B2 AE B3 BF B4 B5
FE6C:  B6 B7 B8
```

<e´> <^> <‘a> <e´> <‘u> <^> <a^> <e^> <^i> <^o> <^u>

```
FE71:  FF FF FF FF FF FF FF FF
       FE79:  FF FF FF . . .
FEFD:  . . . FF FF FF
```

Fill values; not used

```
*******************************
*******************************

FF00:  00          .Byte $00
FF01:  3F          .Byte $3F
FF02:  7F          .Byte $7F
FF03:  01          .Byte $01
FF04:  41          .Byte $41

*******************************

FF05:  78          SEI
FF06:  48          PHA
FF07:  8A          TXA
FF08:  48          PHA
FF09:  98          TYA
FF0A:  48          PHA
FF0B:  AD 00 FF    LDA  $FF00
FF0E:  48          PHA
FF0F:  A9 00       LDA  # $00
FF11:  8D 00 FF    STA  $FF00
FF14:  6C 18 03    JMP  ($0318)

*******************************

FF17:  48          PHA
FF18:  8A          TXA
FF19:  48          PHA
FF1A:  98          TYA
FF1B:  48          PHA
FF1C:  AD 00 FF    LDA  $FF00
FF1F:  48          PHA
FF20:  A9 00       LDA  # $00
FF22:  8D 00 FF    STA  $FF00
FF25:  BA          TSX
FF26:  BD 05 01    LDA  $0105,X
FF29:  29 10       AND  # $10
```

American & International Versions

Copy of the configuration registers

Configuration register (CR)
Load config. register A (LCRA)
Load config.register B (LCRB)
Load config. register C (LCRC)
Load config.register D (LCRD)

Kernal NMI routine

Disable all system interrupts
Store acc contents on stack
Store current X-reg contents
On the stack via the acc
Store current Y-reg contents
On the stack via the acc
Get configuration register in acc
Store configu register on stack
Load config. register with $00
And enable system ROMs
Vector points to NMI routine ($FA40)

Kernal IRQ routine

Store acc contents on stack
Store current X-reg contents
On stack via acc
Store current Y-reg contents
On stack via acc
Get configuration register in acc
Store config value on stack
Load config.register with $00
And enable system ROMs
Put stack pointer in X-reg
Get the CPU status byte stored
Get status byte + test break bit

```
FF2B:  F0 03      BEQ  $FF30      No break , continue as norm
FF2D:  6C 16 03   JMP  ($0316)    Vector points to BRK routine
                                  ($B003)
FF30:  6C 14 03   JMP  ($0314)    Vector points to IRQ routine
                                  ($FA65)
FF33:  68         PLA             Get old config value from stack+
FF34:  8D 00 FF   STA  $FF00      Restore selected configuration
FF37:  68         PLA             Get a byte from the stack and
FF38:  A8         TAY             Restore old contents of the Y-reg
FF39:  68         PLA             Get a byte from the stack and
FF3A:  AA         TAX             Restore old contents of X-reg
FF3B:  68         PLA             Restore old acc contents
FF3C:  40         RTI             Return from the interrupt routine

******************************    Kernal RESET routine

FF3D:  A9 00      LDA  # $00      Load config. register with $00
FF3F:  8D 00 FF   STA  $FF00      And enable all system ROMs
FF42:  4C 00 E0   JMP  $E000      Reset entry

******************************    Kernal vector and entry table


FF45:  FF         .Byte  $FF
FF46:  FF         .Byte  $FF

FF47:  4C FB E5   JMP  $E5FB      Pointer to kernal FSTMOD

FF4A:  4C 3D F2   JMP  $F23D      Pointer to kernal EAINIT

FF4D:  4C 4B E2   JMP  $E24B      Pointer to kernal C64 MODE

FF50:  4C A5 F7   JMP  $F7A5      Pointer to kernal DMA-CALL

FF53:  4C 90 F8   JMP  $F890      Pointer to kernal BOOT-CALL

FF56:  4C 67 F8   JMP  $F867      Pointer to kernal PHOENIX

FF59:  4C 9D F7   JMP  $F79D      Routine: LKUPLA:
                                  search for LFN in table
```

```
FF5C:  4C 86 F7   JMP  $F786     Routine: LKUPSA:
                                 search for SA in table
FF5F:  4C 2A C0   JMP  $C02A     Pointer to kernal SWAPPER

FF62:  4C 27 C0   JMP  $C027     Pointer to kernal DLCHR

FF65:  4C 21 C0   JMP  $C021     Pointer to kernal PFKEY

FF68:  4C 3F F7   JMP  $F73F     Rout. SETBNK:
                                 bank for LSV+filename
FF6B:  4C EC F7   JMP  $F7EC     Pointer to kernal GETCFG

FF6E:  4C CD 02   JMP  $02CD     Pointer to kernal JSRFAR

FF71:  4C E3 02   JMP  $02E3     Pointer to kernal JMPFAR

FF74:  4C D0 F7   JMP  $F7D0     Rout. INDFET:
                                 LDA(fetvec),Y any bank
FF77:  4C DA F7   JMP  $F7DA     Rout. INDSTA:
                                 STA(stavec),Y any bank
FF7A:  4C E3 F7   JMP  $F7E3     Rout. INDCMP:
                                 CMP(cmpvec),Y any bank
FF7D:  4C 17 FA   JMP  $FA17     Pointer to kernal PRIMM

FF80:  00         .Byte  $00

FF81:  4C 00 C0   JMP  $C000     Pointer to kernal CINT

FF84:  4C 09 E1   JMP  $E109     Pointer to kernal IOINIT

FF87:  4C 93 E0   JMP  $E093     Pointer to kernal RAMTAS

FF8A:  4C 56 E0   JMP  $E056     Pointer to kernal RESTOR

FF8D:  4C 5B E0   JMP  $E05B     Pointer to kernal VECTOR

FF90:  4C 5C F7   JMP  $F75C     Pointer to kernal SETMSG

FF93:  4C D2 E4   JMP  $E4D2     Routine SECND:
                                 sec addr for LISTN
```

```
FF96:  4C E0 E4   JMP  $E4E0    Routine TKSA:
                                sec addr for TALK
FF99:  4C 63 F7   JMP  $F763    Pointer to kernal MEMTOP

FF9C:  4C 72 F7   JMP  $F772    Pointer to kernal MEMBOT

FF9F:  4C 12 C0   JMP  $C012    Pointer to kernal KEY

FFA2:  4C 5F F7   JMP  $F75F    Pointer to kernal SETTMO

FFA5:  4C 3E E4   JMP  $E43E    Pointer to kernal ACPTR

FFA8:  4C 03 E5   JMP  $E503    Pointer to kernal CIOUT

FFAB:  4C 15 E5   JMP  $E515    Routine UNTLK:
                                Untlk cmd to serial bus
FFAE:  4C 26 E5   JMP  $E526    Routine UNLSN:
                                Unlsn cmd to serial bus
FFB1:  4C 3E E3   JMP  $E33E    Routine LISTN:
                                Listn cmd to serial bus
FFB4:  4C 3B E3   JMP  $E33B    Routine TALK:
                                Talk cmd to serial bus
FFB7:  4C 44 F7   JMP  $F744    Pointer to kernal READST

FFBA:  4C 38 F7   JMP  $F738    Routine SETLFS:
                                Set file parameters
FFBD:  4C 31 F7   JMP  $F731    Routine SETNAM:
                                Set filename
FFC0:  6C 1A 03   JMP  ($031A)  Vector points to OPEN routine
                                $EFBD
FFC3:  6C 1C 03   JMP  ($031C)  Vector points to CLOSE routine
                                $F188
FFC6:  6C 1E 03   JMP  ($031E)  Vector points to CHKIN routine
                                $F106
FFC9:  6C 20 03   JMP  ($0320)  Vector points to CKOUT routine
                                $F14C
FFCC:  6C 22 03   JMP  ($0322)  Vector points to CLRCH routine
                                $F226
FFCF:  6C 24 03   JMP  ($0324)  Vector points to BASIN routine
                                $EF06
```

```
FFD2:  6C 26 03   JMP  ($0326)
FFD5:  4C 65 F2   JMP  $F265
FFD8:  4C 3E F5   JMP  $F53E
FFDB:  4C 65 F6   JMP  $F665
FFDE:  4C 5E F6   JMP  $F65E
FFE1:  6C 28 03   JMP  ($0328)
FFE4:  6C 2A 03   JMP  ($032A)
FFE7:  6C 2C 03   JMP  ($032C)
FFEA:  4C F8 F5   JMP  $F5F8
FFED:  4C 0F C0   JMP  $C00F
FFF0:  4C 18 C0   JMP  $C018
FFF3:  4C 81 F7   JMP  $F781

FFF6:  FF         .Byte  $FF
FFF7:  FF         .Byte  $FF

FFF8:  24 E2      ($E224)
FFFA:  05 FF      ($FF05)
FFFC:  3D FF      ($FF3D)
FFFE:  17 FF      ($FF17)
```

Vector points to BSOUT routine $EF79

Routine LOADSP: load file

Routine SAVESP: save file

Pointer to kernal SETTIM

Pointer to kernal RDTIM

Vector points to STOP routine $F66E

Vector points to GETIN routine $EEEB

Vector points to CLALL routine $F222

Rout. UDTIM: Set internal 24hr clock

Pointer to kernal SCRORG

Pointer to kernal PLOT

Pointer to kernal IOBASE

C128Mode vector

NMI vector

Reset vector

IRQ vector

8.2 The Zero Page

System variables are stored in *zero page*. These variables include the cursor position, information about the current output device, etc. Two hundred and fifty-six bytes sufficed to store all of this information.

With the C-128 the situation is different, 256 bytes are no longer enough to store all of the system information. The name zero page has been retained since it has come into such wide usage (zero page actually refers to the 256-byte *page* of memory starting at address *zero*).

The zero page offers many possibilities for direct manipulation and contains a wealth of information which the programmer can access (and which he should access). Since this zero page is so immensely important, you will find on the following pages more information on the individual memory addresses. This information will be very helpful to you.

Some addresses in the zero page have meaning only in connection to the corresponding routines in the kernal. For this reason it is very important that you take a closer look at the appropriate passages in the kernal before manipulating the zero page.

Commodore-128 Zero page

****

0000:	0000	6510 data direction - processor port
0001:	0001	6510 data register - processor port
0002:	0002	Storage for bank byte
0003:	0003	Storage for program counter high
0004:	0004	Storage for program counter low
0005:	0005	Storage for CPU status register
0006:	0006	Storage for accumulator
0007:	0007	Storage for X-register
0008:	0008	Storage for Y-register
0009:	0009	Storage for stack pointer
000A:	0010	Look for quotation mark at end of string
000B:	0011	Screen column at last TAB
000C:	0012	Disk flag: 0=LOAD, 1=VERIFY
000D:	0013	Number of elements, input buffer pointer
000E:	0014	Default for array dimensioning (DIM)
000F:	0015	Data-type flag 1:$00=numeric, $FF=string
0010:	0016	Data-type flag 2:$00=float,$80=fixed pnt
0011:	0017	Flag: LIST, read DATA, garbage coll.
0012:	0018	Pntr for FN funct, var type for FOR/NEXT
0013:	0019	Input-flag: $00=INPUT, $40=GET, $98=READ
0014:	0020	Sign of TAN: equality by comparison
0015:	0021	Active I/O device, flag: INPUT comment
0016:	0022 - 0023	Line number, integer value Lo/High
0018:	0024	Pointer to temporary string stack
0019:	0025 - 0026	Last string address
001B:	0027 - 0029	3-byte stack for temporary strings
001E:	0030 - 0032	3-byte stack for temporary strings
0021:	0033 - 0035	3-byte stack for temporary strings
0024:	0036 - 0037	2-byte help pointer index 1
0026:	0038 - 0039	2-byte help pointer index 2
0028:	0040 - 0044	Floating-point result of multiplication
002D:	0045 - 0046	Pointer: Start of BASIC text Lo/Hi
002F:	0047 - 0048	Pointer: Start of BASIC variables Lo/Hi
0031:	0049 - 0050	Pointer: Start of BASIC arrays Lo/Hi
0033:	0051 - 0052	Pointer: End of BASIC arrays + 1 Lo/Hi
0035:	0053 - 0054	Pointer: Start of string memory Lo/Hi
0037:	0055 - 0056	Help pointer for string storage Lo/Hi

0039:	0057 - 0058	Pntr: End string memory,Var.Bank 1 Lo/Hi
003B:	0059 - 0060	Current BASIC line number Lo/Hi
003D:	0061 - 0062	Pntr BASIC text for CHRGET,CHRGOT Lo/Hi
003F:	0063 - 0064	PRINT USING pntr,char search pntr Lo/Hi
0041:	0065 - 0066	Current DATA line number Lo/Hi
0043:	0067 - 0068	Pointer to current DATA address Lo/Hi
0045:	0069 - 0070	Vector pointer for INPUT routine Lo/Hi
0047:	0071 - 0072	Current BASIC variable name Lo/Hi
0049:	0073 - 0074	Pointer to address of current var. Lo/Hi
004B:	0075 - 0076	Mask for AND, LIST pntr, FOR NEXT pntr
004D:	0077 - 0078	Temporary storage for program pointer
004F:	0079	Mask for compare operation >:2, =:4, <:8
0050:	0080 - 0081	Var pntr for FN defin., + for garb coll.
0052:	0082 - 0084	Pntr:descriptor var list-string compares
0055:	0085	Help Flag: $xx=HELP, $xx=LIST
0056:	0086 - 0087	Jump vector for function evaluations
0058:	0088	Oldov
0059:	0089	Area for INSTRING oper. / temp pointer 1
005A:	0090 - 0091	Pointer: block transfer, DIM init.
005C:	0092 - 0093	Pointer: block transfer
005E:	0094	Temp pntr 2,occasionally floating-pt acc
005F:	0095 - 0096	# places before/after dec. for conver.
0061:	0097	Pntr: Dec. pt when reading digit strings
0062:	0098	Exponent sign of the # read (neg. =$80)
0063:	0099	Floating-pt. accumulator 1: Exponent
0064:	0100 - 0103	Floating-pt. accumulator 1: Mantissa
0068:	0104	Floating-pt. accumulator 1: sign
0069:	0105	Pointer: Polynomal evaluation
006A:	0106	Floating-pt. accumulator 2: Exponent
006B:	0107 - 0110	Floating-pt. accumulator 2: Mantissa
006F:	0111	Floating-pt. acc. 2: sign
0070:	0112	Result flag:sign compare Acc 1 to Acc 2
0071:	0113	Floating-pt. accumulator 1: Round off
0072:	0114 - 0115	Pointer: Cassette buffer
0074:	0116 - 0117	Offset value for AUTO command, $00=off
0076:	0118	Hires Flag: 1=BASIC-start set 10k higher
0077:	0119	Sprite number-counter for leading zeros
0078:	0120	Help counter
0079:	0121	Temp storage for indirect loading
007A:	0122 - 0124	Description of error-variable DS$

007D:	0125 - 0126	End-of-stack during program run
007F:	0127	Mode Flag: $xx=RUN mode, $xx=direct mode
0080:	0128	USING pntr for dec pnt.,Stat. DOS parser
0081:	0129	Parstx
0082:	0130	Oldstx
0083:	0131	Current color for graphic mode
0084:	0132	Multi-color Mode: Color 1
0085:	0133	Multi-color Mode: Color 2
0086:	0134	Foreground color
0087:	0135 - 0136	X-direction scale factor
0089:	0137 - 0138	Y-direction scale factor
008B:	0139	Stop drawing, if not background color
008C:	0140 - 0141	Address pointer for graphic routines
008E:	0142	Temp storage 1 for graphic routines
008F:	0143	Temp storage 2 for graphic routines
0090:	0144	Status word for kernal input/output
0091:	0145	Stop Flag: STOP key, RVS key
0092:	0146	Time constants for cassette operations
0093:	0147	Load Flag: $00=LOAD, $01=VERIFY
0094:	0148	Serial bus flag: character in buffer
0095:	0149	Char. in buffer for serial bus
0096:	0150	Sync # for cass, EOT received from tape
0097:	0151	Temporary data address
0098:	0152	Index for file tables, no. of open files
0099:	0153	Standard input device (0 for keyboard)
009A:	0154	Standard output device (3 for screen)
009B:	0155	Parity byte from cassette
009C:	0156	Tape flag: byte received
009D:	0157	Status flag for kernal
009E:	0158	Cassette error pass 1: char error
009F:	0159	Cassette error pass 2: corrected
00A0:	0160 - 0162	24-hr real-time clock : 1/60-sec count
00A3:	0163 - 0164	Temporary storage for serial bus
00A5:	0165	Countdown - SAVE on tape, ser. help ptr.
00A6:	0166	Pointer for cassette buffer
00A7:	0167	Tape short counter, RS-232 input bits
00A8:	0168	Tape read err,RS-232 counter input bits
00A9:	0169	Tape 0 read flag, RS-232 start bit flag
00AA:	0170	Tape READ mode, RS-232 buffer input byte
00AB:	0171	Tape short counter, RS-232 input parity

00AC:	0172 - 0173	Pointer: screen scroll,cass buffer Lo/Hi
00AE:	0174 - 0175	Pointer: program end,cassette end Lo/Hi
00B0:	0176 - 0177	Cassette constant for time
00B2:	0178 - 0179	Pointer: Start of cassette buffer Lo/Hi
00B4:	0180	Tape help pntr,RS232 next bit for scroll
00B5:	0181	EOT char, RS-232 next bit for transfer
00B6:	0182	Tape help pointer, RS-232 byte buffer
00B7:	0183	Length of current filename
00B8:	0184	Logical file number (LFN)
00B9:	0185	Current secondary address (SA)
00BA:	0186	Current decive number (GA)
00BB:	0187 - 0188	Pntr:Address of current filename Lo/Hi
00BD:	0189	Tape pntr, RS-232 rotate parity buffer
00BE:	0190	No. of remaining read/write blocks
00BF:	0191	Serial buffer
00C0:	0192	Flag: cassette motor
00C1:	0193	Start address in/output (Lo), track no.
00C2:	0194	Start address in/output (Hi), sector no.
00C3:	0195 - 0196	Tape LOAD temp. pntr Kernal vector address
00C5:	0197	Tape read/write data range
00C6:	0198	Bank no. current LOAD,SAVE,VERIFY calls
00C7:	0199	Bank no. of current filename $BB,$BC
00C8:	0200 - 0201	Pointer: RS-232 input buffer
00CA:	0202 - 0203	Pointer: RS-232 output buffer
00CC:	0204 - 0205	Pointer: keyboard decoder table
00CE:	0206 - 0207	Pntr to string pos.-kernal PRINT routine
00D0:	0208	Index to keyboard buffer queue
00D1:	0209	Function key call flag
00D2:	0210	Function key string call index
00D3:	0211	Shift flag: Shift=$01, C=$02, Ctrl=$04,old=$08
00D4:	0212	Flag for keypress
00D5:	0213	Flag current pressed key (CHR$(0)=none)
00D6:	0214	Flag for INPUT or GET -- keyboard input
00D7:	0215	Flag for 40/80 column mode
00D8:	0216	Flag for text/graphic screen mode
00D9:	0217	Pointer for char set, RAM/ROM (only bit 2)
00DA:	0218	Pointer for MOVLIN (Lo), <keysiz, bitmask>
00DB:	0219	Pointer for MOVLIN (Hi), <keylen, saver>
00DC:	0220	Number of the function key
00DD:	0221	F-key string length up to current F-key

00DE:	0222	Bank for function key call <sedt1>
00DF:	0223	F-key string length up to current (F-key-1)
00E0:	0224 - 0225	Pointer to running screen line: text RAM
00E2:	0226 - 0227	Pointer running screen line:attribute RAM
00E4:	0228	Lower border of window
00E5:	0229	Upper border of window
00E6:	0230	Left border of window
00E7:	0231	Right border of window
00E8:	0232	Start of running input column
00E9:	0233	Start of running input line
00EA:	0234	End of running input line
00EB:	0235	Current cursor position: line
00EC:	0236	Current cursor position: column
00ED:	0237	Maximum number of screen lines
00EE:	0238	Maximum number of screen columns
00EF:	0239	Temp storage of characters to be put out
00F0:	0240	Memory: previous char (for ESC test)
00F1:	0241	Color code under cursor for char output
00F2:	0242	Color code protection for INSERT/DELETE
00F3:	0243	Flag: RVS mode active
00F4:	0244	Flag: Quote mode active
00F5:	0245	Flag: Insert mode active
00F6:	0246	Flag: Auto insert active
00F7:	0247	Cutoff switching of C-Shift ($80) and Ctrl S ($40)
00F8:	0248	Cutoff of screen scrolling
00F9:	0249	Cutoff of beep tones made by Crtl G
00FA:	0250 - 0254	Free area for user applications
00FF:	0255	Lofbuf

****

Commodore-128 Page-One RAM

****

0100:	0256 - 0271	16-byte area for creating data names
0110:	0272	DOS loop counter
0111:	0273	DOS length of 1st file name
0112:	0274	DOS device numbers, 1st disk drive
0113:	0275 - 0276	DOS address, 1st file name Lo/Hi
0115:	0277	DOS length, 2nd file name
0116:	0278	DOS device number, 2nd disk drive
0117:	0279 - 0280	DOS address, 2nd file name Lo/Hi
0119:	0281 - 0282	Starting address for BLOAD/BSAVE Lo/Hi
011B:	0283 - 0284	End address for BSAVE command Lo/Hi
011D:	0285	DOS logical address
011E:	0286	DOS physical address
011F:	0287	DOS secondary address
0120:	0288	DOS length of a record
0121:	0289	DOS BANK number
0122:	0290 - 0291	DOS 2-byte storage for diskette ID
0124:	0292	DOS flag for disk ID testing
0125:	0293	PRINT USING pointer to starting number
0126:	0294	PRINT USING pointer to end number
0127:	0295	PRINT USING flag for dollar sign ($)
0128:	0296	PRINT USING flag for comma (,)
0129:	0297	PRINT USING counter
012A:	0298	PRINT USING sign of exponent
012B:	0299	PRINT USING pointer to exponent
012C:	0300	PRINT USING counter for whole no. places
012D:	0301	PRINT USING flag for align after dec. pt
012E:	0302	PRINT USING cntr field pos before dec pt
012F:	0303	PRINT USING cntr field pos after dec. pt
0130:	0304	PRINT USING flag for sign (+/-)
0131:	0305	PRINT USING flag for field exponent
0132:	0306	PRINT USING switch
0133:	0307	PRINT USING counter for chars in field
0134:	0308	PRINT USING sign number
0135:	0309	PRINT USING flag for space or asterisk
0136:	0310	PRINT USING pointer to start of field
0137:	0311	PRINT USING pointer for length of format

```
0138:  0312          PRINT USING pointer to end of field

*************************************************************

0139:  0313 - 0510   End of the system stack
01FF:  0511          Start of system stack

*************************************************************

0200:  0512          BASIC and monitor input buffer

*************************************************************

02A2:  0674          FETCH Routine: LDA(ZP),Y from any bank

02A2:  AD 00 FF      LDA  $FF00      You can find a description of
02A5:  8E 00 FF      STX  $FF00      This routine in the
02A8:  AA            TAX             ROM listing at $F800, because
02A9:  B1 FF         LDA  ($FF),Y    The ROM copy is located there.
02AB:  8E 00 FF      STX  $FF00      The "FETVEC" address is:
02AE:  60            RTS             $02AA, or dec. 0682.

*************************************************************

02AF:  0687          STASH Routine: STA(ZP),Y in any bank

02AF:  48            PHA             You can find a description of
02B0:  AD 00 FF      LDA  $FF00      This routine in the
02B3:  8E 00 FF      STX  $FF00      ROM listing at $F80D, because
02B6:  AA            TAX             The ROM copy is located there.
02B7:  68            PLA             The "STAVEC" address: is
02B8:  91 FF         STA  ($FF),Y    $02B9, or dec. 0697.
02BA:  8E 00 FF      STX  $FF00
02BD:  60            RTS

*************************************************************

02BE:  0702          CMPARE Routine: CMP(ZP),Y with any bank

02BE:  48            PHA             You can find a description of
02BF:  AD 00 FF      LDA  $FF00      This routine in the
```

```
02C2:  8E 00 FF     STX  $FF00
02C5:  AA           TAX
02C6:  68           PLA
02C7:  D1 FF        CMP  ($FF),Y
02C9:  8E 00 FF     STX  $FF00
02CC:  69           RTS

*****************************************************************

02CD:  0717         JSRFAR Routine: JSR in any bank and return

02CD:  20 E3 02     JSR  $02E3
02D0:  85 06        STA  * $06
02D2:  86 07        STX  * $07
02D4:  84 08        STY  * $08
02D6:  08           PHP
02D7:  68           PLA
02D8:  85 05        STA  * $05
02DA:  BA           TSX
02DB:  86 09        STX  * $09
02DD:  A9 00        LDA  # $00
02DF:  8D 00 FF     STA  $FF00
02E2:  60           RTS

*****************************************************************

02E3:  0739         JMPFAR Routine: JMP in in any bank no return

02E3:  A2 00        LDX  # $00
02E5:  B5 03        LDA  * $03,X
02E7:  48           PHA
02E8:  E8           INX
02E9:  E0 03        CPX  # $03
02EB:  90 F8        BCC  $02E5
02ED:  A6 02        LDX  * $02
02EF:  20 6B FF     JSR  $FF6B
02F3:  8D 00 FF     STA  $FF00
02F6:  A5 06        LDA  * $06
02F8:  A6 07        LDX  * $07
02FA:  A4 08        LDY  * $08
02FB:  40           RTI
```

ROM listing at $F81C, because
The ROM copy is located there.
The "CMPVEC" address is:
$02C8, or dec. 0712.

You can find a description of
This routine in the
ROM listing under the
address $F82B, because
The ROM copy is located there.

You can find a description of
This routine in the
ROM listing under the
Address $F841, because
The ROM copy is located there.

****

Routine to jump to a function cartridge. The cartridge vector has the address: $02FE-$02FF (dec. 766-767)

```
02FC:  78          SEI          Disable system interrupts
02FD:  4C 00 00    JMP  $0000   Jump to the function cartridge
                                vector
```

****

0300:	0768	3F 4D	($4D3F)	Vector: Error routine (X=error)
0302:	0770	C6 4D	($4DC6)	Vector: Read/exec. BASIC line
0304:	0772	0D 43	($430D)	Vctr: Convert interpreter code
0306:	0774	51 51	($5151)	Vector: Convert to text (List)
0308:	0776	A2 4A	($4AA2)	Vector: Execute the keyword
030A:	0778	DA 78	($78DA)	Vector: Evaluate expression
030C:	0780	21 43	($4321)	Vector: Esc. conversion routine
030E:	0782	CD 51	($51CD)	Vector: Escape list
0310:	0784	A9 4B	($4BA9)	Vector: Execute escape
0312:	0786	FF FF	($FFFF)	Interrupt vector: TIME
0314:	0788	65 FA	($FA65)	Vector for IRQ routine
0316:	0790	03 B0	($B003)	Vector for break entry -Monitor
0318:	0792	40 FA	($FA40)	Vector for NMI routine
031A:	0794	BD EF	($EFBD)	Vector to kernal OPEN routine
031C:	0796	88 F1	($F188)	Vector: kernal CLOSE routine
031E:	0798	06 F1	($F106)	Vector: kernal CHKIN routine
0320:	0800	4C F1	($F14C)	Vector: kernal CKOUT routine
0322:	0802	26 F2	($F226)	Vector: kernal CLRCH routine
0324:	0804	06 EF	($EF06)	Vector to kernal BASIN routine
0326:	0806	79 EF	($EF79)	Vector: kernal BSOUT routine
0328:	0808	6E F6	($F66E)	Vector to kernal STOP routine
032A:	0810	EB EE	($EEEB)	Vector to kernal GETIN routine
032C:	0812	22 F2	($F222)	Vector: kernal CLALL routine
032E:	0814	06 B0	($B006)	Vector to EXMON entry
0330:	0816	6C F2	($F26C)	Vector to kernal LOAD routine
0332:	0818	4E F5	($F54E)	Vector to kernal SAVE routine

```
*************************************************************
```

Copy of the character output, keyboard and decoder vectors.
The originals of these vectors are in ROM at addr. $C065 - $C07A

```
0334:  0820      B9 C7   ($C7B9)   Vector for char output with Ctrl
0336:  0822      05 C8   ($C805)   Vector : char output with Shift
0338:  0824      C1 C9   ($C9C1)   Vector for char output with Esc
033A:  0826      E1 C5   ($C5E1)   Vector for keyboard read
033C:  0828      AD C6   ($C6AD)   Vector to keypress store
033E:  0830      80 FA   ($FA80)   Vector: Keybd decoder table 1a
0340:  0832      D9 FA   ($FAD9)   Vector: Keybd decoder table 2a
0342:  0834      32 FB   ($FB32)   Vector: Keybd decoder table 3a
0344:  0836      8B FB   ($FB8B)   Vector: Keybd decoder table 4a
0346:  0838      80 FA   ($FA80)   Vector: Keybd decoder table 1a
0348:  0840      E4 FB   ($FBE4)   Vector: Keybd decoder table 5a
```

```
*************************************************************
```

```
034A:  0842 - 0851   IRQ keyboard buffer
0354:  0852 - 0861   Bit map table: Tab stops
035E:  0862 - 0865   Bit map table: Line overflow
0362:  0866 - 0875   Table of logical file numbers
036C:  0876 - 0885   Table of device addresses
0376:  0886 - 0895   Table of secondary addresses
```

```
*************************************************************
```

```
0380:  0896          BASIC CHRGET routine
                     (The original is in ROM at address $4279)

0380:  E6 3D         INC  * $3D     Increment BASIC text pointer lo
0382:  D0 02         BNE  $0386     No overflow, then skip
0384:  E6 3E         INC  * $3E     Increment BASIC text pointer hi
```

```
*************************************************************

0386:  0902          BASIC CHRGOT routine
                     (The original is in ROM at address $427F)

0386:  8D 01 FF      STA  $FF01     Enable RAM 0 area
0389:  A0 00         LDY  # $00     Displacement pntr to BASIC text
038B:  B1 3D         LDA  ($3D),Y   Get character from BASIC text
038D:  8D 03 FF      STA  $FF03     RAM 0, enable system ROMs

*************************************************************

0390:  0912          BASIC QNUM routine
                     set zero flag for separator $00 or $3A
                     set carry flag for digit 0 - 9
                     (The original is in ROM at address $4289)

0390:  C9 3A         CMP  # $3A     Char code > digit code?
0392:  B0 0A         BCS  $039B     Yes, then skip
0394:  C9 20         CMP  # $20     Was character a "blank"?
0396:  F0 EB         BEQ  $0380     Yes, then skip blank
0398:  38            SEC            Set carry for subtraction
0399:  E9 30         SBC  # $30     Test for digit (then C = 1)
039B:  38            SEC            Set carry for subtraction
039C:  E9 D0         SBC  # $D0     Restore old value
039E:  60            RTS            Return from subroutine

*************************************************************

039F:  0927          Load from a bank via PCRA and PRCR
                     (The original is in ROM at address $4298)

039F:  8D A6 03      STA  $03A6
03A2:  8D 01 FF      STA  $FF01
03A5:  B1 00         LDA  ($00),Y
03A7:  8D 03 FF      STA  $FF03
03AA:  60            RTS
```

```
*************************************************************
```

```
03AB:  0939          Load from any bank via PCRB and PCRD
                     (The original is in ROM at address $42A4)

03AB:  8D B2 03      STA  $03B2
03AE:  8D 02 FF      STA  $FF02
03B1:  B1 00         LDA  ($00),Y
03B3:  8D 04 FF      STA  $FF04
03B6:  60            RTS
```

```
*************************************************************
```

```
03B7:  0951          Load from any bank via PCRA and PCRC of
                     the address given by zero-page index 1
                     The original is in ROM at address $42B0)

03B7:  8D 02 FF      STA  $FF02
03BA:  B1 24         LDA  ($24),Y
03BC:  8D 04 FF      STA  $FF04
03BF:  60            RTS
```

```
*************************************************************
```

```
03C0:  0960          Load from any bank via PCRB and PCRD of
                     the address given by zero-page index 2
                     (The original is in ROM at address $42B9)

03C0:  8D 01 FF      STA  $FF01
03C3:  B1 26         LDA  ($26),Y
03C5:  8D 03 FF      STA  $FF03
03C8:  60            RTS
```

```
*************************************************************
```

```
03C9:  0969          Load from any bank via PCRA and PCRC of the
                     address given by the zero-page CHRGET pointer
                     The original is in ROM at address $42C2)

03C9:  8D 01 FF      STA  $FF01
03CC:  B1 3D         LDA  ($3D),Y
03CE:  8D 03 FF      STA  $FF03
```

```
03D1:  60            RTS

*************************************************************
```

03D2: 0978 - 0980 Numerical constants BASIC, loaded from ROM
03D5: 0981 Bank for SYS,POKE,PEEK. Set by bank cmd
03D6: 0982 - 0985 Temp storage for INSTRING
03DA: 0986 Bank pointer for strings and number conversion
03DB: 0987 - 0990 4 Byte storage for SSHAPE operations
03DF: 0991 Overflow marker of FAC1
03E0: 0992 Temp storage for sprite control No.1
03E1: 0993 Temp storage for sprite control No.2
03E2: 0994 Packed foreground/background color nibbles
03E3: 0995 Packed foreground/background color nibbles
03E4: 0996 - 1007 Free area

```
*************************************************************
```

03F0: 1008 DMA call routine inthe lower common area (1st K) for initializing the the external memory access

```
03F0:  AE 00 FF     LDX  $FF00
03F3:  8C 01 DF     STY  $DF01
03F6:  8D 00 FF     STA  $FF00
03F9:  8E 00 FF     STX  $FF00
03FC:  60           RTS
```

You can find a description of This DMA call routine for controlling the external memory access in ROM under the original address $F85A

```
*************************************************************
```

03FD: 1021 - 1023 Free area

03FF: 1023 End of the common area, the same in all banks

```
*************************************************************
```

0400: 1024 - 2047 Screen storage

```
*************************************************************
```

0800: 2048 - 2559 512 bytes for BASIC run-time storage

****

0A00:	2560 - 2561	Vector System restart (normal warm-start) ($4003)
0A02:	2562	Kernal Warm/cold-start Initialization status
0A03:	2563	PAL/NTSC system pntr ($FF=PAL,$00=NTSC)
0A04:	2564	System pointer for the NMI and RESET status
0A05:	2565 - 2566	Lower boundary of available RAM in system bank
0A07:	2567 - 2568	Upper boundary of available RAM in system bank
0A09:	2569 - 2570	Indirect IRQ vector for cassette routines
0A0B:	2571	Time comparison for cassette routines
0A0C:	2572	Temp stroage when reading from cassette
0A0D:	2573	Temp storage when reading from cassette
0A0E:	2574	Timeout pointer for fast serial mode
0A0F:	2575	RS-232 NMI status register
0A10:	2576	RS-232 control register
0A11:	2577	RS-232 command register
0A12:	2578 - 2579	RS-232 user baud rate
0A14:	2580	RS-232 status register
0A15:	2581	RS-232 Number of bits to send
0A16:	2582 - 2583	RS-232 baud rate: full bit time (in us)
0A18:	2584	RS-232 Index to the start of the input buffer
0A19:	2585	RS-232 Index to the end of the input buffer
0A1A:	2586	RS-232 Index to the start of the output buffer
0A1B:	2587	RS-232 Index to the end of the output buffer
0A1C:	2588	Intern/extern pointer for fast serial mode
0A1D:	2589 - 2591	Temp storage for the 24hr real-time clock
0A20:	2592	Storage for the size of the keyboard buffer
0A21:	2593	Pause pointer, <Crtl - S> pointer
0A22:	2594	Pointer: Key repetitions
0A23:	2595	Count speed for the key repeat
0A24:	2596	Counter for the key-repeat delay
0A25:	2597	Storage for the last shift pattern of the keyboard
0A26:	2598	Pointer for cursor in flash phase
0A27:	2599	Pointer for cursor on/off (0 = flashing cursor)
0A28:	2600	Count pointer for flashing cursor
0A29:	2601	Character for cursor position
0A2A:	2602	Storage for background color under cursor
0A2B:	2603	Pointer for current cursor mode (if available)
0A2C:	2604	Text screen/character base pointer
0A2D:	2605	Bit map base pointer

0A2E:	2606	Pointer for address (*256) for 80 char video RAM
0A2F:	2607	Pointer for address (*256) for attribute RAM
0A30:	2608	Temp pointer to last line for LOOP4 routine
0A31:	2609	Temp storage (a) for 80-column routines
0A32:	2610	Temp storage (b) for 80-column routines
0A33:	2611	Temp storage (a) for line clear / move
0A34:	2612	Temp storage (b) for line clear / move
0A35:	2613	Color under 80-column cursor before flash
0A36:	2614	Raster line at which the raster int. was generated
0A37:	2615	Storage for the X-register for BANK operations
0A38:	2616	Counter for the PAL system, jiffie adjust
0A39:	2617	Temp storage for for 80-column VDC screen

****

Safety storage for passive-screen variables. This area corresponds to the zero-page area at $E0.

0A40:	2624 - 2625	Pointer to the current screen line: Text RAM
0A42:	2626 - 2627	Pointer to the current screen line: Attribute RAM
0A44:	2628	Lower border of the window (init: $18 = 24)
0A45:	2629	Upper border of the window (init: $00 = 00)
0A46:	2630	Left border of the window (init: $00 = 00)
0A47:	2631	Right border of the window (init: $4F = 79)
0A48:	2632	Start of the current input line (init: $00 = 00)
0A49:	2633	Start of the current input column (init: $00 = 00)
0A4A:	2634	End of the current input line (init: $00 = 00)
0A4B:	2635	Current cursor position: line (init: $00 = 00)
0A4C:	2636	Current cursor position: column (init: $00 = 00)
0A4D:	2637	Max number of screen lines (init: $18 = 24)
0A4E:	2638	Max number of screen columns (init: $4F = 79)
0A4F:	2639	Temp storage for character to output
0A50:	2640	Storage: Previous character (for ESC test)
0A51:	2641	Current color code under cursor (init: $07 = 07)
0A52:	2642	Color code storage (Insert+delete)(init: $07 = 07)
0A53:	2643	Pointer for RVS mode active
0A54:	2644	Pointer for quote mode active
0A55:	2645	Pointer for insert mode active
0A56:	2646	Pointer for auto-insert active
0A57:	2647	Pointer for switch-lock and pause pointer

```
0A58:  2648          Pointer for locking screen-scroll
0A59:  2649          Pointer for locking beep tone (Ctrl-G)

*************************************************************

0A60:  2650 - 2687   Temp storage area for 40 and 80-column
0A80:  2688 - 2719   Buffer for comparison operations
0AA0:  2720 - 2729   Temp counter
0AAA:  2730          Adressing mode for assembler command
0AAB:  2731          Length of the cmd code for assem./disassembler
0AAC:  2732 - 2734   Assembler/disassembler storage for integ. monitor
0AAF:  2735          One-byte temp storage for misc
0AB0:  2736          One-byte temp storage for misc
0AB1:  2737          One-byte temp storage for misc
OAB2:  2738          X-reg storage for indirect subroutine calls
0AB3:  2739          Direction pointer for transfer operations
0AB4:  2740 - 2751   One-byte temp storage
0AC0:  2752          ROM bank for current function key call
0AC1:  2753 - 2756   Table of physical addresses and ID's from
                     inserted expansion cards
0AC5:  2757          System pointer for the combination of vowels with
                     accents in DIN character set (International only)

*************************************************************

0B00:  2816 - 3071   Cassette buffer

*************************************************************

0C00:  3072 - 3327   RS-232 input buffer

*************************************************************

0D00:  3328 - 3583   RS-232 output buffer

*************************************************************

0E00:  3584 - 4095   Area for sprite definition (must be under $1000)
```

****

1000: 4096 - 4105 Programmable function keys (length table)
100A: 4106 - 4351 Programmable function keys (function strings)

****

1100: 4352 - 4400 Buffer for generating DOS output strings
1131: 4401 - 4402 Graphic variable: Current X-position (Lo/Hi)
1133: 4403 - 4404 Graphic variable: Current Y-position (Lo/Hi)
1135: 4405 - 4406 Graphic variable: Dest direction, X-coord (Lo/Hi)
1137: 4407 - 4408 Graphic variable: Dest direction, Y-coord (Lo/Hi)
1139: 4409 - 4410 Variable - graphic lines: X/Y-absolute,X-absolute
113B: 4411 - 4412 Variable for graphic lines: Y-absolute
113D: 4413 - 4414 Variable - graphic lines: X/Y-Signum , X-Signum
113F: 4415 - 4416 Variable for graphic lines: Y-sign
1141: 4417 - 4420 Variable for graphic lines: Factor
1145: 4421 - 4422 Variable for graphic lines: Error value
1147: 4423 Variable for graphic lines: Smaller marker
1148: 4424 Variable for graphic lines: Larger marker
1149: 4425 Variable for angle routine: Sign of the angle
114A: 4426 - 4427 Variable for angle routine: Sine of the angle value
114C: 4428 - 4429 Variable for angle routine: Cosine of the angle val
114E: 4430 - 4431 Variable for angle routine: Angle distance

****

The following 24 bytes are used for a variety of purposes

Variables for circle routines

1150: 4432 - 4433 Circle center: X-coordinate (Lo/Hi)
1152: 4434 - 4435 Circle center: Y-coordinate (Lo/Hi)
1154: 4436 - 4437 Circle radius in X-direction (Lo/Hi)
1156: 4438 - 4439 Circle radius in Y-direction (Lo/Hi)
1158: 4440 - 4443 Rotation angle of the circle (Lo/Hi)
115C: 4444 - 4445 Angle degree for start of arc (Lo/Hi)
115E: 4446 - 4447 Angle degree for end of arc (Lo/Hi)
1160: 4448 - 4449 X-radius * Cos (rotation angle)
1162: 4450 - 4451 Y-radius * Sin (rotation angle)
1164: 4452 - 4453 X-radius * Sin (rotation angle)
1166: 4454 - 4455 Y-radius * Cos (rotation angle)

Parameters used for general purposes

1150: 4432 - 4433	Center for X-coordinate
1152: 4434 - 4435	Center for Y-coordinate
1154: 4436 - 4437	Distance 1 for X-coordinate
1156: 4438 - 4439	Distance 1 for Y-coordinate
1158: 4440 - 4441	Distance 2 for X-coordinate
115A: 4442 - 4443	Distance 2 for Y-coordinate
115C: 4444 - 4445	End of coordinate distance
115E: 4446	Column counter for characters
115F: 4447	Line counter for characters
1160: 4448:	Length counter for string

Variables used for rectangle routines

1150: 4432 - 4433	X-coordinate 1
1152: 4434 - 4435	Y-coordinate 1
1154: 4436 - 4437	Rotation angle
1156: 4438 - 4439	Counter for X-value
1158: 4440 - 4441	Counter for Y-value
115A: 4442 - 4443	Length of a side of the rectangle
115C: 4444 - 4445	X-coordinate 2
115E: 4446 - 4447	Y-coordinate 2

Used for shapes and shape movement

1150: 4432	Place older
1151: 4433	Length pointer
1152: 4434	Following pointer
1153: 4435	Length of the string
1154: 4436	Shape mode set/replace
1155: 4437	Pointer to position in the string
1156: 4438	Old bit-map byte
1157: 4439	Variable for new string or bit-map byte
1158: 4440	Place holder
1159: 4441 - 4442	Column width (X-width) of a shape
115B: 4443 - 4444	Line number (Y-length) of a shape
115D: 4445 - 4446	Temp storage for the column width
115F: 4447 - 4448	Pointer to the shape string for shape storage
1161: 4449	Bit pointer to byte of shape string

****

Area for general graphic variables

1168: 4456	Temp storage for diverse purposes
1169: 4457	Temp storage: Bit counter GSHAPE instruction
116A: 4458	Screen scaling pointer 0=320*200,1=1024*1024
116B: 4459	Temp storage for double-width
116C: 4460	Temp storage for box fill
116D: 4461	Temp storage for bit masks
116E: 4462	Temp counter for numerical values
116F: 4463	Temp pointer for trace mode on/off
1170: 4464 - 4465	Temp storage 1 for renumber routine
1172: 4466 - 4467	Temp storage 2 for renumber routine
1174: 4468	1 byte temp storage
1175: 4469 - 4470	2 byte temp storage
1177: 4471	1 byte temp storage 1 for graphic routines
1178: 4472	1 byte temp storage 2 for graphic routines
1179: 4473	1 byte temp storage for graphic routines
117A: 4474 - 4475	Vector: Convert floating-point to integer ($849F)
117C: 4476 - 4477	Vector: Convert integer to floating-point ($793C)
117E: 4478 - 4565	Speed/direction table for sprites
11D6: 4566 - 4607	42-byte area for copying VIC registers
1200: 4608 - 4609	Previous BASIC line number
1202: 4610 - 4611	Command pointer for BASIC CONT command
1204: 4612	Print Using pointer: Chr$
1205: 4613	Print Using pointer: Fill character
1206: 4614	Print Using pointer: Comma character
1207: 4615	Print Using pointer: Character for decimal point
1208: 4616	Last error number (for TRAP command)
1209: 4617 - 4618	Line number of the last error ($FFFF is OK ind)
120B: 4619 - 4620	Line number to be executed if error occurs
120D: 4621	Temp pointer for TRAP command
120E: 4622 - 4623	Pointer to text of error message
1210: 4624 - 4625	Text-end pointer
1212: 4626 - 4627	Highest address available to BASIC in RAM 0
1214: 4628 - 4629	Temp storage for DO - LOOP
1216: 4630 - 4631	Temp storage for line number
1218: 4632	USR jump
1219: 4633 - 4634	USR address in format Lo/Hi

```
121B: 4635 - 4639   Initial value for RND function
1220: 4640          Degree number for arc
1221: 4641          Pointer to reset status (cold-start or warm-start)

*************************************************************

                    Storage area for music pointers

1222: 4642          <tempo rate>
1223: 4643 - 4648   <voices>
1229: 4649 - 4650   <ntime>
122B: 4651          <octave>
122C: 4652          <sharp>
122D: 4653 - 4654   <pitch>
122F: 4655          <voice>
1230: 4656 - 4658   <wave 0>
1233: 4659          <dnote>
1234: 4660 - 4663   <fltsav>
1238: 4664          <fltflg>
1239: 4665          <nibble>
123A: 4666          <tonnum>
123B: 4667 - 4669   <tonval>
123E: 4670          <parcnt>
123F: 4671 - 4680   <atktab>
1249: 4681 - 4690   <sustab>
1253: 4691 - 4700   <waftab>
125D: 4701 - 4710   <pulslw>
1267: 4711 - 4720   <pulshi>
1271: 4721 - 4725   <filters>

*************************************************************

                    Storage area for interrupt pointer

1276: 4726 - 4728   3-byte interrupt storage
1279: 4729 - 4731   3-byte interrupt address lo storage
127C: 4732 - 4734   3-byte interrupt address hi storage
127F: 4735          <intval>
1280: 4736          <coltyp>
```

****

Storage for SID variables

1281: 4737	Sound: Voice storage
1282: 4738 - 4740	Sound: Time storage lo value (3 byte)
1285: 4741 - 4743	Sound: Time storage hi value (3 Byte)
1288: 4744 - 4746	Sound: Max value lo (3 Byte)
128B: 4747 - 4749	Sound: Max value hi (3 Byte)
128E: 4750 - 4752	Sound: Min value lo (3 Byte)
1291: 4753 - 4755	Sound: Min value hi (3 Byte)
1294: 4756 - 4758	Sound: Direction (3 Byte)
1297: 4759 - 4761	Sound: Step number lo (3 Byte)
129A: 4762 - 4764	Sound: Step number hi (3 Byte)
129D: 4765 - 4767	Sound: Frequency lo (3 Byte)
12A0: 4768 - 4770	Sound: Frequency hi (3 Byte)
12A3: 4771	Temp storage: Time value lo
12A4: 4772	Temp storage: Time value hi
12A5: 4773	Temp storage: Maximum value lo
12A6: 4774	Temp storage: Maximum value hi
12A7: 4775	Temp storage: Minimum value lo
12A8: 4776	Temp storage: Minimum value hi
12A9: 4777	Temp storage: Direction
12AA: 4778	Temp storage: Step number lo
12AB: 4779	Temp storage: Step number hi
12AC: 4780	Temp storage: Frequency lo
12AD: 4781	Temp storage: Frequency hi
12AE: 4782	Temp storage: Pulse-wave width lo
12AF: 4783	Temp storage: Pulse-wave width hi
12B0: 4784	Temp storage: Waveform
12B1: 4785	Temp storage 1 for POT function
12B2: 4786	Temp storage 2 for POT function
12B3: 4787 - 4790	Temp storage for WINDOW operations lo/hi
12B7: 4791 - 4857	Memory pointer for SPRDEF & SAVSPR cmds
12FA: 4858	Definit. mode for SPRDEF and SAVSPR cmds
12FB: 4859	Line counter for SPRDEF and SAVSPR cmds
12FC: 4860 - 4863	Sprite number for SPRDEF and SAVSPR cmds

```
**************************************************************

1300: 4864 - 6143    Unused absolute RAM range
1800: 6144 - 7167    Reserved for function key applications
1C00: 7168 - 8191    Video matrix #2 (1 Kb, bit map color) if needed

**************************************************************

2000: 8192 -16383    VIC bit map (8 Kb) if needed

**************************************************************

4000: 16384          Start of ROM

**************************************************************
```

8.3 Alphabetical listing of the kernal routines

As a user of the kernal and its subroutines you probably have found yourself looking for a certain routine or table. The kernal and the built-in monitor in the Commodore 128 consist of a large number of interesting and useful routines which you can integrate into your own programs in various ways. The problem lies in knowing that a certain routine exists, but not knowing where it can be found and how to access it. Before you start to look in the ROM listing for the routine you need, take a look through this table in which we have listed all of the important routines and tables which may be of interest to you.

$C17C	Adapt attribute RAM address
$B0FC	Addresses of the individual monitor commands (table)
$B88A	Base table for four number systems
$C98E	Bell: create tone
$EF84	BSOUT output not to screen
$E224	C128 mode routine
$F81C	CMPARE routine for FAR operations RAM
$F81C	CMPARE routine for FAR operations ROM
$EE9B	Change IRQ vector for tape operation
$C3F4	Check Commodore key for time delay
$F3A1	Check filename for burst mode
$CA9F	Clear from cursor position to screen end
$CA76	Clear from cursor position to line end
$CA8B	Clear from line start to line end
$CBB1	Clear line overflow bit
$C60A	Commodore/Shift character set switch
$C892	Commodore/Shift switch to 40-column mode
$C89F	Commodore/Shift switch to 80-column mode
$E0CD	Copy NMI and IRQ routines to all banks
$E723	CKOUT routine for RS-232 output
$F16C	CKOUT evaluation on serial bus
$F127	CHKIN evaluation on RS-232
$E795	CHKIN routine for RS-232 input
$F1A9	CLOSE routine for tape operation
$EED0	Check cassette recorder-keyboard
$E980	Check tape header address for validity
$E242	Check EXROM input form cartridge test
$E61B	Check RS-232 send parity
$CAEA	Clear or set auto-insert pointer

$C142	Clear screen window
$C4A5	Clear screen line in 40-column mode
$C4C0	Clear screen line in 80-column mode
$B8D2	Convert acc contents into two ASCII characters (X/A)
$B8C2	Convert acc to two ASCII characters and output
$F755	Coordinate system status word
$E24B	Configure system as Commodore 64
$C40D	Copy a window line (routine: MOVLIN)
$C436	Copy a window line in 80-column mode
$ED51	Copy start address for input/output operations
$F533	Control message: output LOADING
$F50F	Control message: output SEARCHING FOR filename
$F533	Control message: output VERIFYING
$CE0C	Copy character set into VDC RAM
$C320	Conversion from ASCII characters to POKE codes
$C93D	Delete character under cursor
$CA52	Delete current input line
$CA24	Define sceen as window
$F1E4	Delete file entry from table
$F1C1	Delete a file entry
$C91B	Delete character to the left of the cursor
$C3DC	Delete line on screen (with move)
$B050	Display monitor register contents
$B0C5	Determine address of a monitor command
$B641	Determine address of BRANCH commands
$C96C	Determine tab position
$C8A6	Disable or enable Commodore/Shift
$03F0	DMA call routine of common area in RAM
$CAF2	Enable block cursor
$C194	Editor IRQ routine
$C62F	Evaluate decoder table according to shift pattern
$C6AD	Evaluate and store keypress
$C7B6	Execute control code
$C9BE	Execute escape sequences
$C8E3	Execute insert
$02A2	Fetch routine for FAR operations RAM
$F800	Fetch routine for FAR operations ROM
$F7C9	Fetch routine for LSV operations
$F7AE	Fetch routine for character from filename
$C6E7	Flash VIC cursor
$E26B	Function ROM test for C-128 mode
$E569	Get bit from serial bus into carry flag
$CC6A	Get cursor position and set

$C244	Get character from keyboard queue
$CB58	Get character and color at cursor position
$C29B	Get character from screen
$EF5C	Get character from serial bus
$EF48	Get character from cassette
$EF67	Get character from RS-232
$E7CE	GET routine for RS-232
$EEF9	GETIN evaluation not over keyboard
$E5D6	Give fast-mode pulse on serial bus
$E9BE	Increment tape buffer pointer
$C07B	Initialize screen and editor
$C07B	Initialize editor and screen
$B046	Initialization of monitor commands
$B021	Initialize monitor for regular entry
$B014	Initialize monitor after BREAK
$E1DC	Initialize VDC registers
$C37C	Insert line on screen
$EAEB	Interrupt routine for tape read
$ED90	Interrupt routine for tape write
$CCF6	Insert function key string
$02E3	JMPFAR routine RAM
$F841	JMPFAR routine ROM
$02CD	JSRFAR routine RAM
$F82B	JSRFAR routine ROM
$C94F	Jump to tab stop
$E43E	Kernal Acptr routine
$EF06	Kernal BASIN routine
$F934	Kernal boot routine
$EF79	Kernal BSOUT routine
$F106	Kernal CHKIN routine
$EF06	Kernal CHRIN routine
$EF79	Kernal CHROUT routine
$E503	Kernal CIOUT routine
$F14C	Kernal CKOUT routine
$F222	Kernal CLALL routine
$F188	Kernal CLOSE routine
$F226	Kernal CLRCH routine
$F7A5	Kernal DMA call routine
$E5FB	Kernal FSTMODE routine
$F7EC	Kernal GETCFG routine
$EEEB	Kernal GETIN routine
$E24B	Kernal GO64 routine
$F781	Kernal IOBASE routine

$E109 Kernal IOINIT routine
$FF17 Kernal IRQ routine
$C55D Kernal KEY routine ($FC87 in International versions)
$E343 Kernal LISTN routine
$F79D Kernal LKUPLA routine
$F786 Kernal LKUPSA routine
$F265 Kernal LOAD routine
$F772 Kernal MEMBOT routine
$F763 Kernal MEMTOP routine
$FF05 Kernal NMI routine
$EFBD Kernal OPEN routine
$F867 Kernal PHOENIX routine
$FA17 Kernal PRIMM routine
$E093 Kernal RAMTAS routine
$F65E Kernal RDTIM routine
$F744 Kernal READST routine
$FF3D Kernal RESET routine
$E4D2 Kernal SECND routine
$F73F Kernal SETBNK routine
$F738 Kernal SETFLS routine
$F75C Kernal SETMSG routine
$F731 Kernal SETNAM routine
$F665 Kernal SETTIM routine
$F75F Kernal SETTMO routine
$E33B Kernal TALK routine
$E4E0 Kernal TKDA routine
$F5F8 Kernal UDTIM routine
$E526 Kernal UNLSN routine
$E515 Kernal UNTLK routine
$E056 Kernal RESTOR routine
$F53E Kernal SAVE routine
$F66E Kernal STOP routine
$E05B Kernal VECTOR routine
$C67E Key repeat evaluation
$C55D Keybaord matrix read
$F63D Keyboard row selection: RUN/STOP - SHIFT
$C5E1 Keyboard read evaluate
$C6CA Keyboard buffer prepare for function key
$B976 Load bank pointer and program counter from zero page
$E9FB Load program from cassette
$F3EA LOAD routine in burst mode
$F27B LOAD routine from serial bus
$B406 Monitor command: . (assemble a line)

$B194 Monitor command: ; (change register)
$B1AB Monitor command: > (change memory contents
$BA90 Monitor command: @ (disk command)
$B406 Monitor command: A (assemble a line)
$B231 Monitor command: C (compare memory areas)
$B599 Monitor command: D (disassemble memory)
$B3D8 Monitor command: F (fill memory area)
$B1D6 Monitor command: G (Jump to XXXX without return)
$B2CE Monitor command: H (Search for memory contents)
$B1DF Monitor command: J (Jump to XXXX with RTS)
$B337 Monitor command: L (Load a program)
$B152 Monitor command: M (display memory contents)
$B050 Monitor command: R (display register contents)
$B337 Monitor command: S (store a program)
$B234 Monitor command: T (move memory areas)
$B337 Monitor command: V (compare program with memory)
$B0E3 Monitor command: X (exit)
$B981 Monitor command: Convert number to different system
$E805 NMI routine for RS-232
$E8A9 NMI routine for RS-232 output
$E878 NMI routine for RS-232 input
$F915 Output boot sector message
$F0CB Open file on serial bus
$EFF0 OPEN routine for tape operation
$F040 OPEN routine for RS-232
$E75C Output in RS-232 buffer
$CC2F Output acc at cursor position
$FD15 Output combined accent
$CC27 Output space at cursor position
$E3E2 Output byte on serial bus
$C76F Output carriage return to screen
$C2BC Output character at cursor position
$C72D Output character on screen
$F521 Output found filename on screen
$F71E Output system and control messages
$CE8C Prepare byte output on serial bus
$F9FB Prepare acc contents in two ASCII characters (-99)
$EAA1 Prepare cassette synchronization
$C363 Perform linefeed
$E69D Process received bit from RS-232
$CCA2 Program function key
$F4C5 Read data block in burst mode
$E9F2 Read data block from tape

$F4BA	Read data byte in burst mode
$C258	Read an input line terminated by RETURN
$E8D0	Read program header from cassette
$E987	Recalculate tape-end address
$EE57	Recorder operation end
$EEB0	Recorder motor off
$E000	Reset routine
$C651	Repeat keyboard logic
$C77D	Reset quote mode
$F0B0	Reset CIAs to RS-232
$FCAA	Reset decoder table set vectors
$F9B3	Recreate DOS output buffer
$C980	Reset tab stops
$E5FF	RS-232 output
$E68E	RS-232 data-bit number calculate
$E672	RS-232 NMI status set
$E6D4	RS-232 start bittest
$EFB7	RS-232 character output
$C3A6	Scroll screen up
$F5C8	SAVE routine for tape operation
$E99A	Search tape header for name
$CBC3	Search for end of input line
$F202	Search in logical file number table
$CACA	Scroll up
$CABC	Scroll down
$CAE2	Scrolling permit or prohibit
$F23D	Set standard I/O devices
$CA14	Set window borders
$ED5A	Set bit counter for serial output
$CB37	Set or clear bell pointer
$CDF9	Set attribute address for attribute RAM
$C7E5	Set character color in 40-column mode
$C7EC	Set character color in 80-column mode
$CB93	Set line overflow bit
$C8D5	Set cursor flash mode
$CD57	Set cursor at current column
$C33E	Set cursor to end of line
$C150	Set cursor in screen windor at HOME position
$C875	Set cursor to left in window to left
$C867	Set cursor up in window
$C854	Set cursor right in window
$C85A	Set cursor down in window
$CC00	Set cursor one position left in window

$CBED	Set cursor one position right in window
$C932	Set old cursor address again
$CD6F	Set cursor color at cursor position
$F0D5	Set filename to serial bus
$C961	Set or clear tab stop
$E573	Set clock frequency to 1MHz
$C8BF	Set or clear reverse mode
$C207	Set IRQ register
$F39B	Set program end address after LOAD
$02AF	Stash routine for FAR operations RAM
$F80D	Stash routine for FAR operations ROM
$F7BC	Stash routine for LSV operations
$CD2C	Switch 40/80 column modes
$FA65	System IRQ routine
$FA40	System NMI routine
$F7F0	Table of configuration values
$CEB2	Table of function key assignments
$C6DD	Table of funtion key codes
$EEA8	Table of IRQ vectors for tape operation
$CE74	Table of initialization values for 40-column
$CE8E	Table of initialization values for 80-column
$C78C	Table of control codes
$E04B	Table of MMU initialization values
$B0E6	Table of monitor keywords
$E850	Table of timer constants for RS-232 baud rate
$E2F8	Table for VDC initialitzation
$E2C7	Table for VIC initialitzation
$FCC3	Test accent keys and combine accents
$CB74	Test line overflow bit
$C2FF	Test quote character and set pointer
$B7A5	Test separator between command operands
$EA8F	Test the STOP key
$E9DF	Test for tape button
$C8DC	Turn off cursor flash mode
$CB1A	Turn cursor flash off for 40-column mode
$CB2E	Turn cursor flash on for 40-column mode
$CB0B	Turn cursor flash off for 80-column mode
$CB21	Turn cursor flash on for 80-column mode
$CB48	Turn off 80-column reverse
$CB3F	Turn on 80-column reverse
$C8CE	Turn underline mode off
$C8C7	Turn underline mode on
$CAFE	Turn underline cursor on

$C06F	Vector table to ASCII decoder tables
$FE34	Vector table to DIN decoder tables (International Versions only)
$C000	Vector table for editor routines
$C9DE	Vector table for editor routines
$C7B6	Vector table for control code routines
$F3EA	Verify routine in burst mode
$EA7D	Wait for tape I/O termination
$E7EC	Wait for end of RS-232 tranfer
$E5BC	Wait for fast-mode response from bus
$E9E9	Wait for RECORD & PLAY on Datasette
$E9C8	Wait for button on datasette
$EA15	Write tape buffer to tape
$ED69	Write bit to tape
$E919	Write data block to tape
$E919	Write header to tape
$EA1C	Write data block to tape
$EE2E	Write the header

8.4 The Token Table

The Commodore BASIC 7.0 is, in contrast to BASIC 2.0 on the C-64, extended with a number of new commands and instructions. As you know, BASIC commands are not saved in their text forms, but in the form of so-called "tokens". In order to ensure unambiguous identification of tokens and other text characters, the code values 128 to 256 are reserved for the tokens. This is exactly 128 possible values with which a token can be indicated. But BASIC 7.0 has more than 128 different command keywords. For this reason, there are some tokens which require two values to denote a keyword. The BASIC interpreter recognizes the two values as a token. Here is a table of all the command keywords and the token values associated with them.

Command	Token	Command	Token
END	$80	FOR	$81
NEXT	$82	DATA	$83
INPUT#	$84	INPUT	$85
DIM	$86	READ	$87
LET	$88	GOTO	$89
RUN	$8A	IF	$8B
RESTORE	$8C	GOSUB	$8D
RETURN	$8E	REM	$8F
STOP	$90	ON	$91
WAIT	$92	LOAD	$93
SAVE	$94	VERIFY	$95
DEF	$96	POKE	$97
PRINT#	$98	PRINT	$99
CONT	$9A	LIST	$9B
CLR	$9C	CMD	$9D
SYS	$9E	OPEN	$9F
CLOSE	$A0	GET	$A1
NEW	$A2	TAB(	$A3
TO	$A4	FN	$A5
SPC(	$A6	THEN	$A7
NOT	$A8	STEP	$A9
+	$AA	-	$AB
*	$AC	/	$AD
^	$AE	AND	$AF
OR	$B0	>	$B1

Command	Token	Command	Token
=	$B2	<	$B3
SGN	$B4	INT	$B5
ABS	$B6	USR	$B7
FRE	$B8	POS	$B9
SQR	$BA	RND	$BB
LOG	$BC	EXP	$BD
COS	$BE	SIN	$BF
TAN	$C0	ATN	$C1
PEEK	$C2	LEN	$C3
STR$	$C4	VAL	$C5
ASC	$C6	CHR$	$C7
LEFT$	$C8	RIGHT$	$C9
MID$	$CA	GO	$CB
RGR	$CC	RCLR	$CD
POT	$CE $02	BUMP	$CE $03
PEN	$CE $04	RSPPOS	$CE $05
RSPRITE	$CE $06	RSPCOLOR	$CE $07
XOR	$CE $08	RWINDOW	$CE $09
POINTER	$CE $0A	JOY	$CF
RDOT	$D0	DEC	$D1
HEX$	$D2	ERR$	$D3
INSTR	$D4	ELSE	$D5
RESUME	$D6	TRAP	$D7
TRON	$D8	TROFF	$D9
SOUND	$DA	VOL	$DB
AUTO	$DC	PUDEF	$DD
GRAPHIC	$DE	PAINT	$DF
CHAR	$E0	BOX	$E1
CIRCLE	$E2	GSHAPE	$E3
SSHAPE	$E4	DRAW	$E5
LOCATE	$E6	COLOR	$E7
SCNCLR	$E8	SCALE	$E9
HELP	$EA	DO	$EB
LOOP	$EC	EXIT	$ED
DIRECTORY	$EE	DSAVE	$EF
DLOAD	$F0	HEADER	$F1
SCRATCH	$F2	COLLECT	$F3
COPY	$F4	RENAME	$F5
BACKUP	$F6	DELETE	$F7
RENUMBER	$F8	KEY	$F9
MONITOR	$FA	USING	$FB

Command	Token	Command	Token
UNTIL	$FC	WHILE	$FD
BANK	$FE $02	FILTER	$FE $03
PLAY	$FE $04	TEMPO	$FE $05
MOVSPR	$FE $06	SPRITE	$FE $07
SPRCOLOR	$FE $08	RREG	$FE $09
ENVELOPE	$FE $0A	SLEEP	$FE $0B
CATALOG	$FE $0C	DOPEN	$FE $0D
APPEND	$FE $0E	DCLOSE	$FE $0F
BSAVE	$FE $10	BLOAD	$FE $11
RECORD	$FE $12	CONCAT	$FE $13
DVERIFY	$FE $14	DCLEAR	$FE $15
SPRSAV	$FE $16	COLLISION	$FE $17
BEGIN	$FE $18	BEND	$FE $19
WINDOW	$FE $1A	BOOT	$FE $1B
WIDTH	$FE $1C	SPRDEF	$FE $1D
QUIT	$FE $1E	STASH	$FE $1F
FETCH	$FE $21	SWAP	$FE $23
OFF	$FE $24	FAST	$FE $25
SLOW	$FE $26		

8.5 The Character Set

On the following pages you find two character sets, the normal Commodore character set (the only one in the American version) and the DIN (German [Deutshe] Industry Normal) foreign language set. They contain information about the address at which the matrix of the character is located, as well as the value of the POKE code in parentheses.

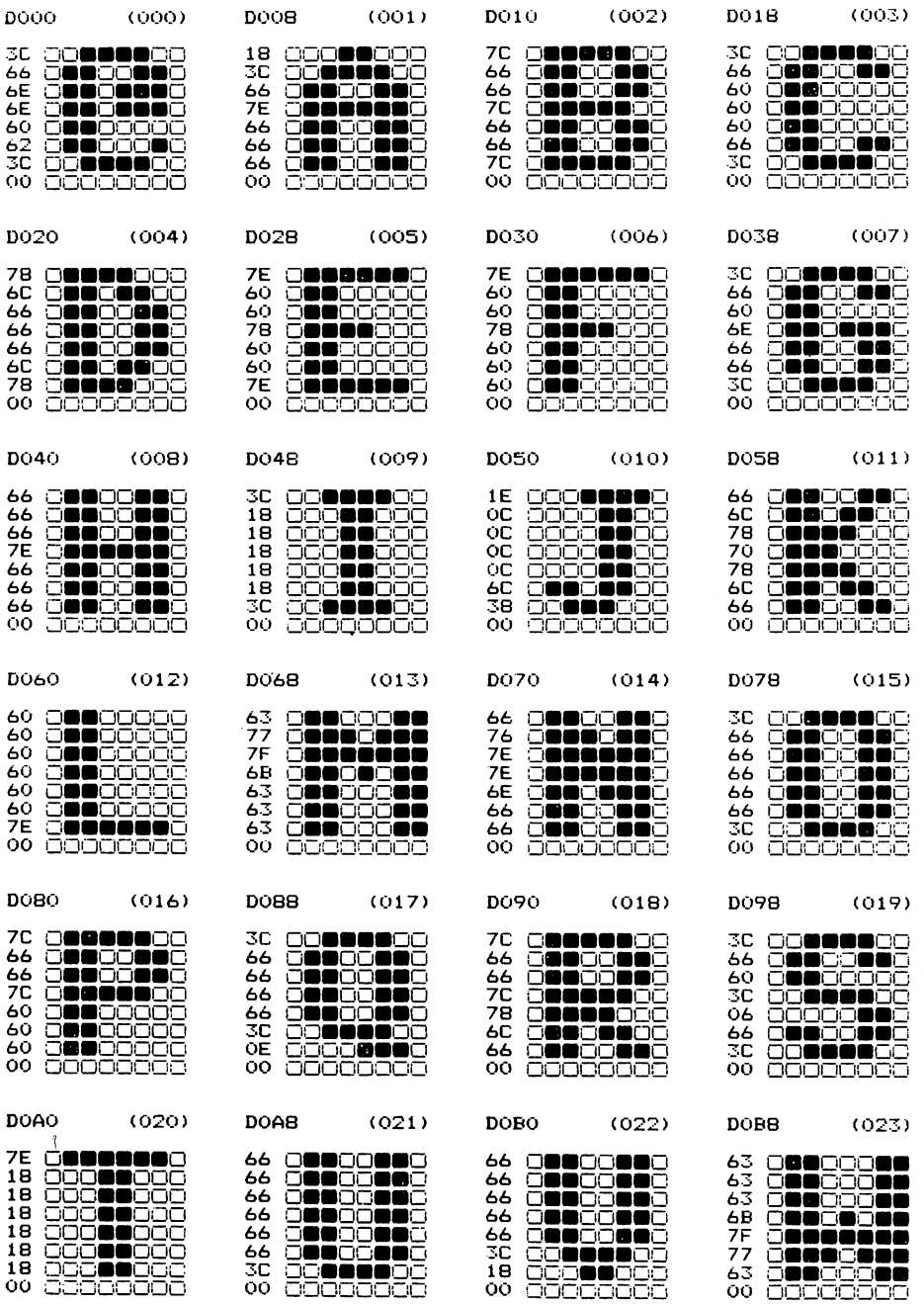
The C-128's sold in Europe contain two character sets, the normal Commodore character set and in the German versions a DIN (German [Deutshe] Industry Normal) character set for foreign languages. C-128s sold in other foreign countries may have a different International character set than the one presented here, we have checked only the American and German versions. See the differences in the ROM listing starting at $FC80 thru $FEFF and at $C012. Notice that the KEY vector at $FF9F in the Kernal Jump Table points to the same location but that the address at that location ($C012) is different for the American ($C55D) and German ($FC87) versions. The German version jumps to the standard keyboard matrix reading routine ($C55D) at address $FCC3. On the International versions you can switch between the two character sets by pressing the ASCII/DIN key (CAPS LOCK on American versions). The key is polled through interrupts, meaning that it is recognized immediately when it is pressed. The character set on the 40-column screen changes immediately and on the 80 column screen the computer pauses for about one second. This is because the computer has to copy the character set to the VDC (80-column controller) memory because this controller does not get its characters from the ROM.

Physically the two character sets, ASCII and DIN, are at the same address, namely $D000. When the ASCII/DIN key is pressed, the two character sets are exchanged via hardware.

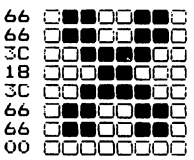
To save space in the book, we have not pictured the reverse characters. To obtain the address of these characters, add the offset $0400 to the base address of the normal character.

You can easily change the character set for the 80-column controller by changing the corresponding addresses in the VDC RAM. Chapter 5 contains more information about this and other aspects of the VDC.

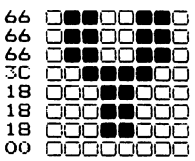
You can also change the VIC character set by changing the character set pointer in CIA 1. More information about this can be found in the chapter on the VIC chip, Chapter 2.


D000 (000)
3C 66 6E 6E 60 62 3C 00
D008 (001)
18 3C 66 7E 66 66 66 00
D010 (002)
7C 66 66 7C 66 66 7C 00
D018 (003)
3C 66 60 60 60 66 3C 00
D020 (004)
78 6C 66 66 66 6C 78 00
D028 (005)
7E 60 60 78 60 60 7E 00
D030 (006)
7E 60 60 78 60 60 60 00
D038 (007)
3C 66 60 6E 66 66 3C 00
D040 (008)
66 66 66 7E 66 66 66 00
D048 (009)
3C 18 18 18 18 18 3C 00
D050 (010)
1E 0C 0C 0C 0C 6C 38 00
D058 (011)
66 6C 78 70 78 6C 66 00
D060 (012)
60 60 60 60 60 60 7E 00
D068 (013)
63 77 7F 6B 63 63 63 00
D070 (014)
66 76 7E 7E 6E 66 66 00
D078 (015)
3C 66 66 66 66 66 3C 00
D080 (016)
7C 66 66 7C 60 60 60 00
D088 (017)
3C 66 66 66 66 3C 0E 00
D090 (018)
7C 66 66 7C 78 6C 66 00
D098 (019)
3C 66 60 3C 06 66 3C 00
D0A0 (020)
7E 18 18 18 18 18 18 00
D0A8 (021)
66 66 66 66 66 66 3C 00
D0B0 (022)
66 66 66 66 66 3C 18 00
D0B8 (023)
63 63 63 6B 7F 77 63 00

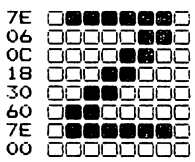
D0C0 (024)



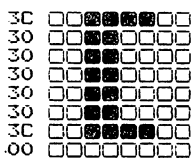
D0C8 (025)



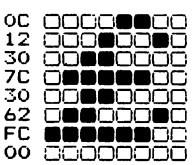
D0D0 (026)



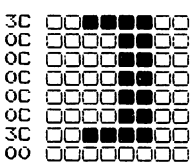
D0D8 (027)



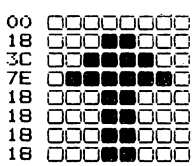
D0E0 (028)



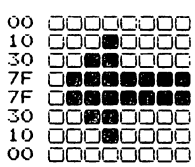
D0E8 (029)



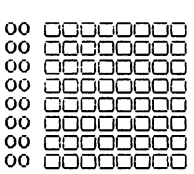
D0F0 (030)



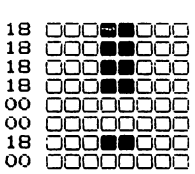
D0F8 (031)



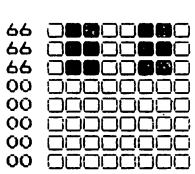
D100 (032)



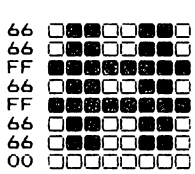
D108 (033)



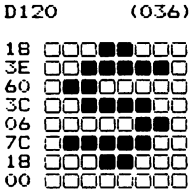
D110 (034)



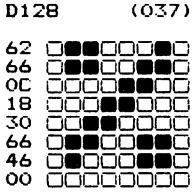
D118 (035)



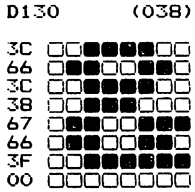
D120 (036)



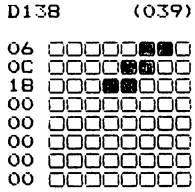
D128 (037)



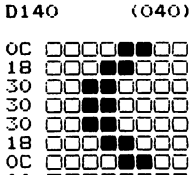
D130 (038)



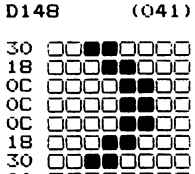
D138 (039)



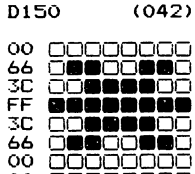
D140 (040)



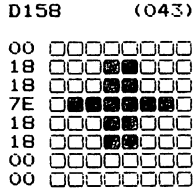
D148 (041)



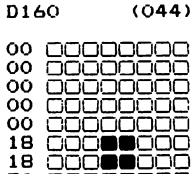
D150 (042)



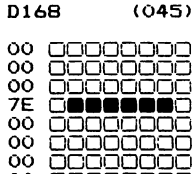
D158 (043)



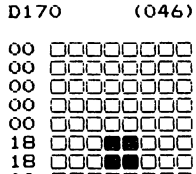
D160 (044)



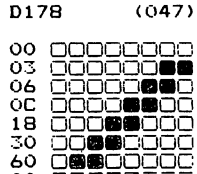
D168 (045)



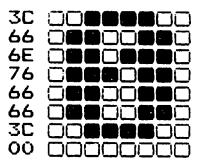
D170 (046)



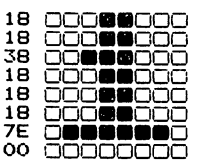
D178 (047)



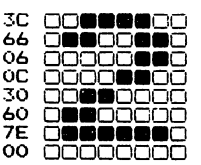
D180 (048)



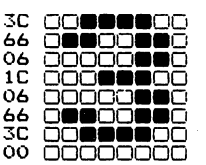
D188 (049)



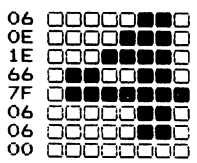
D190 (050)



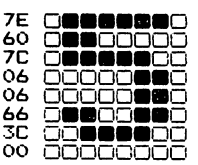
D198 (051)



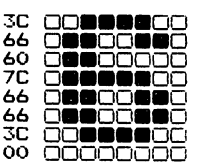
D1A0 (052)



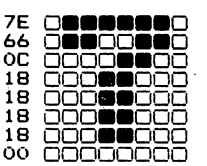
D1A8 (053)



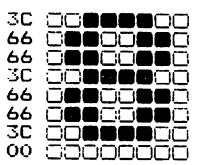
D1B0 (054)



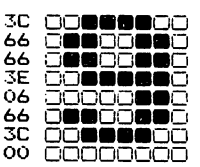
D1B8 (055)



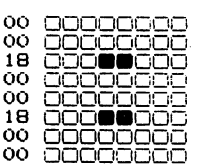
D1C0 (056)



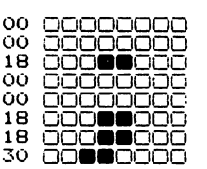
D1C8 (057)



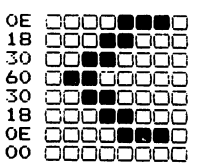
D1D0 (058)



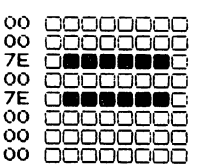
D1D8 (059)



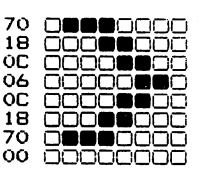
D1E0 (060)



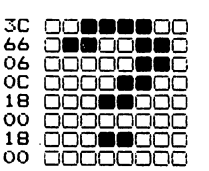
D1E8 (061)



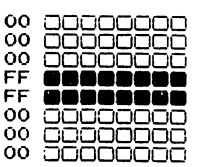
D1F0 (062)



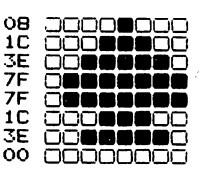
D1F8 (063)



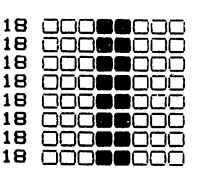
D200 (064)



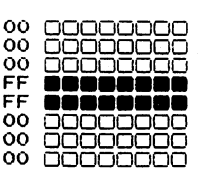
D208 (065)



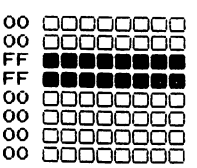
D210 (066)



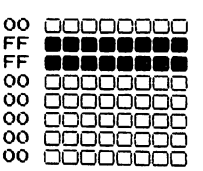
D218 (067)



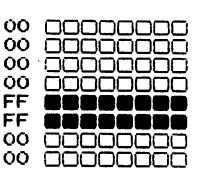
D220 (068)



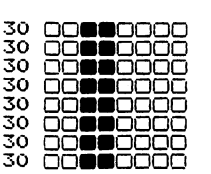
D228 (069)



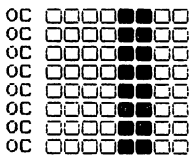
D230 (070)



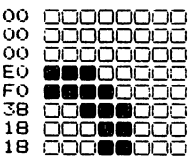
D238 (071)



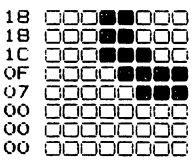
D240 (072)



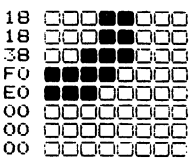
D248 (073)



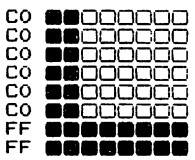
D250 (074)



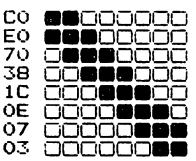
D258 (075)



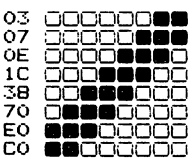
D260 (076)



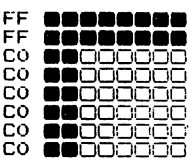
D268 (077)



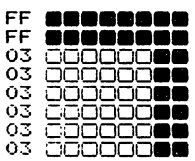
D270 (078)



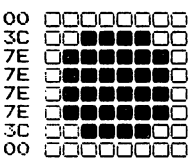
D278 (079)



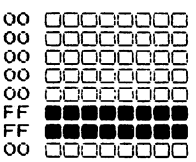
D280 (080)



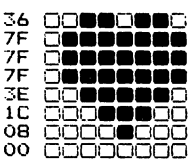
D288 (081)



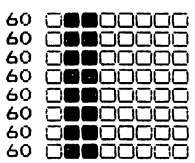
D290 (082)



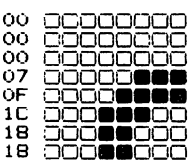
D298 (083)



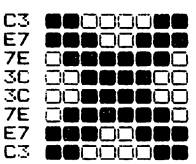
D2A0 (084)



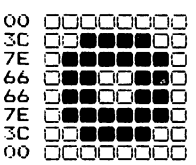
D2A8 (085)



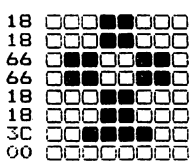
D2B0 (086)



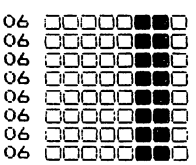
D2B8 (087)



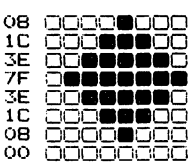
D2C0 (088)



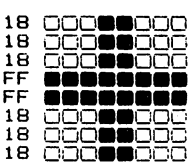
D2C8 (089)



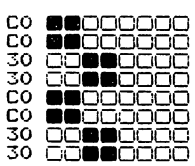
D2D0 (090)



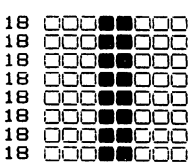
D2D8 (091)



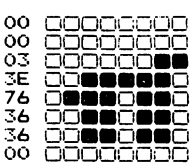
D2E0 (092)



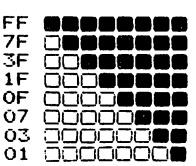
D2E8 (093)



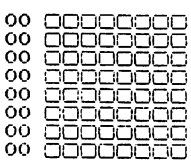
D2F0 (094)



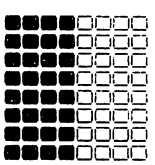
D2F8 (095)



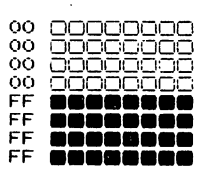
D300 (096)



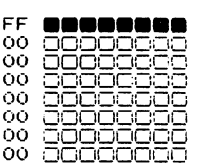
D308 (097)



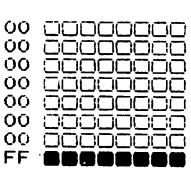
D310 (098)



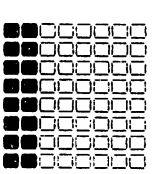
D318 (099)



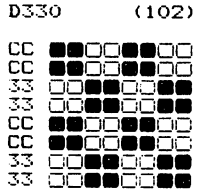
D320 (100)



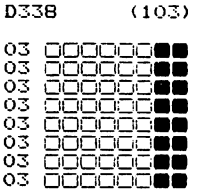
D328 (101)



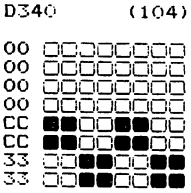
D330 (102)



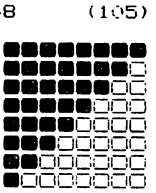
D338 (103)



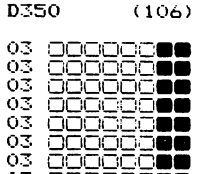
D340 (104)



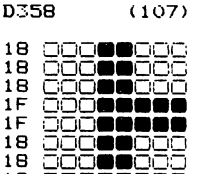
D348 (105)



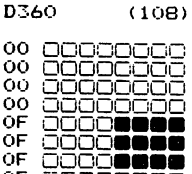
D350 (106)



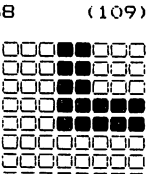
D358 (107)



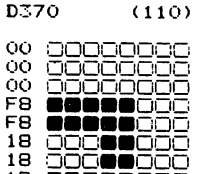
D360 (108)



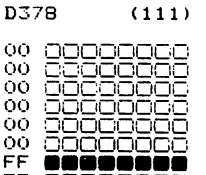
D368 (109)



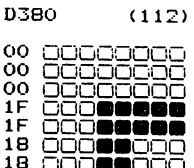
D370 (110)



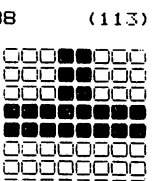
D378 (111)



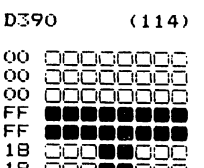
D380 (112)



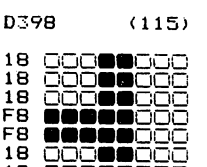
D388 (113)



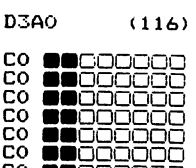
D390 (114)



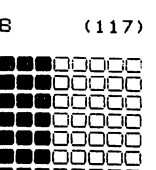
D398 (115)



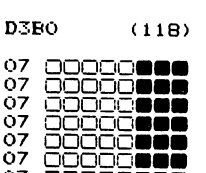
D3A0 (116)



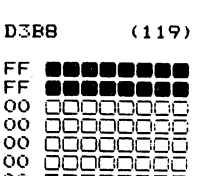
D3A8 (117)

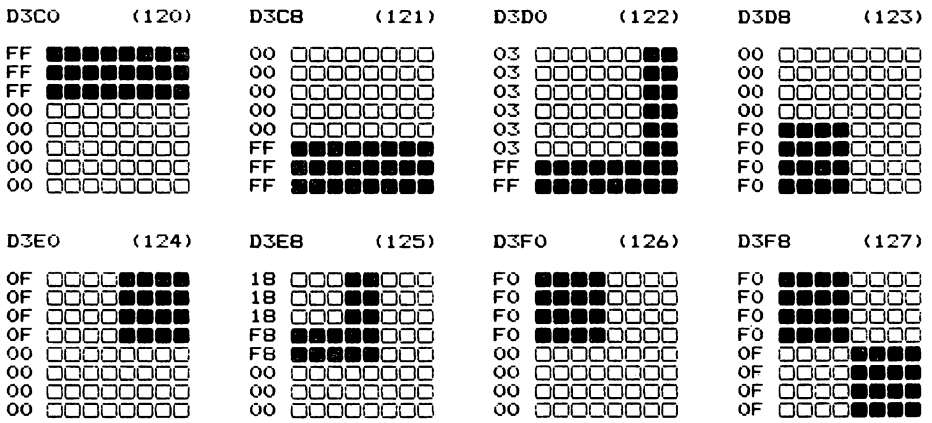


D3B0 (118)

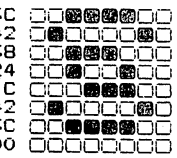


D3B8 (119)

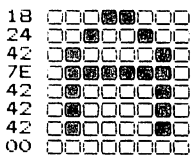



D3C0 (120)
FF
FF
FF
00
00
00
00
00
D3C8 (121)
00
00
00
00
00
FF
FF
FF
D3D0 (122)
03
03
03
03
03
03
FF
FF
D3D8 (123)
00
00
00
00
F0
F0
F0
F0
D3E0 (124)
0F
0F
0F
0F
00
00
00
00
D3E8 (125)
18
18
18
F8
F8
00
00
00
D3F0 (126)
F0
F0
F0
F0
00
00
00
00
D3F8 (127)
F0
F0
F0
F0
0F
0F
0F
0F

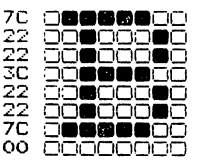
D000 (000)



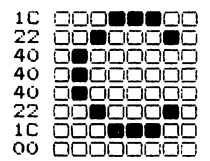
D008 (001)



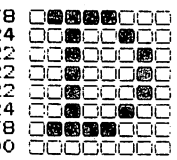
D010 (002)



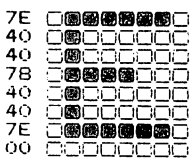
D018 (003)



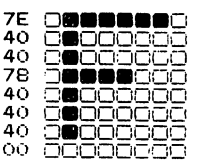
D020 (004)



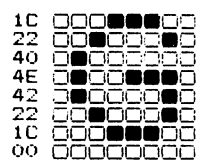
D028 (005)



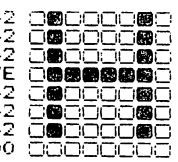
D030 (006)



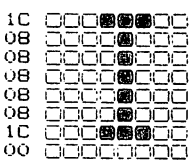
D038 (007)



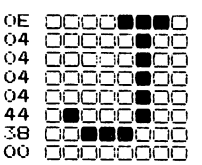
D040 (008)



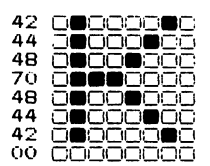
D048 (009)



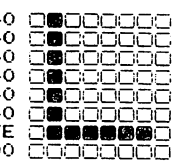
D050 (010)



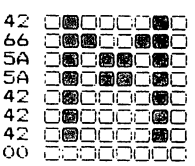
D058 (011)



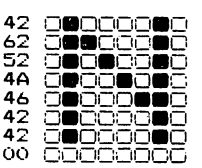
D060 (012)



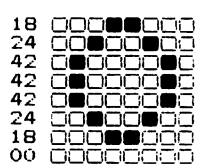
D068 (013)



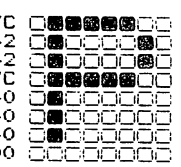
D070 (014)



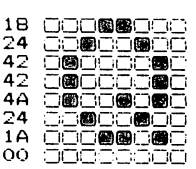
D078 (015)



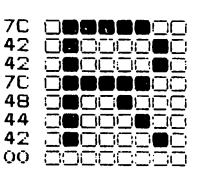
D080 (016)



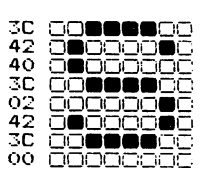
D088 (017)



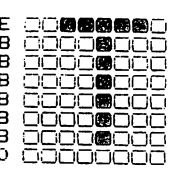
D090 (018)



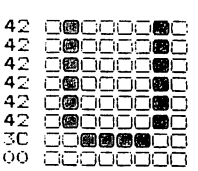
D098 (019)



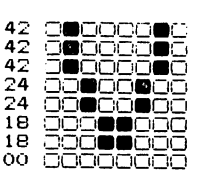
D0A0 (020)



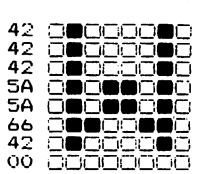
D0A8 (021)

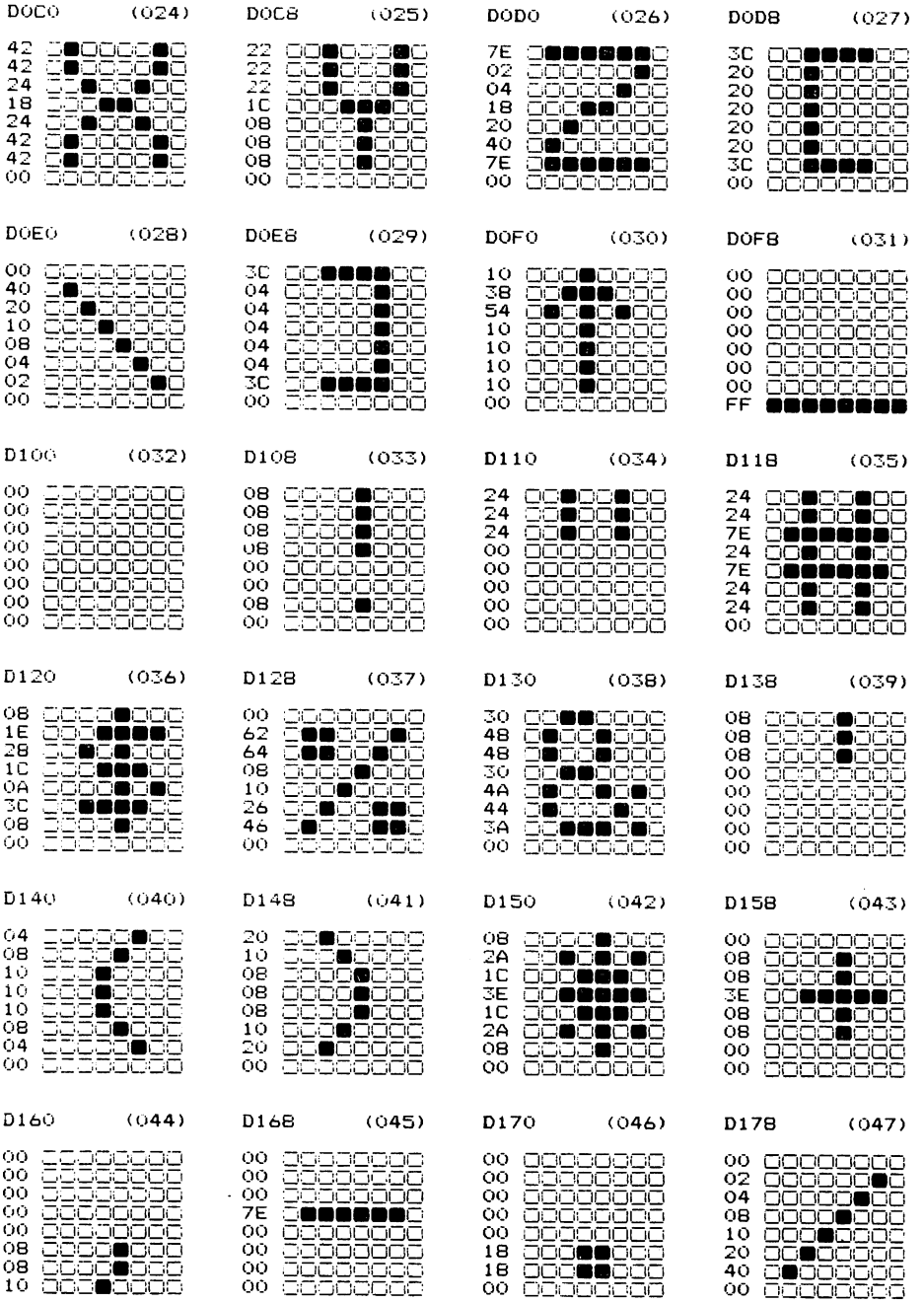


D0B0 (022)

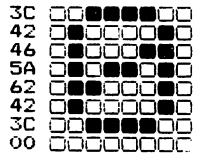


D0B8 (023)

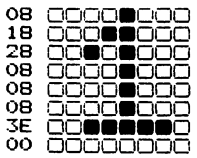



D0C0 (024)
42
42
24
18
24
42
42
00
D0C8 (025)
22
22
22
1C
08
08
08
00
D0D0 (026)
7E
02
04
18
20
40
7E
00
D0D8 (027)
3C
20
20
20
20
20
3C
00
D0E0 (028)
00
40
20
10
08
04
02
00
D0E8 (029)
3C
04
04
04
04
04
3C
00
D0F0 (030)
10
38
54
10
10
10
10
00
D0F8 (031)
00
00
00
00
00
00
00
FF
D100 (032)
00
00
00
00
00
00
00
00
D108 (033)
08
08
08
08
00
00
08
00
D110 (034)
24
24
24
00
00
00
00
00
D118 (035)
24
24
7E
24
7E
24
24
00
D120 (036)
08
1E
28
1C
0A
3C
08
00
D128 (037)
00
62
64
08
10
26
46
00
D130 (038)
30
48
48
30
4A
44
3A
00
D138 (039)
08
08
08
00
00
00
00
00
D140 (040)
04
08
10
10
10
08
04
00
D148 (041)
20
10
08
08
08
10
20
00
D150 (042)
08
2A
1C
3E
1C
2A
08
00
D158 (043)
00
08
08
3E
08
08
00
00
D160 (044)
00
00
00
00
00
08
08
10
D168 (045)
00
00
00
7E
00
00
00
00
D170 (046)
00
00
00
00
00
18
18
00
D178 (047)
00
02
04
08
10
20
40
00

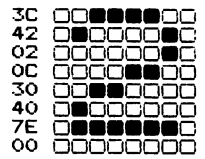
D180 (048)



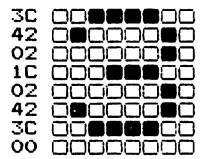
D188 (049)



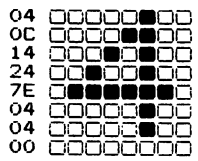
D190 (050)



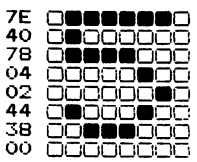
D198 (051)



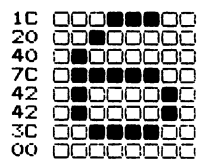
D1A0 (052)



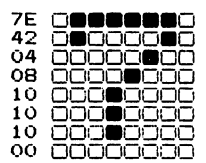
D1A8 (053)



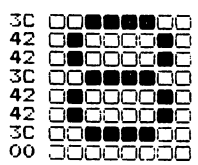
D1B0 (054)



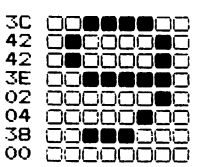
D1B8 (055)



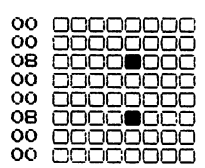
D1C0 (056)



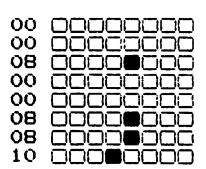
D1C8 (057)



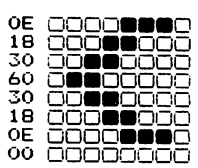
D1D0 (058)



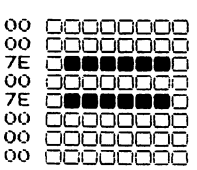
D1D8 (059)



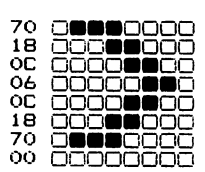
D1E0 (060)



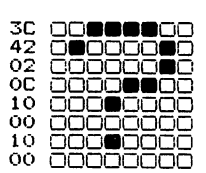
D1E8 (061)



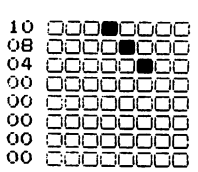
D1F0 (062)



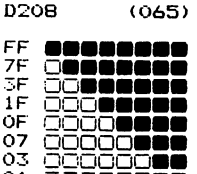
D1F8 (063)



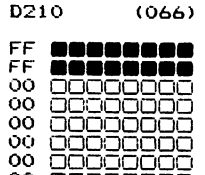
D200 (064)



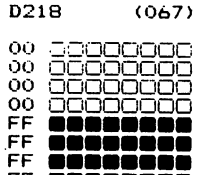
D208 (065)



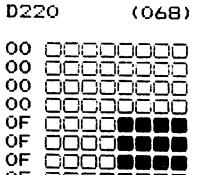
D210 (066)



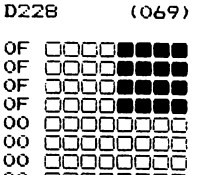
D218 (067)



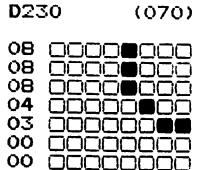
D220 (068)



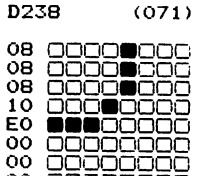
D228 (069)

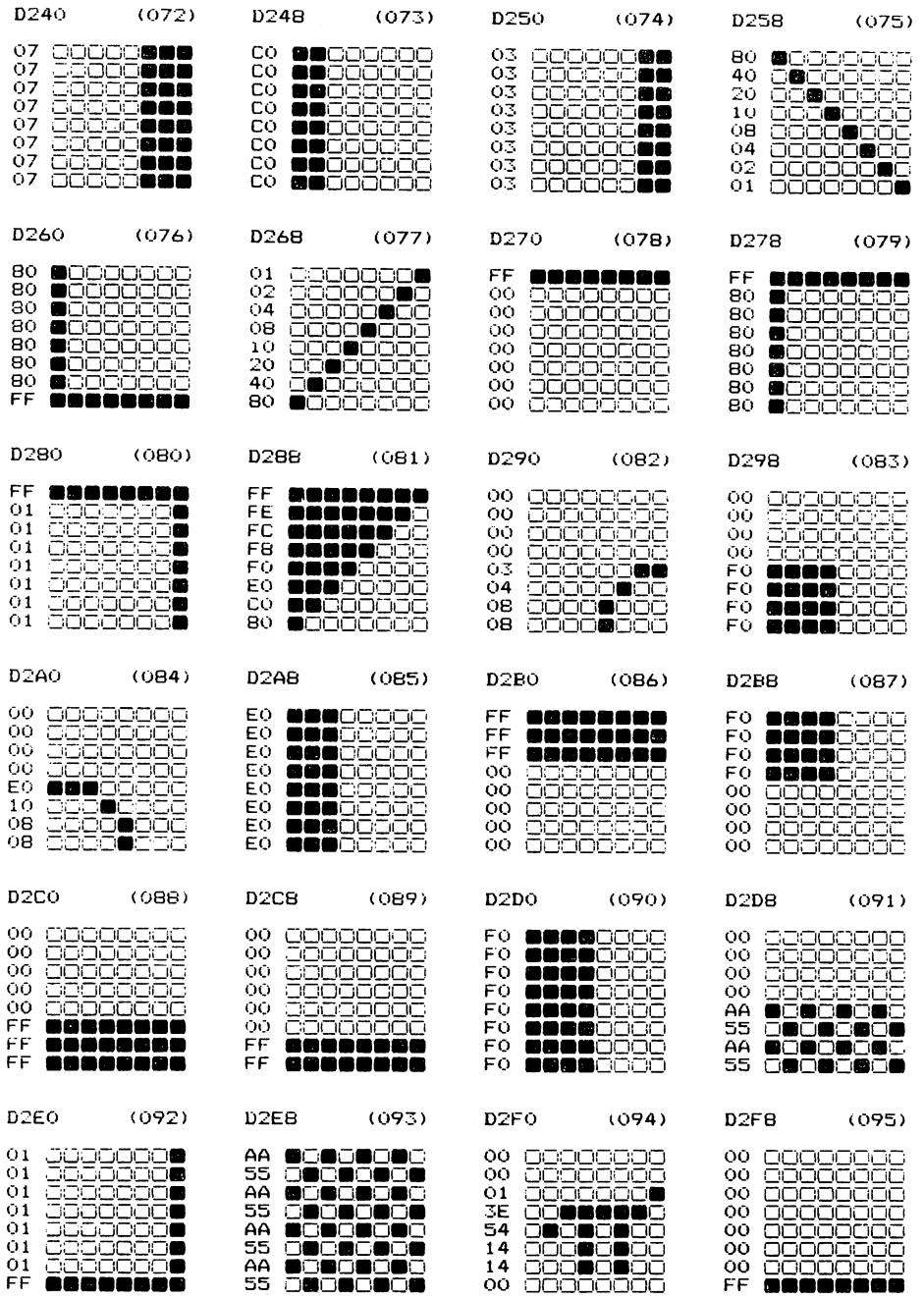


D230 (070)

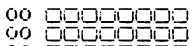


D238 (071)

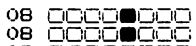



D240 (072)
07 07 07 07 07 07 07 07
D248 (073)
C0 C0 C0 C0 C0 C0 C0 C0
D250 (074)
03 03 03 03 03 03 03 03
D258 (075)
80 40 20 10 08 04 02 01
D260 (076)
80 80 80 80 80 80 80 FF
D268 (077)
01 02 04 08 10 20 40 80
D270 (078)
FF 00 00 00 00 00 00 00
D278 (079)
FF 80 80 80 80 80 80 80
D280 (080)
FF 01 01 01 01 01 01 01
D288 (081)
FF FE FC F8 F0 E0 C0 80
D290 (082)
00 00 00 00 03 04 08 08
D298 (083)
00 00 00 00 F0 F0 F0 F0
D2A0 (084)
00 00 00 00 E0 10 08 08
D2A8 (085)
E0 E0 E0 E0 E0 E0 E0 E0
D2B0 (086)
FF FF FF 00 00 00 00 00
D2B8 (087)
F0 F0 F0 F0 00 00 00 00
D2C0 (088)
00 00 00 00 00 FF FF FF
D2C8 (089)
00 00 00 00 00 00 FF FF
D2D0 (090)
F0 F0 F0 F0 F0 F0 F0 F0
D2D8 (091)
00 00 00 00 AA 55 AA 55
D2E0 (092)
01 01 01 01 01 01 01 FF
D2E8 (093)
AA 55 AA 55 AA 55 AA 55
D2F0 (094)
00 00 01 3E 54 14 14 00
D2F8 (095)
00 00 00 00 00 00 00 FF

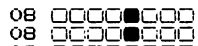
D300 (096)



D308 (097)



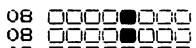
D310 (098)



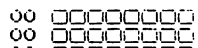
D318 (099)

D320 (100)

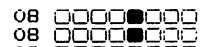
D328 (101)



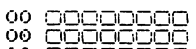
D330 (102)



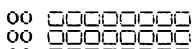
D338 (103)



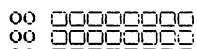
D340 (104)



D348 (105)



D350 (106)



D358 (107)

D360 (108)

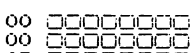
D368 (109)

D370 (110)

D378 (111)

D380 (112)

D388 (113)



D390 (114)

D398 (115)

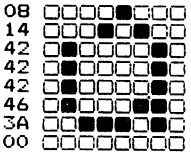
D3A0 (116)

D3A8 (117)

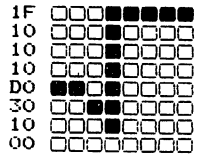
D3B0 (118)

D3B8 (119)

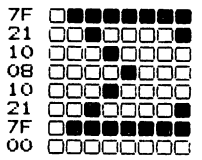
D3C0 (120)



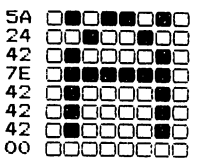
D3C8 (121)



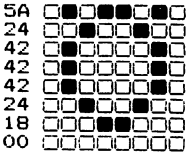
D3D0 (122)



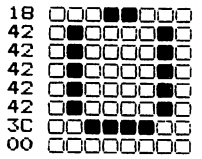
D3D8 (123)



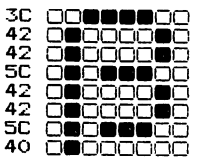
D3E0 (124)



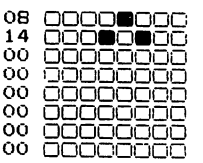
D3E8 (125)



D3F0 (126)



D3F8 (127)



8.6 The Keyboard Matrix

The keyboard of the C-128 is designed in the form of a matrix. Imagine it as a network (or grid) of lines. In the horizontal plane you have 11 lines and in the vertical, 8 lines. When you press a key, you close the normally open contact between a horizontal and a vertical line. The computer can then recognize which key was pressed.

That's the basic principle of the keyboard matrix. In practice it is much more complicated since a connection is not available on an I/O component for each of the 11 horizontal and 8 vertical lines. The Commodore 128 has two components with a total of three ports that have the task of reading the keyboard matrix. Lines PA0-PA7 and PB0-PB7 are available on CIA 1. These 16 lines can be programmed as either input or output. Theoretically it is also possible to transfer 16-bit values via these lines. Lines PA0-PA7 are responsible for the first 8 matrix lines of the keyboard circuit. The missing three lines, believe it or not, are connected to the VIC chip.

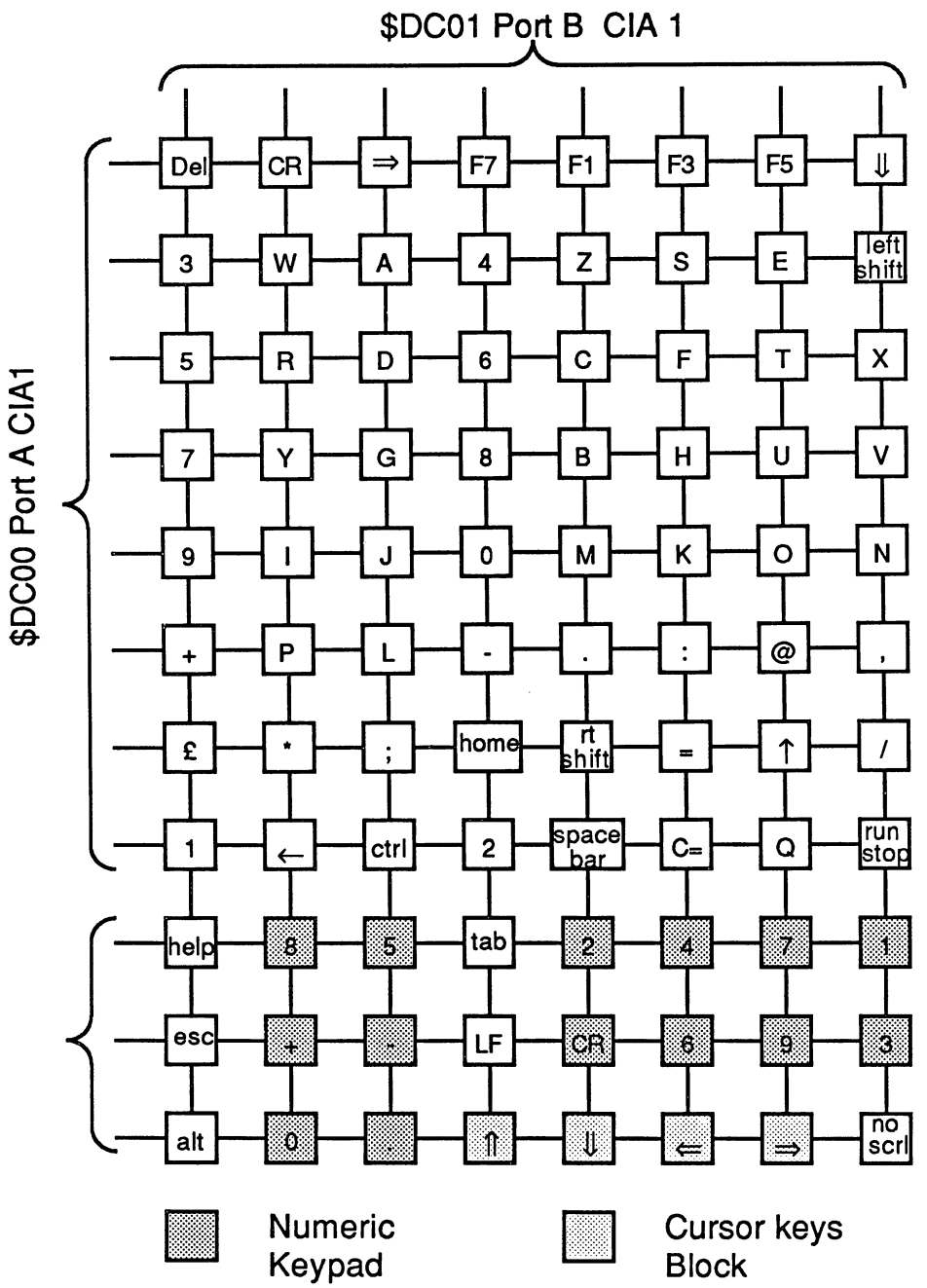
The VIC chip built into your C-128 has 2 more registers than the component used in the Commodore 64. The first is the register at address $D030, is responsible for the clock frequency at which the computer will operate (1 or 2 MHz). This register does not interest us here. The other new register is at address $D02F. The additional three keyboard matrix lines are polled via this register. The register offers us bits 0-3 for this, but only bits 0-2 are used, since only three additional matrix lines need to be polled. The 8 matrix columns are addressed via the lines PB0-PB7 of CIA1 via port B.

The actual keyboard polling follows this pattern. Port A of CIA 1 (lines PA0-PA7 are brought low; that is, the register is loaded with the hex value $00). In addition, the remaining three lines must also be loaded with a low value in the VIC register. Port B (lines PB0-PB7) of CIA 1, switched to input, is then read. If a key is pressed at some point, one of the input lines on port B will also be switched to a low level. This is recognized by reading port B and finding a value other than the high value ($FF). At this point, we can determine that a key was pressed. Which key it is cannot yet be determined.

The exact position within the keyboard matrix is then determined by bringing each of the 11 matrix lines low in turn and reading port B each time. Now we can tell in which line and column of the matrix the key was pressed. A count register is used during this process in order to record the assignment number of the pressed key. Polling the joystick is done in the

same manner as the normal keyboard polling because the joystick connections are wired in parallel to some keys on the keyboard.

In the schematic on the next page you can recognize the physical layout of the keys and their connections to the three ports. One point of interest is that, while the keys on the keypad produce the same results on the screen as the regular digit keys, they can be differentiated from them. This applies for the cursor control keys and the other duplicate keys on the keyboard.


$DC01 Port B CIA 1
$DC00 Port A CIA1
Del CR ⇒ F7 F1 F3 F5 ⇓
3 W A 4 Z S E left shift
5 R D 6 C F T X
7 Y G 8 B H U V
9 I J 0 M K O N
+ P L - . : @ ,
£ * ; home rt shift = ↑ /
1 ← ctrl 2 space bar C= Q run stop
help 8 5 tab 2 4 7 1
esc + - LF CR 6 9 3
alt 0 . ↑ ⇓ ⇐ ⇒ no scrl
Numeric Keypad
Cursor keys Block

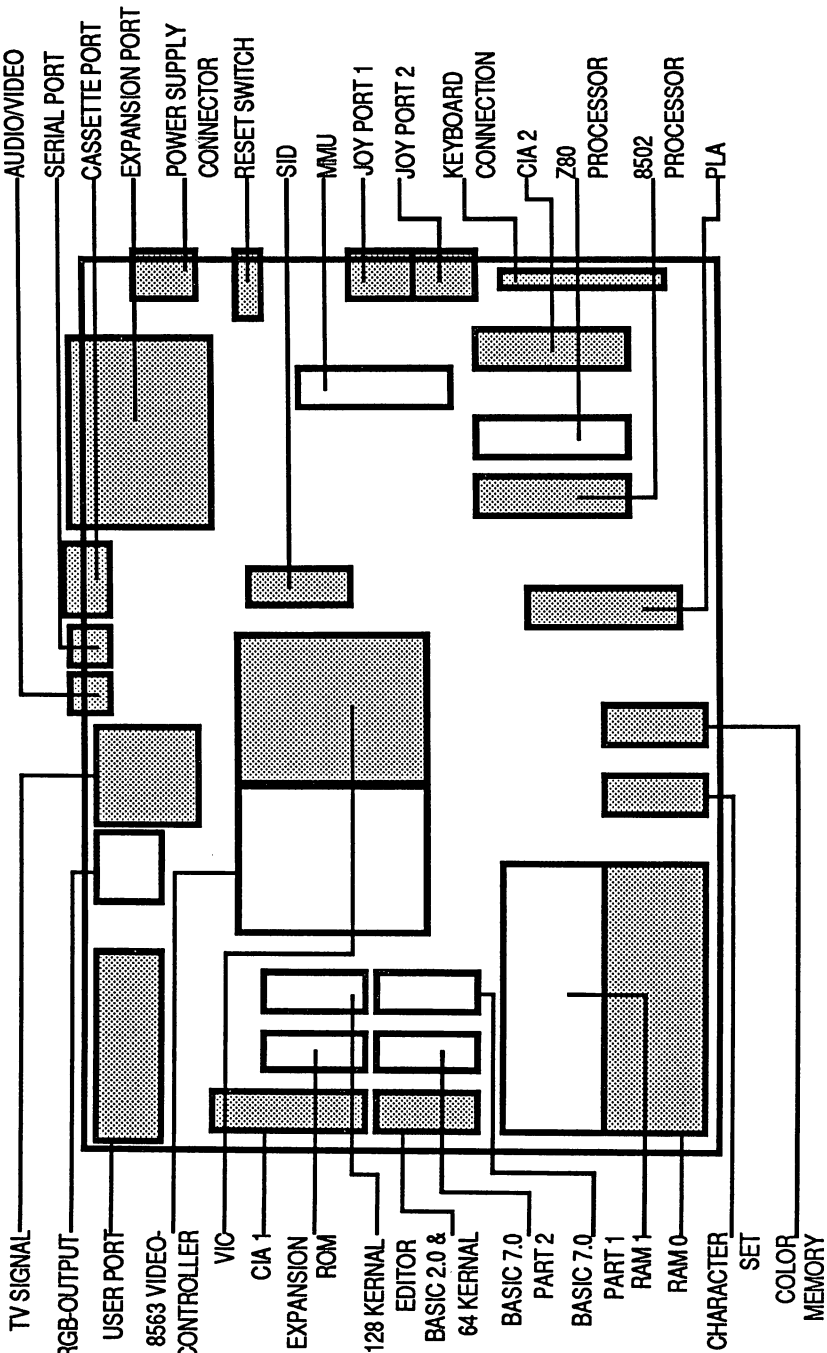
8.7 The Computer Modes

As you must know by now, your Commodore 128 contains *three* computers in one. You can select whether you want to have a:

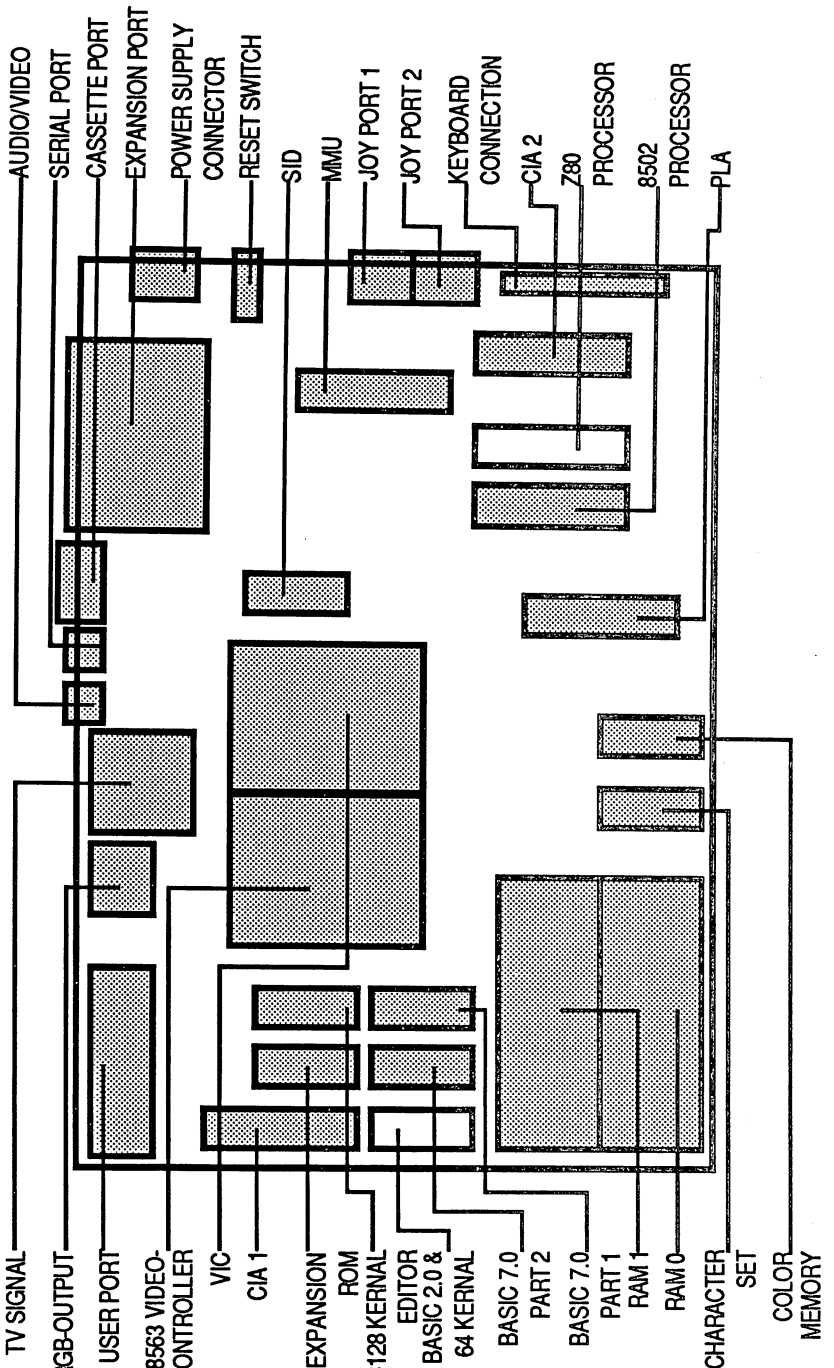
* **CP/M 3.0+ computer**
* **Commodore 128**
* **Commodore 64**

The various components in the computer are switched on or off depending on the computer mode. In order to show you graphically which components are involved, we have made the following three figures. Shaded areas designate the devices active in the given mode while unshaded areas indicate those which are inactive. Inactive means that the MMU does not permit access to these components. In the C64 mode, access to the MMU itself is prohibited (for compatibility reasons).

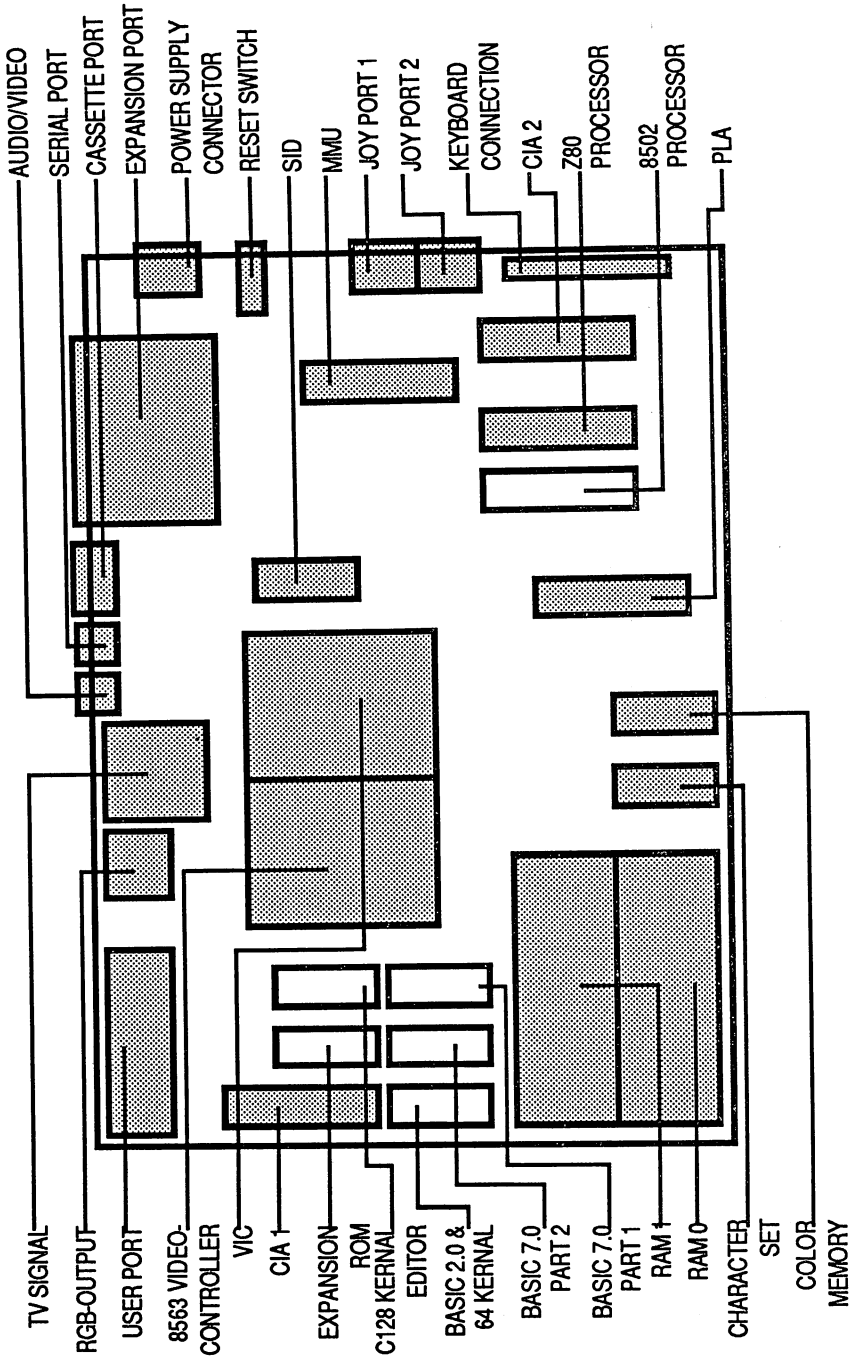
C-64 MODE


AUDIO/VIDEO
SERIAL PORT
CASSETTE PORT
EXPANSION PORT
POWER SUPPLY CONNECTOR
RESET SWITCH
SID
MMU
JOY PORT 1
JOY PORT 2
KEYBOARD CONNECTION
CIA 2
Z80 PROCESSOR
8502 PROCESSOR
PLA
TV SIGNAL
RGB-OUTPUT
USER PORT
8563 VIDEO-CONTROLLER
VIC
CIA 1
EXPANSION ROM
C128 KERNAL
EDITOR
BASIC 2.0 & 64 KERNAL
BASIC 7.0 PART 2
BASIC 7.0 PART 1
RAM 1
RAM 0
CHARACTER SET
COLOR MEMORY

C-128 MODE


AUDIO/VIDEO
SERIAL PORT
CASSETTE PORT
EXPANSION PORT
POWER SUPPLY CONNECTOR
RESET SWITCH
SID
MMU
JOY PORT 1
JOY PORT 2
KEYBOARD CONNECTION
CIA 2
Z80 PROCESSOR
8502 PROCESSOR
PLA
TV SIGNAL
RGB-OUTPUT
USER PORT
8563 VIDEO-CONTROLLER
VIC
CIA 1
EXPANSION
ROM C128 KERNAL
EDITOR
BASIC 2.0 & 64 KERNAL
BASIC 7.0 PART 2
BASIC 7.0 PART 1
RAM 1
RAM 0
CHARACTER SET
COLOR MEMORY

CP/M MODE


AUDIO/VIDEO
SERIAL PORT
CASSETTE PORT
EXPANSION PORT
POWER SUPPLY CONNECTOR
RESET SWITCH
SID
MMU
JOY PORT 1
JOY PORT 2
KEYBOARD CONNECTION
CIA 2
Z80 PROCESSOR
8502 PROCESSOR
PLA
TV SIGNAL
RGB-OUTPUT
USER PORT
8563 VIDEO-CONTROLLER
VIC
CIA 1
EXPANSION
ROM C128 KERNAL
EDITOR
BASIC 2.0 & 64 KERNAL
BASIC 7.0 PART 2
BASIC 7.0 PART 1
RAM 1
RAM 0
CHARACTER SET
COLOR MEMORY

8.7.1 The power-up modes

On the preceding three pages you see three diagrams. These schematic drawings of the chips and circuits in your Commodore 128 should make it clear to you which ROMs and controllers are active in each of the three modes of operation. The active components in each operating mode are shaded.

As a rule, the computer always tries to enter the 128 mode when it is turned on. But there are some special cases in which the computer is directly switched to a different operating mode. This is the case when you insert a CP/M diskette into the disk drive. The CP/M mode with the Z-80 processor active is then enabled via the boot routine in the 128 mode. Another possibility arises when you insert a Commodore 64 cartridge in the expansion port. This is also noted during the power-up procedure and the computer switches directly to the C-64 mode responsible for this cartridge.

Another way of entering the C-64 mode is by way of the `GO 64` command. After an appropriate request for confirmation of the command, the computer is switched to the C-64 mode. It is also possible to get around the BASIC interpreter's confirmation request and enter the C-64 mode directly. You can do this by directly addressing the kernal routine for reconfiguration with a SYS command. The appropriate SYS command is:

SYS 57931 or SYS DEC("E24B")

These are, so to speak, the "official" options for entering another operating mode, especially the C-64 mode. But there are a few "unofficial" ways, which we discovered by accident while documenting the kernal and BIOS.

One such method involves the Commodore key, designated with the Commodore logo and found in the lower left-hand corner of the keyboard. If you hold this key down during the power-up procedure, the computer will enter the C-64 mode directly without trying to load a boot sector from the diskette or entering the 128 mode. The obligatory confirmation question is also avoided with this method. This trick with the Commodore key works not only when the computer is being turned on, but also if you hold it down while pressing the reset button on the right side.

Another interesting option affecting the power-up state of the computer involves holding down the RUN/STOP key while turning the computer on. This causes the computer to enter the 128 mode, but control is immediately passed to the built-in monitor. Furthermore, the kernal boot routine is not executed first. We say "first" because the test to see if the RUN/STOP key is pressed is performed before the kernal boot routine is executed and the test routine then jumps to the monitor in the form of a JSR command. When you exit the monitor with the X command, then the computer resumes operation with the normal boot routine and general initialization, provided you have not changed the return address on the stack.

These methods are of interest both to the assembly-language programmer and to the user who wants to use his old C-64 programs without having to go through the boot routine.

CHAPTER 9

Chapter 9: The Hardware

Imagine the following tricky situation in which the developers are asked to construct a computer that, on the one hand, is completely compatible with the existing C-64, and on the other hand, is to be outfitted with completely new, state-of-the-art features.

This task is difficult enough, but as icing on the cake, a Z-80 microprocessor should be added to the whole thing in such a way that it can peacefully coexist in the same system with the other processor.

You can imagine the difficulties involved. But this idea is not entirely new. Some of you no doubt remember the infamous CP/M cartridge for the Commodore 64, which was supposed to allow it to use the CP/M operating system. This expansion contained a Z-80 processor in addition to a few control components.

So you take a C-64, a CP/M cartridge, an additional 64K of memory, a new operating sytem and BASIC, mix them all together and let it all simmer for a few months.

We have no doubt that the first C-128 prototype used a C-64 as a basis. The tough part must have been in trying to put the whole thing together on a single board.

Fortunately, Commodore has its own semiconductor manufacturing company (MOS). It was clear that there was no way the necessary control components for the coordination of the processors and switching both memory banks and operating systems would fit onto a reasonably-sized PC board using current TTL technology. MOS had to design a special large-scale intergrated circuit, the MMU 8722, to handle all of the management logic.

This undertaking proceeded very well when the VIC chip, taken from the C-64 (but now known as the 8564), and the 6510 processor (now the 8502) were submitted to redesign. A new video controller (8563), which can display 80 columns in color, was added. What good is such a capable computer, which can run CP/M, if it is limited to 40 columns per line?

The address manager of the C-64 was refurbished to become the 8721. It has 23 (significant) input lines and 16 outputs. We will discuss the details of this and other devices shortly.

The question will no doubt arise, as to why we have not included a complete circuit diagram in this book. The diagram takes up 4 sheets of normal-sized paper; each so full of components, that reduction would be out of the question. We therefore decided to divide the circuit into blocks, into bite-sized and (hopefully) clear function groups within the text.

In general the diagrams are designed so that the signal flow is from left to right. At the left you find all input lines and at the right all of the output lines. The I/O block is an exception to this. Here it was not possible to retain this principle because space was too scarce and the interfaces are not clearly assigned as input or output.

Signal names prefixed by a minus sign are active low, meaning that they are "true" (or active) when the logical signal =0. Bus lines are emphasized in the diagrams. Everything having something to do with the address bus is dotted, the data bus has diagonal stripes, and all of the remaining bus-like structures are checkered.

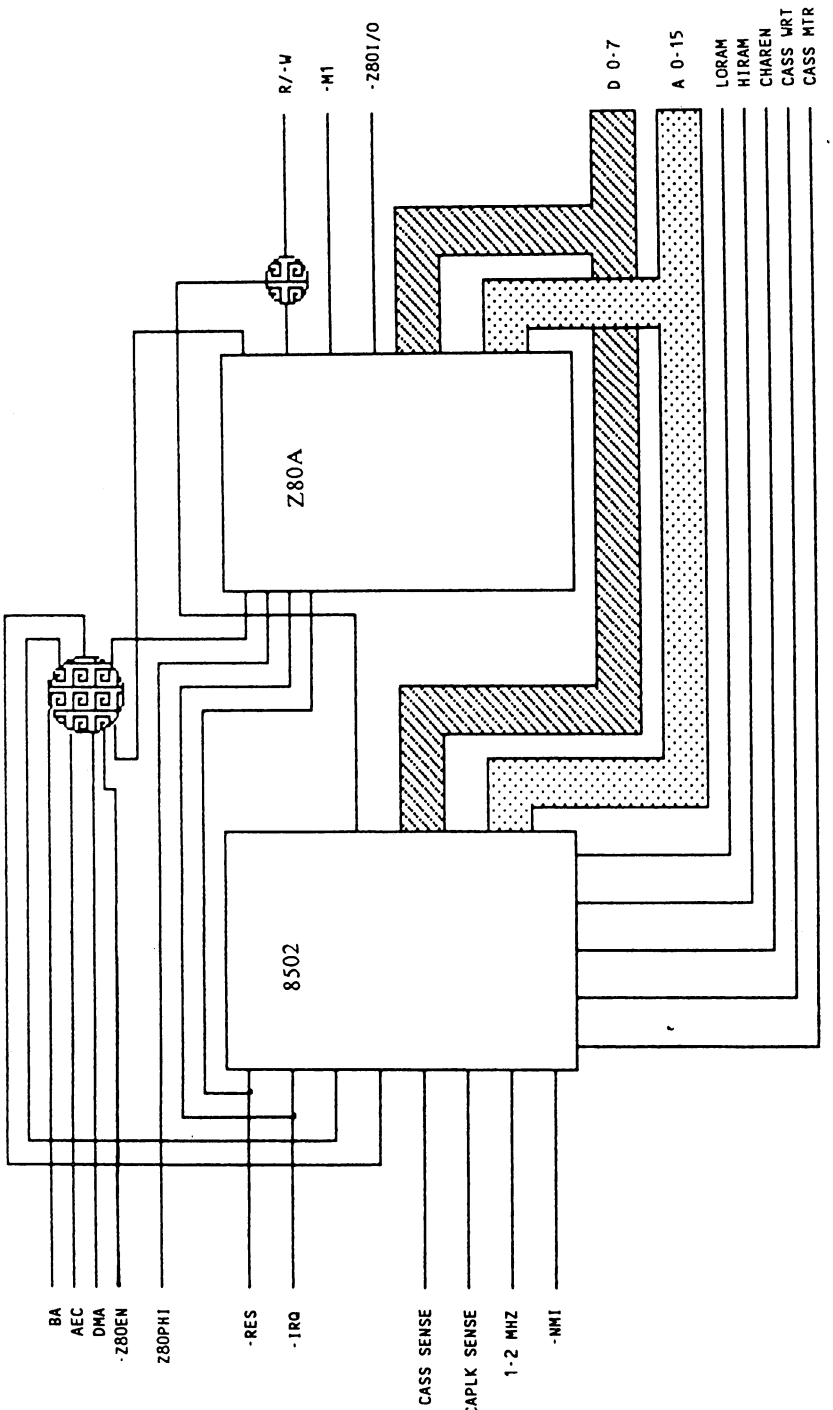
The components filled with the rings are intended to indicate that something special happens to the input signals which cannot be represented individually. In general, there is not a single IC behind it, but a whole system of them.

The CPU

Naturally there isn't just one processor, but two. The block diagram on the next page should, despite its simplicity, convince you that a good deal of switching effort is required to allow two microprocessors to use the same system components, even if not at the same time.

It is clear that only one processor can be running at any given time. The other must wait during this period. The trick is to get the processor in question to actually stop (that is, to interrupt it in such a manner that it can resume its work at a later time without any problems) and not crash when the system bus is blocked off. This can be done in various ways:

The bus must be given up (and this applies to both processors) when the 40-column video controller has to access the RAM, in order to refresh the screen. The lines BA (Bus Available) and AEC (Address Enable Control), both of which come from the VIC chip, are used to signal this condition to the switching logic.


R/-W
-M1
-Z80I/O
D 0-7
A 0-15
LORAM
HIRAM
CHAREN
CASS WRT
CASS MTR
Z80A
8502
BA
AEC
DMA
-Z80EN
Z80PHI
-RES
-IRQ
CASS SENSE
CAPLK SENSE
1-2 MHZ
-NMI

The second possibility is offered by the DMA line (Direct Memory Access), which comes from the expansion slot. Here too, both processors must relinquish the bus because the system bus is controlled from the outside in these cases, by a RAM expansion or other add-on hardware.

The programmer (or CP/M) is responsible for the third variant. Here the two processors can be selected with the -Z80EN line. This signal comes from the MMU.

The meaning of additional input lines:

Z80PHI is the system clock created by the VIC for the Z-80.

-RES resets the processors, which causes the Z-80 to start execution at address 0, and the 8502 to start at the reset vector in the ROM.

-IRQ is the interrupt line connected to both processors by means of which devices like the CIAs can signal the occurrence of an event.

-NMI is also an interrupt, but only for the 8502. This signal is derived from the RESTORE key.

CASS SENSE comes from the Datasette and indicates that the PLAY button is pressed.

CAPLK SENSE indicates the status of the SHIFT LOCK key .

1-2MHZ is the system clock for the 8502 and is provided by the VIC. This line supplies a clock signal of 1 or 2 MHz, depending on bit 0 in register 48 of the VIC.

The output lines:

R/-W tells connected peripheral components whether data is to taken from the bus or whether they are to supply the bus with data on their part.

-M1 is a Z-80 specific signal and means that the processor is currently fetching a command byte (in contrast to an operand) from the data bus. This line is used to prevent access to peripheral ICs during M1 (which could otherwise happen because the I/O addresses in the hardware are "normal" memory addresses).

-Z80I/O signals an I/O access of the Z-80 by means of the command IN or OUT. The lock-out mentioned above is removed by this signal.

D0-7 comprise the data bus.

A0-15 make up the address bus.

LORAM puts RAM in place of the BASIC ROM in the C-64 mode.

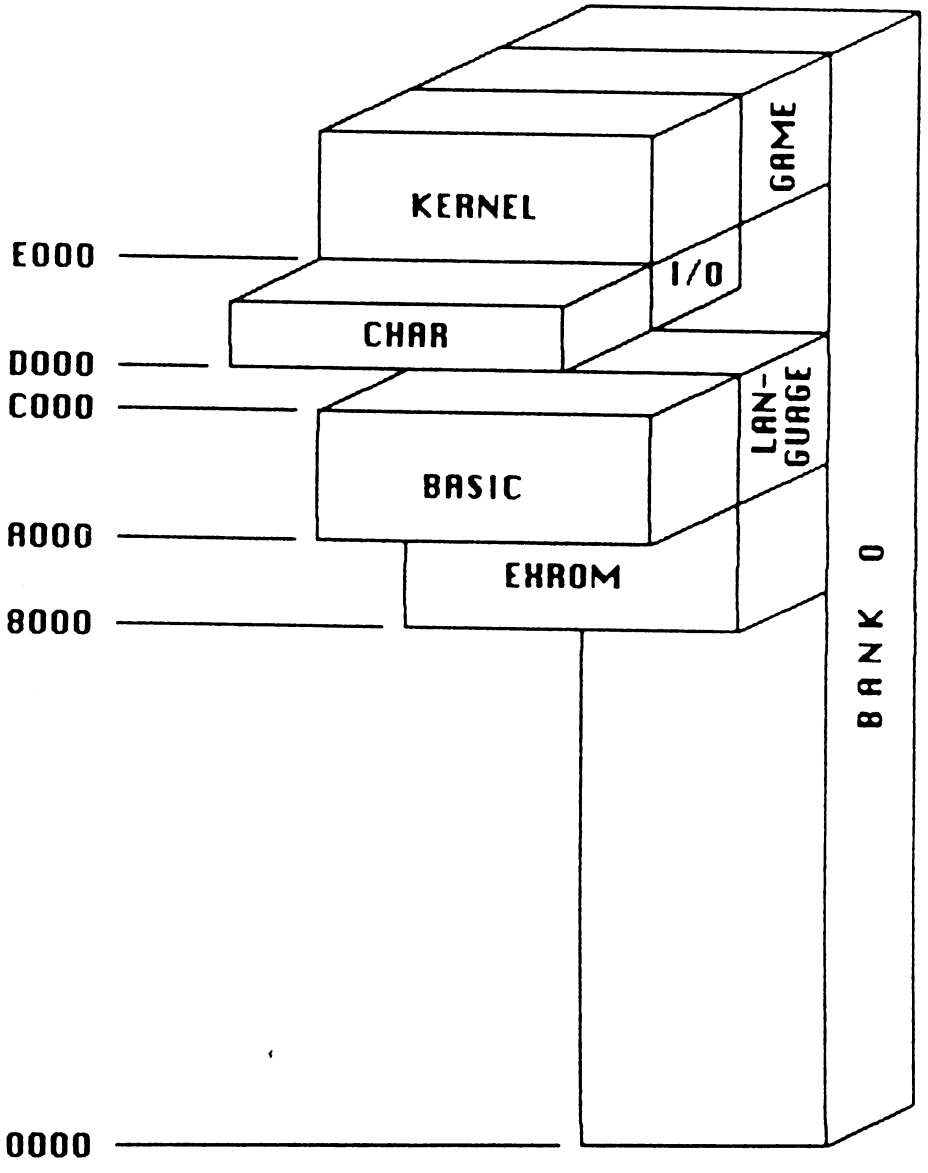
HIRAM is like LORAM, except the kernal ROM is replaced by RAM.
CHAREN makes it possible to read the character generator. Normally the color memory and I/O lie in this range.
CASS WRT is the write line for the Datasette.
CASS MTR controls the Datasette motor.

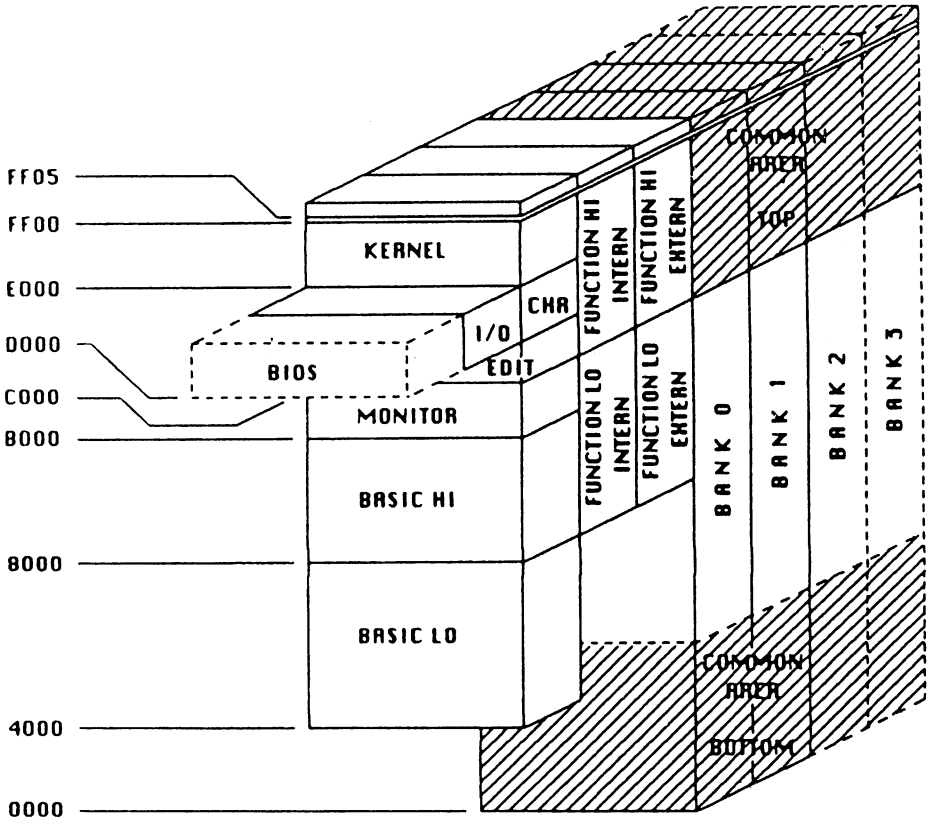
The address logic

In order to give you an idea of the complexity of this function block, we have illustrated the complete memory layouts for the C-64 and 128 modes on the following two pages. Try to imagine the memory in layers. One layer applies as the user surface, in which changing combinations are possible. The MMU is responsible for the global division (operating mode, bank selection, processor). Depending on this, the AM brings the desired portions to the surface, that is, it generates the necessary selection signals.

In connection with this we would like to list the pin layouts of these two ICs, as well as a description of each pin:

2 -RES
3-10 TA8-15. This is the translated address bus. The address A8-15 are "translated" depending on the configuration. For example, the address $0000 must be converted to $D000 during Z-80 operation, because part of the BIOS is located here and after reset the Z-80 starts execution at address 0.
11+12 -CAS0 and -CAS1. These two signals are responsible for the selection of the RAM bank, depending on register 1.
13-15 I/OSEL, ROMBANKHI, and ROMBANKLO. These signals are used to control the AM and result from the combination of bits 0-5 of register 1.
16 GAEC results from the combination of DMA and AEC and permits the MMU to take the lines LA8-15 from the bus.
17 MUX is created by the VIC. The MMU uses this to activate the signals -CAS0 and -CAS1.
18-31 A0-15, whereby the lines A6/7 and A4/5 are externally combined into one signal. The MMU also decodes its selection signal from the address bus.
35-42 D0-7
43 -Z80EN reflects bit 0 of register 5 and is responsible for selecting the processor.


KERNEL
GAME
I/O
CHAR
LAN-
GUAGE
BASIC
EXROM
BANK 0
E000
D000
C000
A000
8000
0000

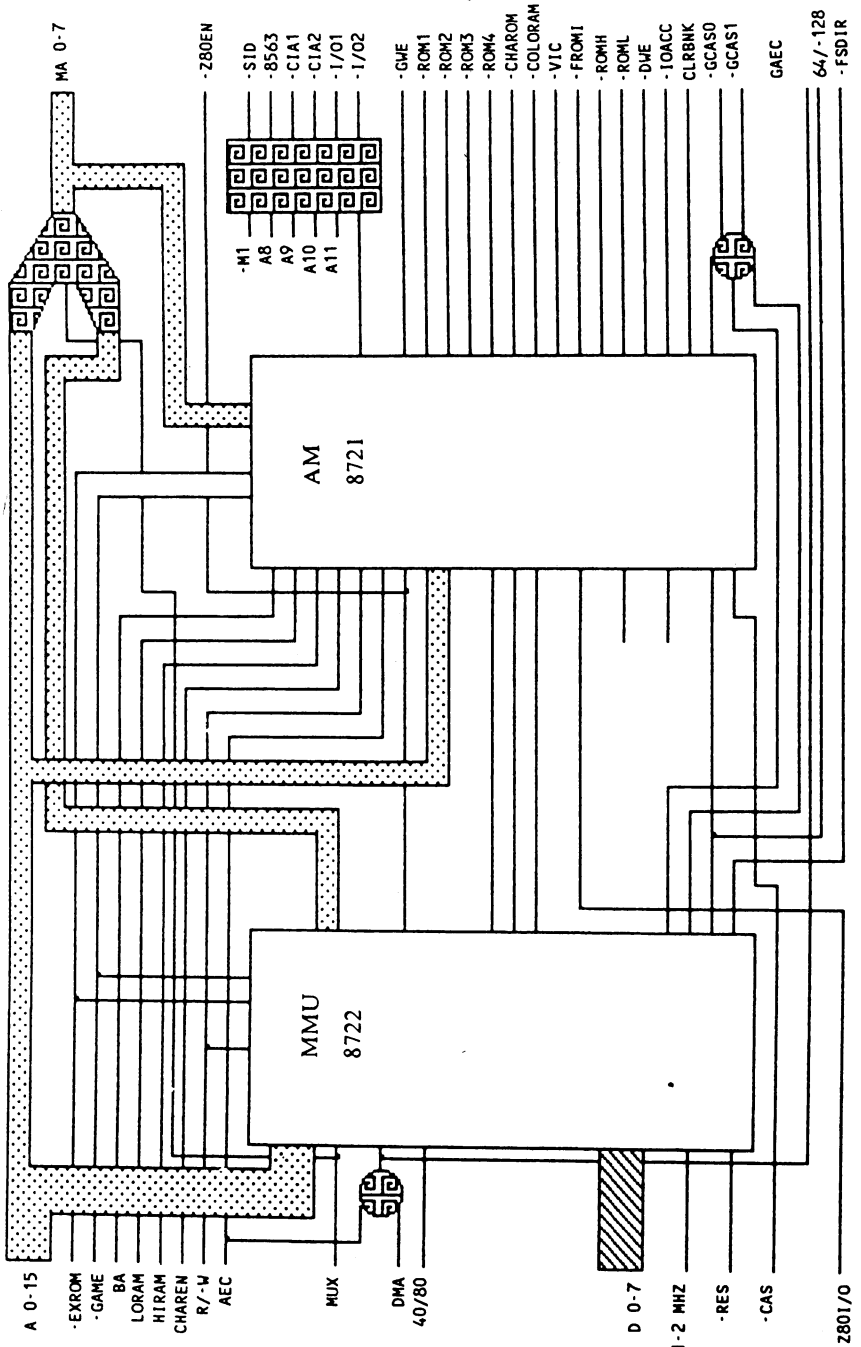

FF05
FF00
E000
D000
C000
8000
8000
4000
0000
KERNEL
I/O
CHR
EDIT
BIOS
MONITOR
BASIC HI
BASIC LO
FUNCTION HI EXTERN
FUNCTION HI INTERN
FUNCTION LO EXTERN
FUNCTION LO INTERN
BANK 0
BANK 1
BANK 2
BANK 3
COMMON AREA TOP
COMMON AREA BOTTOM

44 -FSDIR corresponds to bit 3 of register 5. Here the data direction of the serial bus -SRQIN, which is responsible for the data clock for the high-speed transfer using the 1571 disk drive, is switched.
45-46 -GAME and -EXROM come from the expansion slot and can be read as bits 4 and 5 of register 5.
47 64/-128 is a control line for the AM and corresponds to bit 6 of register 5.
48 40/80 comes from the keyboard and can be read as bit 7 of register 5.

With a few exceptions we will simply list the AM pins, since no set assignment can be given to many of them because of the complex internal combination possibilities. Imagine how many combinations have to be checked with 23 input bits (about eight million). Naturally, not that many are used, since the IC has only 16 output lines, of which a maximum of only four can reasonably be active at any given time.

You can easily recognize the function of the outputs from the names and the block diagrams, since they are really only chip select lines.

1-6 A15-10
7 VICFIX. This input is tied to ground on the board via jumper J2. The significance of this line is not clear.
9 AEC
10 R/-W. This signal is evaluated such that, at least in the C64 mode (nothing is known about the C128 mode), write access to the ROM address area always write to the "hidden" RAM, even if it is not explicitly selected.
11-12 -GAME and -EXROM exchange the BASIC and/or kernal ROM for the software found in a cartridge in the C64 mode, as is the case for games, for example.
13 -Z80EN
14 -Z80I/O
15 64/-128. Here it is decided which kernal or BASIC is active.
16 I/OSEL
17-18 ROMBANKHI and ROMBANKLO
19-20 MA4-5
21 BA
22-23 LORAM and HIRAM
25 CHAREN
26 -VA14


MA 0-7
-Z80EN
-SID
-8563
-C1A1
-C1A2
-I/O1
-I/O2
-M1
A8
A9
A10
A11
-GWE
-ROM1
-ROM2
-ROM3
-ROM4
-CHAROM
-COLORAM
-VIC
-FROMI
-ROMH
-ROML
-DWE
-IOACC
CLRBNK
-GCAS0
-GCAS1
GAEC
64/-128
-FSDIR
AM
8721
MMU
8722
A 0-15
-EXROM
-GAME
BA
LORAM
HIRAM
CHAREN
R/-W
AEC
MUX
DMA
40/80
D 0-7
1-2 MHZ
-RES
-CAS
-Z80I/O

27 128/-256. This signal indicates what type of ROM is in the sockets ROM1 and ROM3. It is possible to have two 16K ROMs in the sockets ROM1/4 and ROM2/3 form one 32K ROM each. It is possible to specify one or the other possibility during production. In the second case, the free sockets ROM2 and ROM4 are not free for other purposes!

30-31 -ROML and ROMHI go to the expansion slot.

32 CLRBANK switches between two possible banks of the color RAM. The dependency of this signal is not yet known. -VA14 may play a role here.

33 -FROMI (Funtion ROM Internal)

34-37 -ROM4 to -ROM1

38 -IOCS is the general selection for all peripheral ICs. The sole exception is the MMU, which decodes itself.

40 -DWE is the write signal for the RAM banks.

41 -CASENB is the address strobe for the RAM banks (simultaneous selection signal).

42 -VIC

43 -IOACC signals to the VIC an access to a peripheral IC, which brings the system clock down to 1 MHz if it was previously at 2MHz. This is necessary because the peripheral components can be supplied only with 1MHz, so this signal synchronizes them to the 8502.

44 -GWE is the write signal for the color RAM.

45 -COLORAM

46 -CHAROM

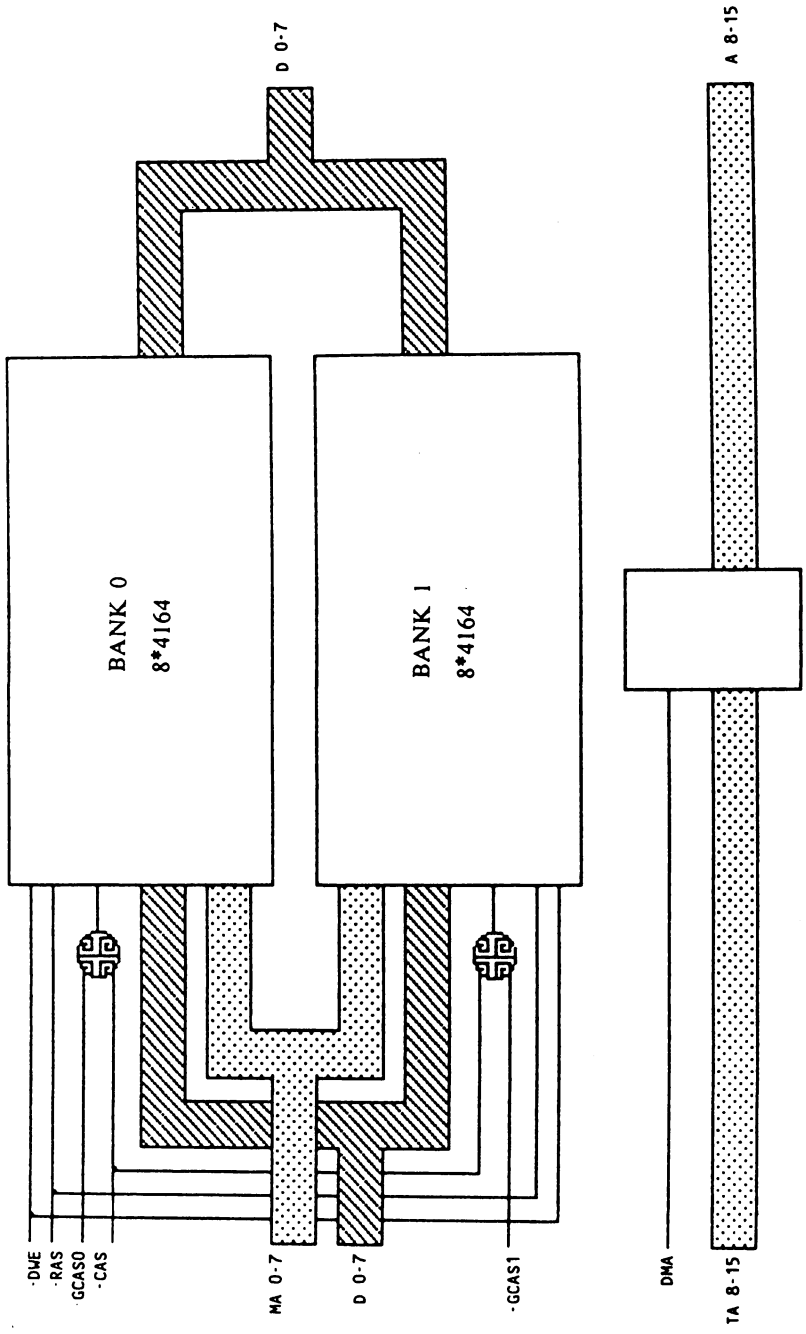
47 -CAS is actually responsible for the creation of -CASENB.

The greatest portion of the address logic is described in the pin descriptions. Worthy of mention is the funnel-shaped symbol in the upper right-hand corner of the diagram. This is the address multiplexer. As you may already know, not all of the address lines are applied to the dynamic RAMs at once, but one half at a time to the same lines. For this reason, the two halves of the address bus must be brought together. The decision as to which half is applied to the lines is taken care of by the MUX signal. The RAM chip recognizes the bottom half on -RAS and the top half on -CAS.

The RAM

The RAM consists of two banks of 64K each. No more are possible! Banks 2 and 3 indicated in the memory map are to be understood as external expansion and are accessed in a different manner.

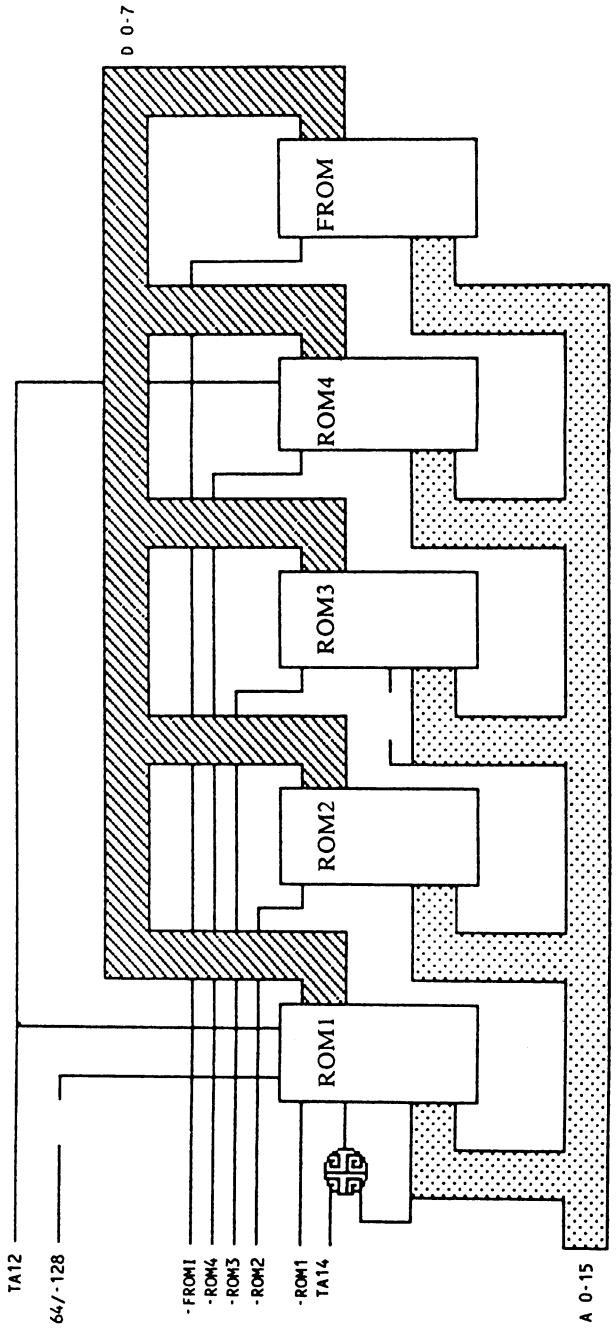
One interesting thing in the block diagram is the buffer at the bottom. In cases of memory access from the outside, this supplies the top half of the address bus.



The ROMs

The function block contains the combined "intelligence" of the computer. The selection signals for the individual ROMs come from the AM. Worthy of mention is the function of the 64/128 signal. If a 32K ROM is inserted in ROM1, this signal switches between the two halves. The lower half contains the kernal for the C-128 and the upper half contains the entire operating system software for the C-64. Jumper J6 must be connected on the PC board for this.

TA12 provides for the conversion of the area at $D000 to $0000 for the Z-80 operation.


D 0-7
FROM
ROM4
ROM3
ROM2
ROM1
TA12
64/-128
-FROM1
-ROM4
-ROM3
-ROM2
-ROM1
TA14
A 0-15

40 column

The jumble of bus lines in this section is because the VIC controls the system bus itself in order to get the information necessary to refresh the screen picture from the RAM. To do this, it must stop the currently active processor by means of the BA and AEC lines, so that no concurrent RAM accesses can occur. As much as possible, however, it chooses a time when the processors will not be disturbed. It uses the clock gaps during which the computer is not accessing the bus. An exception to this is when sprites are displayed. Here the VIC must get the entire point map, which in the case of a "normal" character it would get from the character generator, from the RAM, which naturally takes time.

So that the VIC "knows" when it may do something, it is in charge not only of the screen display, but of the clock generation for the whole computer. So at any time it is informed about the current state of the system. To display a character, it first gets the ASCII value of the character from the RAM, then the corresponding bit pattern from the character generator, and finally the color information from the color RAM.

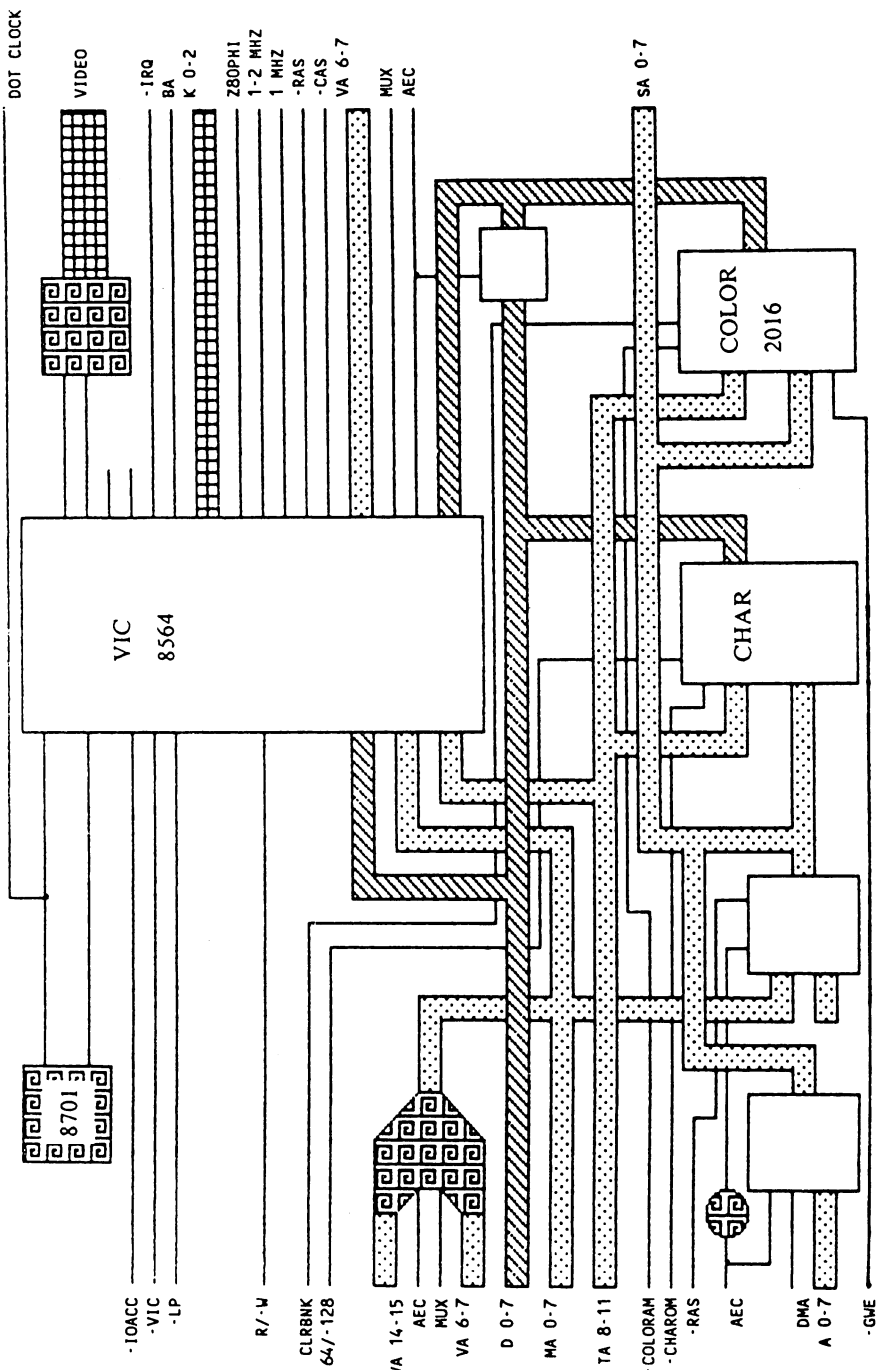
This last point, by the way, is a special case: The color RAM is connected directly to the VIC via its own four-bit wide data bus, so that the ASCII value and the color arrive simultaneously when the refresh address is given.

Another interesting feature is the composition of the RAM address. It is placed on the bus in two halves, whereby the base address of the video RAM is formed from bits VA6-7 of the VIC and bits VA14-15 of CIA2. The video RAM is therefore movable within a large area.

In the upper left-hand corner is the master clock. This is an oscillator running at 17.73447 MHz. This strange frequency was chosen for the color creation in the PAL standard. The VIC produces the various system clocks from this clock.

If by chance you are familiar with the VIC in the C-64, you may notice something unusual about this version of the VIC. A bus heads to the right with the designation K0-2. These lines are responsible for the column selection of the ten-key pad.

We should mention the little box in the lower left corner. This is a device similar to the buffer in the RAM block for the lower half of the address bus.


DOT CLOCK
VIDEO
-IRQ
BA
K 0-2
Z80PH1
1-2 MHZ
1 MHZ
-RAS
-CAS
VA 6-7
MUX
AEC
SA 0-7
VIC
8564
COLOR
2016
CHAR
8701
-IOACC
-VIC
-LP
R/-W
CLRBNK
64/-128
VA 14-15
AEC
MUX
VA 6-7
D 0-7
MA 0-7
TA 8-11
-COLORAM
-CHAROM
-RAS
AEC
DMA
A 0-7
-GWE

80 column

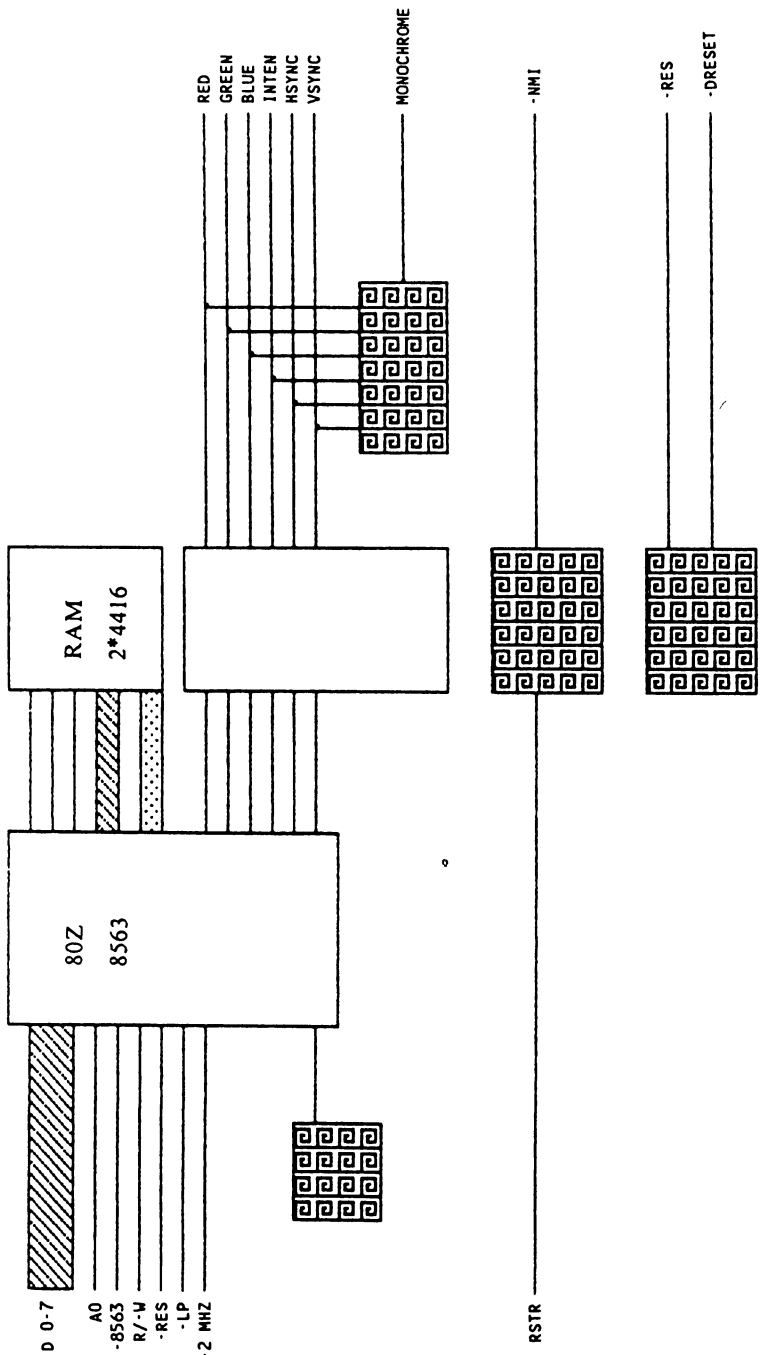
Although this section represents one of the most interesting features of the computer, it offers little in the way of circuit technology. The reason for this is the 80-column video controller which MOS (a subsidiary of Commodore) developed for this computer. This video controller contains all of the logic for accessing the video RAM, color RAM, and character generator, as well as the necessary circuitry for creating the screen picture.

The reason the function diagram is so uncluttered is that the interface to the system consists only of the data bus and one address line. But the main reason is that only a single memory component can be seen, namely a dynamic RAM with 16K. Actually there two IC's with 4 bits each. But there is nothing resembling a character generator or color RAM.

The trick lies in the fact that everything is done in the RAM, even the character generator. Since this normally consists of a ROM, because the bit patterns of the characters are normally unchangeable, the character generator from the 40-column section is copied into this RAM when the 80-column mode is switched on.

You may wonder how the data gets to the video RAM since it has no direct connection to the system. All communication with the video RAM is done through the controller. First, the controller loads the low order byte of the register that specifies the RAM address. Next the data is loaded. Then the desired address is passed, followed by the data. The controller ensures that the data end up in the right place. This isn't the fastest way of doing things, of course, but it works.

The little box to the left of the video controller is an oscillator that runs at 16 MHz. This is the normal frequency for the Dot Clock in 80-column monitors. The two boxes in the lower middle create a clean reset signal (from which -DRESET is derived. -DRESET is used to reset one of the built-in disk drivesand to form the -NMI pulse from the RESTORE key.


RED
GREEN
BLUE
INTEN
HSYNC
VSYNC
MONOCHROME
-NMI
-RES
-DRESET
RAM
2*4416
80Z
8563
D 0-7
A0
-8563
R/-W
-RES
-LP
1-2 MHZ
RSTR

Input/Output

This section looks quite chaotic largely because all of the connections to the outside world run through it.

Let's start at the top left. There we find the two joystick ports. Their digital components, the stick movement and fire button, are wired in parallel to the keyboard matrix. This is why characters appear on the screen when the stick is moved. The analog components (such as those for the paddles) are multiplexed by an analog switch because the SID has only two analog inputs, but two pairs must be read.

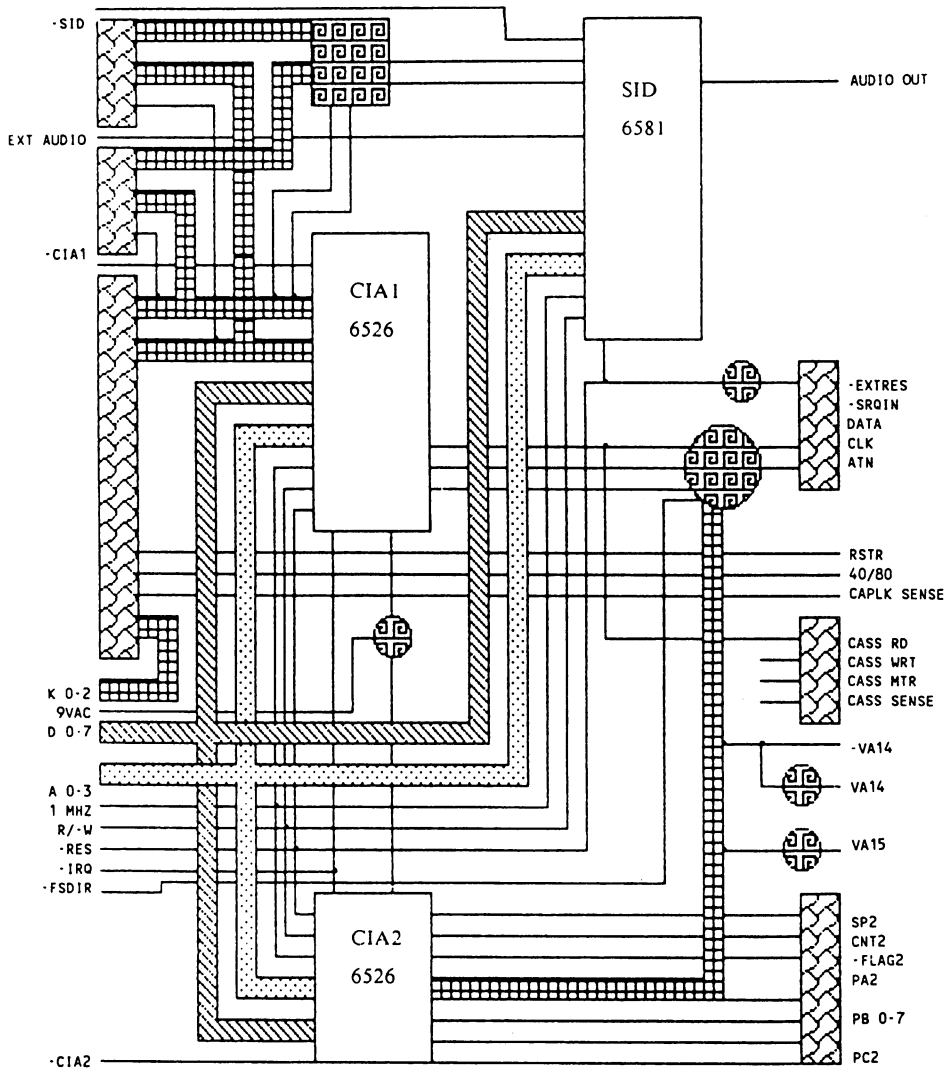
Below the analog switch is the CIA1. This has by far the largest share of work to do. It is responsible for reading the keyboard, as well as the serial bus. Here Commodore makes an improvement over the C-64. Instead of constructing the data bytes from the disk drive one bit at a time, this task is automatically assumed by the CIA. An entire byte is simply loaded into the shift register and the CIA shifts it out to SP automatically. It works the same way in the opposite direction. The bit speed is dependent on the clock at SP.

Here the -FSDIR signal from the MMU takes on significance. It is, as already mentioned, a direction switch for this same clock which is always supplied by the sending device, sometimes the computer and sometimes the disk drive. This clock is sent over the -SRQIN line since this line is not used by devices that cannot use the fast serial mode.

A good half of CIA2 is dedicated to the user port, but part of it is also used for the serial bus. Bits VA14-15 are also created for switching the video RAM.

On the left side is a signal name which you probably cannot place (normally the signal names are self-evident). This is 9VAC. This is nothing other than 9-Volt alternating current from the power supply. What purpose does this voltage serve on the board? Quite simple: This signal is rectified and limited and used as the clock for the real-time clocks in the CIAs.

This is the end of our little excursion into the hardware. We hope that you have gained at least some insight into the operation of the computer.


-SID
EXT AUDIO
-CIA1
SID
6581
AUDIO OUT
CIA1
6526
-EXTRES
-SRQIN
DATA
CLK
ATN
RSTR
40/80
CAPLK SENSE
CASS RD
CASS WRT
CASS MTR
CASS SENSE
K 0-2
9VAC
D 0-7
-VA14
VA14
VA15
A 0-3
1 MHZ
R/-W
-RES
-IRQ
-FSDIR
CIA2
6526
SP2
CNT2
-FLAG2
PA2
PB 0-7
PC2
-CIA2

CHAPTER 10

Chapter 10: Decimal-Hexadecimal-Binary Conversion Table

Dec.	Hex	Binary	Dec.	Hex	Binary
#000	$00	%00000000	#001	$01	%00000001
#002	$02	%00000010	#003	$03	%00000011
#004	$04	%00000100	#005	$05	%00000101
#006	$06	%00000110	#007	$07	%00000111
#008	$08	%00001000	#009	$09	%00001001
#010	$0A	%00001010	#011	$0B	%00001011
#012	$0C	%00001100	#013	$0D	%00001101
#014	$0E	%00001110	#015	$0F	%00001111
#016	$10	%00010000	#017	$11	%00010001
#018	$12	%00010010	#019	$13	%00010011
#020	$14	%00010100	#021	$15	%00010101
#022	$16	%00010110	#023	$17	%00010111
#024	$18	%00011000	#025	$19	%00011001
#026	$1A	%00011010	#027	$1B	%00011011
#028	$1C	%00011100	#029	$1D	%00011101
#030	$1E	%00011110	#031	$1F	%00011111
#032	$20	%00100000	#033	$21	%00100001
#034	$22	%00100010	#035	$23	%00100011
#036	$24	%00100100	#037	$25	%00100101
#038	$26	%00100110	#039	$27	%00100111
#040	$28	%00101000	#041	$29	%00101001
#042	$2A	%00101010	#043	$2B	%00101011
#044	$2C	%00101100	#045	$2D	%00101101
#046	$2E	%00101110	#047	$2F	%00101111
#048	$30	%00110000	#049	$31	%00110001
#050	$32	%00110010	#051	$33	%00110011
#052	$34	%00110100	#053	$35	%00110101
#054	$36	%00110110	#055	$37	%00110111
#056	$38	%00111000	#057	$39	%00111001
#058	$3A	%00111010	#059	$3B	%00111011
#060	$3C	%00111100	#061	$3D	%00111101
#062	$3E	%00111110	#063	$3F	%00111111
#064	$40	%01000000	#065	$41	%01000001
#066	$42	%01000010	#067	$43	%01000011
#068	$44	%01000100	#069	$45	%01000101
#070	$46	%01000110	#071	$47	%01000111
#072	$48	%01001000	#073	$49	%01001001
#074	$4A	%01001010	#075	$4B	%01001011
#076	$4C	%01001100	#077	$4D	%01001101

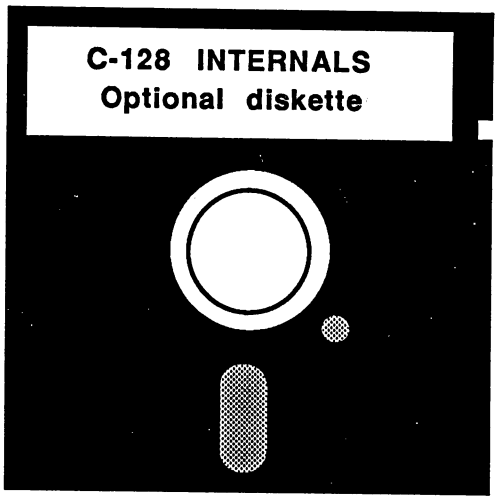
Dec.	Hex	Binary	Dec.	Hex	Binary
#078	$4E	%01001110	#079	$4F	%01001111
#080	$50	%01010000	#081	$51	%01010001
#082	$52	%01010010	#083	$53	%01010011
#084	$54	%01010100	#085	$55	%01010101
#086	$56	%01010110	#087	$57	%01010111
#088	$58	%01011000	#089	$59	%01011001
#090	$5A	%01011010	#091	$5B	%01011011
#092	$5C	%01011100	#093	$5D	%01011101
#094	$5E	%01011110	#095	$5F	%01011111
#096	$60	%01100000	#097	$61	%01100001
#098	$62	%01100010	#099	$63	%01100011
#100	$64	%01100100	#101	$65	%01100101
#102	$66	%01100110	#103	$67	%01100111
#104	$68	%01101000	#105	$69	%01101001
#106	$6A	%01101010	#107	$6B	%01101011
#108	$6C	%01101100	#109	$6D	%01101101
#110	$6E	%01101110	#111	$6F	%01101111
#112	$70	%01110000	#113	$71	%01110001
#114	$72	%01110010	#115	$73	%01110011
#116	$74	%01110100	#117	$75	%01110101
#118	$76	%01110110	#119	$77	%01110111
#120	$78	%01111000	#121	$79	%01111001
#122	$7A	%01111010	#123	$7B	%01111011
#124	$7C	%01111100	#125	$7D	%01111101
#126	$7E	%01111110	#127	$7F	%01111111
#128	$80	%10000000	#129	$81	%10000001
#130	$82	%10000010	#131	$83	%10000011
#132	$84	%10000100	#133	$85	%10000101
#134	$86	%10000110	#135	$87	%10000111
#136	$88	%10001000	#137	$89	%10001001
#138	$8A	%10001010	#139	$8B	%10001011
#140	$8C	%10001100	#141	$8D	%10001101
#142	$8E	%10001110	#143	$8F	%10001111
#144	$90	%10010000	#145	$91	%10010001
#146	$92	%10010010	#147	$93	%10010011
#148	$94	%10010100	#149	$95	%10010101
#150	$96	%10010110	#151	$97	%10010111
#152	$98	%10011000	#153	$99	%10011001
#154	$9A	%10011010	#155	$9B	%10011011
#156	$9C	%10011100	#157	$9D	%10011101
#158	$9E	%10011110	#159	$9F	%10011111

Dec.	Hex	Binary	Dec.	Hex	Binary
#160	$A0	%10100000	#161	$A1	%10100001
#162	$A2	%10100010	#163	$A3	%10100011
#164	$A4	%10100100	#165	$A5	%10100101
#166	$A6	%10100110	#167	$A7	%10100111
#168	$A8	%10101000	#169	$A9	%10101001
#170	$AA	%10101010	#171	$AB	%10101011
#172	$AC	%10101100	#173	$AD	%10101101
#174	$AE	%10101110	#175	$AF	%10101111
#176	$B0	%10110000	#177	$B1	%10110001
#178	$B2	%10110010	#179	$B3	%10110011
#180	$B4	%10110100	#181	$B5	%10110101
#182	$B6	%10110110	#183	$B7	%10110111
#184	$B8	%10111000	#185	$B9	%10111001
#186	$BA	%10111010	#187	$BB	%10111011
#188	$BC	%10111100	#189	$BD	%10111101
#190	$BE	%10111110	#191	$BF	%10111111
#192	$C0	%11000000	#193	$C1	%11000001
#194	$C2	%11000010	#195	$C3	%11000011
#196	$C4	%11000100	#197	$C5	%11000101
#198	$C6	%11000110	#199	$C7	%11000111
#200	$C8	%11001000	#201	$C9	%11001001
#202	$CA	%11001010	#203	$CB	%11001011
#204	$CC	%11001100	#205	$CD	%11001101
#206	$CE	%11001110	#207	$CF	%11001111
#208	$D0	%11010000	#209	$D1	%11010001
#210	$D2	%11010010	#211	$D3	%11010011
#212	$D4	%11010100	#213	$D5	%11010101
#214	$D6	%11010110	#215	$D7	%11010111
#216	$D8	%11011000	#217	$D9	%11011001
#218	$DA	%11011010	#219	$DB	%11011011
#220	$DC	%11011100	#221	$DD	%11011101
#222	$DE	%11011110	#223	$DF	%11011111
#224	$E0	%11100000	#225	$E1	%11100001
#226	$E2	%11100010	#227	$E3	%11100011
#228	$E4	%11100100	#229	$E5	%11100101
#230	$E6	%11100110	#231	$E7	%11100111
#232	$E8	%11101000	#233	$E9	%11101001
#234	$EA	%11101010	#235	$EB	%11101011
#236	$EC	%11101100	#237	$ED	%11101101
#238	$EE	%11101110	#239	$EF	%11101111
#240	$F0	%11110000	#241	$F1	%11110001

Dec.	Hex	Binary	Dec.	Hex	Binary
#242	$F2	%11110010	#243	$F3	%11110011
#244	$F4	%11110100	#245	$F5	%11110101
#246	$F6	%11110110	#247	$F7	%11110111
#248	$F8	%11111000	#249	$F9	%11111001
#250	$FA	%11111010	#251	$FB	%11111011
#252	$FC	%11111100	#253	$FD	%11111101
#254	$FE	%11111110	#255	$FF	%11111111

INDEX

Optional Diskette



For your convenience, the program listings contained in this book are available on a 1541 formatted floppy disk.

You should order the diskette, if you want to use the programs, without typing them in from the listings in the book.

All programs on the diskette have been fully tested. You can change the programs for your particular needs. The diskette is available for $14.95 + $2.00 ($5.00 foreign) for postage and handling.

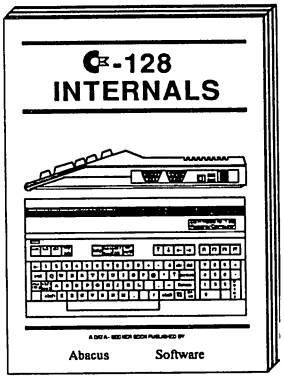
When ordering, please give your name and shipping address. Enclose a check, money order or credit card information. Mail your order to:

Abacus Software
P.O. Box 7211
Grand Rapids, MI 49510

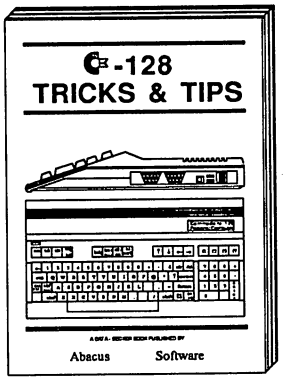
Or for fast phone service, call **616/241-5510.**

AUTHORATATIVE, COMPREHENSIVE, DEFINITIVE

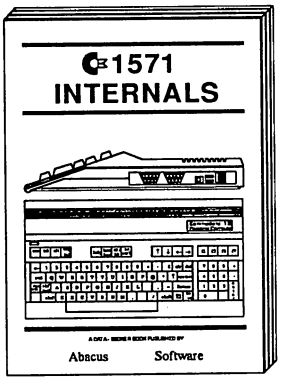
booksbooksbooks



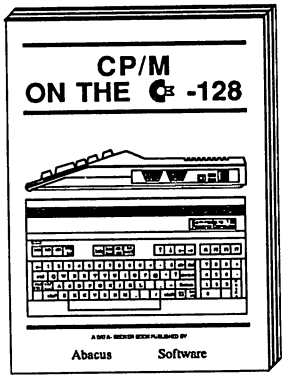
With ROM listings
Avail. Nov. $19.95



For the programmer
Avail. Nov. $19.95

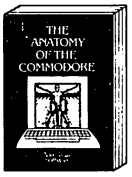


With ROM listings
Avail. Dec. $19.95

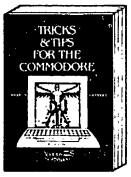


Especially for the '128
Avail. Dec. $19.95

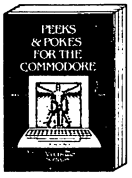
...and more books.



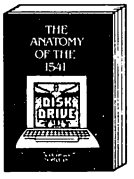
For two years a best seller. C-64 internals w/ROM listings. $19.95



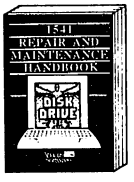
Favorite among programmers. 75,000+ sold worldwide. $19.95



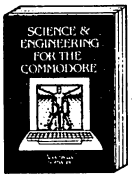
Quickhitting, easy-to-use routines for every C-64 owner. $14.95



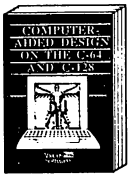
All time best seller. Revised & expanded. ROM listings. $19.95



Brand new! Complete maintenance and repair procedures. $19.95

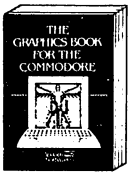


Intro to computers and world of science. Real examples. $19.95



CAD techniques using C-128/C-64. Many program examples. $19.95

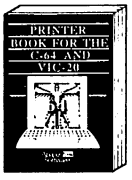
Learn to design and write your own compiler. With examples. $19.95



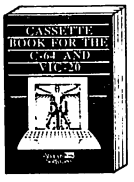
Most in depth treatment available. Dozens of techniques. $19.95

Intro to machine language geared to C-64. Assembler incl. $14.95

Techniques never covered before interrupts, controllers, etc. $14.95

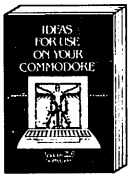


All about using printers and '64. Graphics, text, interfaces. $19.95



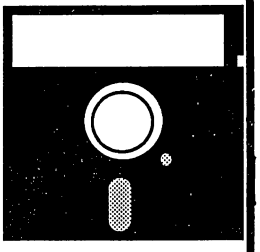
A must for cassette owner. Hi speed cassette system. $19.95

Write your own adventures. Learn strategy, motivation. $14.95



Dozens of interesting pro-ects for your '64. Easy to read. $12.95

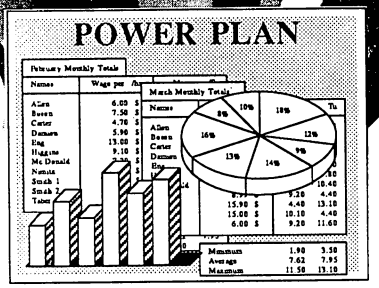
OPTIONAL DISKETTES are also available for each of our book titles. Each diskette contains the programs found in the book to save you the time of typing them in at the keyboard. Price of each diskette is $14.95.



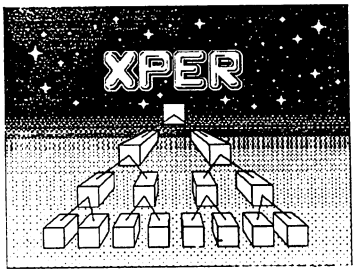
Call **now**, for the name of your nearest dealer. Or order directly from ABACUS with your MC, VISA or AMEX card. Add $4.00 for postage and handling. Foreign orders add $6.00 per book. Other software and books are also available. Call or write for **free** catalog. Dealer inquiries welcome - over 1200 dealers nationwide. Call **616 / 241-5510**

Abacus Software

P.O. Box 7211 Grand Rapids, MI 49510
Phone 616/241-5510 Telex 709-101



For C-64/C-128 on diskette



For C-64/C-128 on diskette

POWER & KNOWLEDGE ...at your fingertips

PowerPlan - SUPER SPREADSHEET
Start with an easy to learn spreadsheet, convenient menus and 90+ help screens. Add fast, shortcut commands for the advanced user. Build in a full range of flexible features for use with complex worksheets. Combine it with graphics for 2D/3D charts and graphs so you can display your "what-if' data both visually and numerically. Finally price it low enough for everyone's budget. That's what we call *powerful* software. **$49.95**

XPER - KNOWLEDGE BASE SOFTWARE
Ordinary data bases are good at memorizing and playing back facts. But *expert systems* help you wade through hundreds of items to make important decisions. **XPER** has an easy-to-use entry editor to quickly build your knowledge base from raw information; a sophisticated inquirer to guide you through the complex decision-making criteria; complete data editing and reporting features for analyzing your data. **$59.95**

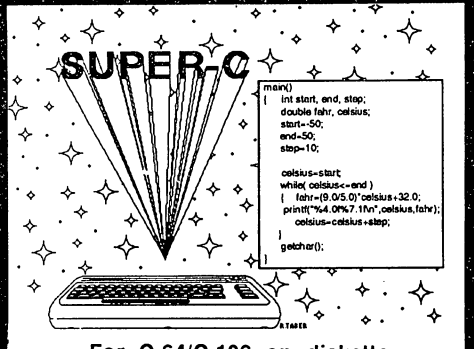
Call now for **free** software and book catalog and the name of your local dealer. If he is out of stock, have your dealer order our quality products for you. To order by credit card call 616/241-5510. We accept MC, VISA and AMEX. Add $4.00 postage and handling per order (foreign $8.00 per item). Michigan residents add 4% sales tax.

Abacus Software

P.O. Box 7211 Grand Rapids, MI 49510 For Fast Service Call (616) 241-5510

SOFTWARE As Real As Tomorrow

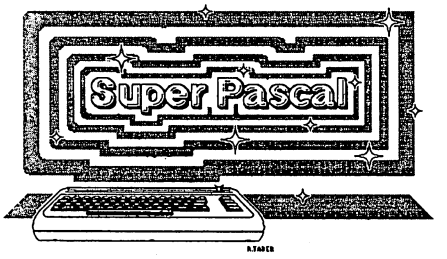
...available today!



For C-64/C-128 on diskette

Super C - Most advanced C package for the C-64/C-128. Since **Super C** supports the full K&R language (w/o bit fields), programs are transportable to other computers. It's a perfect learning tool for schools and industry. **Super C** package includes a complete source editor with 80 column display using horizontal scrolling, search/replace, 41K source files. Linker binds up to 7 separate modules. I/O library supports standard functions like printf and fprintf. Includes runtime package. **$79.95**

Compiler and Software Development System



For C-64/C-128 on diskette

Super Pascal - Not just a compiler, but a complete development system. It rivals even Turbo Pascal© in features. **Super Pascal** includes an advanced source file editor; a full Jensen & Wirth compiler; system programming extensions, a builtin assembler for specialized requirements, and a new high speed DOS which is 3X faster than standard 1541. Produces fast machine code. Supports program overlays, high precision 11 digit arithmetic, debugging tools, graphic routines and more. **$59.95**

Call now for our **free** software and book catalog and the name of your nearest dealer. If he's out of stock, have him order our products for you. Credit card orders call **616/241-5510**. Add $4.00 shipping and handling per order (foreign add $8.00/item).

Abacus Software

P.O. Box 7211 Grand Rapids, MI 49510
For fast service call 616/241-5510

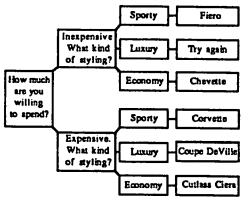
SUPER PRODUCTIVITY

POWER PLAN

Names	Wage per Hr	Mo	Tu
Andew	6.00 $	7.50	8.10
Busen	7.50 $	7.70	3.50
Carter	4.70 $	6.80	3.50
Damien	5.90 $	1.90	10.60
Gerrd	13.00 $	11.50	10.00
Higgins	9.10 $	6.50	7.80
Mc Donald	7.20 $	9.00	10.40
Nimitz	8.99 $	9.20	4.40
Smith 1	15.90 $	4.40	13.10
Smith 2	15.00 $	10.10	4.40
Wimpy	6.00 $	9.20	11.60
Minimum		1.90	3.50
Average		7.62	7.95
Maximum		11.50	13.10

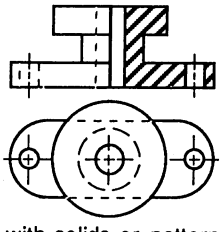
Powerful spreadsheet *plus* builtin graphics - display your important data visually as well as numerically. You'll learn fast with the 90+ HELP screens. Advanced users can use the shortcut commands. For complex spreadsheets, you can use POWER PLAN's impressive features: cell formatting, text formatting, cell protection, windowing, math functions, row and column sort, more. Then quickly display your results in graphics format in a variety of 2D and 3D charts. Includes system diskette and user's handbook. **$49.95**

XPER - expert system



XPER is the first *expert system* - a new breed of intelligent software for the C-64 & C-128. While ordinary data base systems are good at reproducing facts, **XPER** can help you make decisions. Using its simple entry editor, you build the information into a *knowledge base.* XPER's very efficient searching techniques then guide you through even the most complex decision making criteria. Full reporting and data editing. Currently used by doctors, scientists and research professionals. **$59.95**

CADPAK Revised Version



CADPAK is a superb design and drawing tool. You can draw directly on the screen from keyboard or using optional lightpen. POINTs, LINEs, BOXes, CIRCLEs, and ELLIPSEs; fill with solids or patterns; free-hand DRAW; ZOOM-in for intricate design of small section. Mesuring and scaling aids. Exact positioning using our *AccuPoint* cursor positioning. Using the powerful **OBJECT EDITOR** you can define new fonts, furniture, circuitry, etc. Hardcopy to most printers. **$39.95**
McPen lightpen, optional **$49.95**

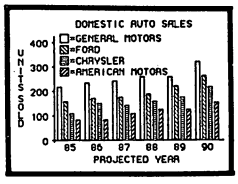
DATAMAT - data management

INVENTORY FILE			
Item Number	_____	Description	_____
Onhand	_____	Price	_____
Location	_____		
Reord. Pt.	_____	Reord. Qty.	_____
Cost	_____		

"Best data base manager under $50"
RUN Magazine

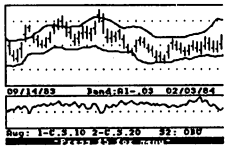
Easy-to-use, yet versatile and powerful features. Clear menus guide you from function to function. Free-form design of data base with up to 50 fields and 2000 records per diskette (space dependent). Simple data base design. Convenient and quick data entry. Full data editing capabilities. Complete reporting: sort on multiple fields and select records for printing in your specific format. **$39.95**

CHARTPAK



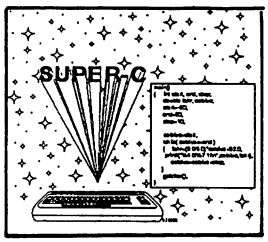
Make professional quality charts from your data in minutes. Quickly enter, edit, save and recall your data. Then interactively build pie, bar, line or scatter graph. You can specify scaling, labeling and positioning and watch **CHARTPAK** instantly draw the chart in any of 8 different formats. Change the format immediately and draw another chart. Incudes statistical routines for average, deviation, least squares and forecasting. Hardcopy to most printers. **$39.95**
CHARTPLOT-64 for 1520 plotter **$39.95**

TAS - technical analysis

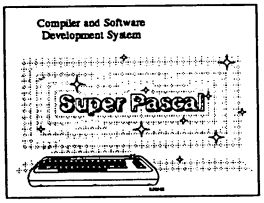


Technical analysis charting package to help the serious investor. Enter your data at keyboard or capture it through **DJN/RS** or Warner Services. Track high, low, close, volume, bid and ask. Place up to 300 periods of information for 10 different stocks on each data diskette. Build a variety of charts on the split screen combining information from 7 types of moving averages, 3 types of oscillators, trading bands, least squares, 5 different volume indicators, relative charts, much more. Hardcopy to most printers. **$59.95**

SUPER LANGUAGES

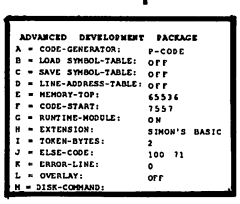


The most advanced **C** development package available for the C-64 or C-128 with very complete source editor; full K&R compiler (w/o bit fields); linker (binds up to 7 separate modules); and set of disk utilities. Very complete editor handles search/replace, 80 column display with horizontal scrolling and 41K source files. The I/O library supports standard functions like printf and fprintf. Free runtime package included. For C-64/C-128 with 1541/1571 drive. Includes system diskette and user's handbook. **$79.95**



Not just a compiler, but a complete development system. Rivals Turbo Pascal© in both speed and features. Produces fast 6510 machine code. Includes advanced source file editor; full Jensen & Wirth compiler with system programming extensions, new high speed DOS (3 times faster); builtin assembler for specialized requirements. Overlays, 11-digit arithmetic, debugging tools, graphics routines, much more. Free runtime package. Includes system diskette and complete user's handbook. **$59.95**

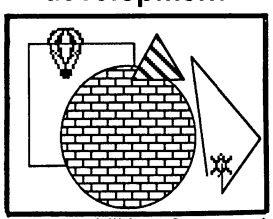
BASIC-64 full compiler



```
ADVANCED DEVELOPMENT PACKAGE
A = CODE-GENERATOR:        P-CODE
B = LOAD SYMBOL-TABLE:     OFF
C = SAVE SYMBOL-TABLE:     OFF
D = LINE-ADDRESS-TABLE:    OFF
E = MEMORY-TOP:            65536
F = CODE-START:            7557
G = RUNTIME-MODULE:        ON
H = EXTENSION:             SIMON'S BASIC
I = TOKEN-BYTES:           2
J = ELSE-CODE:             100 ?1
K = ERROR-LINE:            0
L = OVERLAY:               OFF
M = DISK-COMMAND:
```

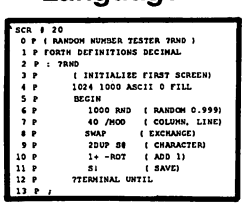
The most advanced **BASIC** compiler available for the C-64. Our bestselling software product. Compiles to superfast 6510 machine code or very compact speed-code. You can even mix the two in one program. Compiles the complete BASIC language. Flexible memory management and overlay options make it perfect for all program development needs. **BASIC 64** increases the speed of your programs from 3 to 20 times. Free runtime package. Includes system diskette and user's handbook. **$39.95**

VIDEO BASIC development



The most advanced graphics development package available for the C-64. Adds dozens of powerful commands to standard BASIC so that you can use the hidden graphics and sound capabilities. Commands for hires, multicolor, sprite and turtle graphics, simple and complex music and sound, hardcopy to most printers, memory management, more. Used by professional programmers for commerical software development. Free runtime package. Includes system diskette and user's handbook. **$39.95**

FORTH Language



```
SCR # 20
 0 P ( RANDOM NUMBER TESTER ?RND )
 1 P FORTH DEFINITIONS DECIMAL
 2 P : ?RND
 3 P       ( INITIALISE FIRST SCREEN)
 4 P      1024 1000 ASCII 0 FILL
 5 P      BEGIN
 6 P         1000 RND   ( RANDOM 0.999)
 7 P         40 /MOD    ( COLUMN, LINE)
 8 P        SWAP        ( EXCHANGE)
 9 P         2DUP 58    ( CHARACTER)
10 P         1+ -ROT    ( ADD 1)
11 P         SI         ( SAVE)
12 P      ?TERMINAL UNTIL
13 P ;
```

Our **FORTH** language is based on the Forth 79 standard, but also includes much of the 83 level to give you 3 times vocabulary of fig-Forth. Includes full-screen editor, complete Forth-style assembler, set of programming tools and numerous sample programs to get you deeply involved in the FORTH language. Our enhanced vocabulary supports both hires and lores graphics and the sound synthesizer. Includes system diskette with sample programs and user's handbook. **$39.95**

Other software also available!
Call now for free catalog and the name of your nearest dealer. Phone: **616/241-5510.**

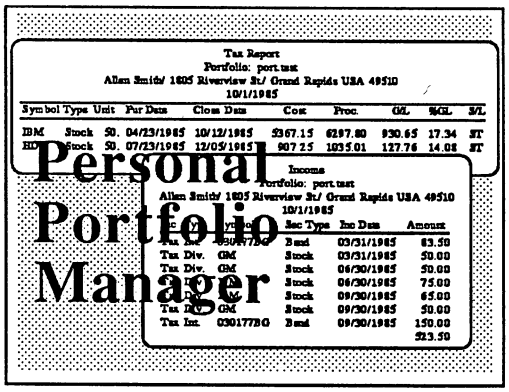
Abacus Software

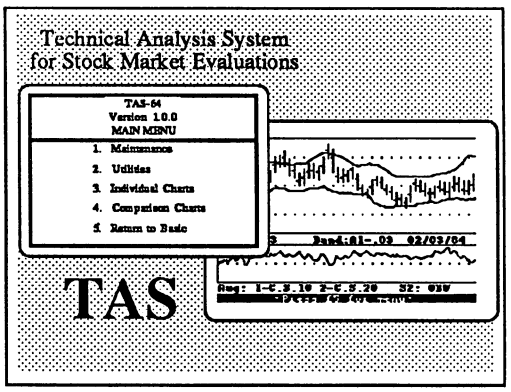
P.O. Box 7211 Grand Rapids, MI 49510 616/241-5510

For fast service call **616/241-5510.** For postage and handling, include $4.00 per order. Foreign orders include $8.00 per item. Money orders and checks in U.S. dollars only. Mastercard, Visa and Amex accepted.

Dealer Inquiries Welcome
More than 1200 dealers nationwide

Manage your Money
on your Commodore 128 or 64





Personal Portfolio Manager

- online data collection thru DJNRS or Warner Computer or manual entry
- manage stocks, bonds, options, mutual funds, treasury bills, others.
- record dividends, interest and transactions for year end tax requirements
- unique report generator produces reports in any desired format
- 30 day money back guarantee

$39.95 + $4.00 shipping

Technical Analysis System

- online data collection thru DJNRS or Warner Computer or manual entry
- 7 moving averages, 5 volume indicators, least squares, trading band, comparison and relative charts, more.
- 300 trading days for up to 10 stocks per disk. Unlimited number of disks
- Hardcopy of charts
- 30-day money back guarantee

$59.95 + $4.00 shipping

You Can Count On

Abacus Software

P.O. Box 7211 Grand Rapids, MI 49510 Phone 616/241-5510 Telex 709-101

COMMODORE 128

THE AUTHORITATIVE INSIDERS' GUIDE

INTERNALS

This book guides you deep into the heart of the Commodore 128. **128 Internals** is written for those of you who want to push your computer to the limits. This book contains the complete, fully commented ROM listings of the operating system kernal. Here is a list of just some of the things that you can expect to read about:

- Using the interrupts
- Assembly language programming and Kernal routines
- Z-80 processor and the boot ROM
- Peripherals and the ports
- Programming for sound and music
- Programming the various graphic modes
- Understanding and using the Input/Output ports
- Programming the Memory Management Unit (MMU)
- Using the 80-column chip -
  - getting 640 X 200 point resolution
  - getting more than 25 lines on the screen
  - smooth scrolling
  - copying blocks in screen memory
  - character length and width management

About the authors:
Klaus Gerits is the Director of Product Development at Data Becker Software House. Joerg Scheib, a highly experienced programmer and book author, and Frank Thrun, an avid Commodore programmer, are also members of the Data Becker development staff based in Duesseldorf, W. Germany.

ISBN 0-9164439-42-9

A Data Becker book published by

You Can Count On Abacus Software