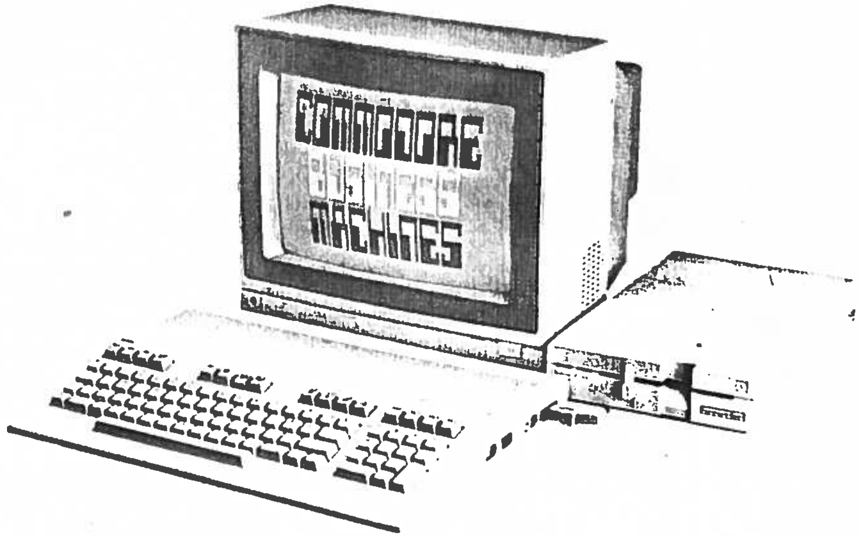
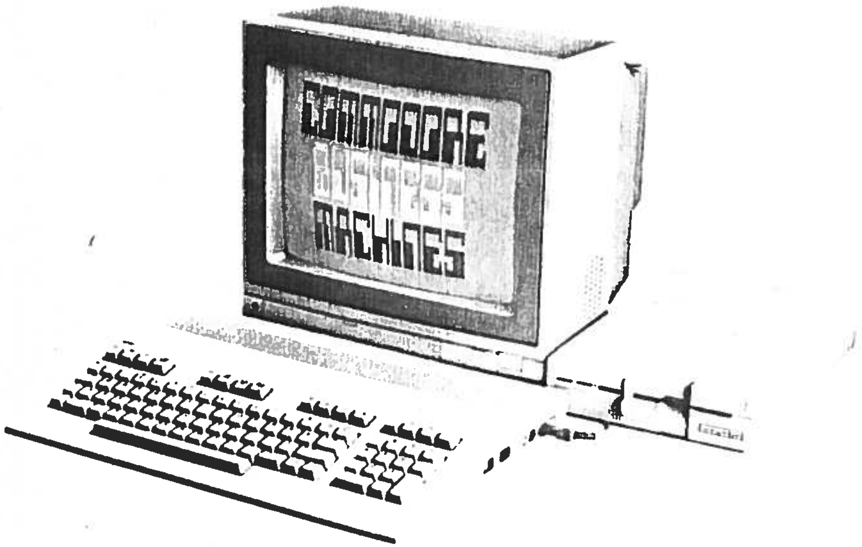
# Tricks and Tips FOR THE C-128




FIRST PUBLISHING LTD

A Data Becker Book from First Publishing Ltd.

# Tricks and Tips FOR THE C-128




FIRST PUBLISHING LTD

A Data Becker Book from First Publishing Ltd.

# Tricks and Tips

## for the

# C-128

By Tobias Weltner, Ralf Hornig and Jens Trapp

A Data Becker Book

Published by

First Publishing Ltd
20B Horseshoe Park
Pangbourne
Berks.
Tel: 07357 5244

ISBN 0 948015 531

Printed and Bound by Athenæum Press
Newcastle upon Tyne

# TABLE OF CONTENTS

# FOREWORD

We've written this book for every Commodore 128 owner who wants to make better use of his or her machine. Whether you want to create your own character set, use the higher computing speed (C-128 FAST mode) in your C-64 programs, or use the ROM routines, you'll find this book full of practical information. Some of the topics covered in this book include: Banking and memory configurations; VIC-II chip registers; windows; multitasking; command extensions; important memory locations; and many, many sample programs.

We've tested and debugged all of the BASIC programs, as well as the BASIC loaders that match the machine language listings. An optional companion disk is available containing all of the programs in this book. See the ordering instructions at the end of this book for details. We used a modified version of the *LISTING CONVERTER* program (found in chapter 3) to transfer the program listings from the C-128 to the computer this book was edited with. There should be no errors as far as the listings themselves are concerned. The text proper will describe the operation of the programs. Before we go into the depths of the C-128, we'll just remind you of Murphy's Laws on Programming*:

1) Once a running version of a program is ready, it's already obsolete.
2) All other programs cost more and take longer to run.
3) If a program is useful, odds are it can be replaced.
4) If a program is useless, it will be documented.
5) Every completed program takes up all the memory, whether it was written that way or not.
6) The value of a program is proportionate to the time taken in mass-producing it.

7) Program development takes so long that by the time you get it running, you'll have to revise it to keep up with the times.

Have fun,

The Authors

Rinteln, Germany, August 1985

* Source: A. Bloch, "*Why What Can Go Wrong, WILL*", Goldman, 1977

# CHAPTER 1

# GRAPHICS ON THE COMMODORE 128

Graphics are an intriguing subject, particularly when we're talking about high-resolution graphics (as opposed to the graphic symbols built into the C-128 character set). For those who were stopped from writing professional software for the Commodore 64 because of its 40-column screen, this new machine offers another possibility. If you look on the back of the computer, you'll see two jacks marked **RF** and **VIDEO**; these allow you to connect the C-128 to a television set (RF) and a composite monitor (such as the Commodore 1701). These two jacks give you 40 columns, as with the C-64, but the 128 has one more jack -- an interface marked **RGB**! An RGB monitor (Red Green Blue) is much more expensive when compared to, say, the 1701, but with the RGB monitor you get better picture quality, higher resolution and, most importantly, an 80-column screen.

The next few pages discuss exactly what can be done with the graphics screen. Numerous sample programs illustrate these discussions. Remember that one major difference exists between the C-64 and C-128 graphics: the C-128 allows two completely independent screens, which we'll discuss next.

## 1.1 SWITCHING: 40/80 COLUMNS

Before going into any detail, we should take a look at switching between the 40-column and 80-column screens. The setting of the 40/80 DISPLAY key on the upper section of the keyboard determines the screen mode when powered on. After power-up, the switch is inoperative unless the reset button is pressed. To switch modes within a program use the following commands:

| | |
|---|---|
| `ESC+X` | (switches direct mode on) |
| `PRINT CHR$(27)+"X"` | (switches mode outside program) |
| `SYS 49194` | (works like the switching from BASIC, but also works in machine language) |

## 1.2 THE 40-CHARACTER SCREEN

The 40 column screen is controlled by the VIC 8564 chip. The VIC 8564 is similar to the VIC 6564 chip in the 64, but it contains an additional two registers (more on this later). When using the 40 column mode, the screen can be displayed on either a television or a composite monitor, but NOT on an RGB monitor.

Video RAM and color memory use the same memory ranges as those in the 64:





```
SCREEN RAM  $0400 - $07FF  (decimal 1024 - 2023)
COLOR  RAM  $D800 - $DBFF  (decimal 55296-56295)
```

Both ranges are in normal address space, and can be manipulated using PEEK and POKE:

```
10 FOR A=0 TO 255
20 POKE 1024+A,A
30 NEXT A
40 :
50 FOR A=0 TO 255
60 X=INT(RND(1)*16):REM RANDOM NUMBER (COLOR)
70 POKE 55296+A,X
80 NEXT A
```

The statement:

```
POKE 1024+column+(40*line),0-255
```

puts a character onto a forty-column screen; while the statement:

```
POKE 55296+column+(40*line),0-15
```

puts a color into the matching memory location. In either statement, `column` can range in value from 0 to 39 and `line` can range from 0 to 24.

As in the C-64, screen memory can be moved in 1K steps, but color memory is not relocatable.

## 1.3 THE 40-COLUMN CHARACTER GENERATOR

The actual design of a character is stored in a ROM known as the character generator. The character generator is found in memory at $D000 to $DFFF. You can't read the character generator by normal means, since it lies in ROM, and you can't write to it at all.

The internal divisions of the character generator look like this:

CHARACTER SET 1 (UPPER CASE and BLOCK GRAPHICS)

| | |
|---|---|
| $D000 - $D1FF | Upper case letters |
| $D200 - $D3FF | Block characters |
| $D400 - $D5FF | Reverse upper case letters |
| $D600 - $D7FF | Reverse block graphics |

CHARACTER SET 2 (UPPER CASE and LOWER CASE)

| | |
|---|---|
| $D800 - $D9FF | Lower case letters |
| $DA00 - $DBFF | Upper case letters |
| $DC00 - $DDFF | Lower case reversed |
| $DE00 - $DFFF | Upper case reversed |

The following short program allows you to read the character generator:





```
90   PRINT "{CLR HOME} PRINT THE CHARACTER
      GENERATOR ROM"
91   PRINT : PRINT
100  REM  OUTPUT CHAR GENERATOR
120  :
130  FOR X = 0 TO 255:  REM  256 CHARACTERS
140  FOR Z = 0 TO 7:  REM  8 BYTES EACH
150  AD = 53248 + X * 8 + Z
180  BANK 14: C =  PEEK (AD): BANK 15
210  :
220  FOR Y = 7 TO 0 STEP  - 1:  REM  8 PIXELS/BYTE
230  IF C >  = 2 ^ Y THEN C = C - 2 ^ Y:
     PRINT "*";:  ELSE  PRINT ".";
240  NEXT Y
250  :
260  PRINT
270  NEXT Z
280  PRINT
290  NEXT X
300  END
```

The pattern of each character is stored in eight bytes; each byte is divided into eight bits. So, one character has a matrix of 64 bits. Each of these bits can be turned on or off. The character generator is taken directly from ROM in 40-column mode. Therefore to make any changes to a character's pattern, we have to copy the character generator into RAM. The ROM character generator cannot be moved or changed, but the copied generator can be moved to a different memory location.

We'll begin from a cold start in BASIC. Type the following in direct mode:

```
POKE 56,48:POKE 58*256,0:NEW
```

Any program currently in memory will be lost. To copy the character set from the character generator to RAM, use the following:

```
10 REM COPY CHARGEN $D000 TO $2000
20 BANK 14 : REM READ OUT CHARGEN
30 FOR X=0 TO 4095
40 POKE DEC("2000")+X,PEEK(DEC("D000")+X)
50 NEXT X
60 BANK 15 : REM NORMAL CONFIG.
```

Here's where we see how slow BASIC is; the entire procedure takes more than a minute! The program moves the character generator from ROM to the RAM area near the start of BASIC.

Now the character generator is in RAM. We know this, but the computer doesn't; you'll have to tell the computer to use the copied character set (64 fans will note that the address 53272 is the same). The memory contents of this location determines the starting addresses of the character generator and screen RAM. We'll skip the latter for the moment, and have a look at the character generator itself. Here's an overview of possible starting combinations:

| SCREEN MEMORY | | CHARACTER SET | |
|---|---|---|---|
| 0000xxxx | 0 | xxxx000x | 0 |
| 0001xxxx | 1024 | xxxx001x | 2048 |
| 0010xxxx | 2048 | xxxx010x | 4096 |
| 0011xxxx | 3072 | xxxx011x | 6144 |
| 0100xxxx | 4096 | xxxx100x | 8192 |
| 0101xxxx | 5120 | xxxx101x | 10240 |
| 0110xxxx | 6144 | xxxx110x | 12288 |
| 0111xxxx | 7168 | xxxx111x | 14336 |
| 1000xxxx | 8192 | | |
| 1001xxxx | 9216 | | |
| 1010xxxx | 10240 | | |
| 1011xxxx | 11264 | | |
| 1100xxxx | 12288 | | |
| 1101xxxx | 13312 | | |
| 1110xxxx | 14336 | | |
| 1111xxxx | 15360 | | |





The character set can be moved in 2K increments. This has two disadvantages: First, the character set can only be moved within the first 16K of memory. Second, the normal ROM character set is located at 6144, but that's normal BASIC RAM. How can that be?

The VIC chip can only address 16K, which in this case is the first 16K of memory. The address 6144 represents the offset within this 16K block. We can determine which 16K block is involved by changing the contents of address 56576:

16K range:

| | | |
|---|---|---|
| $0000 - $3FFF | 0 - 16383 | POKE 56576,199 |
| $4000 - $7FFF | 16384 - 32767 | POKE 56576,198 |
| $8000 - $BFFF | 32768 - 49151 | POKE 56576,197 |
| $C000 - $FFFF | 49152 - 65535 | POKE 56576,196 |

When the last 16K block ($C000-$FFFF) is used (default), the character generator resides at 49152 + 4096 = 53248 ($D000). If you check the first table, you'll see that the normal character set (upper case/graphics) begins at $D000. The first two bits in 56576 represent address bits 14 and 15 of the character generator.

Keep in mind that the screen memory is also moved in 16K steps. The high byte of screen RAM has to be moved:

POKE 2619,4+X*64

X=0-3 (matches 16 banks 0-3)

To let the computer know that the new character set is at $2000, you'll have to change the contents of 53272; here we find a substantial difference between the C-128 and the C-64. Where the C-64 required only a simple POKE, here we don't have that luxury -- what you POKE will be reset by the C-128 operating system. So, to get around this, we'll have to deal with a byte in zero-page memory:

```
$0A2C(2604) VIC TEXT SCREEN/CHAR BASE POINTER
```

This looks more complicated than it actually is. POKEing into 53272 on the C-64 is equivalent to POKEing into 2604 on the C-128. The contents of this address automatically writes to 53272.

Switching to our new character set can be accomplished with this statement:

```
POKE 2604,PEEK(2604) AND NOT 2+4+8 OR 8
```

In other words, bits 1 to 3 (controlling the position of the character generator) are cleared, and bit 3 is set. So, address 53272 gives us these contents: xxxx100x

This statement sets the new character generator at 8192. Bear in mind that all we've done is move the character set around; the procedure isn't finished. Notice how odd the characters look on the screen.





## 1.3.1 CHANGING THE CHARACTER SET

Now, type in the following BASIC program:

```
10 REM @  SIGN TO SQUARE
20 FOR X = 0 TO 7: REM 8 BYTES PER CHARACTER
30 READ CO
40 POKE 8192+0*8+X,CO
50 NEXT X
60 DATA 255,129,129,129,129,129,129,255
```

The @ sign changes before our eyes to a square (see "Defining your Own Characters" in the 80-column section for more information).

## 1.3.2 40-COLUMN CHARACTER EDITOR

Fortunately, you don't need to design your new 40-column character set by hand: With a few small changes, you can use the 80-column character editor (Chapter 1.16.1). First move the start of BASIC to protect your character set from overwriting your BASIC program. In direct mode, enter:

```
POKE 46,58:POKE14848,0;NEW
```

Then change the following lines of the 80-column character editor:

```
4000 BANK 14
4010 FOR X = 0 TO 4096
4020 POKE DEC("2000")+X,PEEK(DEC("D000")=X)
4030 NEXT X
4050 POKE 2619,4
```

```
4060 POKE 5676,199
4070 POKE 2604,PEEK(2604) AND NOR 2+4+8 OR 8
4080 BANK 15
5005 - 5045 :  DELETE
5065 AD=8192+8*A+Y
5070 POKE AD,W
5075         :  DELETE
```

Edit the characters, then exit the program and RUN 40000 to enable your custom character set.

## 1.4 MOVING SCREEN MEMORY

Screen memory normally resides in $0400 - $07FF (1024 - 2023), but it can be relocated. Remember these two addresses:

```
2604 VIC TEXT SCREEN/CHAR BASE POINTER
2619 VIC TEXT SCREEN BASE
```

We can move screen memory anywhere in memory, in 1K steps. See Chapter 1.3 for the table showing possible addresses. Meanwhile, let's get started on uses for relocating screen memory.





### 1.4.1 WORKING WITH SEVERAL SCREENS

As the title suggests, you have the option of using several screens at once (called "page-flipping") in 40-column mode. We'll illustrate this using three screens. Since these three screens will use part of normal BASIC memory, you must move the start of BASIC so that your programs don't overwrite the new screen memory. In direct mode, enter the following statements:

```
POKE 46,40: POKE10240,0: NEW
```

The following BASIC program POKEs a machine language program into memory which can be used to "page flip" between three screens by using the <F1>, <F3>, and <F5> keys.

```
2000 FOR X = 4864 TO 4950
2010 READ A : CS=CS+A: POKE X,A
2020 NEXT X
2030 IF CS <> 7857 THEN PRINT CHR$(7); LIST
2040 DATA  120,169,24,141,20,3,169,19,141,21,3,
           169,0,141,0,16
2050 DATA 141,2,16,141,4,16,88,96,72,138,72,
           166,213,224,4,208
2060 DATA 13,169,20,141,44,10,169,4,141,59,
           10,76,80,19,224,5
2070 DATA 208,13,169,132,141,44,10,169,32,
           141,59,10,76,80,19,224
2080 DATA 6,208,13,169,148,141,44,10,169,
           36,141,59,10,76,80,19
2090 DATA 104,170,104,76,101,250,255
```

Here's a listing of the machine language program that is POKEd into memory by the above BASIC program:

```
1300 78       SEI         :interrupt off
1301 A9 18    LDA #$18    :low-byte of new IRQ
1303 8D 14 03 STA $0314   :store low-byte
1306 A9 13    LDA #$13    :high-byte of new IRQ
1308 8D 15 03 STA $0315   :store it
130B A9 00    LDA #$00    :Length of function keys
130D 8D 00 10 STA $1000   :F1 on
1310 8D 02 10 STA $1002   :F3 on
1313 8D 04 10 STA $1004   :F5 on
1316 58       CLI         :interrupt again permitted
1317 60       RTS         :back to BASIC
```

NEW IRQ:

```
1318 48       PHA         :put accu
1319 8A       TXA         :and X-reg
131A 48       PHA         :on stack
131B A6 D5    LDX $D5     :load X w/ pressed key
131D E0 04    CPX #$04    :F1?
131F D0 0D    BNE $132E   :no, then read next
1321 A9 14    LDA #$14    :new starting address
1323 8D 2C 0A STA $0A2C   :and set
1326 A9 04    LDA #$04    :new screen at $0400
1328 8D 3B 0A STA $0A3B   :and set it
132B 4C 50 1A JMP $1350   :end F1
132E E0 05    CPX #$05    :F3?
1330 D0 0D    BNE $133F   :no, then read next
1332 A9 84    LDA #$84    :new starting address
1334 8D 2C 0A STA $0A2C   :and set
1337 A9 20    LDA #$20    :new screen at $2000
1339 8D 3B 0A STA #$0A3B:and set
133C 4C 50 1A JMP $1350   :end F3
133F E0 06    CPX #$06    :F5?
1341 D0 0D    BNE $1350   :no, then ready
1343 A9 94    LDA #$94    :new starting address
1345 8D 2C 0A STA #$0A2C:and set
1348 A9 24    LDA #$24    :new screen at $2400
134A 8D 3B 0A STA $0A3B   :and set
134D 4C 50 1A JMP $1350   :end F5
1350 68       PLA         :Put back old
1351 AA       TAX         :X-reg and
1352 68       PLA         :accumulator values,
1353 4C 65 FA JMP $FA65   :and return normal IRQ
```





The principle is the same for 80-column mode (see Chapter 1.17).

After RUNning the program, you have your choice of three independent screen pages, called by F1, F3 and F5. These screens will initially be full of strange characters -- clear the individual screens with:

`PRINT CHR$(147)` (or with PRINT"{CLR/HOME}")

You can switch screens in program mode with a POKE to address 213:

```
POKE 213,4 (normal screen)
POKE 213,5 (2nd screen at $2000)
POKE 213,6 (3rd screen at $2400)
```

Here is the memory configuration used by the routine:

| | SCR 1(normal) | SCR 2 | SCR 3 |
|---|---|---|---|
| Start of screen memory | $0400 | $2000 | $2400 |
| End of screen memory | $07FF | $23FF | $27FF |
| Color RAM -- start | $D800 | $D800 | $D800 |
| Color RAM -- end | $DBFF | $DBFF | $DBFF |
| BASIC start | $2800 | $2800 | $2800 |
| Press | F1 | F3 | F5 |

```
1300 78          SEI          ;interrupt off
1301 A9 18       LDA #$18     :low-byte of new IRQ
1303 8D 14 03    STA $0314    :store low-byte
1306 A9 13       LDA #$13     :high-byte of new IRQ
1308 8D 15 03    STA $0315    :store it
130B A9 00       LDA #$00     :Length of function keys
130D 8D 00 10    STA $1000    :F1 on
1310 8D 02 10    STA $1002    :F3 on
1313 8D 04 10    STA $1004    :F5 on
1316 58          CLI          :interrupt again permitted
1317 60          RTS          :back to BASIC
```

NEW IRQ:

```
1318 48          PHA          :put accu
1319 8A          TXA          :and X-reg
131A 48          PHA          :on stack
131B A6 D5       LDX $D5      :load X w/ pressed key
131D E0 04       CPX #$04     :F1?
131F D0 0D       BNE $132E    :no, then read next
1321 A9 14       LDA #$14     ;new starting address
1323 8D 2C 0A    STA $0A2C    :and set
1326 A9 04       LDA #$04     :new screen at $0400
1328 8D 3B 0A    STA $0A3B    :and set it
132B 4C 50 1A    JMP $1350    :end F1
132E E0 05       CPX #$05     :F3?
1330 D0 0D       BNE $133F    :no, then read next
1332 A9 84       LDA #$84     :new starting address
1334 8D 2C 0A    STA $0A2C    :and set
1337 A9 20       LDA #$20     :new screen at $2000
1339 8D 3B 0A    STA #$0A3B:and set
133C 4C 50 1A    JMP $1350    :end F3
133F E0 06       CPX #$06     :F5?
1341 D0 0D       BNE $1350    :no, then ready
1343 A9 94       LDA #$94     :new starting address
1345 8D 2C 0A    STA #$0A2C:and set
1348 A9 24       LDA #$24     :new screen at $2400
134A 8D 3B 0A    STA $0A3B    :and set
134D 4C 50 1A    JMP $1350    :end F5
1350 68          PLA          :Put back old
1351 AA          TAX          :X-reg and
1352 68          PLA          :accumulator values,
1353 4C 65 FA    JMP $FA65    :and return normal IRQ
```





The principle is the same for 80-column mode (see Chapter 1.17).

After RUNning the program, you have your choice of three independent screen pages, called by F1, F3 and F5. These screens will initially be full of strange characters -- clear the individual screens with:

`PRINT CHR$(147)` (or with PRINT"{CLR/HOME}")

You can switch screens in program mode with a POKE to address 213:

```
POKE 213,4 (normal screen)
POKE 213,5 (2nd screen at $2000)
POKE 213,6 (3rd screen at $2400)
```

Here is the memory configuration used by the routine:

| | SCR 1(normal) | SCR 2 | SCR 3 |
|---|---|---|---|
| Start of screen memory | $0400 | $2000 | $2400 |
| End of screen memory | $07FF | $23FF | $27FF |
| Color RAM -- start | $D800 | $D800 | $D800 |
| Color RAM -- end | $DBFF | $DBFF | $DBFF |
| BASIC start | $2800 | $2800 | $2800 |
| Press | F1 | F3 | F5 |

## 1.5 THE 80-COLUMN SCREEN

The C-128 isn't just for games. To make it a more practical machine, it has 80-column capability, thanks to a special graphic processor called the VDC 8563 (Video Display Controller). We should mention that this chip displays 80 columns only on a RGB monitor. We'll talk about that later.

## 1.6. SCREEN AND COLOR RAM

Now we enter completely new territory. While screen and color RAM for 40-column mode is in normal RAM, and can be controlled by PEEKing and POKEing, 80-column video RAM is outside of normal RAM, and can't be changed using normal PEEKs and POKEs! Don't panic yet -- the next couple of pages show how we can gain control of 80-column video controller. On the next page is a list of registers for the 80-column controller; we'll cover each register in detail.



## 1.7 REGISTERS OF THE 80-COLUMN CONTROLLER

| | |
|---|---|
| 00 | READ: Status, LP,VBlank,-,-,-,-,-<br>WRITE: Bits 0-5 of desired register |
| 01 | Characters per line |
| 02 | Shift screen window (horizontal/character-wise) |
| 03 | Shift screen window (horizontal/pixel-wise) |
| 04 | Vertical synchronization |
| 05 | Vertical total |
| 06 | Lines per screen page |
| 07 | Shift screen window (vertical/line-wise) |
| 08 | Interface mode |
| 09 | Matrix register -- vertical |
| 10 | Cursor mode -- begin scan |
| 11 | End scan |
| 12 | Screen memory start address -- HI |
| 13 | LO of 12 |
| 14 | Cursor position -- HI |
| 15 | LO of 14 |
| 16 | Light pen vertical |
| 17 | Light pen horizontal |
| 18 | Channel address HI |
| 19 | LO of 18 |
| 20 | Addribute-RAM start address -- HI |
| 21 | LO of 20 |
| 22 | Matrix register/display horizontal |
| 23 | Matrix display vertical |
| 24 | Smooth-scroll vertical |
| 25 | Smooth-scroll horizontal |
| 26 | Color |
| 27 | Address shifting |
| 28 | Character generator basic address -- HI |
| 29 | Underline-Cursor-Scan-Line |
| 30 | Repeat register |
| 31 | Channel, byte read/write in video RAM |
| 32 | Block start address -- HI |
| 33 | LO of 32 |
| 34 | Start of screen representation |
| 35 | End of screen representation |
| 36 | refresh-rate |

# 1.8 THE VIDEO DISPLAY CONTROLLER (VDC)

We'll now cover the functions of the individual registers using examples.

These registers are indirectly addressed. That means that only registers 0 and 1 can be accessed. If you want to see the contents of register 26, you'd type this in:

```
A=DEC("D600")
POKE A,26:PRINT PEEK(A+1)
```

The register number is written into register 0 ($D600); register 1 ($D601) acts as the channel for writing to or reading from the desired register:

```
10 INPUT"REGISTER";R
20 INPUT"VALUE";V
30 POKE DEC("D600"),R
40 POKE DEC("D601),V
```

## Access to the Video RAM

As already mentioned, the VDC video has 16K of RAM outside of the normal address range. To access the VDC video RAM, use the following method:

```
10  BA = DEC("D600")
110 INPUT "ADDRESS OF THE VIDEO RAM";V
120 INPUT "VALUE";W
130 HO = INT(V/256): LO= V-(256*HI)
140 POKE BA, 18: POKE BA+1, HI
150 POKE BA, 19: POKE BA+1, LO
160 POKE BA, 31: POKE BA+1, W
```





```
170 WAIT BA, 32
180 POKE BA, 30: POKE BA+1, 1
```

Program explanation:

| | |
|---|---|
| 100 | Store the base address of the VDC in BA |
| 110 | Prompt for desired video RAM address (V) |
| 120 | Prompt for value you want written into video RAM (W) |
| 130 | Separate V into low and high bytes |
| 140 | Put high byte desired address into register 18 |
| 150 | Put low byte into register 19 |
| 160 | Byte value into register 31 |
| 170 | Wait command until we reach memory address BA |
| 180 | Register 30 filled with 1 (character output) |

Here is the video RAM layout on power-up:

$0000 - $07FF Screen refresh memory (SCR-RAM)
(dec. 0-2047)

$0800 - $0FFF Attribute RAM (e.g. color memory)
(dec. 2048-4095)

$1000 - $1FFF free
(dec. 4096 - 8191)

$2000 - $3FFF character generator
(dec. 8192-16385)

Now we'll try to write to video RAM. Start the above BASIC program and input 0 as the desired address; then enter a value between 0 and 255. A character should appear in the upper left-hand corner of the 80-column screen -- the character displayed depends upon the value you enter.

To tell you the truth, this method of accessing video RAM is pretty unreliable, but you can repeatedly write to this RAM (as long as it's within bounds!). On the other hand, this isn't a method for a serious programmer. We'll give you another method in the next section.

## 1.9 PRACTICAL VIDEO RAM ACCESS

What does the operating system do when a key is pressed while in 80-column mode? It seems to work perfectly. Well, let's explore how the operating system treats characters in this mode. Of particular interest is a ROM routine which you can easily call yourself. Here it is:

```
SYS 49155,CHARACTER,COLOR
```

```
CHARACTER   =char #(0-255)
COLOR       =char. color (0-16)
```

Unlike the previous BASIC routine, this routine always works. Plus, this routine works for both the 40- and 80-column screens. This means that it's possible to program for both screens at once (or two separate monitors).





Once the novelty of the above routine wears off, you may wonder how to put these characters on different lines of the screen. The position of the character is read from memory locations 224 and 225 (current cursor position). If you want a character in a specific place, you'll have to play around a bit with these memory locations:

```
10 AD = CLMN + PEEK(238)* LINE
20 S1 = PEEK (224): S2 = PEEK(225)
30 HI = (AD/256): LO = AD-(256*HI)
40 POKE 244, LO: POKE 225, HI
50 SYS 49155, CHARACTER, COLOR
60 POKE 224, S1: POKE 225, S2
70 END
```

CLMN:0-79 (0-39)
LINE :0-24

| | |
|---|---|
| 238 | Maximum length of screen |
| 224 | Cursor position -- LOW |
| 225 | Cursor position -- HIGH |
| 49155 | Start address -- ROM routine |

## 1.10 POKE SIMULATION

This machine language routine does away with all the compromises of the previous techniques; we call it a "modified POKE command". Basically, it's a pseudo-POKE for 80-column video RAM.

```
1800 48       PHA        :Get char from stack
1801 8A       TXA        :low byte address
1802 48       PHA        :placed on stack
1803 98       TYA        :high byte address
1804 48       PHA        :put on stack
1805 A9 02    LDA #$02
```

```
1807 8D 28 0A STA $0A28   :set cursor flag
180A A2 12    LDX #$12    :VDC register 18
180C 68       PLA         :get high byte back
180D 20 1B 1C JSR $181B   :set register
1810 E8       INX         :VDC register 19
1811 68       PLA         :get back low byte
1812 20 1B 1C JSR $181B   :set register
1815 A2 1F    LDX #$1F    :VDC register 31
1817 68       PLA         :get back character
1818 4C 1B 1C JMP $181B   :set register -- ready
--------------------------------------------------
181B 8E 00 D6 STX $D600   :register 0
181E 2C 00 D6 BIT $D600   :bit 7 set?
1821 10 FB    BPL $1C8E   :no -- then test again
1823 8D 01 D6 STA $D601   :give value in D601
1826 60       RTS
```

For those of you who don't program in machine language, here is the BASIC loader.

```
 5   REM  1.10A
10   FOR X = 6144 TO 6182
20   READ A: CS = CS + A:  POKE X,A
30   NEXT X
40   IF CS <  > 3411 THEN  PRINT  CHR$ (7);:  LIST
50   DATA 72,138,72,152,72,169,2,141,40,10,162,18
60   DATA 104,32,27,24,232,104,32,27,24,162,31,104
70   DATA 76,27,24,142,0,214,44,0,214,16,251,141
80   DATA 1,214,96
```

Now you have an extended POKE command at your disposal, which uses the following format:

```
SYS DEC ("1800"),CHR,LO,HI
```





CHR = character/byte-value (0-255)
LO = low byte of desired address
HI = high byte of desired address

Try this:

```
10 INPUT "ADDRESS";AD
20 HI=INT(AD/256):LO=AD-(256*HI)
30 SYS DEC("1800"),3,LO,HI
```

Given an address of 0; immediately a "C" (3 = screen code C) appears at the HOME area of the 80-column screen. Restart the routine, and input a value of 2048 -- now the "C" is in cyan; you've written it to *attribute* RAM (2048-3047).

One byte of attribute RAM is configured as follows:

| | |
|---|---|
| BIT 0 | brightness |
| BIT 1 | blue |
| BIT 2 | green |
| BIT 3 | red |
| BIT 4 | blink |
| BIT 5 | underline |
| BIT 6 | reverse video |
| BIT 7 | 2nd character set |

The use of the first four bits is obvious: combining these bits results in the 16 available colors. Setting bit 4 causes the corresponding character to blink rapidly.

```
1807 8D 28 0A STA $0A28   :set cursor flag
180A A2 12    LDX #$12    :VDC register 18
180C 68       PLA         :get high byte back
180D 20 1B 1C JSR $181B   :set register
1810 E8       INX         :VDC register 19
1811 68       PLA         :get back low byte
1812 20 1B 1C JSR $181B   :set register
1815 A2 1F    LDX #$1F    :VDC register 31
1817 68       PLA         :get back character
1818 4C 1B 1C JMP $181B   :set register -- ready
-------------------------------------------------------
181B 8E 00 D6 STX $D600   :register 0
181E 2C 00 D6 BIT $D600   :bit 7 set?
1821 10 FB    BPL $1C8E   :no -- then test again
1823 8D 01 D6 STA $D601   :give value in D601
1826 60       RTS
```

For those of you who don't program in machine language, here is the BASIC loader.

```
 5   REM  1.10A
10   FOR X = 6144 TO 6182
20   READ A: CS = CS + A:  POKE X,A
30   NEXT X
40   IF CS < > 3411 THEN PRINT  CHR$ (7);:  LIST
50   DATA 72,138,72,152,72,169,2,141,40,10,162,18
60   DATA 104,32,27,24,232,104,32,27,24,162,31,104
70   DATA 76,27,24,142,0,214,44,0,214,16,251,141
80   DATA 1,214,96
```

Now you have an extended POKE command at your disposal, which uses the following format:

```
SYS DEC ("1800"),CHR,LO,HI
```





| | |
|---|---|
| CHR | = character/byte-value (0-255) |
| LO | = low byte of desired address |
| HI | = high byte of desired address |

Try this:

```
10 INPUT "ADDRESS";AD
20 HI=INT(AD/256):LO=AD-(256*HI)
30 SYS DEC("1800"),3,LO,HI
```

Given an address of 0; immediately a "C" (3 = screen code C) appears at the HOME area of the 80-column screen. Restart the routine, and input a value of 2048 -- now the "C" is in cyan; you've written it to *attribute* RAM (2048-3047).

One byte of attribute RAM is configured as follows:

| | |
|---|---|
| BIT 0 | brightness |
| BIT 1 | blue |
| BIT 2 | green |
| BIT 3 | red |
| BIT 4 | blink |
| BIT 5 | underline |
| BIT 6 | reverse video |
| BIT 7 | 2nd character set |

The use of the first four bits is obvious: combining these bits results in the 16 available colors. Setting bit 4 causes the corresponding character to blink rapidly.

Just for fun, put this new line into the program we just typed in:

```
30 SYS DEC("1800"),2^0+2^3+2^4,LO,HI
```

Now when you input 2048 for AD, the first character on screen blinks pink (light red)! You can also put characters into reverse video or underscore them by setting bit 6 or bit 5 (respectively). Bit 7 lets you change character sets, just as <SHIFT/C=> does. On the 80-column screen it's possible to have BOTH character sets onscreen at the same time, unlike in 40-column mode. This means that you have 512 characters to work with!

## 1.11 THE CHARACTER GENERATOR

Having 512 characters at your finger tips may seem like a lot, but for special purposes (games, math characters, special alphabets, etc.), the in-house character set just isn't enough. So, you have to go in and design the missing characters on your own. This is somewhat easier to do in 80-column mode, since the character generator is already in RAM, and can be changed from there without having to copy it from ROM or moving the start of BASIC.





## 1.12 READING THE CHARACTER GENERATOR

Now we'll read out the character generator from video RAM:

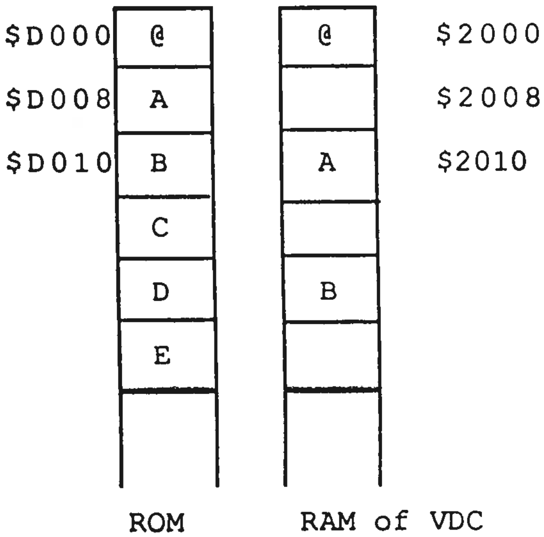
```
20    BA = 8192
30    A =  DEC ("D600"): B = A + 1
40    FOR X = 0 TO 7
50    AD = BA + X + 8 * W:  IF AD > 16383 THEN  END
60    HI =  INT (AD / 256): LO = AD - (256 * HI)
70    POKE A,18:  POKE B,HI
80    POKE A,19:  POKE B,LO
90    POKE A,31: CH =  PEEK (B)
100    FOR Y = 7 TO 0 STEP  - 1
110    IF CH >  = 2 ^ Y THEN CH = CH - 2 ^ Y:
       PRINT "*";:  ELSE  PRINT ".";
120    NEXT Y
130    PRINT
140    NEXT X
150    W = W + 1
160    GOTO 40
```

Variables used:

| | |
|---|---|
| BA | :Base of character generator |
| A | :Base of VDC |
| AD | :Current address in character generator |
| HI | :High byte of AD |
| LO | :Low byte of AD |
| CH | :Read-out value of character line |
| W | :Counter |

RUNning the program causes all the characters to be displayed in an enlarged matrix on the screen. You'll note that eight spaces follow every character. Also note that the 80-column character generator of the C-128 is quite different from those of the VIC-20, or even the C-64.

You see, every character is made up of 16 bytes, as opposed to the eight bytes of the VIC or 64. Contrary to the way it sounds, there is no waste here; normal circumstances give eight bytes per character unused. These "empty bytes" serve a specific purpose -- the VDC 8563 can produce character matrices in either 8 X 8 or 16 X 8 format; see the figure below.








## 1.13 BIG SCRIPT WITH STRINGS

Here is another useful program using the "Character Generator". There's nothing stopping us from making enlarged characters. Try this program.

```
10      Z$ = "*":  GOSUB 60000:  PRINT Z$:  END
60000   Z =  ASC (Z$): Z$ = "":  IF Z AND 128
        THEN X = Z AND 127 OR 64:  GOTO 60040
60010   IF  NOT Z AND 64 THEN X = Z:  GOTO 60040
60020   IF Z AND 32 THEN X = Z AND 95:  GOTO 60040
60030   X = Z AND 63
60040   L$ = "": U$ = ""
60050   FOR W = 0 TO 7
60060   L$ = L$ +  CHR$ (157)
60070   U$ = U$ +  CHR$ (145)
60080   NEXT W
60090   D$ =  CHR$ (17)
60100   REM  CHARACTER GENERATOR SELECTION
60110   A =  DEC ("D600"): B = A + 1
60120   :
60130   FOR Z = 0 TO 7:  REM  8 BYTES
60140   AD = 8192 + X * 16 + Z
60150   HI =  INT (AD / 256): LO = AD - (256 * HI)
60160   :
60170   POKE A,18:  POKE B,HI
60180   POKE A,19:  POKE B,LO
60190   POKE A,31: C =  PEEK (B)
60200   :
60210   FOR Y = 7 TO 0 STEP  - 1:  REM  8 PIXEL
        LINES/CHARACTER
60220   IF C >  = 2 ^ Y THEN C = C - 2 ^ Y: ZE$ =
        ZE$ + "*":ELSE ZE$ = ZE$ + "{SPACE}"
60230   NEXT Y
60240   Z$ = Z$ + ZE$ + L$ + D$: ZE$ = ""
60250   NEXT Z
60260   Z$ =  LEFT$ (Z$, LEN (Z$) - 9) + U$
60270   RETURN
```

## 1.14 PRINTING BANNERS AT HOME

Now that we have enlarged characters on the screen, we might as well print them to the printer. This program will let you print banners and posters of any length. You have a choice of sizes from 8 to 80 times larger than normal:

```
10  S$ = "*": L$ = " "
20  REM  BANNER-PRINTER
30  :
40  PRINT "{CLR HOME}{RVS ON} BANNER-PRINTER {RVS
    OFF}"
50  PRINT "{CRSR DOWN}{CRSR DOWN}THIS PROGRAM
    DEMONSTRATES THE "
60  PRINT "{CRSR DOWN}USE OF THE PRINTER FOR
    VERTICAL PRINTING"
70  PRINT "{CRSR DOWN}{CRSR DOWN}TO PRINT A
    LETTER,"
80  PRINT "{CRSR DOWN}TYPE IN SIZE AND THEN HIT
    'RETURN' ."
85  PRINT :  PRINT  TAB( 7)"{CRSR DOWN}{CRSR
    DOWN}{CRSR DOWN}HIT ANY KEY TO CONTINUE"
90  GET  KEY A$
100 OPEN 4,4
110 PRINT "{CLR HOME}{RVS ON} SIDEWAYS-PRINTER
    {RVS OFF}"
120 PRINT "{CRSR DOWN}{CRSR DOWN}  ENTER SIZE:
    {CRSR DOWN}{CRSR DOWN}"
130 INPUT "{CRSR DOWN}{CRSR DOWN}HEIGHT (1-10)
    ";HO
140 INPUT "WIDTH  (1-...)";BR
150 PRINT "AT COLON-TYPE LETTER TO PRINT"
155 PRINT "HIT [RETURN] TO STOP"
160 PRINT ": ";:  POKE 208,0: GET KEY A$:PRINT A$;
165 IF A$ =  CHR$ (13) THEN  END
170 :
180 AC =  ASC (A$)
190 IF AC AND 128 THEN CO = AC AND 127 OR 64:
    GOTO 230
```





```
200 IF  NOT AC AND 64 THEN CO = AC:  GOTO 230
210 IF AC AND 32 THEN CO = AC AND 95:  GOTO 230
220 CO = AC AND 63
230 REM  ** CHAR CODES  **
240 BANK 14
250 FOR A = 0 TO 7
260 CH(A) =  PEEK (53248 + (8 * CO) + A)
270 NEXT A
280 BANK 15
290 :
300 REM  ** CHAR ROTATE **
310 FOR A = 0 TO 7
320 K(A) = 0
330 NEXT A
340 FOR A = 0 TO 7
350 FOR B = 7 TO 0 STEP - 1
360 W = 2 ^ B
370 IF CH(A) >= W THEN CH(A) = CH(A) - W: K(7 -
    B) = K(7 - B) + 2 ^ A
380 NEXT B,A
390 :
400 REM  ** CHAR OUTPUT **
410 FOR I = 0 TO 7
420 Q$ = ""
430 FOR J = 7 TO 0 STEP  - 1
440 WI = K(I) AND 2 ^ J
450 IF WI THEN  FOR U = 1 TO HO: Q$ = Q$ + S$:
    NEXT U:  GOTO 470
460 FOR U = 1 TO HO: Q$ = Q$ + L$:  NEXT U
470 NEXT J
480 REM  ** DELETE UNNECESSARY BEGINNING SPACES **
490 LX =  LEN (Q$) - 1
500 IF  RIGHT$ (Q$,1) = " " THEN Q$ =  LEFT$
    (Q$,LX):  GOTO 490
510 FOR U = 1 TO BR
520 PRINT# 4,Q$
530 NEXT U
540 NEXT I
550 GOTO 160
```

## 1.15 DEFINING YOUR OWN CHARACTER SETS

Now we'll redefine the built-in character set using our pseudo POKE program from Chapter 1.10. To help you, we've supplied the following program.

Normally, every character is built within an 8 X 8 matrix. A good monitor will allow you to see these individual points. The "A" character looks like this when enlarged:

```
76543210
...**...0
..****..1
.**..**.2
.******.3
.**..**.4
.**..**.5
.**..**.6
........7
```

This is what you get when you run the character generator reading program. You have this same matrix in which to design each of your own characters (if you want still more, read on). Every line of a character consumes one byte of memory, and each pixel within a byte is equal to one bit. Each bit shows a point when it is set (1).

RUN the following program, which turns the "@" sign into a square:

```
10 A = DEC("D600"): B= A+1
20 BA = 8192: ZE= 0
30 FOR X = 0 TO 7
40 AD = BA + X + (8*ZE)
50 HI = INT(AD/256) : LO = AD - (HI*256)
```





```
60 READ CH
70 SYS ("DEC1800"),CH,LO,HI
80 NEXT X
90 END
100 DATA 255,129,129,129,129,129,129,255
```

Now for an explanation of what happens:

| | |
|---|---|
| 10 | A=base if VDC, B=register 1 |
| 20 | BA=base of char. generator, ZE= character to be altered (A=1, B=2, etc.) |
| 30 | change 8 bytes |
| 40 | byte address=base of char. gen+byte number+8*char. number |
| 50 | AD converted into low byte/high byte format |
| 60 | read new byte values |
| 70 | modified POKE routine POKEs value into video RAM |
| 100 | DATA for square: |

```
76543210
********0 255 (2^7+2^6+2^5+2^4+2^3+2^2+2^1+2^0)
*......*1 129 (2^7+2^0)
*......*2 129 (2^7+2^0)
*......*3 129 (2^7+2^0)
*......*4 129 (2^7+2^0)
*......*5 129 (2^7+2^0)
*......*6 129 (2^7+2^0)
********7 255 (2^7+2^6+2^5+2^4+2^3+2^2+2^1+2^0)
```

Now, change line 100 to this

```
100 DATA 60,66,157,161,161,157,66,60
```

or this:

```
100 DATA 66,157,161,161,161,161,157,66
```

and press the "@" key.

## 1.16 80-COLUMN CHARACTER EDITOR

You could redefine the entire character set by hand, and POKE it into memory. You don't need to, though; use this program instead.

```
1      ZN = 5100
2      MA = 7
3      DIM D(MA),W$(MA)
10     D$ = "........"
20     REM  CHARACTER-EDITOR (80-COLUMN-CHARSET)
30     FOR Y = 0 TO 7: D(Y) = 0:  NEXT Y
31     PRINT "{CLR HOME}{RVS ON}{WHT}80-COLUMN
       CHARSET EDITOR-EDIT CHARACTER {RVS OFF}"
32     MF = 1
40     PRINT
50     PRINT "ENTER CHARACTER TO EDIT, THEN
       'RETURN'  -->";:  GET  KEY C$
55     PRINT C$:  PRINT
60     AC =  ASC (C$)
70     IF AC AND 128 THEN CO = AC AND 127:  GOTO 110
80     IF  NOT AC AND 64 THEN CO = AC:  GOTO 110
90     IF AC AND 32 THEN CO = AC AND 95:  GOTO 110
100    CO = AC AND 63
110    AD = 53248 + CO * 8
120    C$ = ""
130    :
140     PRINT "{CRSR DOWN}{RVS ON}{GRN} ORIGINAL
        {RVS OFF}        {RVS ON}{L GRN}   USER
        {RVS OFF}"
150    BANK 14
160    FOR X = 0 TO MA
170    CZ =  PEEK (AD + X):  IF X > 7 THEN CZ = 0
180    FOR Y = 7 TO 0 STEP  - 1
190    IF CZ >  = 2 ^ Y THEN C$ = C$ +"{WHT}*{GRN}"
       : CZ = CZ - 2 ^ Y:  ELSE C$ = C$ + "."
200    NEXT Y
210    PRINT "{GRN}";C$;X; TAB( 16);"{L GRN}";:
       PRINT  USING "##";X;
211    PRINT  TAB( 19);D$
```





```
220    C$ = ""
230    NEXT X
231    PRINT "{RVS ON}{GRN}             {RVS OFF}
       {RVS ON}{L GRN}             {RVS OFF}"
232    BANK 15
240    :
260    REM  EDITOR-ROUTINE
270    PRINT "{WHT}";
271    OPEN 1,0
280    FOR Y = 0 TO MA
290    CHAR ,16,7 + Y
300    PRINT " ->";
301    WINDOW 19,7 + Y,27,7 + Y,MF
302    POKE 244,1:  INPUT# 1,W$(Y)
310    W$(Y) =  LEFT$ (W$(Y),8)
311    WINDOW 0,0,39,24:  REM  0,0,79,24 FOR 80
       COLUMN
320    FOR X = 1 TO 8
330    IF  MID$ (W$(Y),X,1) = "*" THEN D(Y) = D(Y)
       + 2 ^ (8 - X)
340    NEXT X
350    NEXT Y
351    CLOSE 1
360    CHAR ,0,22:  PRINT "CORRECTIONS (Y/N) ?";:
       GET  KEY A$
370    IF A$ = "Y" THEN MF = 0:  FOR Y = 0 TO 7:
       D(Y) = 0:  NEXT Y:  GOTO 260
380    PRINT :  PRINT "USE CHARACTER (Y/N) ?";:
       GET  KEY A$
390    IF A$ = "N" THEN 30
400    REM  USE CHARACTER
1000   PRINT "{CLR HOME}{CRSR DOWN}{CRSR DOWN}";
1010   ZN = ZN + 10
1020   PRINT ZN;"DATA ";CO;",";
1030   FOR X = 0 TO MA:  PRINT D(X);"{CRSR
       LEFT},";:  NEXT X
1031   PRINT "{CRSR LEFT} "
1032   PRINT ZN + 10;"DATA -1"
1039   PRINT :  PRINT "GOTO 30"
1040   PRINT "{HOME}";
1050   FOR Y = 842 TO 845:  POKE Y,13:  NEXT Y
1060   POKE 208,4
1070   END
```

```
5000  REM  BASIC-LOADER CHARACTER DEFINITION
5005  FOR X = 6144 TO 6182
5010  READ A
5015  CS = CS + A
5020  POKE X,A
5025  NEXT X
5030  IF CS <  > 3411 THEN  PRINT  CHR$ (7):  LIST
      5035 - 5050:  END
5035  DATA 72,138,72,152,72,169,2,141,40,10,162,18
5040  DATA 104,32,27,24,232,104,32,27,24,162,31
5045  DATA 104,76,27,24,142,0,214,44,0,214,16,251
5046  DATA 141,1,214,96
5050  READ A:  IF A =  - 1 THEN  END
5055  FOR Y = 0 TO 7
5060  READ W
5065  AD = 8192 + 16 * A + Y
5070  HI =  INT (AD / 256): LO = AD - (256 * HI)
5075  SYS  DEC ("1800"),W,LO,HI
5080  NEXT Y
5085  GOTO 5050
5110  DATA 1,0,0,0,0,0,0,0,0
5120  DATA 1,0,0,0,0,0,0,0,0
5130  DATA -1
```

Variables:

| | |
|---|---|
| ZN: | first line number of DATA statements |
| MA: | matrix 8*MA+1 |
| D(x): | DATA used to calculate character |
| W$(x): | given character line |
| C$: | character to be changed |
| AC: | ASCII code of C$ |
| CO: | screen code of ASCII character |
| CZ: | read-out ROM data |
| CS: | checksum |





Program description:

Don't let the size of this program throw you off; for a well-equipped character editor, it's really a very short program! Be sure to SAVE the program before running it.

Once started, the generator will ask for the first character that you want changed; just press the desired key. Two 8 X 8 character matrices will appear; the left matrix will contain the original character, while the right matrix is for you to do your "paperwork". Asterisks ("*") represent set points, and periods (".") stand for no point.

When you've finished modifying each line, press the <RETURN> key. Once the character is done, you have the option of going in to make corrections. If you wish to do so, press "Y", and make your corrections. If everything is right, just press <RETURN>.

Assuming that the finished character is to your satisfaction, the character editor figures out the DATA statements for you. Once all the characters are changed, stop the program with the <RUN/STOP> key, and type in:

```
DELETE -5000.
```

The editor/generator is deleted, leaving you with a BASIC loader starting at line 5000 (you might want to change the numbering now, using the RENUMBER command). This loader goes into your program, changing only the characters you wished to change -- the other characters appear as normal.

50 - 120 Commodore-specific screen code is stored in C$ (very different from the ASCII code).

140 - 232 **Screen mask is designed.** The memory configuration is switched with BANK 14, by which the ROM character set at $D000 can be read. Finally, a character is read out bit-by-bit, and the bits are set. After reading ROM, configuration switches back to BANK 15.

260 - 351 Editor routine: Screen is opened for data, using INPUT without question mark. Every input line of the user matrix is defined in a window.

360 - 400 Prompts: Any corrections? If so, MF will be set to 0, the window will remain uncleared. D(Y) will be cleared.

1000 - 1039 DATA line of the last character will be printed out in CHAR. CODE, NUMBERS format. Last DATA statement will be -1.

1040 - 1070 Keyboard buffer fills with <RETURN>s, and line is rewritten. For more information on how this works, see Chapter 5.

5000 - 5085 Start of the BASIC loader being produced, with the modified POKE implemented in lines 5005-5045. This section also contains a read loop for the character DATA still to be added.





## 1.17 WORKING WITH MULTIPLE SCREENS

### Video RAM layout:

| | |
|---|---|
| $0000 - $07FF | screen RAM (dec. 0 - 2047) |
| $0800 - $0FFF | color RAM, etc. (dec. 2048 - 4095) |
| $1000 - $1FFF | free (4096 - 8191) |
| $2000 - $3FFF | character generator (8192 - 16385) |

Notice the memory from $1000 to $1FFF. This 4K in the middle of video RAM is unused. You can make good use of this area -- 4K is equal to 2*2K, and you can see that the screen memory ($0000-$07FF) is 2K in size. This free area gives us space to store two additional screen "pages" in addition to normal one displayed on the monitor. There are a number of uses for this. For example, you can have an invisible screen on which graphics are drawn, while you work on the visible screen, and flip back and forth between screens. Or, one screen can have a program listing, the second a disk directory, and the third the program run of the listing on the first screen.

### Implementation:

The screen memory is normally found at $0000 in video RAM, but this isn't a hard and fast rule. Registers 12 and 13 of the VDC contain the high and low byte of screen memory's starting address. Three addresses must be changed to move screen memory:

| | |
|---|---|
| VDC register 12: | high byte of the new starting address |
| VDC register 13: | low byte of the new starting address |

Address 2606($0A2E): high byte - new starting address

Let's say we want to move the start of screen memory from $0000 to $1000. The high byte of this address is 16 (1*4096/256):

```
10    A=DEC("D600"):B=A+1
20    HI=16:LO=0
30    POKE A,12:POKE B,HI
40    POKE A,13:POKE B,LO
50    POKE 2606,HI
```

The screen will be filled with garbage; this is normal. Clear the screen with PRINT CHR$(147). Now you can use this screen as you normally would. When you want to return to your old screen, change line 20 to this:

```
20    HI=0:LO=0
```

Now we're back where we started (although the cursor may not be visible, you'll be able to type).

The machine language program below uses all of the free video memory to give you three screens. It uses the interrupt, and starts immediately after execution:

**Initialization:**

```
1B00 78        SEI         :ignore interrupt
1B01 A9 1C     LDA #$1C
1B03 A0 18     LDY #$18
1B05 8D 15 03 STA $0315   :change interrupt pntr (lo)
1B08 8D 14 03 STA $0314   :change interrupt pntr (hi)
1B0B 58        CLI         :leave interrupt
1B0C A9 00     LDA #$00
1B0E 8D 00 10 STA $1000   :F1 cleared
```





```
1B11 8D 02 10 STA $1002   :F3 cleared
1B14 8D 04 10 STA $1004   :F5 cleared
1B17 60        RTS         :ready
```

New Interrupt

```
1B18 48        PHA         :save accumulator
1B19 8A        TXA
1B1A 48        PHA         :save X register
1B1B A5 D5     LDA $D5     :read keyboard
1B1D C9 58     CMP #$58    :no key? Then go
1B1F F0 3A     BEQ $1C5F   :to normal IRQ routine
1B21 A2 OC     LDX #$0C    :VDC register 12
1B23 C9 04     CMP #$04    :F1?
1B25 F0 0D     BEQ $1C38   :yes -- then goto $1C38
1B27 C9 05     CMP #$05    :F3?
1B29 F0 0E     BEQ $1C3D   :yes -- then goto $1C3D
1B2B C9 06     CMP #$06    :F5? No -- then
1B2D D0 2C     BNE $1C5F   :goto normal IRQ routine
1B2F A9 00     LDA #$00    :screen at $0000
1B31 4C 3E 1B JMP $1B3E   :set register
1B34 A9 10     LDA #$10    :screen at $1000
1B36 4C 3E 1B JMP $11B3E  :set register
1B39 A9 18     LDA #$18    :screen at $1800
1B3B 4C 3E 1B JMP $1B3E   :set register
1B3E 8E 00 D6 STX $D600   :desired register in REG 0
1B41 2C 00 D6 BIT $D600   :bit 7 set?
1B44 10 FB     BPL $1C45   :wait.  Write value
1B46 8D 01 D6 STA $D601   :in video RAM into REG 1
1B49 8D 2E 0A STA $D601   :set pointer in zeropage
1B4C A2 00     LDX #$0D    :VDC register 13
1B4E A9 00     LDA #$00    :low byte =0
1B50 8E 00 D6 STX $D600   :set register
1B53 2C 00 D6 BIT $D600   :bit 7 set?
1B56 10 FB     BPL $1C57   :wait.  Write byte
1B58 8D 01 D6 STA $D601   :in video RAM into REG 1
1B5B 68        PLA         :return X register
1B5C AA        TAX
1B5D 68        PLA         :return accumulator
1B5E 4C 65 FA JMP $FA65   :back to IRQ routine
```

FOR BASIC programmers here is the BASIC loader.

```
10 FOR X = 6912 TO 7008
20 READ A: CS = CS + A:  POKE X,A
30 NEXT X
40 IF CS < > 9318 THEN  PRINT  CHR$ (7);:  LIST
45 SYS 6912
50 DATA 120,169,27,160,24,141,21,3,140,20,3,88
60 DATA 169,0,141,0,16,141,2,16,141,4,16,96
70 DATA 72,138,72,165,213,201,88,240,58,162,12,201
80 DATA 4,240,13,201,5,240,14,201,6,208,44,169
90 DATA 0,76,62,27,169,16,76,62,27,169,24,76
100 DATA 62,27,142,0,214,44,0,214,16,251,141,1
110 DATA 214,141,46,10,162,13,169,0,142,0,214,44
120 DATA 0,214,16,251,141,1,214,104,170,104,76,101
130 DATA 250
```

RUNning the initialization program causes our IRQ vector (Interrupt Request vector) in the second half of the routine to be added to the normal IRQ routine (the IRQ is what the computer executes every 1/60 second). This routine then gives you three separate 80 column screens to work with. You can shift screens in program mode (by pressing F1, F3 or F5; be sure to clear each new screen before use), or in direct mode (POKE 213,4; POKE 213,5; or POKE 213,6, respectively).

## 1.18 MANIPULATING THE VDC 8563

Here's another feature of the new 80-column display controller. To show you that we're not praising this chip too much, there is a demonstration program below which will show you just how versatile the VDC 8563 is. At this point, you may want to review the VDC register listing in Chapter 1.7.



When we manipulate the display controller, the entire normal screen representation is put on the stack. In your experiments, you should remember a complete recovery of the VDC controller is often possible only by switching the computer off. By the same token, you won't cause any internal damage from playing with the registers.

### Moving the Screen Windows

Those of you who owned VIC-20s in "the old days" remember that the entire screen could be moved around. This effect is accomplished on the 80-column C-128 using the VDC registers 2 and 7:

| | |
|---|---|
| 02 | Shifts screen window horizontally & character-wise |
| 07 | Shifts screen window vertically & line-wise |

Let's try it out:

```
10 REM MOVING THE SCREEN WINDOW
20 A=DEC("D600"):B=A+1
30 FOR X=0 TO 255
40 POKE A,2:POKE B,X
50 POKE A,7:POKE B,X
60 NEXT X
70 END
```

This listing will make the screen wander diagonally. You can return it to normal by pressing <RUN-STOP/RESTORE>.

Simulating an explosion may be more to your tastes. The quality of a game often depends on its realism; try this program out. We leave the sound effects to your discretion.

```
10  REM  EXPLOSIONS SIMULATION
15  A =  DEC ("D600"): B = A + 1
20  FOR X = 0 TO 50
30  Y =  INT ( RND (1) * 2) + 101
40  Z =  INT ( RND (1) * 2) + 31
50  POKE A,2:  POKE B,Y
60  POKE A,7:  POKE B,Z
70  NEXT X
80  POKE A,2:  POKE B,102
90  POKE A,7:  POKE B,32
```

## 1.19 MANIPULATING SCREEN FORMAT

You're presently in 80-column mode which, as the name implies, has 80 characters per line, and 25 lines per screen page. Let's say that we want to change this format for now. This is a relatively easy task, using VDC registers 1 and 6:

| | |
|---|---|
| 01 | Characters per line (default 80) |
| 06 | Lines per screen page (default 80) |

NOTE: You can't get any more than 80 columns. Our goal, here, is to increase the number of lines on the screen using this formula:

(new number of columns) * (new number of lines) = 2000





If the total is greater or less than 2000, we may run into trouble. Try this, using 30 lines * 62 characters:

```
10   REM NEW SCREEN FORMAT 62 * 30
20   A=DEC("D600"):B=A+1
30   POKE A,1:POKE B,62
40   POKE A,6:POKE B,30
50   END
```

All this program does is change the screen format to 30 X 62. You can design any format in principle with this program. Perhaps you can make use of this in games simulating a mineshaft, or deep well, or having a sprite move offscreen.

## 1.20 FOR MONOCHROME MONITOR OWNERS

If you're the lucky owner of a "green screen" (or amber, or whatever), you obviously can't take advantage of the C-128's colors. At best, you get two shades of monitor color. What do you need that 2K of attribute RAM for? It's there for screen development, but in the case of monochrome output, it's just collecting dust, so to speak. If we want to use that RAM, we consult VDC register 25:

| | |
|---|---|
| 25 | Smooth Scroll horizontal |

Bit 6 of this register declares whether attribute RAM is on or not:

```
10   REM DEACTIVATING ATTRIBUTE RAM
20   A=DEC("D600"):B=A+1
30   POKE A,25:POKE B,PEEK(B) AND NOT 64
```

This brief routine switches off attribute RAM ($0800 - $0FFF), and turns it over to you to use for screen memory. Naturally, this routine can also be used by RGB monitor owners who wish to do without color.

### The Curtain Falls . . . .

Normally, the screen window sits within a prescribed border. These edges can be adjusted by VDC registers 34 and 35:

| | |
|---|---|
| 34 | Start of screen representation |
| 35 | End of screen representation |

Rather than go into lengthly explanations, here's a demo program:

```
10    A =  DEC ("D600"): B = A + 1
20    INPUT X,Y
30    POKE A,34:  POKE B,X
40    POKE A,35:  POKE B,X - Y
60    GOTO 20
```

The left and right borders can also be moved without disturbing screen contents. Try this:

```
10    A =  DEC ("D600"): B = A + 1
20    FOR X = 0 TO 40
30    POKE A,34:  POKE B,46 - X
40    POKE A,35:  POKE B,46 + X
50    FOR T = 1 TO 10:  NEXT T
60    NEXT X
```





## 1.21 THE 8x16 CHARACTER MATRIX

You played around with custom characters a few pages ago; you'll recall that each character is built into an 8 X 8 matrix:

```
76543210
........0
........1
........2
........3
........4
........5
........6
........7
```

Now that you've had some experience in character design, you may not want to be limited to the 8 X 8 matrix; it's too small to make a spaceship character, and much too large for a small character. In the paragraphs to follow, we'll show you how to change the size of the character matrix itself.

Let's peek into the registers that control matrix size (registers 22 and 23):

| | |
|---|---|
| 22 | Matrix display (horizontal) |
| 23 | Matrix display (vertical) |

## Relative Changing of the Matrix

Each register lists how many pixels are in a character matrix; default of both registers is eight, governed by the first four bits of register 22 and the first five bits of register 23.

```
10    A =  DEC ("D600"): B = A + 1
20    FOR X = 0 TO 8
30    POKE A,22:  POKE B, PEEK (B) AND  NOT 7 OR X
40    FOR T = 1 TO 100:  NEXT T
50    NEXT X
60    FOR X = 0 TO 8
70    POKE A,23:  POKE B,X
71    FOR T = 1 TO 100:  NEXT T
80    NEXT X
90    END
```

Here's a neat little arrangement for game use:

```
10    A =  DEC ("D600"): B = A + 1
20    FOR X = 0 TO 8
30    POKE A,22:  POKE B, PEEK (B) AND  NOT 7 OR X
40    POKE A,23:  POKE B,X
50    FOR T = 1 TO 200:  NEXT T
60    NEXT X
70    END
```

This is only a relative size change, and leaves us with an 8 X 8 matrix, most of which simply goes unused.





## 1.22 TOTAL 16X8 MATRIX MANIPULATION

Let's try developing a 16X8 matrix. In other words, we'll create an 8-column matrix of 16 lines. We'll find the needed numbers at registers 4 and 9 of the VDC:

| | |
|---|---|
| 04 | Vertical synchronization |
| 09 | Vertical matrix register |

Register 9 declares the number of lines in a character. To raise the matrix to 16 points, we'll have to double the amount in register 9, and change the synchronization in 04:

```
10 REM 16 * 8 MATRIX
20 A=DEC("D600"):B=A+1
30 POKE 228,16
40 READ X
50 IF X=-1 THEN END
60 READ Y
70 POKEA,X:POKEB,Y
80 GOTO 20
90 :
100 DATA 9,15
110 DATA 6,17
120 DATA 23,15
130 DATA 4,19
140 DATA 7,19
150 DATA -1
```

After starting the program, the screen looks funny; there's a big space between the screen lines, but the characters are still clear. That space is due to the enlarged matrix; the space is the additional 8 pixels (see Chapter 1.12).

Type PRINT CHR$(27)+"R" in direct mode; this reverses the screen contents, and lets you see the scope of the character expansion. This project gives you 17 lines of 80 characters, with a 16 X 8 matrix. Let's figure out the total resolution:

```
PRINT 17*80*(16*8)
```

This gives you 174,080 pixels!! Since the video RAM is limited to 16K, these points can't be easily set (e.g., bit-mapping).

**Program Explanation:**

| | |
|---|---|
| 10 | A=VDC,B=REG 01 |
| 20 | Bottom window border is set to keep cursor from scrolling offscreen. |
| 30 | VDC loaded with new value. |
| 60 | Read DATA |
| 70 | Switch character size from 8 to 16 pixels. |
| 80 | Limit to 17 lines per screen. |
| 90 | 16 X 8 pixels per character. |
| 100 | 20 lines (+ border) instead of 40. |
| 110 | Bring up screen contents in proper format. |





## 1.23 DOUBLE-HEIGHT CHARACTERS

So, what can we do with that 16X8 matrix? We can create double-height characters, and this next program lets you do just that.

```
10    REM  POKE-ROUT
20    FOR X = 6144 TO 6182
30    READ A: CS = CS + A:  POKE X,A
40    NEXT X
50    IF CS < > 3411 THEN  PRINT "DATA-ERROR":  END
60    DATA 72,138,72,152,72,169,2,141,40,10,162,18
70    DATA 104,32,27,24,232,104,32,27,24,162,31,104
80    DATA 76,27,24,142,0,214,44,0,214,16,251
85    DATA 141,1,214,96
90    REM  COPY
100   FOR W = 0 TO 255:  REM  256 CHARS
110   FOR K = 0 TO 7:  REM  8 LINES EACH
120   AD = 53248 + W * 8 + K
130   BANK 14: K(K)  =  PEEK (AD): BANK 15
140   NEXT K
150   FOR K = 0 TO 15:  REM  15 LINE SET
160   AD = 8192 + W * 16 + K
170   HI =  INT (AD / 256): LO = AD - (256 * HI)
180   SYS  DEC ("1800"),K( INT (K / 2)),LO,HI
190   NEXT K
200   NEXT W
```

This routine is *SLOW* in BASIC. For impatient readers, we'll give you a machine code listing:

```
0B00 A2 03     LDX #$03      :Read loop
0B02 BD 41 0B  LDA 0B41,X    :Load starting address
0B05 95 FA     STA $FA,X     :into free zero page
0B07 DEX                     :Everything read in?
0B08 10 F8     BPL $0B02     :No -- continue
0B0A A2 01     LDX #$01      :Set bank
0B0C 8E 00 FF  STX $FF00     :configuration
0B0F A0 00     LDY #$00      :to 0+vector
```

```
0B11 B1 FA     LDA ($FA),Y   :Read in start address
0B13 48        PHA           :Get it
0B14 A2 00     LDX #$00      :Set bank
0B16 8E 00 FF  STX $FF00     :configuration
0B19 A6 FC     LDX $FC       :Low video RAM address
0B1B A4 FD     LDY $FD       :High video RAM address
0B1D 20 46 0B  JSR $0B46     :POKE subroutine
0B20 E6 FC     INC $FC       :Low=Low+1
0B22 D0 02     BNE $0B26     :Low greater than 0
0B24 E6 FD     INC $FD       :Low=0:High=High+1
0B26 68        PLA           :Get High ROM
0B27 A6 FC     LDX $FC       :Low video RAM address
0B29 A4 FD     LDY $FD       :High video RAM address
0B2B 20 46 0B  JSR $0B46     :POKE subroutine
0B2E E6 FC     INC $FC       :Low=Low+1
0B30 D0 02     BNE $0B34     :Low greater than 0
0B32 E6 FD     INC $FD       :Low=0:High=High+1
0B34 E6 FA     INC $FA       :Low ROM=Low ROM+1
0B36 D0 02     BNE $0B3A     :Still >0?  NO--
0B38 E6 FB     INC $FB       :High ROM=High ROM+1
0B3A A4 FB     LDY $FB       :Load
0B3C C0 E0     CPY #$E0      :Reached end of ROM
0B3E 90 CA     BCC $0B0A     :yet?  NO--go on
0B40 60        RTS           :Return to BASIC
0B41 00 D0 00  20 00         :Pntr starting address
0B46 48        PHA           :Get character
0B47 8A        TXA
0B48 48        PHA           :Get low byte
0B49 98        TYA
0B4A 48        PHA           :Get high byte
0B4B A9 02     LDA #$02
0B4D 8D 28 0A  STA $0A28     :Set cursor flag
0B50 A2 12     LDX #$12      :VDC REG 18
0B52 68        PLA           :Set low byte of
0B53 20 61 0B  JSR $0B61     :register
0B56 E8        INX           :VDC REG 19
0B57 68        PLA           :Set high byte
0B58 20 61 0B  JSR $0B61     :of register
0B5B A2 1F     LDX #$1F      :VDC REG 31
0B5D 68        PLA           :Character
0B5E 4C 61 0B  JMP $0B61     :Register set, ready
0B61 8E 00 D6  STX $D600     :Desired register given
0B64 2C 00 D6  BIT $D600     :Bit 7 set?
```





```
0B67 10 FB     BPL $0B64     :NO--then wait
0B69 8D 01 D6  STA $D601     :for given value
0B6C 60        RTS           :Return
```

There is, of course, a matching BASIC loader:

```
5000  FOR X = 2816 TO 2924
5010  READ A: CS = CS + A:  POKE X,A
5020  NEXT X
5030  IF CS <> 13689 THEN PRINT CHR$ (7);: LIST
5040  DATA 162,3,189,65,11,149,250,202,16,248,162,1
5050  DATA 142,0,255,160,0,177,250,72,162,0,142,0
5060  DATA 255,166,252,164,253,32,70,11,230,252,208,2
5070  DATA 230,253,104,166,252,164,253,32,70,11,230,252
5080  DATA 208,2,230,253,230,250,208,2,230,251,164,251
5090  DATA 192,224,144,202,96,0,208,0,32,0,72,138
5100  DATA 72,152,72,169,2,141,40,10,162,18,104,32
5110  DATA 97,11,232,104,32,97,11,162,31,104,76,97
5120  DATA 11,142,0,214,44,0,214,16,251,141,1,214
5130  DATA 96
```

## Defining the 16X8 Matrix

Let's continue by defining some 16X8 characters. This procedure has already been covered in the chapter on "Designing your own Characters". Make the following corrections in the 80 column character editor program:

```
2 MA=15
5055 FOR Y=0 TO 15
```

## 1.24 MOVING THE VIDEO RAM

Video RAM is divided into four sections:

| | |
|---|---|
| $0000 | 2K Screen memory |
| $0800 | 2K Attribute RAM |
| $1000 | Free |
| $2000 | 8K Character generator |

You'll find the purpose of the free 4K ($1000) in Chapter1.17; for the moment, we're talking about the other three areas. It's possible to move attribute RAM and screen RAM in 256-byte steps, while the character generator can only be moved 8K at a time. Here are the VDC's registers for controlling this:

| | |
|---|---|
| 12 | HIgh byte of screen memory |
| 13 | LOw byte of screen memory |
| 20 | HIgh byte - attribute RAM |
| 21 | LOw byte - attribute RAM |
| 28 | HIgh byte - character generator (bits 5-7) |

### Moving Attribute RAM

The program below moves attribute RAM into any area of video RAM. Please note that you are limited to 256-byte steps.





```
10    REM ATTRIBUTE RAM SHIFTER
20    INPUT "NEW STARTING ADDRESS";AD$
25    AD=DEC(AD$)
30    IF AD/256=INT(AD/256)THEN 40:ELSE PRINT
      "256K STEPS!":GOTO 20
40    HI=INT(AD/256):LO=AD-(256*HI)
50    A=DEC("D600"):B=A+1
60    POKE A,20:POKE B,HI
70    POKE A,21:POKE B,LO
80    POKE 2607,HI
```

The value at line 80 is a special number; it's not enough to simply change the VDC registers or they will remain unchanged. At the same time, a specified address in zero page will be loaded with the high-byte of the new starting address:

| | |
|---|---|
| 2606 | :HIgh byte of screen RAM |
| 2607 | :HIgh byte of attribute RAM |

Try the address "1000"; attribute RAM is moved to the free area. Type a few different characters on the screen and see what you get. Now, run the attribute RAM shifter routine again, and specify "800" as a starting address. The characters take on normal color again.

You can actually use this technique in page-flipping routines (i.e., set up a different color memory on each screen page, and switch back and forth). We suggest the addresses between "1000" and "1800" as the most suitable.

Just to show you what happens when you enter an illegal address, run the attribute RAM shifter program again, and enter "0000", which will put attribute RAM in the same range as screen RAM. The result: every character has its own color and shape!

Or enter "2000", which puts us in the character generator; certain characters lose their normal appearance.

### Moving Screen RAM

This function is analogous to moving attribute RAM. You'll find the important addresses in the preceding sections.

We believe that shifting screen RAM is a useful extra, since it allows you to perform the page-flipping trick (see Chapter 1.17).

## 1.25 COLOR FOR THE 80-COLUMN SCREEN

Color? You bet! 80-column mode gives you a choice of 16 character colors, by using <CTRL 1-8> and <C= 1-8>. For now, though, we'll concern ourselves with changing the border and background colors. There too we have 16 colors to choose from, but we'll have to POKE the colors in. In 40-column mode addresses 53280 and 53281 are used; 80-column mode utilizes register 26 of the VDC for background. Here's what you do to change register 26:

```
POKE DEC("D600"),26:POKE DEC("D601"),X    [X=COLOR]
```

You could use color control characters within a PRINT statement, but it's not the best method. POKEing the color into address 241 is a better method:





```
POKE 241,X [X=COLOR NUMBER FROM 0 TO 15]
```

Look for a moment at the last 4 bits of a byte in attribute RAM:

BIT 4: blinking
BIT 5: underscored
BIT 6: reverse video
BIT 7: 2nd character set

It was once quite difficult to get these functions; the only way you could attain some of these functions was by your own programming efforts.

Two methods of accessing the second character set are to press <C=> and SHIFT simultaneously, or to use a PRINT character string. These other functions still aren't very simple to get at, but programming them has gotten a lot easier!

```
POKE 241,PEEK(241) OR 2^4:PRINT"THIS LINE BLINKS!"
```

```
POKE 241,PEEK(241) OR 2^5:PRINT "UNDERSCORED!"
```

Here's a sample program:

```
10 PRINT "THIS MATTER IS ";
20 POKE 241,PEEK(241)OR 2^5
30 PRINT"IMPORTANT";
40 POKE 241,PEEK(241) AND NOT 2^5
50 PRINT"!"
60 END
```

The word IMPORTANT will blink if you replace 2^5 with 2^4. Using "2^5+2^4" instead of 2^5 alone will simultaneously underline *AND* blink the word. Naturally, 2^6 will bring up reverse video, and 2^7 will call the second character set.

## 1.26 CUSTOM CHARACTER GENERATOR

Earlier in this section, we explained the design of the character generator used by the 80-column controller. You'll also remember our mentioning that each character has a 16 X 8 matrix. We have already used these 16 bytes in 16 X 8 definition. We'd like to take that a step further.

Let's assume that you start with a normal 8 X 8 matrix. Only 8 bytes are used in character definition, with the remaining 8 bytes hiding somewhere in the background. The machine language routine below (we call it "Swapper") trades off one set of 8 bytes for the other set. This means that you can design another character set, and switch off between "standard" and your own custom characters.

```
0B00 A9 00      LDA #$00    :Store first low byte
0B02 85 4C      STA $4C     :pointer
0B04 A9 20      LDA #$20    :Store first high byte
0B06 85 4D      STA $4D     :pointer
0B08 A9 08      LDA #$08    :Store second low byte
0B0A 85 43      STA #$4E    :pointer
0B0C A9 20      LDA #$20    :Store second high byte
0B0E 85 4F      STA $4F     :pointer
0B10 A0 07      LDY #$07    :8 bytes
0B12 98         TYA
0B13 48         PHA         :on stack
0B14 A5 4C      LDA $4C     :Low byte of first pointer
```





```
0B16 A6 4D      LDX $4D     :Hi-byte of first pointer
0B18 20 8C 0B   JSR $0B8C   :PEEK subroutine
0B1B 48         PHA         :Put char. on stack
0B1C A5 4E      LDA $4E     :Lo-byte of second pointer
0B1E A6 4F      LDX $4F     :Hi-byte of second pointer
0B20 20 8C 0B   JSR $0B8C   :PEEK subroutine
0B23 A6 4C      LDX $4C     :Low byte of first pointer
0B25 A4 4D      LDY $4D     :Hi-byte of first pointer
0B27 20 65 0B   JSR $0B65   :POKE subroutine
0B2A 68         PLA         :Read character
0B2B A6 4E      LDX $4E     :Lo-byte of second pointer
0B2D A4 4F      LDY $4F     :Hi-byte of second pointer
0B2F 20 65 0B   JSR $0B65   :POKE subroutine
0B32 E6 4C      INC $4C     :Low byte 1=Low byte 1+1
0B34 D0 02      BNE $0B3E   :Still >0?
0B36 E6 4D      INC $4D     :NO--High 1=High 1+1
0B38 E6 4E      INC $4E     :Low byte 2=Low byte 2+1
0B3A D0 02      BNE $0B3E   :Still >0?
0B3C E6 4F      INC $4F     :NO--High 2=High 2+1
0B3E 68         PLA         :Back to read-in value
0B3F A8         TAY
0B40 88         DEY
0B41 10 CF      BPL $0B12   :Copy more
0B43 A5 4F      LDA $4F     :Hi-byte of second pointer
0B45 C9 40      CMP #$40    :reached $4000 yet?
0B47 B0 1B      BCS $0B64   :YES--ready
0B49 A9 08      LDA #$08    :Low=Low+ 8 char. bytes
0B4B 65 4C      ADC $4C     :added
0B4D 85 4C      STA $4C     :Store some more
0B4F A9 00      LDA #$00    :add 0
0B51 65 4D      ADC $4D     :add carry
0B53 85 4D      STA $4D     :Store some more
0B55 A9 08      LDA #$08    :Low2=Low2+8 char. bytes
0B57 65 4E      ADC $4E     :added
0B59 85 4E      STA $4E     :Store some more
0B5B A9 00      LDA #$00    :add 0
0B5D 65 4F      ADC $4F     :add carry
0B5F 85 4F      STA $4F     :Store some more
0B61 4C 10 0B   JMP $0B10   :Loop
0B64 60         RTS         :Go back to BASIC
0B65 48         PHA         :Hold char.
0B66 8A         TXA
0B67 48         PHA         :Get low byte
```

```
0B68 98         TYA
0B69 48         PHA           :Get high byte
0B6A A9 02      LDA #$02
0B6C 8D 28 0A   STA $0A28     :Set cursor flag
0B6F A2 12      LDX #$12      :VDC REG 18
0B71 68         PLA           :Put back high byte
0B72 20 80 0B   JSR $0B80     :Set register
0B75 E8         INX           :VDC REG 19
0B76 68         PLA           :Get low-byte
0B77 20 80 0B   JSR $0B80     :Set register
0B7A A2 1F      LDX #$1F      :VDC REG 31
0B7C 68         PLA           :Get byte
0B7D 4C 80 0B   JMP $0B80     :Ready; set register
0B80 8E 00 D6   STX $D600     :REG set
0B83 2C 00 D6   BIT $D600     :Wait
0B86 10 FB      BPL $0B83     :Waited long enough?
0B88 8D 01 D6   STA $D601     :Value given
0B8B 60         RTS           :Ready
0B8C 48         PHA           :Get low byte
0B8D 8A         TXA
0B8E 48         PHA           :Get high byte
0B8F A2 12      LDX #$12      :REG 18 VDC
0B91 68         PLA           :High byte in accumulator
0B92 20 80 0B   JSR $0B80     :Set register
0B95 E8         INX           :REG 19 VDC
0B96 68         PLA           :Low byte in accumulator
0B97 20 80 0B   JSR $0B80     :Set register
0B9A A2 1F      LDX #$1F      :REG 31 VDC
0B9C 8E 00 D6   STX $D600     :Set register
0B9F 2C 00 D6   BIT $D600     :Wait
0BA2 10 FB      BPL 0B9F      :Long enough?
0BA4 AD 01 D6   LDA $D601     :Read value from video RAM
0BA7 85 FE      STA $FE       :Put on
0BA9 60         RTS           :Ready
```



First Publishing Tricks and Tips for the C-128

Here's the matching BASIC loader.

```
5000    FOR X = 2816 TO 2985
5010    READ A: CS = CS + A:  POKE X,A
5020    NEXT X
5030    IF CS <> 17057 THEN PRINT CHR$ (7);:  LIST
5040    DATA 169,0,133,76,169,32,133,77,169,8,133,78
5050    DATA 169,32,133,79,160,7,152,72,165,76,166,77
5060    DATA 32,140,11,72,165,78,166,79,32,140,11,166
5070    DATA 76,164,77,32,101,11,104,166,78,164,79,32
5080    DATA 101,11,230,76,208,2,230,77,230,78,208,2
5090    DATA 230,79,104,168,136,16,207,165,79,201,64,176
5100    DATA 27,169,8,101,76,133,76,169,0,101,77,133
5110    DATA 77,169,8,101,78,133,78,169,0,101,79,133
5120    DATA 79,76,16,11,96,72,138,72,152,72,169,2
5130    DATA 141,40,10,162,18,104,32,128,11,232,104,32
5140    DATA 128,11,162,31,104,76,128,11,142,0,214,44
5150    DATA 0,214,16,251,141,1,214,96,72,138,72,162
5160    DATA 18,104,32,128,11,232,104,32,128,11,162,31
5170    DATA 142,0,214,44,0,214,16,251,173,1,214,133
5180    DATA 254,96
```

RUNning the routine by typing SYS DEC("0B00") turns the screen black, after which the cursor reappears.

The character generator will be switched in the normal manner. The second set of eight bytes per character would normally read null. So, switching the character gives you spaces to define. You can call back the original character set by calling the routine again. Now load the character editor in Chapter 1.16; you can produce new characters to your heart's content. One small change will have to be made in the BASIC program, though, at line 5065:

```
5065 AD=8192+16*A+Y+8
```

Now start the loader. Swap your character generators with the routine, and go to it. You can also have two self-defined character sets, rather than one "standard" and one "custom" (for games, etc.).

## 1.27 SYSTEM ROUTINES

Maybe you've been looking for special routines to use with 80-column mode, like instant access to video RAM, or controller initialization. Well, the built-in ROM routines for the 40-column screen aren't limited to that mode (the operating system works on both screens). It's possible, then, to use the built-in ROM routines in 80-column mode through programming.

Now, on to the routines themselves. Each routine has two addresses; the first is the jump table for the editor, while the second is the starting address of the routine proper. Which address you use is up to you.

### Screen Initialization

The following routine initializes the screen, somewhat akin to using <RUN-STOP/RESTORE>:

```
SYS 49152/SYS 49275
```





### Color Output of a Character

This was mentioned in Chapter 1.9. in connection with the 80-column screen. The routine simply prints a character of a specified color on the screen, with the position depending on the contents of locations 224 and 225:

```
SYS 49155,CHAR,COLOR
SYS 52276,CHAR,COLOR
```

CHAR: (0-255) Character in screen code (A=1,B=2,etc.)

COLOR:(0-15) Color the character should be (0=black,1=white, etc.) 80-column mode also allows values from 0 to 255. The additional 4 bits have these meanings:

| | |
|---|---|
| BIT 4: | Blinking |
| BIT 5: | Underscore |
| BIT 6: | Reverse video |
| BIT 7: | 2nd character set |

### ASCII Output

The preceding subsection mentioned the "screen code". This code is Commodore-specific, and *NOT* standard; but you can get ASCII output like this:

```
SYS 49164, ASCII code
SYS 50989, ASCII code
PRINT CHR$(ASCII code)
```

Conversely, you can find out the ASCII code by typing:

```
PRINT ASC("K")
```

which would give us the ASCII code number for K.

## PRINT AT Simulation

Basically, this routine lets you format things on screen (called PRINT AT in some BASIC versions). Here is a command which works in conjunction with the machine language call CHAR:

```
SYS 49176,A,column, row:PRINT...
SYS 52330,A,column, row:PRINT...
```

You could conceivably use this to display a command line onscreen (with warnings and system status addressed to the user). You see this a lot on professional software; now you can have it in a user-friendly atmosphere.

If you want to see just where the PRINT AT command has written an item, check these locations:

```
10 REM PRINT AT WITH RETURN
20 SP=PEEK(236):REM CURRENT COLUMN STORED
30 ZE=PEEK(235):REM CURRENT ROW STORED
40 :
50 SYS 49176,0,5,10:PRINT"COMMAND LINE"
60 :
70 SYS 49176,0,SP,ZE:REM BACK AGAIN
80 PRINT"BACK THERE"
90 END
```





All this routine needs to do is get the current cursor position from the operating system.

## Definition of Character Sets

This routine is for the 80-column mode only. If, by some chance, you don't like your custom character set, or it doesn't work very well, this routine copies the original character set into the VDC's video RAM.

```
SYS 49191 / SYS 52748
```

## 40/80-Column Toggling

We've mentioned this little trick before:

```
SYS 49194 / SYS 52526
```

For more information on these routines, please see Chapter 11.2: The Kernal.

## 1.28 HIGH-RESOLUTION GRAPHICS

The C-128 has a new BASIC (BASIC 7.0), with a host of graphic commands to make hi-res programming easier. Trouble is, the hi-res commands only operate in 40-column mode. We don't understand why you can't use them in 80-column mode; the doubled resolution (640*200 pixels) would come in handy.

The following pages will show you how to do 80-column hi-res graphics, and how to use this in your programs. If you're without an RGB monitor, you'll have to amuse yourself with standard graphic commands.

### Bit-Map Mode

Those of you former C-64 and VIC-20 owners probably remember "bit-map mode" in high-resolution programming: Essentially, it switches the screen from normal to high-resolution mode. Video RAM is no longer divided into screen memory, attribute RAM and the character generator. The computer works bit-for-bit with video RAM. In other words, for every set (on) bit, a point is written to the screen.

Our 80-column screen would give us a resolution along the lines of:

```
16000 BYTES * 8 BITS = 128,000 SCREEN POINTS
```

--which you can have either set or unset.





Switching on the bit-map mode is accomplished by registers 20 and 25:

| | |
|---|---|
| 20 | Attribute RAM starting address (HIGH) |
| 25 | Bit 7: Hi-Res On/Off |

Let's turn the bit-map on with this little program:

```
10 A=DEC("D600"):B=A+1
20 POKE A,20:POKE B,0
30 POKE A,25:POKE B,128
```

Immediately, the screen is a jumble of points and lines. There is method to this madness, though: every bit in the 16K of video RAM has been switched ON. Using the modified Poke routine from Chapter 1.10, try this:

```
SYS DEC("1800"),128,0,0
```

That turns one point on at the upper left-hand corner of the screen. Now that you have the principle, here's a program that draws a sine wave on the screen. The PLOT routine works faster than some machine code routines; it manages this through a) the modified POKE (see "POKE Simulation"), b) the modified PEEK, and c) ERASE (clearing the graphic screen).

Type this program in first and start it; it's the graphics initialization routine.

```
10   FOR X = 6144 TO 6182
20   READ A : CS = CS + A : POKE X, A
30   NEXT X
40   IF CS <> 3411 THEN PRINT "DATA ERROR IN 40"
50   DATA 72, 138,72,152,72,169,2,141,40,10,162,18
60   DATA 104,32,27,24,232,104,32,27,24,162,31,104
70   DATA 76,27,24,142,0,214,44,0,214,16,251,141
80   DATA 1,214,96
```

```
85    CS = 0
90    FOR X = 6656 TO 6697
100   READ A: CS = CS + A:  POKE X,A
110   NEXT X
120   IF CS <> 4356 THEN  PRINT "DATA ERROR IN 120"
130   DATA 72,138,72,162,18,104,32,30,26,232,104,32
140   DATA 30,26,162,31,142,0,214,44,0,214,16,251
150   DATA 173,1,214,133,254,96,142,0,214,44,0,214
160   DATA 16,251,141,1,214,96
165   CS = 0
170   FOR X = 6400 TO 6453
180   READ A: CS = CS + A:  POKE X,A
190   NEXT X
200   IF CS <> 6426 THEN  PRINT "DATA ERROR IN 200"
210   DATA 162,64,169,0,160,0,133,254,72,138,72,162
220   DATA 18,165,254,32,42,25,232,152,32,42,25,162
230   DATA 31,169,0,32,42,25,104,170,104,200,208
240   DATA 228,230,254,202,208,223,96,142,0,214,44
250   DATA  0,214,16,251,141,1,214,96
```

And now for the sine wave:

```
10   REM  80 COL SINE WAVE PLOT PROGRAM
20   A =  DEC ("D600"): B = A + 1
30   REM  PLOT:SYS DEC("1800"),BYTE,LO,HI
40   REM  ERASE:SYS DEC("1900")
50   REM  PEEK:SYS DEC("1A00"),LO,HI:
          PRINT PEEK(254);
60   :
70   POKE A,25:  POKE B,128:  POKE A,20:  POKE B,0
80   SYS  DEC ("1900")
110 FOR X = 0 TO 639
120 Y =  INT ( SIN (X / 10) * 100) + 100
130 GOSUB 170
140 NEXT X
150 END
170 REM  PLOT
180 AN = 80 * Y
190 Z1 =  INT (X / 8)
200 AD = AN + Z1: HI =INT (AD/256):LO =AD-256 * HI
210 SYS  DEC ("1A00"),LO,HI
220 PE =  PEEK (254) OR 2 ^ (7 - (X - Z1 * 8))
```





```
230 SYS  DEC ("1800"),PE,LO,HI
240 RETURN
```

The modified POKE has been described previously. Here is the ERASE routine:

```
1900 A2 40     LDX #$40    :Clear 64 pages
1902 A9 00     LDA #$00    :from $0000
1904 A0 00     LDY #$00
1906 85 FE     STA $FE     :Store high byte
1908 48        PHA         :and retrieve
1909 8A        TXA
190A 48        PHA         :Retrieve pages
190B A2 12     LDX #$12    :VDC REG 18
190D A5 FE     LDA $FE     :Load high-byte
190F 20 2A 19  JSR $192A   :Set register
1912 E8        INX         :VDC REG 19
1913 98        TYA         :Low byte into accumulator
1914 20 2A 19  JSR $192A   :Set register
1917 A2 1F     LDX #$1F    :VDC REG 31
1919 A9 00     LDA #$00    :Write 0 into video RAM
191B 20 2A 19  JSR $192A   :Set register
191E 68        PLA         :Get high byte
191F AA        TAX         :in X-register
1920 68        PLA         :Get pages
1921 C8        INY
1922 D0 E4     BNE $1908   :Loop
1924 E6 FE     INC $FE     :High=High+1
1926 CA        DEX         :Page=Page-1
1927 D0 DF     BNE $1908   :Still >0?  Keep going.
1929 60        RTS         :Return to BASIC
192A 8E 00 D6  STX $D600   :Set register
192D 2C 00 D6  BIT $D600   :Wait
1930 10 FB     BPL $192D   :Waited long enough?
1932 80 01 D6  STA $D601   :Value given
1935 60        RTS         :Return
```

These graphic commands aren't exactly the fastest. It won't be long, though, before someone brings out an 80-column graphic extension.

## 1.29 CHARACTER GENERATORS -- AGAIN

We close with a short program that will perform a headstand -- literally. We suggest that you review the POKE routine ($1800) and PEEK routine ($1A00) in the previous chapters.

Here's one for the normal 8 X 8 matrix:

```
10 FOR X = 0 TO 511
20 FOR X2 = 0 TO 3
30 A1 =8199+8*X-X2: H1= INT(A1/256): L1= A1-256*H1
40 SYS  DEC ("1A00"),L1,H1
50 W1 =  PEEK ( DEC ("FE"))
60 A2 =8192+8*X+X2: H2 =INT(A2/256): L2 =A2-256*H2
70 SYS  DEC ("1A00"),L2,H2
80 W2 =  PEEK ( DEC ("FE"))
90 SYS  DEC ("1800"),W2,L1,H1
100 SYS  DEC ("1800"),W1,L2,H2
110 NEXT X2
120 NEXT X
```

And one for the 16 X 8 matrix (you must have previously defined a 16X8 matrix to see the results of this program):

```
10  FOR X = 0 TO 511
20  FOR X2 = 0 TO 7
30  A1 =8207+16*X-X2: H1=INT(A1/256):L1=A1-256*H1
40  SYS  DEC ("1A00"),L1,H1
50  W1 =  PEEK ( DEC ("FE"))
60  A2=8192+16*X+X2: H2=INT(A2/256):L2=A2-256*H2
70  SYS  DEC ("1A00"),L2,H2
80  W2 =  PEEK ( DEC ("FE"))
90  SYS  DEC ("1800"),W2,L1,H1
100 SYS  DEC ("1800"),W1,L2,H2
110 NEXT X2
120 NEXT X
```





# CHAPTER 2

# BASIC 7.0 GRAPHICS COMMANDS

## 2.1 THE CIRCLE COMMAND

CIRCLE is one of the most versatile commands in BASIC 7.0. As its name suggests, it can draw circles. But it can also draw lines, triangles, rectangles, ellipses and other geometric shapes. The command uses the following format:

```
CIRCLE clr,x,y,xr,xy,sa,ea,r,i
```

These parameters are defined as follows::

| | |
|---|---|
| `clr` | Number of color memory (0-3) |
| `x,y` | Coordinates for center point |
| `xr` | Radius in x-direction |
| `yr` | Radius in y-direction |
| `sa` | Starting angle of the circle |
| `ea` | End angle of the circle |
| `r` | Angle for rotation |
| `i` | Angle for drawn circle segments |

To draw a circle use the first four parameters; the rest are for finer details, as we shall soon see. The first value gives color memory; the next two indicate midpoint coordinates; and the next, the radius.

Drawing an ellipse requires the previous coordinates, plus the Y-register radius (if unequal to the X-register radius):

```
CIRCLE 0,160,100,10,30
```

**If you wish to turn the ellipse at the midpoint, you'll have to change the last** value. For example:

```
CIRCLE 0,160,100,10,30,0,360,45
```

These parameters give the number of degrees drawn of the ellipse/circle. The sixth and seventh values tell at which angles the circle/ellipse begins and ends. We can get a half-circle by doing this:

```
CIRCLE 0,160,100,30,30,0,180
```

How can we get squares out of this command? The last parameter performs that function, using these numbers:

| | |
|---|---|
| 0-44 | Circle (higher the number, the "rounder") |
| 45 | Octagon |
| 60 | Hexagon |
| 75 | Pentagon |
| 90 | Square & Rectangle |
| 91-119 | Unequal rectangle |
| 120 | Equilateral triangle |
| 121-179 | Triangle (higher the value, the more unequal) |
| 180-255 | Lines |





## 2.2 PIE CHARTS

BASIC 7.0's graphic commands offer a lot to the user. One of the little extensions of the CIRCLE command that we've written is a fascinating one: **PIE CHARTS. You can imagine how useful this program can be for** calculation programs, statistics and the like. Naturally, our routine isn't as good as something "store-bought"; we'll leave it to you to improve it....

Now on to the program. One problem that cropped up was the fact that the color memory in hi-res mode isn't the same size as the graphic memory. So, we had to color in an entire field with 8 X 8 points. We could have used multicolor mode instead, but then we would have ended up with a pie with sharp edges.

These problems only occur in every other segment. You can turn segment colors off altogether (delete 340-390) to avoid some of these difficulties.

```
10   REM  PIE CHARTS
20   GRAPHIC 0,1
30   INPUT "SEGMENT SIZE";N
35   IF N = 0 THEN  END
40   DIM A(N)
45   DIM P(N)
50   DIM T$(N)
60   FOR I = 1 TO N
70   PRINT I". SEGMENT"
80   INPUT "VALUE";A(I)
90   P = P + A(I)
100  INPUT "TEXT ";T$(I)
110  NEXT I
120  INPUT "ANY CHANGES (Y/N) ";A$
130  IF  LEFT$ (A$,1) = "Y" THEN BEGIN
140  FOR I = 1 TO N
150  PRINT I".  "T$(I),A(I)
```

```
160   NEXT I
170   INPUT "SEGMENT NUMBER ";I
180   P = P - A(I)
190   INPUT "VALUE";A(I)
200   P = P + A(I)
210   INPUT "TEXT ";T$(I)
220   GOTO 120
230   BEND
240   REM  DRAW PIE CHART
250   GRAPHIC 1,1
260   COLOR 1,1
270   CIRCLE ,160,100,80
280   G = 0
290   FOR I = 1 TO N
300   P(I) = A(I) / P * 100
310   G = G + P(I)
320   CIRCLE ,160,100,80,0,0,80,G * 3.6
330   IF I / 2 < > INT (I / 2) THEN 520
340   REM  DRAW SEGMENTS
350   COLOR 1,I / 2 + .5
360   CIRCLE ,160,100,80,0,0,40,G * 3.6-P(I) * 1.8
370   X =  RDOT (0)
380   Y =  RDOT (1)
390   PAINT ,X + 2,Y + 1
400   NEXT I
410   REM  TEXT PRINT
420   G = 0
430   FOR I = 1 TO N
440   G = G + P(I)
450   COLOR 1,I / 2 + .5
460   CIRCLE ,160,100,80,0,89,90,G * 3.6-P(I) * 1.8
470   COLOR 1,1
480   X =  RDOT (0) / 8
490   Y =  RDOT (1) / 8
500   IF X < 16 THEN X = X -  LEN (T$(I))
510   CHAR ,X,Y,T$(I)+STR$(INT(10*P(I) + .5) / 10)
520   NEXT I
525   CHAR ,14,24,"'SPACE' TO CONT."
530   GET  KEY A$
540   GRAPHIC 0
550   PRINT "{RVS ON}N{RVS OFF}EW PIE CHART"
555   PRINT "{RVS ON}O{RVS OFF}LD PIE CHART"
556   PRINT "'SPACE' TO END"
```





```
560   GET  KEY A$
570   IF A$ = "N" THEN  RUN
580   IF A$ = "O" THEN 120
```

The input of different portions results with absolute values and not percentages; every portion can include a text string of your choice (as long as it's not too large a string).

## 2.3 BAR GRAPHS

Let's go on to another graphic aid for calculation programs: Bar graphs. Our programs are limited to 8 "blocks" per character block (it loses something on the 40- character screen). Block length is up to you; and the highest value would be 100 (as in percent). You'll note that we've taken great advantage of the vertical resolution (you'll find the calculations in line 280).

```
10    REM  BAR GRAPHS
20    GRAPHIC 0
30    SCNCLR
40    DIM A(8)
50    DIM F(8)
60    INPUT "GRAPH NAME";U$
70    U$ = "          " + U$
80    FOR I =  LEN (U$) TO 39
90    U$ = U$ + " "
100   NEXT I
110   INPUT "NUMBER OF BARS (1-8)";D
120   IF D < 1 OR D > 8 THEN 110
130   FOR I = 1 TO D
140   PRINT "BAR #"I
150   INPUT "HOW MUCH";A(I)
160   IF A(I) > MAX THEN MAX = A(I)
170   INPUT "COLOR (1-16)";F(I)
180   IF F(I) < 1 OR F(I) > 16 THEN 170
```

```
190    NEXT I
200    PRINT "USE THIS DATA?"
210    GET  KEY A$
220    IF A$ = "N" THEN  RUN
230    GRAPHIC 1
240    SCNCLR
250    COLOR 1,1
260    CHAR ,0,0,U$,1
270    FOR I = 1 TO D
280    A(I) = 190 - A(I) / MAX * 170
290    COLOR 1,F(I)
300    BOX ,I * 38 - 24,A(I),I * 38,190,0,1
310    DRAW ,I*38-24,A(I) TO I*38-18,A(I)-6 TO I*38
       + 6,A(I)-6 TO I*38 + 6,184 TO I*38,190
320    DRAW ,I * 38,A(I) TO I * 38 + 6,A(I) - 6
330    NEXT I
340    GET  KEY A$
350    GRAPHIC 0
360    SCNCLR
370    PRINT "(O) OLD GRAPH"
380    PRINT "(N) NEW GRAPH"
390    PRINT "(X) EXIT"
400    GET  KEY A$
410    IF A$ = "X" THEN  END
420    IF A$ = "O" THEN  GRAPHIC 1:  GOTO 340
430    IF A$ = "N" THEN  RUN
440    GOTO 370
```





## 2.4 FUNCTION PLOTTER

Who hasn't wished for smooth arcs? Maximum - minimum curve representations? The endless mysteries of a tangent? This program is a complete curve plotter. It takes your input, and plots it:

```
10    REM  FUNCTION PLOTTER
20    GRAPHIC 0,1
30    DEF  FN Y(X) =  COS (X)
40    INPUT "BEGINNING FUNCTION RANGE ";A
50    INPUT "ENDING FUNCTION RANGE  ";E
60    IF A =  > E THEN 40
70    INPUT "REGISTER  X-VALUE ";A1
80    INPUT "REGISTER  Y-VALUE ";E1
90    IF A1 =  > E1 THEN 70
100   S = (E - A) / 320
110   S2 = (E1 - A1)
120   GRAPHIC 1,1
130   FOR I = 0 TO 319
140   A = A + S
150   X = 200 - ( FN Y(A) - A1) / S2 * 199
160   IF X =  < 199 AND X =  > 0 THEN  DRAW 1,I,X
170   NEXT I
180   GET  KEY A$
190   GRAPHIC 0
```

After RUNning the program, it prompts you for four values: the range in which the function will begin and end (be sure it's only as big as your screen); the X-register (-5 to 9); and the Y-register (0 to 9).

If you wish to explore other functions, change line 30. To get a sine wave, do this:

```
30 DEF FN Y(X)= SIN(X)
```

This will not calculate in degrees, but in radians (360 degrees are exactly 2 * PI radians). Two ground rules: X- value must be no less than 0 and no more than 3.1415; and the Y-value can only be between -1 and 1.

## 2.5 WINDOWS

Windows are the newest buzzword in computerese. No new computer, no new BASIC, is without this feature; and the C-128 is no exception. Unfortunately, you only get one window on the C-128; but it is possible to get multiple windows with a little finagling.

### 2.5.1 HOW TO DO WINDOWS

You've learned from your handbook that a window can be set up in direct mode; no problem there, but how do you find it? The borders of the window can be found by moving the cursor around. The simplest method to get out of the window space is to use <RUN-STOP/RESTORE>, which dumps the window. These methods have the disadvantage of stopping a running program. We have an answer. Isn't a window just a small screen, and a screen an enlarged window? Well, we could conceivably draw the window to fit the screen:

```
WINDOW 0,0,79,24 (80-col. screen)
WINDOW 0,0,39,24 (40-col. screen)
```





### 2.5.2 READING WINDOW COORDINATES

You have already heard of the command RWINDOW. You can determine the row (RWINDOW(0)), column (RWINDOW(1)) and character mode (RWINDOW(2)). If you want complete coordinates, you'll need to handle the matter a bit differently. Zero page memory has four extra bytes in which the coordinates for the current window are stored; these are addresses 228-231:

```
10 WINDOW 1,11,20,22
20 PRINT "BOTTOM BORDER:";PEEK(228)
30 PRINT "TOP    BORDER:";PEEK(229)
40 PRINT "LEFT   BORDER:";PEEK(230)
50 PRINT "RIGHT  BORDER:";PEEK(231)
```

To experiment with this program, hit RUN-STOP/RESTORE and change the window parameters.

### 2.5.3 SETTING UP ALTERNATE WINDOWS

We mentioned earlier that the screen can only have one window, as indicated by locations 228 ($E4) to 231 ($E7). Here are the default values:

```
228 ($E4)   :24
229 ($E5)   : 0
230 ($E6)   : 0
231 ($E7)   :79 (80-col. mode)
            :39 (40-col. mode)
```

Now, if you type in WINDOW 1,2,3,14 -- those values will change to:

```
228 ($E4)   :14
229 ($E5)   : 2
230 ($E6)   : 1
231 ($E7)   : 3
```

(regardless of screen size)

You can manage machine language programming of windows, just by altering 228 to 231 decimal. POKE 228,10, for example, sets the bottom at 10.

**A word of warning:** You will run into problems if you try making windows larger than the screen.

## 2.5.4 VERTICAL SCROLLING

The WINDOW command allows you to produce vertical scrolling without hassles. The program looks like this:

```
10 INPUT "TEXT";A$
20 INPUT"SPEED";G
30 WINDOW 10,10,10,20
40 FOR I=1TO LEN(A$)
50 PRINT MID$(A$,I,1)
60 FOR T=1 TO G
70 NEXT T
80 NEXT I
90 GOTO 40
```





The window width is reduced to one, so the characters scroll beneath one another. Unfortunately, this system doesn't work with an pre-established window.

## 2.5.5 THE WINDOW AS INPUT LINE

This program makes it easy to limit the user's access to the cursor keys, and keep the user in the confines of an input line.

```
 5 REM F = 39 (40 COL.) F = 79 (80 COL.)
10 REM INPUT LINE
20 PRINT CHR$(27);"M":REM scrolling stopped
40 PRINT"NAME?";
50 OPEN 1,0
60 WINDOW PEEK(236),PEEK(235),F,PEEK(235),1
70 INPUT#1,A$
80 CLOSE 1
90 WINDOW 0,0,F,24
```

NOTE: The constant F (lines 60 & 90) should be replaced with 39 (40-col. screen) or 79 (80-col.).

Address 236 has the present column, and 235 the current line.

## 2.5.6 PRINT AT WITH WINDOWS

Many books and magazines have written about simulating PRINT AT on the C-64. The C-128 has a command called CHAR, used to simulate PRINT AT. But this isn't always the best method -- for example, you can't control the length of the output. Also, there might be control characters in the string to be printed. If you use the WINDOW, you'll come out ahead:

```
 5 REM F = 39 (40 COL.) F = 79 (80 COL.)
10 REM PRINT AT WITH WINDOWS
20 INPUT"ROW";RW
30 INPUT"COLUMN";CL
40 INPUT"LENGTH";LN
50 F=PEEK(231)
55 PRINT CHR$(27);"M":REM SCROLLING STOPPED
60 WINDOW CL,RW,CL+LN,LINE
65 PRINT"HELLO"
70 WINDOW 0,0,F,24
```

The variable F again represents the screen width. The system automatically figures the mode out at line 50.

## 2.5.7 CLEARING A PARTIAL SCREEN

One part of the WINDOW command we haven't touched on is its ability to clear window contents. You have your choice of two methods:

A. `10 WINDOW 10,10,20,20:PRINT"(CLR/HOME)"`

B. `10 WINDOW 10,10,20,20,1`





The uses are clear for this. We may want to erase the window, so that we can use the screen for other things. On the other hand, you may want to just clear the lower half of the 40-column screen:

```
10 WINDOW 0, 12, 39, 24, 1
20 WINDOW 0,  0, 39, 24(, 0)
```

You can, of course, use these commands in direct mode. Just type it in one line, separating the two commands with a colon (:).

## 2.5.8 SECURING WINDOW CONTENTS

Let's say you're in the middle of a word processing program. You've filled the entire screen, and now you want to save the text. You go to the main menu, which appears in a screen window. Good software will still retain the text under the window, and even let you go back to the text, erasing the window. The WINDOW command on the 128 doesn't do this.

Now, once the window is cleared, the contents are lost forever. What to do? We can make the window simulate a 40-column screen:

```
10   REM  SAVE SCREEN WHILE USING WINDOWS
20   GRAPHIC 0,1
30   REM  WRITE SOMETHING ON SCREEN
40   FOR I = 1024 TO 2024
50   POKE I,J
60   J = J + 1
70   IF J > 255 THEN J = 0
80   NEXT I
90   X1 = 5:  REM  DEFINE WINDOW
```

```
100   X2 = 12
110   X3 = 35
120   X4 = 22
125   REM  SAVE CONTENTS UNDER WINDOW
130   GOSUB 60000
140   REM  USE WINDOW
150   WINDOW X1,X2,X3,X4,1
160   INPUT "TEXT";A$
170   REM  RETURN SCREEN TO NORMAL
180   WINDOW 0,0,39,24,0
190   GOSUB 60090:  REM  RECALL INFO
200   GET  KEY B$
210   GRAPHIC 0,1
220   END
60000  REM  SAVE SCREEN UNDER WINDOW
60010  DIM X(350)
60020  FOR I = X2 TO X4
60030  FOR J = X1 TO X3
60040  X(Z)  =  PEEK (I * 40 + J + 1024)
60050  Z = Z + 1
60060  NEXT J
60070  NEXT I
60080  RETURN
60090  REM  RECALL SCREEN
60100  Z = 0
60110  FOR I = X2 TO X4
60120  FOR J = X1 TO X3
60130  POKE I * 40 + J + 1024,X(Z)
60140  Z = Z + 1
60150  NEXT J
60160  NEXT I
60170  RETURN
```

The interesting part of this program starts at line 60000. Line 60010 defines an array called X(). The array size is dependent on the size of the windows. If you use this routine in your own programs, make sure no other arrays or variables exist with this name. If you give the coordinates for one window, you can put this value directly in the loop instead of using variables. If you use several windows you should use variables.



First Publishing Tricks and Tips for the C-128

Like all BASIC programs, this one has a small disadvantage: It's too slow. Here's a second version, in machine language:

```
1400 A5 E7     LDA $E7      :Right window border
1402 18        CLC
1403 E5 F6     SBC $E6      :Minus left border
1405 85 FF     STA $FF      :=length of window
1407 E6 FF     INC $FF      :Length=length+1
1409 A9 04     LDA #$04     :Screen start
140B 85 FC     STA $FC      :store
140D A2 00     LDX #$00
140F 8A        TXA
1410 E4 E5     CPX $E5      :$E5 = 1st window line?
1412 F0 0B     BEQ $141F    :YES--then $141F
1414 E8        INX
1415 18        CLC
1416 69 28     ADC #$28     :Next line
1418 90 F6     BCC $1410    :Still not all
141A E6 FC     INC $FC      :Raise high-byte pointer
141C 4C 10 14  JMP $1410
141F 18        CLC
1420 65 E6     ADC $E6      :Window start - low
1422 90 02     BCC $1426
1424 E6 FC     INC $FC      :Raise high-byte by 1
1426 85 FB     STA $FB      :Store low-byte
1428 85 FD     STA $FD      :in $FB and $FD
142A A5 FC     LDA $FC      :High-byte
142C 69 F7     ADC #$11     :17 there
142E 85 FE     STA $FE      :and in high-byte of the
1430 A0 FF     LDY #$FF     :2nd counter
1432 C8        INY
1433 AD 6B 14  LDA $146B    :Read or write?
1436 D0 1A     BNE $1452    :Write
1438 B1 FB     LDA($FB),Y   :Char. on screen
143A 91 FD     STA($FD),Y   :Store
143C C4 FF     CPY $FF      :Y-reg = length?
143E D0 F2     BNE $1432    :UNEQUAL--continue
1440 E8        INX
1441 E4 E4     CPX $E4      :X-reg = bottom edge?
1443 90 14     BCC $1459    :smaller
1445 AD 6B 14  LDA $146B    :Written or read?
1448 D0 04     BNE $144E    :Written, read next
```

```
144A EE 6B 14 INC $146B
144D 60       RTS          :Back to BASIC
144E CE 6B 14 DEC $146B
1451 60       RTS          :Return to BASIC
1452 B1 FD    LDA($FD),Y   :Char. from memory
1454 91 FB    STA($FB),Y   :written to screen
1456 4C 3C 14 JMP $143C
1459 A5 FB    LDA $FB      :Low-byte in accumulator
145B 18       CLC
145C 69 28    ADC #$28     :Next line
145E 90 04    BCC $1464    :No overflow
1460 E6 FC    INC $FC      :Raise high-byte by 1
1462 E6 FE    INC $FE      :Raise high-byte of 2nd
                            counter by 1
1464 85 FB    STA $FB      :Low-byte in 1st counter
1466 85 FD    STA $FD      :Low-byte in 2nd counter
1468 4C 30 14 JMP $1430
146B 00       BRK          :Byte for read  (0)
                            or write       (1)
```

Here's the BASIC loader for the window saving routine. A sample program has been included with the BASIC loader to demonstrate the speed of the routine.

```
0    GOTO 180
10   REM  TEST PROGRAM
20   GRAPHIC 0
30   FOR I = 1024 TO 2023
40   POKE I,A
50   A = A + 1
60   IF A > 255 THEN A = 0
70   NEXT I
80   WINDOW 10,10,20,20
90   SYS 5120
100   PRINT "{CLR HOME}"
110   GET  KEY A$
120   PRINT A$;
130   IF A$ <  >  CHR$ (13) THEN 110
140   SYS 5120
150   WINDOW 0,0,39,24
160   GRAPHIC 0
170   END
```





```
180    REM  ROUTINE SAVE WINDOW
190    FOR I = 5120 TO 5227
200    READ A
210    S = S + A
220    POKE I,A
230    NEXT I
240    IF S <  > 16107 THEN BEGIN
250    PRINT "?ERROR IN DATA"
260    END
270    BEND
280    GOTO 10
290    DATA 165,231,24,229,230,133,255,230
300    DATA 255,169,4,133,252,162,0,138
310    DATA 228,229,240,11,232,24,105,40
320    DATA 144,246,230,252,76,16,20,24
330    DATA 101,230,144,2,230,252,133,251
340    DATA 133,253,165,252,105,17,133,254
350    DATA 160,255,200,173,107,20,208,26
360    DATA 177,251,145,253,196,255,208,242
370    DATA 228,228,232,144,20,173,107,20
380    DATA 208,4,238,107,20,96,206,107
390    DATA 20,96,177,253,145,251,76,60
400    DATA 20,165,251,24,105,40,144,4
410    DATA 230,252,230,254,133,251,133,253
420    DATA 76,48,20,0
```

The routine itself begins at line 180. The first section contains the sample program. The window in which you to read and write text must always be active.

## 2.5.9 SIMULATING SEVERAL WINDOWS

We can simulate a number of windows with the help of the BASIC program in Chapter 2.5.8:

```
10     REM  SAVE WINDOW CONTENTS
15     DIM X(40,30)
20     GRAPHIC 0,1
30     REM  FILL SCREEN WITH TEXT
40     FOR I = 1024 TO 2024
50     POKE I,J
60     J = J + 1
70     IF J > 255 THEN J = 0
80     NEXT I
85     REM  PUT 40 WINDOWS ON SCREEN
90     FOR K = 0 TO 39
100    X1 =  INT ( RND (1) * 25) + 5
110    X2 =  INT ( RND (1) * 14) + 3
115    REM  SAVE CONTENTS UNDER WINDOW
120    GOSUB 60000
130    WINDOW X1,X2,X1 + 4,X2 + 3
140    FOR W = 1 TO 20:  REM  PRINT #'S IN WINDOW
150    PRINT  CHR$ (K + 48);
160    NEXT W
165    NEXT K
170    REM  TAKE OFF WINDOWS ON SCREEN
180    FOR K = 39 TO 0 STEP  - 1
190    X1 = X(K,29)
200    X2 = X(K,30)
210    WINDOW X1,X2,X1 + 4,X2 + 3
215    REM  REPLACE CONTENTS
220    GOSUB 60090
230    NEXT K
240    GET  KEY A$
250    GRAPHIC 0,1
260    END
60000 REM  MEMORIZE CONTENTS UNDER WINDOW
60010 Z = 0
60020 FOR I = X2 TO X2 + 3
```





```
60030 FOR J = X1 TO X1 + 4
60040 X(K,Z) =  PEEK (I * 40 + J + 1024)
60050 Z = Z + 1
60060 NEXT J
60070 NEXT I
60074 X(K,29) = X1
60076 X(K,30) = X2
60080 RETURN
60090 REM  RECALL CONTENTS
60100 Z = 0
60110 FOR I = X2 TO X2 + 3
60120 FOR J = X1 TO X1 + 4
60130 POKE I * 40 + J + 1024,X(K,Z)
60140 Z = Z + 1
60150 NEXT J
60160 NEXT I
60170 RETURN
```

First, a screen is displayed, then 40 windows are produced, then the windows are cleared. The final result is the starting screen.

This program shows you how to use multiple windows. Using multiple windows in your own programs entails writing individual routines for each window.

The program works with 40 windows, the largest number possible in C-128 variable memory. The array X() uses 40 * 1004 bytes (40 refers to the number of windows, of course). If you use fewer in your program, decrease the number proportionately.

It's possible to extend a window over the entire screen. This would require 1000 bytes for screen contents, and four for window coordinates, resulting in 1004. Again, a smaller window requires a smaller number.

## 2.6 SPRITE HANDLING

Another indicator of BASIC 7.0's versatility is its sprite handling commands. The sprite generating and editing features make sprite work very easy (as opposed to the C-64, where sprite handling is made very difficult with BASIC 2.0).

It goes without saying that games make up the majority of sprite applications. You can also use sprites instead of regular characters (see Chapter 3.5). The next program was designed for just that; it copies the character of your choice into the sprite editor, where you can make your own lettering (eight sprites are available).

```
5    REM  2.6A
10   REM  CHAR. COPIER TO SPRITE
20   INPUT "WHICH SPRITE";S
30   IF S < 1 OR S > 8 THEN 20
40   INPUT "COL  (0-13) ";Z
50   IF Z < 0 OR Z > 13 THEN 40
60   INPUT "ROW    (0-2) ";Y
70   IF Y < 0 OR Y > 2 THEN 60
80   REM  ERASE SPRITE
90   FOR I = 0 TO 62
100  B$ = B$ +  CHR$ (0)
110  NEXT I
120  SPRSAV B$,S
130  IF Z = 0 AND Y = 0 THEN 170
140  FOR I = 1 TO Z * 3 + Y
150  A$ = A$ +  CHR$ (0)
160  NEXT I
170  INPUT "INPUT CHAR. CODE ";C
180  FOR I = C * 8 TO C * 8 + 7
190  A = 0
200  FOR J = 0 TO 7
210  BANK 14
220  F =  PEEK (53248 + I)
```





```
230  IF (F AND 2 ^ J) = 2 ^ J THEN A = A + 2 ^ J
240  NEXT J
250  A$ = A$ +  CHR$ (A) +  CHR$ (0) +  CHR$ (0)
260  NEXT I
270  SPRSAV A$,S
280  REM  SPRITE IMAGE GENERATOR
290  POKE 842,48 + S
300  POKE 843,13
310  POKE 208,2
320  SPRDEF
```

You need to enter the screen code of the character you want. After you've done that, the system copies the character into the SPRDEF mode, where you have sprite commands at your fingertips. Then again, there isn't a whole lot you can do with a character built within an 8X8 matrix; the majority of the sprite field is left unused. The second program doubles the size of the sprites, allowing you to use multicolor mode, and, for instance, make a sprite with a shadow trailing behind it.

```
10   REM  DOUBLE SIZE CHAR COPY TO SPRITE
20   INPUT "WHICH SPRITE";T
30   IF T < 1 OR T > 8 THEN 20
40   INPUT "COL.  (0-5) ";Z
50   IF Z < 0 OR Z > 5 THEN 40
60   INPUT "ROW    (0-1) ";Y
70   IF Y < 0 OR Y > 1 THEN 60
80   REM  SPRITE ERASE
90   FOR I = 0 TO 62
100  B$ = B$ +  CHR$ (0)
110  NEXT I
120  SPRSAV B$,T
130  IF Z = 0 AND Y = 0 THEN 170
140  FOR I = 1 TO Z * 3 + Y
150  A$ = A$ +  CHR$ (0)
160  NEXT I
170  INPUT "INPUT CHAR CODE   ";S
175  REM  CHAR COPIER
180  FOR I = S * 8 TO S * 8 + 7
190  A(0) = 0
```

```
200   A(1) = 0
210   FOR J = 0 TO 3
220   FOR Q = 0 TO 1
230   BANK 14
240   F =  PEEK (53248 + I)
250   IF (F AND 2 ^ (J + Q * 3)) = 2 ^ (J + Q * 3)
      THEN A(Q) =A(Q) + 2 ^ (J * 2) + 2 ^ (J * 2+1)
260   NEXT Q
270   NEXT J
280   A$ = A$ + CHR$ (A(1)) + CHR$ (A(0)) +CHR$ (0)
290   A$ = A$ + CHR$ (A(1)) + CHR$ (A(0)) +CHR$ (0)
300   NEXT I
310   SPRSAV A$,T
320   REM  ACTIVATE SPRITE IMAGE GENERATOR
330   POKE 842,48 + T
340   POKE 843,13
350   POKE 208,2
360   SPRDEF
```

## 2.6.1 DESIGN IN LISTING

One thing is a little puzzling about sprites: where are the sprite contents going to be stored? We've designed the two following programs to fix that problem. The first program reads DATA statements to form a sprite. That is, there's no DATA per se, but rather strings, they are faster and simpler for the computer to handle, and changes are easily made on these sprites. Like most of the BASIC listings, this one, too, is slow; but it beats buying a program.

```
10    REM  DESIGN IMAGE LISTING 1
20    INPUT "HOW MANY SPRITES";S
30    POKE 53296,1 : REM SET FAST MODE
40    FOR T = 1 TO S:  PRINT "READING IN SPRITE #"T
50    A$ = ""
```





```
60    FOR G = 0 TO 20
70    READ B$
80    IF  LEN (B$) < 24 THEN B$ = B$ + ".": GOTO 80
90    FOR I = 0 TO 2
100   A = 0
110   FOR J = 0 TO 7
120   IF MID$ (B$,I*8+J+1,1) ="*" THEN A =A+2^(7-J)
130   NEXT J
140   A$ = A$ +  CHR$ (A)
150   NEXT I
160   NEXT G
170   SPRSAV A$,T
180   SPRITE T,1,T,1,1,1,0
190   MOVSPR T,T * 10#T
200   NEXT T
210   POKE 53296,0
1000    REM  012345678901234567890123
1010    DATA *********
1020    DATA *********
1030    DATA .******
1040    DATA ..******
1050    DATA ...******
1060    DATA ....******
1070    DATA ....************
1080    DATA ....**************
1090    DATA .....****************
1100    DATA ......****************
1110    DATA .......*****************
1120    DATA ......****************
1130    DATA .....****************
1140    DATA ....**************
1150    DATA ....************
1160    DATA ....******
1170    DATA ...******
1180    DATA ..******
1190    DATA .******
1200    DATA ********
1210    DATA ********
1220    REM  012345678901234567890123
```

As you can see from the listing, a set bit in a sprite is shown as an asterisk, and an unset bit a period.

While the data is being read in, the clock frequency is doubled (FAST), which increases execution time, but turns off the 40-column screen. When the routine finishes, the screen returns to normal, and a sprite is displayed on the screen.

Not every program uses this method of reading sprites as DATA statements, so we wrote the second program in this chapter, which automatically cranks out DATA statements.

```
10    REM  DESIGN IMAGE LISTING 2
20    INPUT "HOW MANY SPRITES";S
30    FOR T = 1 TO S
40    FOR G = 0 TO 62 STEP 3
50    B$ = ""
60    FOR I = 0 TO 2
70    FOR J = 7 TO 0 STEP  - 1
80    IF (PEEK(3520+G+I+T*64) AND 2^J) = 2^J THEN
      B$ = B$ + "*":  ELSE B$ = B$ + "."
90    NEXT J
100   NEXT I
110   PRINT  STR$ (10000 + G + 100 * T);" DATA";B$
120   PRINT "GOTO 170{CRSR UP}{CRSR UP}{CRSR UP}";:
      REM  3*CURSOR ^
130   POKE 842,13
140   POKE 843,13
150   POKE 208,2
160   END
170   NEXT G
180   NEXT T
190   REM  DATA AT 10100-
```

Once the routine is done, remove the old DATA lines with the DELETE command.





## 2.6.2 COMFORTABLE SPRITE EDITING

After defining a sprite, you might like to alter it. For example, you've designed a spaceship, and you discover that it wasn't that great a design. Instead of redoing it completely, just run it through this program. Ah, but there's more -- it can also expand the sprite in two axis *AND* rotate the sprite.

```
10    REM  *************** SPRITE-HANDLING
20    DIM B(600)
30    DIM A(62)
40    REM  *************** MENU
50    INPUT "COLOR 1";F
60    INPUT "COLOR 2";D
70    INPUT "COLOR 3";E
80    SPRCOLOR D,E
90    PRINT "1 :  VERTICAL MIRROR"
100   PRINT "2 :  HORIZONTAL MIRROR"
110   PRINT "3 :  ROTATE 180 DEG."
120   PRINT "4 :  DISPLAY SPRITE"
130   PRINT "5 :  INVERT"
140   PRINT "6 :  ROTATE 90 DEG."
150   PRINT "7 :  ROTATE 270 DEG."
160   INPUT "COMMAND";B
165   IF B = 0 THEN  END
170   IF B < 1 OR B > 7 THEN  GOTO 10
180   INPUT "SPRITE-NUMBER ";S
190   IF S < 1 OR S > 9 THEN  GOTO 180
200   IF B<5 THEN INPUT"MULTICOLOR (ON=1/OFF=0) ";M
210   :  ELSE M = 0
220   POKE 53296,1
230   IF B > 3 THEN 270
240   FOR I = 0 TO 62
250   A(I) =  PEEK (3520 + S * 64 + I)
260   NEXT I
270   ON B GOSUB 360,430,560,420,600,830,650
280   REM  ******* SPRITE VERTICAL MIRROR
290   POKE 53296,0
```

```
300   GRAPHIC 0,1
310   SPRITE S,1,F,1,1,1,M
320   MOVSPR S,100,100
330   GET  KEY A$
340   SPRITE S,0
350   GOTO 90
360   REM  ************ VERTICAL MIRROR
370   FOR I = 0 TO 60 STEP 3
380   FOR J = 0 TO 2
390   POKE 3520 + S * 64 + I + J,A(60 - I + J)
400   NEXT J
410   NEXT I
420   RETURN
430   REM  ************ HORIZONTAL MIRROR
440   FOR I = 0 TO 60 STEP 3
450   FOR J = 0 TO 2
460   W = 0
470   FOR X = M TO 7 STEP M + 1
480   IF (A(I+2-J) AND 2^X) =2^X THEN W = W+2^(7-X)
490   IF M =1 AND (A(I +2-J) AND 2^(X-1)) = 2^(X-1)
      THEN W = W+2^(7-(X-1))
500   NEXT X
510   A(I + 2 - J) = W
520   POKE 3520 + S * 64 + I + J,A(I + 2 - J)
530   NEXT J
540   NEXT I
550   RETURN
560   REM  ********* ROTATE 180 DEG.
570   GOSUB 360
580   B = 2
590   GOTO 240
600   REM  ************** INVERT
610   FOR I = 0 TO 62
620   POKE3520+S*64+I,255AND(-PEEK(3520+S*64+I)-1)
630   NEXT I
640   RETURN
650   REM  ********* ROTATE 270 DEG.
660   FOR I = 0 TO 20
670   FOR J = 0 TO 2
680   FOR G = 0 TO 7
690   B((I*3+J)*8+7-G) = (PEEK(3520+S*64+I*3+J)
      AND 2^G)/2^G
700   NEXT G
```





```
710   POKE 3520 + S * 64 + I * 3 + J,0
720   NEXT J
730   NEXT I
740   FOR I = 0 TO 2
750   FOR J = 0 TO 20
760   FOR G = 0 TO 7
770   IF B(I * 192 + J + (7 - G) * 24) = 0 THEN 790
780   POKE 3520+S*64+I+(20-J)*3,
      PEEK (3520+S*64+I+(20-J)*3) OR 2^G
790   NEXT G
800   NEXT J
810   NEXT I
820   RETURN
830   REM  ********** ROTATE 90 DEG.
840   GOSUB 650
850   B = 2
860   GOTO 240
```

During the recalculation, the 40-character screen flickers a lot, which tells us that the "high speed" mode (FAST) is on again. The system will return to normal when the program is finished, and the new sprite is displayed on the screen. Pressing any key will return you to the menu.

Here's a quick rundown of the functions:

1. Mirroring

   The sprite will be turned upside-down. The point DATA will *not* be inverted.

2. Mirroring by axis.

   Hard to describe; splits inverted sprite.

3. Turn 180 Degrees

   Self-explanatory.

4. Display Sprite
This lets you see the sprite at any time.

5. Reverse
Not to be confused with mirroring -- the sprite comes up as a "negative" (i.e., reverse video). Multicolor mode is not in this program, but you can still use the sprite in that mode later.

6. Turn 90 Degrees
Self-explanatory.

7. Turn 270 Degrees
Self-explanatory.

C. Copy sprite
Self-explanatory.

Final note: Pressing "C" copies the original sprite, so you have a backup if you destroy your original during experimentation.



# CHAPTER 3