

# **C-128 80 COLUMN DISPLAY MANAGER**

Developed by *BCS Technology Limited*

Copyright ©1987. All rights reserved.

See the last page of this document for details.



# TABLE OF CONTENTS

Introduction. . . . .	1
Numbered Functions. . . . .	3
80CDM Calling Syntax.. . . .	4
Using 80CDM. . . . .	6
Numbered Functions Descriptions.. . . .	8
Copy Screen. . . . .	9
Exchange Screens.. . . .	11
Redirect Display.. . . .	12
Enable/Disable Status Line.. . . .	13
Screen Blanker.. . . .	15
Screen Adjuster. . . . .	16
Export Screen. . . . .	17
Import Screen. . . . .	19
Status Line Writer. . . . .	20
Programming Considerations.. . . .	22
Interrupt-Driven Access. . . . .	24
Driving the Status Line from BASIC.. . . .	25
Software License.. . . .	27



## C-128 80 COLUMN DISPLAY MANAGER

The C-128 80 Column Display Manager (**80CDM**) is a programming utility that has been designed to supplement the Commodore C-128's 80-column video display implementation, especially the windowing functions.

The following operations are available through 80CDM when running on a C-128 with 64 kilobytes (KB) of video memory (VRAM):

- **Screen copy.** Copies the visible display (the “live” screen) and its attributes to any one of ten available hidden screen buffers. It is also possible to copy any hidden screen buffer to any other, or to the live screen.
- **Screen exchange.** Swaps the live screen and an exchange screen buffer, the latter which is a separate buffer from the ten buffers available to the screen copy function.
- **Display redirection.** Sends print output to any of the ten hidden screen buffers, as well as to the exchange buffer, allowing a program to prepare a screen “behind the scenes” and then later copy it to the live screen.
- **Screen export.** Copies a selected screen to a designated area of system memory (RAM).
- **Screen import.** Copies a designated area of system RAM to a selected screen.
- **Static status line.** Reorganizes the display to generate an extra text row, this row being available as a reverse video status line. 80CDM includes a function to display text on the status line, starting at any desired column. Another function may be used to vertically center the raster on the screen when the status line is enabled.
- **Screen blanker.** Turns off the display to reduce the risk of static image burn-in.

On a C-128 with 16 KB VRAM, the screen exchange, status line and display adjustment features will be inoperable, and only one hidden screen buffer will be available.

The screen copy and exchange functions work by storing display and attribute bytes into blocks of normally unused VRAM, utilizing the block-copy feature of the 8563/8568 VDC hardware. Along with the copying of VRAM, 80CDM stores the associated operating system (kernel) variables into tables for later use. When a stored screen is copied to the live screen, the kernel variables are copied as well, and the visible display takes on the characteristics it had when the screen was originally stored. The result is that everything associated with the appearance of the display, including the window size and position, is restored.

Supermon 816 is copyrighted software and may be redistributed with limitations. Please see page 27 for more information.

## 80CDM FUNCTIONS

Access to 80CDM is via a static jump table. The first address in the jump table is for invoking the “numbered” functions, that is, functions that are selected by passing an index to 80CDM. Other functions within 80CDM have individual addresses in the jump table and will be described later on.

The numbered functions in 80CDM are as follows:

- 0 Initialize 80CDM.
- 1 Copy screen.
- 2 Exchange screens.
- 3 Redirect display.
- 4 Enable/disable status line.
- 5 Screen blanker.
- 6 Adjust display vertical position.
- 7 Export screen to RAM.
- 8 Reserved; do not use.
- 9 Import screen from RAM.

On a 16 KB VRAM system, even numbered functions other than zero will not be available.

80CDM loads into RAM-0 starting at \$E000 (57344), which is under the kernel ROM. As a result, 80CDM is about as unobtrusive as anything can be on a C-128. Programming considerations will be discussed later on.

## GENERAL CALLING SYNTAX

The general form of a BASIC call to an 80CDM numbered function is as follows:

```
BANK 0:SYS DM,FC,P1,P2
```

In the above, **DM** is 80CDM's function execution address, currently at \$E000 or 57344, **FC** is the function number as described above and **P1** and **P2** are function-specific parameters, which may not be required in every case. The call must always be preceded by **BANK 0** in order to expose 80CDM. Otherwise, a crash will occur.

The general form of an assembly language call to 80CDM is as follows:

```
lda #%00111111
sta mmucr           ;$FF00, map out all ROMs
lda #func           ;binary function number
ldx #parm1          ;binary function-specific parameter #1
ldy #parm2          ;binary function-specific parameter #2
jsr dmfunc          ;numbered function entry point at $E000
bcc okay            ;function call successful
bcs error           ;error code returned in .A
```

Following any call, 80CDM will return status information, usually to indicate if the call was satisfactorily completed. The general form for retrieving status from 80CDM in BASIC is:

```
RREG EE,,,C:C=© AND 1):IF C THEN ...process error...
```

*80CDM does not interact with BASIC in any way.* Therefore, the above error checking method must be immediately executed after an 80CDM call to determine success or failure. **EE** will return an error code whose meaning may vary from function to function. If no error has occurred **EE** will hold the function number that was used to call 80CDM.

Note: The expression `C=© AND 1)` works around a problem with the way BASIC 7.0 handles Boolean AND statements involving signed one-byte integers. Directly testing `C AND 1` with an IF - THEN statement may not behave as expected.

There are three error codes that may be returned by any function call:

64 (\$40)	80 column display not active.
128 (\$80)	Invalid function number.
192 (\$C0)	80CDM not initialized.



Error 128 (\$80) will be returned if the function number is totally out of range, such as specifying function number 47, or if the function is not available, for example, specifying an even numbered function on a system with 16 KB of VRAM.

Following an assembly language call in which an error occurred, indicated by carry being set, 80CDM will return the error code in .A. Generally speaking, .X and .Y will return their entry values. If the function call was successful, .A will return its entry value. All of this will be discussed in more detail.

It is essential in assembly language programs to correctly configure the memory map before calling 80CDM, as the usual system state is to have the kernel ROM mapped in. Forgetting to write the correct configuration to the MMU will invariably cause a fatal error.

Function calls to 80CDM when the 40-column display is active will immediately abort with error 64 (\$40). The C-128 should be running in FAST mode for best performance.

Incidentally, SYS DM, where DM is 80CDM's jump table starting address, will execute faster than SYS <ADDR>, where <ADDR> is a hard-coded numeric or hexadecimal address, as the latter requires a time-consuming double numeric conversion in BASIC. Hard-coding addresses is never good programming practice.

## USING 80CDM

80CDM must be BLOAded into RAM-0 and initialized before use. The initialization procedure from BASIC is as follows:

```
BLOAD "80CDM.BIN",B0:REM non-relocating load into bank $00
DM=DEC("E000"):REM defined base address
BANK 0:SYS DM,0:REM initialize 80CDM
RREG VR,NS,FF,C:C=© AND 1):REM get return status
IF C THEN PRINT "ERROR";VR;"OCCURRED."
```

During initialization, you may see a momentary disruption to the 80 column display as 80CDM sizes the available video RAM. If 64KB of VRAM is detected on a C-128 with the 1985 revision ROMs, the VDC will be configured to use that extra RAM, which will result in the screen will be cleared to reestablish a coherent display. No such clearing operation will occur with the C-128DCR, since it shipped with 64KB of VRAM.

Initialization will fail with an error if the 40-column display is active, in which case VR will be 64 and the values in NS and FF will be meaningless. Otherwise, RREG will return the following information:

**VR** The amount of VRAM detected, either 16 (16 KB) or 64 (64 KB).

**NS** The number of screen buffers available for use, either 1 (16 KB system) or 10 (64 KB system).

**FF** Available functionality, either 0 for limited functionality (16 KB system) or 1 for full functionality (64 KB system).

“Limited functionality” means that the screen exchange, status line and display adjustment features will not be available.

To initialize 80CDM from assembly language, assuming it has already been loaded into RAM-0, code as follows:

```
lda #%00111111
sta mmucr          ;set configuration
lda #0             ;function number
jsr dmfunc         ;initialize 80CDM
bcs error          ;40-col display active
```

Upon return, .A will correspond to the VR variable in the BASIC form of the call, .X will correspond to the NS variable, and .Y will correspond to the FF variable, all of these being returned as binary values.

Once initialized, 80CDM will be ready for use. Subsequent calls to initialize 80CDM will have no effect but will return the status values described above. During initialization, 80CDM will lower the top-of-BASIC pointers at \$1212 and \$1213, reducing the total space available for BASIC programs from 58 KB to 50 KB. This procedure is necessary to protect 80CDM from being “stepped on” by an exceptionally large BASIC program.

Note: If you plan to run Clock-Calendar 128 (**CC128**) in conjunction with 80CDM it is recommended that you load and initialize 80CDM before starting CC128. Refer to the CC128 documentation for more information.

## NUMBERED FUNCTION DESCRIPTIONS

This section will describe each numbered function and how to use it. Before elucidating, it is appropriate to mention an 80CDM feature called “vertical retrace delay” or **VRD**.

Functions that affect the visible display may cause momentary disruption due to the speed at which the contents of VRAM can be changed by 80CDM. For example, copying a hidden screen to the live screen may cause raster flicker (“tearing”) or herringbones due to the abrupt transition from the image that existed immediately before the copy operation to the image resulting from the copy procedure. This effect is usually seen when the transition occurs while the visible screen raster is being painted.

This problem was anticipated in the design of 80CDM. The VRD feature makes it possible to arrange for display changes to occur during the vertical retrace period, which is when the VDC is preparing to paint a new frame on the screen. During vertical retrace, the electron beam in the video monitor’s CRT will be turned off. If VRD is enabled, the transition between displays will occur at that time, reducing or eliminating “tearing.”

To enable VRD, logically OR the function number with \$80 (FC OR 128 in BASIC). Enabling VRD will have some negative effect on performance, as 80CDM will wait until the VDC indicates it is in vertical retrace before executing the selected function. In most cases, the difference in performance won’t be perceptible.

Functions that cannot use VRD or do not affect the live screen (e.g., copying from one hidden screen buffer to another) will behave as though VRD was not enabled.

- **Copy Screen - Function 1**

To copy a screen from BASIC, code as follows:

```
BANK 0:SYS DM,1,S,T
```

where **S** is the source screen number and **T** is the destination or “target” screen. Allowable screen numbers are 0 (zero) for the visible or live screen, and 1 to 10 for a 64 KB VRAM system. Screens 0 and 1 are the only permissible screen numbers on a 16 KB VRAM system.

On a 64 KB VRAM machine, copying the contents of one hidden screen to another is an acceptable procedure. For example:

```
BANK 0:SYS DM,1,3,10
```

will copy the contents of screen number 3 to screen number 10, making screen number 10 an exact duplicate of screen number 3.

From assembly language, code as follows:

```
lda #%00111111
sta mmucr          ;set configuration
lda #1             ;function number
ora #$80           ;set bit 7 to enable VRD
ldx #source        ;source screen number
ldy #target        ;target screen number
jsr dmfunc         ;call function ($E000)
bcs error          ;function not completed
```

In the assembly language call, screen numbers are binary, e.g., screen 10 will be \$0A.

Possible error returns from screen copy are:

- 1 (\$01)      Invalid source screen number. This error will occur if the screen number is completely out of range (e.g., screen number 47), is out of range for the system (e.g., screen number 2 on a 16 KB system), or the source screen contains no valid data.
- 2 (\$02)      Invalid target screen number. This error will occur if the target screen number is completely out of range or is out of range for the system.

3 (\$03) Target and source screen numbers are the same.

Copying to screen 0 will cause an instantaneous change to the visible display. It is recommended that VRD be used with such a copy to avoid display “tearing.”

- **Exchange Screens - Function 2**

This function swaps the live screen with the exchange (swap) screen. That is, the live screen is copied to the swap screen and the swap screen is copied to the live screen, causing an instantaneous change to the visible display. On the next call, the process will be repeated and, following two successive swaps, what was originally the live screen will again become the live screen.

To swap the live screen with the exchange screen from BASIC, code as follows:

```
BANK 0 : SYS DM,2
```

From assembly language, code as follows:

```
lda #%00111111
sta mmucr           ;set configuration
lda #2              ;function number
ora #$80            ;to enable VRD
jsr dmfunc          ;call function ($E000)
bcs error           ;function not completed
```

Possible error returns from this function are:

- 1 (\$01)      Invalid source screen number. This error will occur if the exchange screen contains no valid data. Display redirection (described next) must first be used to load the exchange screen buffer with valid display data before a swap can be performed.

An invalid function error will always occur on a 16 KB VRAM system.

- **Redirect Display - Function 3**

This function configures the C-128's screen kernel so the display is printed to one of the hidden screen buffers rather than to the visible display. Once a screen buffer has been so prepared it may be copied to any other screen buffer or to the live screen.

To redirect the display from BASIC, code as follows:

```
BANK 0:SYS DM,3,,T
```

where **T** is the screen number to which the display is to be redirected. To redirect to the exchange buffer, use screen number 11. Redirecting to screen number 0 will return display output to the live screen. The only valid values for **T** on a 16 KB VRAM system are 0 and 1.

From assembly language, code as follows:

```
lda #%00111111
sta mmucr           ;set configuration
lda #3              ;function number
ldy #target         ;target screen number
jsr dmfunc          ;call function ($E000)
bcs error           ;function not completed
```

Possible error returns from this function are:

2 (\$02)            Invalid target screen number.

Calling this function does not change the state of the display in any way, which is to say the screen kernel variables that determine window size, cursor position, etc., will not be disturbed. All that will change is where the output goes into VRAM. Redirecting to screen number 6, for example, will only direct output to the area of VRAM designated as screen number 6. Therefore, it is recommended that the window be reverted to full size and the screen be cleared if the redirection target is not the visible screen. Otherwise it is possible that the display you are trying to build won't be what was expected due to the unknown initial state of the redirection target.

You should not invoke this function from immediate mode, as redirecting the display to anything other than the visible screen will prevent you from seeing what you are typing.



- **Enable/Disable Status Line - Function 4**

This function turns on or turns off the status line on a 64 KB VRAM system. The status line is always displayed in reverse video and is not affected by other activity on the screen.

To enable the status line from BASIC, code as follows:

```
BANK 0 : SYS DM,4,1,A
```

where A is a combined color and character set attribute. The color number may be any one of the following:

0 (\$00)	black	8 (\$08)	dark red
1 (\$01)	dark grey	9 (\$09)	red
2 (\$02)	dark blue	10 (\$0A)	dark violet
3 (\$03)	blue	11 (\$0B)	violet
4 (\$04)	dark green	12 (\$0C)	dark yellow
5 (\$05)	green	13 (\$0D)	yellow
6 (\$06)	dark cyan	14 (\$0E)	grey
7 (\$07)	cyan	15 (\$0F)	white

One of the higher intensity colors, such as grey, is recommended for most applications.

Passing one of the above color numbers as the status line attribute will cause text to be displayed using the upper case/PET graphics character set. To display status line text in mixed (upper case/lower case) characters, OR the color number with 128. For example, to turn on the status line in grey with mixed case text from BASIC:

```
BANK 0:SYS DM,4,1,14 OR 128
```

The above would be coded as follows in assembly language:

```
lda #%00111111
sta mmucr          ;set configuration
lda #4             ;function number
ora #$80           ;to enable VRD
ldx #1             ;enabling the status line
ldy #%10001110     ;light gray & mixed case text
jsr dmfunc         ;call function ($E000)
bcs error          ;function not completed
```

To disable the status line, code as follows in BASIC:

```
BANK 0 : SYS DM,4,0
```

Disabling the status line in assembly language would be coded as follows:

```
lda #%00111111
sta mmucr          ;set configuration
lda #4             ;function number
ora #$80           ;to enable VRD
ldx #0             ;disabling the status line
jsr dmfunc         ;call function ($E000)
bcs error          ;function not completed
```

You can change the status line attribute after it has been enabled by calling the function as though the status line is being enabled. Such a procedure will not erase any status line text, if present.

Possible error returns from this function are:

4 (\$04) Invalid control switch. Only 0 and 1 are allowed.

5 (\$05) Invalid attribute.

As the status line occupies an extra screen row, the vertical alignment of the raster may be sub-optimal when the status line has been enabled. You can correct the alignment with function number 6 (see page 16).

- **Screen Blanker - Function 5**

This function turns on or off the visible display and thus may be used to prevent static image burn-in to the screen. From BASIC, turn off the display as follows:

```
BANK 0:SYS DM,5,1
```

To turn on the display from BASIC:

```
BANK 0:SYS DM,5,0
```

The assembly language equivalent would be:

```
lda #%00111111
sta mmucr          ;set configuration
lda #5             ;function number
ora #$80           ;to enable VRD
ldx #1             ;turn off display
...or...
ldx #0             ;turn on display
jsr dmfunc         ;call function ($E000)
bcs error          ;function not completed
```

Possible error returns from this function are:

0 (\$00)      80CDM busy, relevant if called from an IRQ handler.

4 (\$04)      Invalid control switch.

It is permissible to call this function from an interrupt handler. Such a procedure will be discussed later on.

- **Raster Position Adjustment - Function 6**

This function may be used to make vertical adjustments to the raster position when the status line is enabled. Each call to this function will vertically reposition the raster one scan line, the direction being determined by the switch value passed into the function

The general form of the function call in BASIC is:

```
BANK 0:SYS DM,6,SW
```

where the switch **SW** is one of the following values:

SW	Effect
1	move raster up
128	move raster down
0	restore default adjustment

A limited adjustment range is available—exceeding it will cause display disruption. Restoring the default adjustment relocates the raster to the position that existed at the time the status line was enabled.

The assembly language equivalent of the above would be:

```
lda #%00111111
sta mmucr          ;set configuration
lda #6             ;function number
ora #$80           ;to enable VRD
ldx #switch        ;adjustment switch, see above
jsr dmfunc         ;call function ($E000)
bcs error          ;function not completed
```

Possible error returns from this function are:

4 (\$04)      Invalid control switch.

- **Export Screen - Function 7**

This function may be used to export any valid screen to RAM-0. The exported screen may be saved to a file for later recall via the import screen function described on page 19.

To export a screen, code as follows in BASIC:

```
BANK 0:SYS DM,7,S,RP
```

where **S** is the source screen number and **RP** is the starting page in RAM-0 to which the screen is to be exported. For example:

```
BANK 0:SYS DM,7,0,64
RREG RP,EL,EH,C : C=© AND 1)
IF C THEN PRINT "ERROR";RP;"OCCURRED" : END
EA = EH * 256 + EL
```

The above code will export the visible screen to RAM-0 starting at address 16384 (\$4000), which is the 64th RAM page in the system. Assuming no error occurred, the starting page (64 in this example) will be returned in **RP** and the exported screen will occupy RAM-0 up to and including address **EA-1**. This information could be subsequently used to store the screen into a binary (PRG) file or into a RAM expander for later recall.

The above example would be coded as follows in assembly language:

```
lda #%00111111
sta mmucr          ;set configuration
lda #7             ;function number
ldx #0             ;live screen is source
ldy #>$4000        ;system RAM starting page
jsr dmfunc         ;call function ($E000)
bcs error          ;function not completed

sta rampag         ;starting RAM-0 page
stx endadr         ;end of export +1 LSB
sty endadr+1       ;end of export +1 MSB
```

Possible error returns are:

- 1 (\$01) Invalid source screen number. If the source screen is not the visible display it must have previously been loaded with data, either by copying, redirection or by importing (page 19).

Exported screens always start on an even page boundary. The structure of the exported image is as follows—offsets are in hexadecimal relative to the start of the target RAM-0 page:

- 0000 Source screen number, e.g., \$0A for screen number 10.
- 0001 Number of bytes copied from display and attribute RAM, in little endian format. An export of a default screen will store \$07D0 (2000) into this location.
- 0003 Display VRAM screen codes, the total being equal to the value stored at offset \$0001.
- aaaa Attribute VRAM codes, the total being equal to the value stored at offset \$0001. \$aaaa will vary, depending on the screen configuration. For an export of a default screen, \$aaaa will be \$07D3.
- kkkk Zero page kernel variables associated with the screen, a total of \$1A (26) bytes. As with \$aaaa, \$kkkk will vary, depending on the screen configuration. For an export of a default screen, \$kkkk will be \$0FA3.
- llts Line link and tab stop bitmaps associated with the screen, a total of \$0E (14) bytes. As with \$kkkk, \$llts will vary, depending on the screen configuration. For an export of a default screen, \$llts will be \$0FBD.
- cscs Cursor shadow register associated with the screen, one byte. As with \$llts, \$cscs will vary, depending on the screen configuration. For an export of a default screen, \$cscs will be \$0FCB.
- fgbg Foreground/background color associated with the screen, one byte. As with \$cscs, \$fgbg will vary, depending on the screen configuration. For an export of a default screen, \$fgbg will be \$0FCC.

Depending on the screen configuration, up to 4 KB of contiguous RAM will be required to export a screen. 80CDM does not check the specified RAM-0 page to determine if it is safe to use. It is your responsibility to pick an area in RAM-0 where interference will not occur. Anywhere in RAM-0 above BASIC is okay, including the 4 KB under the I/O block. Naturally, avoid overwriting 80CDM or anything else essential to the operation of the system.

80CDM disables interrupt request processing during the export operation to maximize performance. A typical export operation on a machine running at 2 MHZ takes about 100 milliseconds.

- **Import Screen - Function 9**

This function may be used to import a previously-exported screen image to any valid screen, including the live screen. Prior to calling this function, a valid screen image must have been placed into RAM-0, starting on an even page boundary (see the export function on page 17 for a description of the image structure).

To import a screen, code as follows in BASIC:

```
BANK 0:SYS DM,9,RP,T
```

where **RP** is the starting page in RAM-0 from which the screen is to be imported and **T** is the target screen number. For example:

```
BANK 0:SYS DM,9,64,0
RREG EE,,,C:C=© AND 1)
IF C THEN PRINT "ERROR";EE;"OCCURRED"
```

The above code will import a screen image from RAM-0 starting at address 16384 (\$4000), which is the 64th RAM page in the system. Since the import is to the visible screen in this example, the display will immediately take on the characteristics of the image.

The above example would be coded as follows in assembly language:

```
lda #%00111111
sta mmucr           ;set configuration
lda #9              ;function number
ldy #>$4000         ;system RAM starting page
ldx #0              ;live screen is target
jsr dmfunc          ;call function ($E000)
bcs error           ;function not completed
```

Possible error returns are:

2 (\$02)      Invalid target screen number.

80CDM does not check the structure of the screen image in RAM-0. If the content is improperly formatted or if the number of bytes in the display and attribute segments does not conform to the VDC's configuration, display corruption is highly likely. 80CDM disables interrupt request processing during the import operation to maximize performance. A typical import operation on a machine running at 2 MHZ takes about 100 milliseconds.

**STATUS LINE WRITER**

The status line writer function, whose 80CDM jump table entry point is at \$E007 or 57351 (DM+7), allows you to display text on the status line, starting at any desired column up to the screen width. The text character set is determined by the attribute value that was passed to 80CDM when the status line was enabled (function 4).

To write text from assembly language, code as follows:

```

lda #%00111111
sta mmucr          ;set configuration
lda #col           ;zero-based column at which to print
ora #%10000000     ;set bit 7 to clear status line
ldx #<string       ;null-terminated text string address LSB
ldy #>string       ;null-terminated text string address MSB
jsr dsline         ;call function ($E007)
bcs error

```

Column values range from \$00 to width-1, where width is the display width in columns (usually 80). Setting bit 7 of the column value will cause 80CDM to clear the status line before writing the text. Setting both .X and .Y to zero will clear the status line only—the value in .A will be ignored.

Possible error returns are:

0 (\$00)	80CDM busy, relevant if called from an IRQ handler.
5 (\$05)	The text string is too long relative to the starting column value, also possible if the text string does not have a null terminator.

Programming example:

```

lda #%00111111
sta mmucr          ;set configuration
lda #$0a           ;start at column 10
ora #%10000000     ;clear status line before writing
ldx #<text         ;text string address LSB
ldy #>text         ;text string address MSB
jsr dsline         ;call function ($E007)
bcs error          ;error, error code in .A
;
text .byte "This is a test.", $00

```



The text string should be printable PETSCII characters, not screen codes. Non-printing characters in the text string will produce unpredictable results.

To accomplish the above from BASIC, it is necessary to POKE the text string into RAM-0, append a zero byte and then call the status line writer function as follows:

```
BANK 0 : SYS DM+7,CC,AD AND 255,AD/256
```

where CC is the starting column and AD is the address in RAM-0 to which the string was POKEd. If AD is zero the status line will be cleared. To clear the status line before displaying the text pass CC OR 128 as the column value. Alternatively, you can use the DSLINE.BIN utility, a separately available program described on page 25, to access the status line writer. DSLINE.BIN has a BASIC-friendly call syntax and requires no POKEing.

It is permissible to call this function from an interrupt handler.

## PROGRAMMING CONSIDERATIONS

As described in the introduction, 80CDM runs underneath the kernel ROM, where it is not likely to interfere with anything else. During initialization, 80CDM will relocate the top of the BASIC program text area to protect itself from a very large program. If an oversized BASIC program is subsequently loaded, an out-of-memory error will occur rather than “stepping on” 80CDM.

Once 80CDM has been loaded and initialized, you must be careful to not accidentally overwrite any part of the bank \$00 memory range 57344 (\$E000) to 660620 (\$ECCC) inclusive. Doing so will either cause strange behavior or system fatality when 80CDM is called, the latter very likely if 80CDM is called from within an interrupt handler.

80CDM reads the VDC configuration registers at startup, makes some adjustments to the VDC's memory map, e.g., relocating display attribute memory to make room for the status line, and thereafter assumes that the VDC's configuration will remain unchanged. If your application changes the VDC configuration in some way, such as setting up extra display rows, be sure to do this before initializing 80CDM. Otherwise, 80CDM's understanding of how the VDC is operating and how VRAM is organized may be wrong, opening the door to display corruption when various functions are used.

In the same context, be careful with directly accessing the VDC hardware via the screen kernel ROM routines (*never **PEEK** or **POKE** any VDC register*), as you may change the VDC's configuration in a way that is incompatible with 80CDM.

Pressing Stop/Restore will revert the VDC and screen kernel variables to their power-on defaults. Should the use of Stop/Restore be necessary to regain control of your machine, be sure to reload and initialize 80CDM. Although the program itself will be intact following Stop/Restore, the system configuration will be reverted to the power-on condition and will not agree with what 80CDM thinks it should be.

If you wish to call 80CDM from any assembly language program, especially an interrupt handler, you should first determine that 80CDM is present in memory. 80CDM includes a “presence signature” that may be examined to determine if the program has been loaded, the pointers to this signature being available at \$E003 (LSB) and \$E004 (MSB) in RAM-0 (equal to DM+3 or 57347 and DM+4 or 57348, respectively). The following code shows how to detect 80CDM's presence:

```

        lda #%00111111
        sta mmucr      ;set configuration
        ldx dmsigptr    ;80CDM signature pointer ($E003) LSB
        ldy dmsigptr+1  ;80CDM signature pointer ($E004) MSB
        stx zpptr      ;zero page pointer
        sty zpptr+1
        ldy #0
;
10001  lda (zpptr),y    ;80CDM's signature (we hope!)
        cmp dmsig,y     ;our version
        beq 10002       ;so far, so good
;
        jmp no_dm       ;80CDM not present or overwritten
;
10002  iny
        tax             ;test for null terminator
        bne 10001       ;not done testing
;
;...80CDM present, keep going...
;
dmsig .byte "80CDM...BCS2008",$00

```

The text string at DMSIG is case-sensitive. Note that the above test sequence doesn't determine if 80CDM has been initialized, only that it is present in memory and apparently intact.

## INTERRUPT-DRIVEN ACCESS

The screen blanker (function number 5) and the status line writer may be called from an interrupt handler on a non-interference basis. 80CDM prevents a conflict between a foreground call and an IRQ call by use of an internal semaphore. Foreground calls always have priority. If more than one interrupt handler calls 80CDM, the first handler will have priority over the second.

In some cases, it may be desirable to determine the state of the semaphore before making an 80CDM call. You can test the semaphore as follows:

```
lda #%00111111
sta mmucr          ;set configuration
jsr tstsem         ;test the semaphore ($E00A)
bcc clear          ;semaphore is cleared
bcs set            ;semaphore is set
```

If the semaphore is set and a call is made to an 80CDM function, error 0 (\$00), also called “busy,” will be returned, indicating that 80CDM is processing a foreground call.

Another consideration with calling an 80CDM function as part of interrupt processing has to do with screen kernel conflicts. 80CDM directly manipulates the VDC hardware and if called from an interrupt handler, may inadvertently alter a chip register that was configured by the screen kernel immediately prior to the interrupt, possibly causing display corruption. You can detect this condition by probing the stack to see if the address of the interrupted code is in the range \$0C53C to \$0CE4C inclusive. If it is, the call to 80CDM should be deferred.

## DRIVING THE STATUS LINE FROM BASIC

Calling 80CDM's status line writer from BASIC is clumsy and slow. To simplify status line text displays from BASIC, a separate program, DSLINE status line writer, may be used. DSLINE is a small machine code interface to 80CDM that can process the column and text in a way that is easier to code in BASIC.

From BASIC, call DSLINE as follows:

```
BLOAD "DSLINE",B0:      REM load binary into RAM-0
SL=DEC("0B00"):        REM define execution address
BANK 15:SYS SL,,,,,C,S$: REM call DSLINE
```

where **C** is the column at which to start writing and **S\$** is any string variable or string expression containing the text to be written to the status line.

A call of the form:

```
BANK 15:SYS SL,,,,,C,"LITERAL TEXT STRING"
```

is permissible, as is something like:

```
BANK 15:SYS SL,,,,,C,A$+MID$(B$,2,5)+C$
```

A call such as:

```
BANK 15:SYS SL,,,,,C," "
```

will clear the status line. The same will happen if a null string expression or variable is passed. Avoid embedding non-printing characters in the string, as undesirable results may occur.

Upon successful processing of the text passed in **S\$**, DSLINE will return control to your program. In the event of an error, your program with one of the following BASIC errors:

Syntax	Self-explanatory, usually caused by an incorrect number of commas between the <b>SL</b> and <b>C</b> parameters or failure to pass a string parameter. Note that five commas are required between <b>SL</b> and <b>C</b> .
Illegal Direct	DSLINE is a run-mode only program, as it uses the BASIC input buffer at \$00200 to process the string.

String Too Long	The text string length relative to the starting column value is excessive. The maximum permissible string length is $MC - C$ , where <b>MC</b> is the display width in columns (normally 80) and <b>C</b> is the column parameter passed in the call to <code>DSL</code> INE.
Type Mismatch	Self-explanatory.
Unimplemented Command	80CDM was not detected in memory, has not been initialized or the status line has not been enabled. This error will always occur on a 16 KB VRAM system, since the status line function cannot be supported on such a machine.

The current version of `DSL`INE loads into the tape cassette buffer at 2816 (\$0B00) in RAM-0 and uses memory up to and including 3069 (\$0BFD). If that memory range must be overwritten it will be necessary to reload the program before attempting to use it. If you wish to relocate the program, edit the `_ORIGIN_` assignment in the source code and reassemble. See `DSL`INE.ASM, which is included with the distribution. Be sure to read all notes in the source code before making changes.

**80 Column Display Manager** copyrighted ©1987 by *BCS Technology Limited*. All rights reserved.

**BASIC Status Line Writer** copyrighted ©1987 by *BCS Technology Limited*. All rights reserved.

---

Permission is hereby granted to use, copy, modify and distribute **80 Column Display Manager**, either as stand-alone code or as part of a package. In all cases, *BCS Technology Limited* requires that a copyright acknowledgment worded as follows must be present in any documentation included with the distribution of **80 Column Display Manager**:

**Portions of this software copyright ©1987 by BCS Technology Limited. All rights reserved.**

The above copyright acknowledgment must also appear at least once in the source code of the package into which **80 Column Display Manager** has been integrated. If distributing **80 Column Display Manager** as stand-alone code, the copyright header present in the original source code published by *BCS Technology Limited* must be retained unedited.

Redistribution of this software in any form must be at no charge to the end user. **This code or any part thereof, including but not limited to any derivation, MAY NOT be incorporated into any package intended for sale, unless written permission to do so has been granted by *BCS Technology Limited*.**

**THERE IS NO WARRANTY OF ANY KIND WITH THIS SOFTWARE.** While it is believed that all code will perform as described, the user assumes all risk in connection with the incorporation of this software into any system. If any of the aforementioned provisions are not acceptable, **do not distribute or use this software, and immediately delete it from any system on which it has been installed.**

---