Gemini Microcomputers Ltd

GM832

SVC
(SUPER VIDEO CONTROLLER)

SOFTWARE MANUAL

ISSUE 2

23·07·84

TABLE OF CONTENTS

1. Command Summary....2

2. General....5

3. Hardware....6

4. Software....8
4.1. Alphanumeric mode....8
4.2. Graphics mode....9

5. On-board switches and links....10

6. GM832 Host Interface....11

7. GM832 Control codes....12
7.1. Single byte control codes....12
7.2. Escape Sequences....14

8. SVC features....23
8.1. Software Clock....23

9. Caveats....24
9.1. Inadvertent requests....24
9.2. Nested escape sequences....24
9.3. Escape sequences and BASIC....24
9.4. System peculiarities....25

10. Keyboards....26
10.1. The Gemini GM821 keyboard....26
10.2. The Gemini GM827 and GM852 keyboards....26
10.2.1. Defining keys from the keyboard....26
10.2.2. Defining keys by software....27
10.2.3. KEYCHAIN....28

Appendices

A. Writing your own programs for the SVC....30
A.1. General....30
A.2. Mode....30
A.3. Stack....31
A.4. Registers....31
A.5. Utility subroutines....31

B. CRTC Information....32

C. Function Key codes....33

D. Changing the default Function key settings....34

E. Block Graphics....35

F. PUTVID & GETVID Example....36

1. Command Summary

NOTE:
1. These are the control codes that the SVC responds to. They are normally sent to the SVC by a User program or the Host's operating system. (See section 6 - PUTVID.)
2. Programs that request information from the SVC will have to read the information back from the SVC. (See section 6 - GETVID.)
3. It may not be possible to issue commands to the SVC direct from the keyboard attached to a system, as the operating system may modify the characters typed. (e.g. the standard CP/M line input routine would echo "^[" to the SVC if the ESCAPE key were pressed, rather than the ESCAPE code. However note that this problem can be solved in Gemini CP/M systems by selecting "EDIT mode", where all characters are echoed to the SVC exactly as typed).
   (Also see section 9.2 - Nested Escape sequences).

HEX	ASCII	DECIMAL	COMMAND	(Special key provided)
General				
07	^G	7	Bell - sound on-board buzzer	
08	^H	8	Backspace	(BACK SPACE)
0A	^J	10	Linefeed	
0D	^M	13	Carriage return	(RETURN)
Cursor Movement				
1C	^\	28	Cursor left	(←)
1D	^]	29	Cursor right	(→)
1E	^^	30	Cursor up	(↑)
1F	^/	31	Cursor down	(↓)
1B 0C	<ESC> ^L	27 12	Cursor home	
1B 3D...	<ESC> "="...	27 61...	Cursor addressing	
Additional cursor operartion				
1B 3F	<ESC> "?"	27 63	Return cursor coordinates and character	
1B 44	<ESC> "D"	27 68	Delete cursor	
1B 45	<ESC> "E"	27 69	Enable cursor	
1B 59...	<ESC> "Y"...	27 89...	Define cursor type	
Screen editing				
0B	^K	11	Delete line and scroll up	(SHIFT/↑)
0E	^N	14	Insert line	(SHIFT/↓)
16	^V	22	Delete character in line	(SHIFT/←)
17	^W	23	Insert character in line	(SHIFT/→)
1A	^Z	26	Clear screen & home cursor	
1B 16	<ESC> ^V	27 22	Delete character in screen	
1B 17	<ESC> ^W	27 23	Insert character in screen	
1B 25	<ESC> "%"	27 37	Delete to end-of-screen	
1B 2A	<ESC> "*"	27 42	Delete to end-of-line	

Screen format

1B 31	<ESC> "1"	27 49	Select 80 wide format
1B 32	<ESC> "2"	27 50	Select 40 wide format
1B 33	<ESC> "3"	27 51	Select user-defined format
1B 34	<ESC> "4"	27 52	Select graphics mode
1B 46...	<ESC> "F"...	27 70...	Define user format
1B 70...	<ESC> "p"...	27 112...	Select display/update partition
1B 42	<ESC> "B"	27 66	Blank screen
1B 56	<ESC> "V"	27 86	Video on (Unblank)
1B 49	<ESC> "I"	27 73	Video Invert screen
1B 4A	<ESC> "J"	27 74	Video normal screen (non-inverted)
1B 41	<ESC> "A"	27 65	Alternate char. generator is default
1B 4E	<ESC> "N"	27 78	Normal character generator is default
1B 4D	<ESC> "M"	27 77	Memory lock on
1B 4F	<ESC> "O"	27 79	Memory lock off

Character set

1B 67...	<ESC> "g"...	27 103...	Set language
1B 43...	<ESC> "C"...	27 67...	Define character
1B 63...	<ESC> "c"...	27 99...	Define character set
1B 62...	<ESC> "b"...	27 98...	Read back character set
1B 47	<ESC> "G"	27 71	Construct block graphics character set
1B 48	<ESC> "H"	27 72	Copy normal char. set to alternate & invert
1B 68	<ESC> "h"	27 104	Copy normal char. set to alternate char.gen

Block graphics

1B 47	<ESC> "G"	27 71	Construct block graphics character set
1B 52...	<ESC> "R"...	27 82...	Reset point X,Y
1B 53...	<ESC> "S"...	27 83...	Set point X,Y
1B 54...	<ESC> "T"...	27 84...	Test point X,Y

256x256 graphics

1B 52	<ESC> "R"...	27 82...	Reset point X,Y
1B 53...	<ESC> "S"...	27 83...	Set point X,Y
1B 54...	<ESC> "T"...	27 84...	Test point X,Y
1B 6C...	<ESC> "l"...	27 108...	Draw/Erase/Complement a line
1B 6F...	<ESC> "o"...	27 111...	Draw/Erase/Complement a circle
1B 6D...	<ESC> "m"...	27 109...	Move cursor to X,Y
1B 77	<ESC> "w"...	27 119	Fill a polygon
1B 64...	<ESC> "d"...	27 100...	Graphics screen dump

Keyboard

1B 66...	<ESC> "f"...	27 102...	Define function key(s)
1B 66 64	<ESC> "fD"	27 102 68	Reset function key definitions
1B 66 3F	<ESC> "f?"	27 102 63	Return current function key definitions
1B 6B	<ESC> "k"	27 107	Test keyboard status
1B 4B	<ESC> "K"	27 75	Get keyboard character
1B 58	<ESC> "X"	27 88	Get one line of input
1B 3E...	<ESC> ">"...	27 62...	Download soft-key display

Clock

1B 74...	<ESC> "t"...	27 116...	Set clock time
1B 74 3D..	<ESC> "t="...	27 116 61.	Position clock display
1B 74 45	<ESC> "tE"	27 116 69	Enable clock display
1B 74 44	<ESC> "tD"	27 116 68	Disable clock display
1B 74 3F	<ESC> "t?"	27 116 63	Return clock time

Miscellaneous

1B 5A	<ESC> "Z"	27 90	Return contents of current line
1B 61...	<ESC> "a"...	27 97...	Set/reset attributes
1B 57...	<ESC> "W"...	27 87...	High speed write to display
1B 4C...	<ESC> "L"...	27 76...	Load user program
1B 55	<ESC> "U"	27 85	Execute user program
1B 50	<ESC> "P"	27 80	Return light pen coordinates
1B 76	<ESC> "v"	27 118	Return SVC-MON version number

2. General

The Gemini Super Video Controller (SVC) is an 80-BUS compatible 8"x8" card that handles the character display for a Gemini (or Nascom) computer system. It is an enhanced version of the the earlier Gemini Intelligent Video Controller (IVC), and is upwards compatible with it, supporting all the functions of the earlier card. The SVC contains its own on-board processor (a Z80B running at 6MHz) and communicates with the host system via three I/O ports. As well as handling the video display, the SVC can optionally support a keyboard as well, providing full buffering and permitting "type ahead". In conjunction with the Gemini GM827 or GM852 keyboards it also supports programmable function keys.

By using its own on-board processor the SVC offers a powerful and fast character display with a multitude of features without absorbing any of the power of the host system's processor, or reducing the amount of memory available to programs running on the host system. In addition the SVC supports a 256 x 256 bit-mapped graphics mode.

The SVC accepts 8-bit characters from the host system. All the standard printable ASCII characters (whose codes are in the range 20H-7FH), and all the characters for the alternate character set (whose codes are in the range 80H-FFH), are placed directly into the display at the current cursor position. Characters in the range 00-1FH are interpreted as control characters, and are used to control the extensive features of the SVC.

3. Hardware

The SVC uses an on-board Z80B running at 6MHz to control the display in response to commands passed to it by the Host system. The display memory is shared between the on-board Z80B and the display controller, and a hardware based approach is used to ensure that there is no conflict between the two when accessing the shared memory. (This hardware arbitation scheme provides a significant speed increase when compared to the software arbitration used in the earlier IVC.) This results in an interference-free display that can be updated at a high rate.

The Video card has 2K of RAM on-board for program workspace. The control software uses only 1K bytes of this RAM, leaving 1K free. Provision has been made for the down-loading of a user program to this area, and its execution. This routine could be a specialised screen-handling function, or may have nothing whatever to do with the Video card. In the latter case the Video card can be used as another processor (which indeed it is) to carry out some parallel processing with the host system when it is not updating the display. This is covered more fully in appendix A.

The Video card will also support a keyboard. This may be either a keyboard with a parallel interface (e.g. GM827) or a keyboard with a serial interface (e.g. GM852S). Whenever a key on the attached keyboard is pressed, the resultant character is read and stored in an internal buffer. The Host system can retrieve characters from this buffer by sending the appropriate "Escape sequence" (see below). Note that the keyboard character is captured irrespective of whether the Host system has requested a character. This means that even if the Host system goes off to perform a lengthy operation (like inserting a new line into the middle of a large file) no characters will be lost as they will be queued in the internal keyboard buffer, and passed on to the Host system as and when it requests them. The keyboard buffer will hold up to 64 characters from the keyboard.

The SVC display memory is 8kbytes in size, and its usage varies with the selected display mode. The various standard modes are illustrated below:

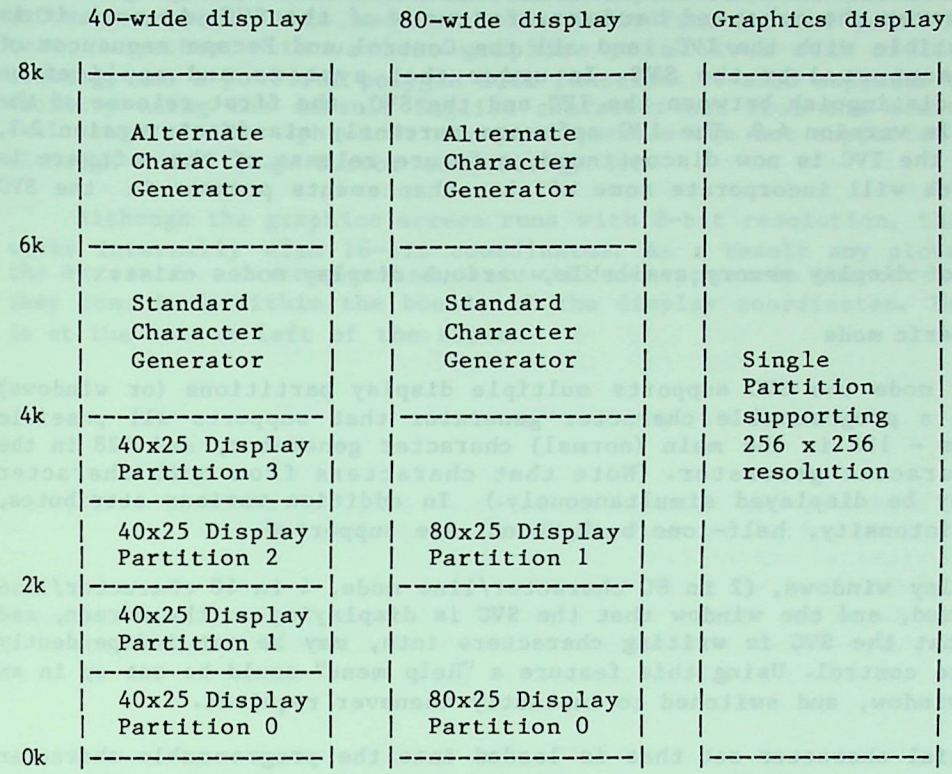

Fig. Display memory image

4. Software

The SVC monitor is a development of the IVC monitor, which has been extended to cover the enhanced hardware features of the SVC. As such it is upwards compatible with the IVC, and all the Control and Escape sequences of the IVC are supported by the SVC. In order that systems and application software may distinguish between the IVC and the SVC, the first release of the SVC software is version 4.0. The IVC software currently stands at version 2.1, and, although the IVC is now discontinued, a future release of the software is envisaged which will incorporate some of the enhancements present in the SVC monitor.

With 8k of display memory available, various display modes exist:

4.1. Alphanumeric mode

In alpha mode the SVC supports multiple display partitions (or windows) together with a programmable character generator that supports all possible 256 characters - 128 in the main (normal) character generator, and 128 in the alternate character generator. (Note that characters from both character generators may be displayed simultaneously.) In addition various attributes, (blink, half intensity, half-tone background) are supported.

The display windows, (2 in 80 character/line mode, 4 in 40 character/line mode), are fixed, and the window that the SVC is displaying on the screen, and the window that the SVC is writing characters into, may be set independently under software control. Using this feature a "Help menu" could be set up in an alternative window, and switched to instantly whenever required.

The initial character set that is loaded into the programmable character generator is held within the monitor ROM. The basic character set (English) can be modified to suit various foreign languages, and the selection is made by 3 on-board switches. If the required language is not available then, as on the IVC, the complete character set may be downloaded by a user program.

On power-up the main character generator is initialised with the selected character set, and the alternate character generator is loaded with the inverse of the main set. Thus, until the initial settings are modified by software, outputting an ASCII character with its most significant bit (msb) set to the SVC will result in that character being displayed in inverse video.

The SVC supports four character attributes that are 'switched on' by setting the most significant bit of a displayed character. Under software control the effect of the msb can be set to be any combination of the available attributes. The four character attributes are:

1) Use the alternate character generator. (Normally used to provide inverse video).
2) Provide a half-tone background to the character.
3) Display the character in reduced intensity.
4) Blink the character.

NOTE: As only one bit is available as an attribute bit, all characters with this bit set will display the same set of attributes.

4.2. Graphics mode

In graphics mode the SVC offers a 256 x 256 resolution display. The software supports this mode with the basic functions of pixel set, reset and test, together with various graphics primitives such as line and circle drawing, and a powerful polygon fill function. It also supports a limited text facility using the default English character set from the monitor EPROM. The full range of the alpha-mode escape sequences are not supported in this mode, although a few (e.g. Cursor addressing) are.

Although the graphics screen runs with 8-bit resolution, the SVC software works internally with 16-bit coordinates. As a result any plots that run off the screen do not wrap-round, but will re-appear in the correct place when they come back within the bounds of the display coordinates. The origin (0,0) is at the bottom left of the screen.

5. On-board switches and links

A four-pole DIL switch will be found on the SVC. This is used to define the keyboard in use, and the default character set. The switches should be set as follows:

SWITCH 2	3	4	
ON	ON	ON	English
ON	ON	OFF	Swedish
ON	OFF	ON	Danish
ON	OFF	OFF	German
OFF	ON	ON	French
OFF	ON	OFF	USA
OFF	OFF	ON	English
OFF	OFF	OFF	English

Fig. Selection of character set

Switch 1

OFF	Keyboard has function keys. (e.g. GM827 or GM852.)
ON	Keyboard has no function keys. (e.g. GM821.)

Note that there are links on the SVC which need to be set according to the type of keyboard connected. (Serial or parallel interface.) Details of these links can be found in the optional SVC Hardware Manual.

6. GM832 Host Interface

To the host system the video board appears as three I/O ports. One port is a bi-directional data port through which bytes are passed to and from the video card, one port is a read-only Status port which holds the two hardware 'buffer full/empty' flags used in data transfers, and the final port, which is Read/Write, is used to reset the video card's processor. As the onboard Z80B is not connected to the 80-BUS reset line the host system can be reset at any time without disturbing the video display, or losing the current video card configuration. (The current screen format and programmed character set will remain intact.)

N.B. The latter assumes that the Host's software does not reset the Video board while re-initialising the host system.

Port	Dir.	Function
0B1H	R/W	Data transfer to/from Video card.
0B2H	R/O	Status port for Data registers. Bit 0 Set if Write Buffer is full. Clear if Write Buffer is empty. Bit 7 Set if Read Buffer is empty. Clear if Read Buffer is full.
0B3H	R/W	Accessing Port resets Video card.

Shown below are examples of the simple driver routines that are required to interface the card. Routines such as these are already incorporated in RP/M, Gemini CP/M systems, and other software that directly supports the SVC. (See also Appendix F.)

```
                ;
                ;  PUTVID - Transfer the character in A to the Video card
                ;
F5      PUTVID: PUSH    AF          ;Save character
DB B2   PV0:    IN      A,(0B2H)    ;Read flags
0F              RRCA                ;Flag to carry
38 FB           JR      C,PV0       ;Loop if buffer still full
F1              POP     AF          ;Get character back
D3 B1           OUT     (0B1H),A    ;Put in buffer
C9              RET                 ;Done

                ;
                ;  GETVID - Read a character from the Video card
                ;           to the A register
                ;
DB B2   GETVID: IN      A,(0B2H)    ;Read flags
07              RLCA                ;Flag to carry
38 FB           JR      C,GETVID    ;Loop if buffer empty
DB B1           IN      A,(0B1H)    ;Read character
C9              RET                 ;Return with it
```

7. GM832 Control codes

The video card responds to a variety of control codes which provide extensive facilities for control of the card. As well as providing for the usual character functions, additional functions are provided including the ability to down-load another program to the video card's workspace, and to execute it.

The commands divide into two types, single byte control codes, and multiple byte control sequences. The multiple byte sequences all start with the control code "Escape" (1BH) and so are referred to as "Escape sequences". The single byte codes handle the usual Cursor functions, while the Escape sequences provide the more elaborate features.

NOTE:
1. These are the control codes that the SVC responds to. They are normally sent to the SVC by a User program or the Host's operating system. (See section 6 - PUTVID.)
2. Programs that request information from the SVC will have to read the information back from the SVC. (See section 6 - GETVID.)
3. It may not be possible to issue commands to the SVC direct from the keyboard attached to a system, as the operating system may modify the characters typed. (e.g. the standard CP/M line input routine would echo "^[" to the SVC if the ESCAPE key were pressed, rather than the ESCAPE code. - However note that this problem can be solved in Gemini CP/M systems by selecting "EDIT mode", where all characters are echoed to the SVC exactly as typed).
   (Also see section 9.2 - Nested Escape sequences).

7.1. Single byte control codes

Shown below are the single byte control codes and the corresponding keyboard characters that generate them. The convention of using a preceding up-arrow (^) to designate a control character has been adopted. Thus linefeed (Hex code 10) is generated by typing control/J which is shown as ^J. The cursor movement codes correspond to those generated by the Gemini GM827 and GM852 keyboards. The insert/delete line and insert/delete character codes correspond to combinations of the shift key and the same cursor control keys. ([] refer to specific keys on the Gemini keyboards).

All of these codes can be produced directly on the Gemini keyboards, either directly, or in conjunction with the shift and/or control keys. (But see NOTE above).

Hex	ASCII	Function
07	^G	Bell. Sounds the on-board buzzer.
08	^H	[BACKSPACE] Destructive backspace. The cursor is moved one position to the left, and the character now under the cursor is overwritten with a space. If the cursor is in the home position the code has no effect.

0A ^J Line feed. The cursor is moved down one line. If it is already at the bottom of the screen, then the entire screen is scrolled up by one line, and the bottom line (containing the cursor) is cleared.

0B ^K [Shift/↑]
Delete Line and Scroll up. The line in which the cursor is positioned is deleted, and the lines below are scrolled up the screen, with a blank line appearing at the bottom. If the cursor is on the bottom line, then this will be cleared.

0D ^M [RETURN]
Carriage return. The cursor is returned to the start of the line in which it is currently positioned.

0E ^N [Shift/↓]
Insert line. The line holding the cursor, and all the lines below it, are scrolled down the screen. A blank line is inserted at the current cursor position. The bottom line of the display is lost.

16 ^V [Shift/←]
Delete character from line. The character currently under the cursor is deleted, and the remaining characters on the line are shifted left one position. A space is entered in the last character position of the line.

17 ^W [Shift/→]
Insert character in line. A space is inserted at the current cursor position. The character under the cursor, and all characters to the right of it, are shifted one character to the right. The character at the end of the line is lost.

1A ^Z Home and clear. The cursor is returned to the Home position, (not necessarily the top of the screen - see <ESC> "M"), and the screen cleared from the cursor.

1C ^\ [←]
Cursor left. The cursor is moved left one position. If it was at the start of a line it is moved to the end of the preceeding line. It will not move past the Home position.

1D ^] [→]
Cursor right. The cursor is moved right one position. If it was at the end of a line it moves to the start of the next line. It will not move past the end-of-screen.

1E ^^ [↑]
Cursor up. The cursor is moved up one line on the display. It will not move past the "Home" line.

1F ^/ [↓]
Cursor down. The cursor is moved down one line on the display. If it is on the bottom line of the display, the code will have no effect.

7.2. Escape Sequences

Code sequence	Function
1B 0C	[<ESC> ^L] Cursor home. This sequence moves the cursor to the home position at the top left of the screen.
1B 16	[<ESC> ^V] Delete character from screen. The character currently under the cursor is deleted and all characters to the right of it, and below it, are shifted left one position. The character at the start of the next line is moved to the end of the line above, and so on down the display. A space is inserted at the end of the bottom line of the display.
1B 17	[<ESC> ^W] Insert character in screen. A space is inserted at the current cursor position. The character under the cursor, and all characters to the right and below, are shifted one character position to the right. The character at the end of each line where a shift occurs is moved to the start of the line below. The character at the end of the screen is lost.
1B 25	[<ESC> "%"] Delete to end-of-screen. The screen is cleared from the current cursor position.
1B 2A	[<ESC> "*"] Delete to end-of-line. The current line is cleared from the cursor position.
1B 31	[<ESC> "1"] Select 80 wide format. The inbuilt 80-character wide screen format is selected.
1B 32	[<ESC> "2"] Select 40 wide display format. The in-built 40-character wide screen format is selected.
1B 33	[<ESC> "3"] Select the user-programmed format. (See <ESC> "F".) If no format has been programmed the default "power-up" format is used.
1B 34	[<ESC> "4"] Select Graphics mode. The SVC is switched into "Graphics mode" where the display now supports a pixel graphics screen offering a resolution of 256 x 256. Characters can still be written to the screen, but the display only supports 32 characters per line, and the English character set will always be used. Individual character attributes are not supported, and nor are most of the character orientated Escape sequences. (e.g. Clear-to-end-of-line is not supported.)

1B 3D..... [<ESC> "=" RR CC]
Cursor addressing. The cursor is positioned to row RR and column CC. RR and CC are offset by 20H. i.e. to position to row 8, column 45 - RR=20H+8 and CC=20H+45, giving a code sequence of - 1B 3D 28 4D. The top left-hand corner of the screen has coordinates 0,0. If either of the coordinates are invalid the cursor position remains unaltered.

1B 3E... [<ESC> ">" NN ..... 00]
Dowload and set a 'soft key' display. This allows a defined number of lines to be locked at the bottom of the screen. These lines will not be scrolled or cleared during the normal use of the display. Optionally the data for the locked lines may be included in the Escape sequence.
The byte NN specifies how many lines are to be locked, where NN = the number of lines + ASCII '0' (30H). i.e. NN = '3' (33H) is a request for 3 lines to locked. If NN ='0', or is out of the range of the screen, then that is taken by the SVC as an instruction to unlock the bottom lines of the display, and the Escape sequence terminates. However, if NN represents a valid number of lines, then all following bytes received by the SVC will be down-loaded sequentially into the locked area. The load is terminated when either the area is full, or a NULL (0) is received.
For example this Escape sequence can be used to set up a 'soft key' display area at the bottom of the screen to show the current definitions of the programmable function keys on the Gemini Keyboard. Note that control characters in the range 00-1Fh are not normally displayed by the SVC, so this part of the character set could be re-programmed to suit the display, and the appropriate control characters (excluding 00) downloaded as part of the ESC > ... sequence.

1B 3F [<ESC> "?"]
Cursor coordinates. The current X,Y position of the cursor is returned via the SVC data port, together with the character at that position. They are returned in the order <row> <col> <char>. <row> and <col> are the actual coordinates (with no offset) and the top left-hand corner of the screen has the coordinates 0,0.

1B 41 [<ESC> "A"]
Selects the alternate character set as the default set. The msb of all characters is complemented before they are stored in the screen memory. (See also <ESC> "N".)

1B 42 [<ESC> "B"]
Blank the screen. The video output from the card is inhibited resulting in a blank screen, but the SVC continues to receive and process characters as normal. This allows screen displays to be set up "unseen", or can be used to briefly blank the screen while the screen format is changed. (See <ESC> "V").

1B 43..... [<ESC> "C" XX YY.....ZZ]

Define a character. This sequence allows a single character to be programmed into the character generator. The hex code XX is the ASCII code for the character to be programmed, which is then followed by the 16 bytes of the dot rows that will form the character in the order row 0 through to row 15. All sixteen must be provided, even if the display does not require them all. (The standard display uses ten raster lines per character.) By default they will load to the appropriate character position in the alternate character generator. (The characters which are displayed by codes in the range 80H-FFH, 128-255 decimal.) However if byte XX already has its MSB set the corresponding character in the normal character generator will be programmed. (Characters in the code range 00H-7FH, 0-127 decimal.) See also <ESC> "c".

1B 44 [<ESC> "D"]

Deletes the cursor from the display. (See <ESC> "E".)

1B 45 [<ESC> "E"]

Enables the cursor. The cursor re-appears on the screen if it had previously been switched off by <ESC> "D".

1B 46..... [<ESC> "F" AA....LL ZZ]

Define screen format. This sequence downloads a setting for the CRTC to the video card. AA....LL represent the twelve bytes that are to be set in registers 0 to 11 of the CRTC (see Appendix B for appropriate values) and ZZ is the byte that selects the dot clock appropriate for the 80-wide alpha-numeric display, the 40-wide alpha-numeric display, or the graphics mode display. The values for ZZ are shown below:

	msb lsb
80-wide	XXXXXX10
40-wide	XXXXXX11
Graphics	XXXXXX01

X = "don't care"

Note that data is only required for twelve of the CRTC's registers, the Display start address and Cursor address are added by the on-board software. The values are only programmed into the CRTC on receipt of the <ESC> "3" sequence.

1B 47 [<ESC> "G"]

Construct the block-graphics characters in the character generator. The block-graphics character set is constructed in the character generator at character addresses 0C0H-0FFH. (See Appendix E)

1B 48 [<ESC> "H"]

Copy the complement of the contents of the normal character generator to the alternate character generator. As a result setting bit 7 of a character will result in it being displayed in inverse video.

1B 49 [<ESC> "I"]
Display the entire screen in inverse video.

1B 4A [<ESC> "J"]
Display the entire screen in normal video.

1B 4B [<ESC> "K"]
Keyboard input. The next character from the keyboard is returned through the data port. If there are already characters stored in the keyboard buffer, then the next character will be returned immediately from there, otherwise there will be a delay until a key is pressed. (See also <ESC> "k".)

1B 4C.... [<ESC> "L" LL HH .....]
Load a user routine to workspace ram. This sequence allows a user program to be loaded to the workspace RAM of the video card (see Appendix A).

1B 4D [<ESC> "M"]
Memory lock on. All lines on the screen above the current line are "locked" on the display. If the screen scrolls these lines will not scroll. The "cursor up" key will not move the cursor into this area, nor will the "Home & Clear" code remove them. However the cursor addressing sequence allows the cursor to be positioned in this area. The "memory lock" function allows headings etc to be placed at the top of the screen, and to be preserved even if the display is subsequently scrolled or cleared. (See <ESC> "O".)

1B 4E [<ESC> "N"]
Normal character set. Characters are stored in the display as received, Bit 7 is not complemented. (See <ESC> "A".)

1B 4F [<ESC> "O"]
Memory lock off. Turns off the memory lock function. (See <ESC> "M".)

1B 50 [<ESC> "P"]
Returns a pair of coordinates from the light pen. The coordinates of the selected character cell are returned as <row> followed by <column>. (Similar to <ESC> "?", but no character is returned.) Depending on the design of light pen used these coordinates may need a small adjustment to arrive at the correct character cell.

1B 52.... [<ESC> "R" XX YY] ALPHA mode only
Resets block graphics point X,Y. Two bytes holding the X and Y coordinates respectively follow the lead-in sequence. These bytes include an offset of 20H in a similar manner to <ESC> "=". Point 0,0 is at the top left of the screen (and so is addressed by a coordinate pair of 20H 20H). The maximum coordinate values depend upon the current screen format. If the coordinate pair represents an illegal point then the request is ignored.

1B 52... [<ESC> "R" Xlo Xhi Ylo Yhi] GRAPHICS mode only
Resets the pixel X,Y. The display origin (0,0) is at the bottom left of the screen.

1B 53.... [<ESC> "S" XX YY] ALPHA mode only
Sets block graphics point X,Y. (See <ESC> "R"... .)

1B 53... [<ESC> "S" Xlo Xhi Ylo Yhi] GRAPHICS mode only
Sets the pixel X,Y. (See <ESC> "R"... .)

1B 54.... [<ESC> "T" XX YY] ALPHA mode only
Test block graphics point X,Y. (See <ESC> "R"... above.) The requested block graphics point is tested. If the point is OFF a byte of 0 is returned, if the point is ON a byte of 01 is returned, and if the coordinates were illegal then a byte of 02 is returned.

1B 54... [<ESC> "T" Xlo Xhi Ylo Yhi] GRAPHICS mode only
Tests the pixel X,Y. The requested graphics point is tested. If the pixel is OFF a byte of 00 is returned, if the pixel is ON a byte of 01 is returned, and if the coordinates are not within the display area a byte of 02 is returned. (See <ESC> "R"... .)

1B 55 [<ESC> "U"]
Execute User program. The program previously loaded by the <ESC> "L" command is executed. (See appendix A for details.)

1B 56 [<ESC> "V"]
Video. Turns on a previously blanked display. (See <ESC> "B".)

1B 57.... [<ESC> "W" LO HO LC HC MM .....]
Write to Display. This sequence allows characters to be moved at high speed direct to the display memory without any of the normal checks for control characters. This command is useful when it is required to update the screen at a very high rate, but it puts the onus on the user to get the screen format correct as the normal screen formatting characters (carriage return, line feed etc), are treated as normal printing characters and are placed directly into the display. The lead-in sequence is followed by a screen offset address (LO HO - lo byte followed by hi byte). This 16-bit number represents the offset (in characters) from the start of the display to the address at which the characters are to load. Next (LC HC) comes the 16-bit byte count of the number of characters to be loaded. The next byte, (MM), is ignored. It must be present, and was used by the IVC monitor to signify whether the load was to be 'transparent' (i.e. characters were only moved to the display during the blanking intervals), or 'direct'. If the byte MM was equal to 'T' (54H) or 't' (74H) the load to the IVC was done transparently, and as a consequence was not as fast. Any other value resulted in the load being immediate, and as a result interference could occur on the screen unless the display was blanked.

With the SVC the highest possible transfer speed can be obtained by first blanking the display as this also has the effect of disabling the display memory arbitration circuit, thus giving full priority to the Z80B.

1B 58 [<ESC> "X"]
Keyboard line input. Obtain one line of input from the keyboard connected to the SVC. When this sequence is received the video card will internally echo characters from the keyboard to the display until the "return" key is pressed. When this occurs the contents of the screen line currently holding the cursor are queued ready for reading by the Host system. Trailing blanks are removed from the line, and it is terminated by a carriage-return. While in the "line input" mode all the control codes can be used to move the cursor about the screen and to modify its contents, (i.e. On-screen editing can be used), and it is only when the code for "carriage return" (ODH) is detected that the mode terminates.

1B 59.... [<ESC> "Y" AA BB]
Define cursor type. This command defines the characteristics of the cursor. Byte AA is loaded to register 10 of the CRTC controller, and byte BB to register 11. (See Appendix B.)

1B 5A [<ESC> "Z"]
Returns the contents of the line currently holding the cursor. Trailing blanks are removed, and the returned characters are terminated by a carriage-return.

1B 61 .. [<ESC> "a" NN]
Set/reset attributes. This sequence is used to select the attributes that apply when bit 7 of a displayed character is set. The lower 4 bits of the byte NN are used as follows:

BIT	effect	
0	BLINK.	Setting this bit will cause all selected characters on the display to blink at a regular rate.
1	HALF INTENS	Setting this bit will cause all selected characters on the display to be shown with reduced intensity.
2	HALF TONE	If this bit is set, all the selected characters will be displayed with a half-tone background.
3	ALT. SET	This is the default setting, and selects the alternate character generator. If this has not been re-programmed, this results in the character being displayed in inverse video. (If necessary this can be set up again using <ESC> "H".)

The attributes may be set in any combination. e.g. sending NN=XXXX1001 would result in inverse blinking characters.

1B 62 .. [<ESC> "b" CC]
Read back the currently programmed character set. This sequence allows the bit-patterns in the character generator to be read back to the Host system. Either the pattern for a single character can be read, or the entire contents of the character generator may be requested. The response is controlled by the parameter CC. As usual the most-significant bit of CC controls which of the character generators is accessed. (Setting the msb selects the alternate character generator.) The lower 7 bits of CC select the character pattern required. If the character 00 or character 80H is requested, then the entire contents of the appropriate character generator are returned, (2048 bytes). Otherwise the 16 bytes associated with the specified character are returned. N.B. all 16 bytes are returned, although there are normally only 10 active lines on the display.

1B 63... [<ESC> "c" GG XX....ZZ]
Load character set. This sequence is used to load a complete character set to either of the programmable character generators. The byte GG specifies whether it is to the normal or alternate one. If GG=0 then the character set is loaded to the alternate character generator, if it is non-zero then the load is to the normal character generator (if programmable). XX...ZZ are the 2048 bytes that are to be loaded. The first sixteen bytes are the rows for character 00, the next sixteen for character 01... etc. (See <ESC> "C".)

1B 64 .. [<ESC> "d" CC] GRAPHICS mode only.
Graphics screen dump. The entire graphics screen memory (8192 bytes) is returned to the HOST, starting with the top left-hand corner of the display. If CC = 'p' or 'P' then the SVC performs the necessary translation to suit a dump for a dot-matrix printer running in graphics mode.
i.e. If the 'P' option is selected, then the display is read out in lines 8 dots high. (Each byte returned represents 8 dots in the Y-axis, and a single dot in the X-axis. There will be 256 bytes per scan of the screen width, and a total of 32 'scans'.) Otherwise each byte represents 8 dots in the X-axis, and one dot in the Y-axis. (In this case there will be 32 bytes for each scan of the screen width, and a total of 256 'scans'.)

1B 66... [<ESC> "f" XX YY....ZZ]
Define a string for a Function key. This sequence is used to change the definitions of the function keys on the extended keyboard, or to request the current definition table. See section 10.

1B 67 .. [<ESC> "g" CC]
Select language character set. This sequence re-programs the normal character generator with the language character set selected by the lower 3 bits of the byte CC.

CC	
00	English
01	Swedish
02	Danish
03	German
04	French
05	USA
06	English
07	English

NOTE. This should be followed by an ESC H if the correct inverse character set is required as well.

1B 68 [<ESC> "h"]
Copy the contents of the normal character generator to the alternate character generator. As a result setting bit 7 of a character will have no visible effect. (This will only be true if the 'alternate character set' is the only attribute enabled.)

1B 6B [<ESC> "k"]
Keyboard status. The card returns a byte of 0 if no characters are waiting to be read from the keyboard buffer. If one or more characters are waiting, then a byte of 0FFH is returned. The actual characters themselves are obtained by using the sequence <ESC> "K".

1B 6C ... [<ESC> "l" Xlo Xhi Ylo Yhi PEN] GRAPHICS mode only
Draw a line. Draws a line from the current graphics coordinates to the specified point X,Y using the pen type requested. The PEN options are:

```
00000000 - No change.
XXXXXX01 - Pen down. (Turns pixels on.)
XXXXXX10 - Pen up.   (Turns pixels off.)
XXXXXX11 - Xor.      (Complements pixels.)
           X = don't care
```

1B 6D ... [<ESC> "m" Xlo Xhi Ylo Yhi] GRAPHICS mode only
Move. Moves the graphics cursor to the point X,Y.

1B 6F ... [<ESC> "o" Rlo Rhi PEN] GRAPHICS mode only
Circle. Draws a circle on the display of radius R, centred on the current graphics coordinates using the pen type specified by PEN. (See <ESC> "l".) A true circle is drawn (in terms of the pixels), but its exact appearance will be determined by the Width and Height settings of the video monitor in use.

1B 70 .. [ESC "p" NN] ALPHA mode only
Select display/update partitions (or windows). This sequence selects which partition (or window) of the display memory is updated by the incoming characters, and which window is actually displayed on the screen. Bits 0 and 1 of NN select the window to be updated, while bits 2 and 3 select the window to be displayed:

```
                msb     lsb
          NN =  XXXXDDUU
      where:  XX = don't care
              DD = no. of display window (00 01 10 or 11)
              UU = no. of update window  (00 01 10 or 11)
```

1B 74 ... [<ESC> "t" NN ...]
Screen clock option. This sequence allows the SVC software clock to be set, read, enabled and disabled.
ESC t E - Enables the clock display.
ESC t D - Disables the clock display update.
ESC t = X Y - Defines the position of the clock on the screen.
ESC t HHMMSS - Sets the clock to HH MM SS.
ESC t ? - Returns the current time in the format HHMMSS.
(See the CLOCK section.)

1B 76 [<ESC> "v"]
Return version number. The card returns the version number of the SVC software. A single byte is returned, with 40H representing version 4.0, 41H representing 4.1, ... 50H representing version 5.0 and so on.
NOTE. The SVC version numbers start at 4.0 in order to distinguish them from the earlier Gemini IVC card whose version numbers started at 1.0 but will never reach 4.0.

1B 77 [<ESC> "w"] GRAPHICS mode only
Graphics fill. This sequence will fill in a bounded area of the graphics screen starting at the current graphics cursor. The fill proceeds outward from the starting point and will turn ON all pixels within the bounded area, whatever its shape. The fill routine requires a reasonable amount of workspace if it is to handle complex shapes, and so it utilises the 1kbyte USER AREA within the SVC. If a user downloads a program to the SVC (see ESC L), then the fill routine automatically adjusts itself to use 3/4 of the remaining space. i.e. if a 512-byte user program is downloaded, then the fill routine will use the last 384-bytes of the user area for its workspace. Due allowance for this should be made by the user program. (N.B. the space is only used if an "ESC w" is received, otherwise it is left untouched.)
The fill routine can fill quite complex shapes with as little as 128 bytes of workspace.

8. SVC features

This section covers the clock feature of the SVC in greater detail.

8.1. Software Clock

The SVC supports a software clock by decrementing an internal counter once per displayed frame (50 times a second). Every time the counter reaches zero it is reset to 50, and an ASCII clock is incremented. The ASCII clock is held as HH:MM:SS, and each pair of characters is incremented by a common routine. The gives the clock the idiosyncrasy that the hours will count up from 00 to 59 before resetting, a feature that will only be noticed by those who run their system overnight. On power-up the clock is zeroed, but subsequent resets will not disturb the current setting.

The Escape sequence <ESC> "t"..... is provided to support the clock. The various possibilities are listed below:

ESC t = X Y - defines where on the screen the clock will appear. The default position is at the right-hand end of the top line of the display, although this may be changed by sending the ESC t = X Y sequence, where X and Y are the desired screen position for the clock display. The parameters required in the positioning sequence (= X Y) are identical those of the "<ESC> = X Y" cursor addressing sequence.

ESC t HHMMSS - Initialises the clock display to HH:MM:SS. The characters represented by H,M and S should be ASCII numeric characters.

ESC t ? - results in the current contents of the clock being returned to the Host system as 6 ASCII characters in the form HHMMSS.

ESC t E - Enables the display of the clock. As a result, once per second immediately following the internal update of the clock, the clock is copied across into the display memory at the defined address. If the screen display is static, then the clock will just appear to 'tick' away like any digital clock. However, if the display is scrolling, then the clock will disappear off the top of the screen as the line it is written to scrolls. It will re-appear at its defined place when it is next updated, one second later. (This effect could be prevented by 'locking' the top line of the display. (See <ESC> "M".))

ESC t D - Disables the display of the clock. Note that this does not remove the clock from the display, it only stops the screen update of the clock. Thus there will be a stationary display on the screen until the screen is cleared or the line is scrolled off the top.

9. Caveats

9.1. Inadvertent requests

Some of the escape sequences request data from the Video card (e.g. <ESC> "?"). If one of these sequences is sent (or typed) by accident, (perhaps an object program is listed by mistake, and part of the code imitates one of the sequences), then the Video card could 'hang up' waiting for the requested bytes to be read. To prevent this occuring the routine that transfers bytes to the Host system also checks the incoming data buffer. If the Host system continues to send bytes to the Video card then the current output operation is aborted and the Video card returns to accepting input characters in the normal way.

9.2. Nested escape sequences

The SVC software accepts a limited depth (4) of nested escape sequences. However this only applies to the two character sequences (e.g. ESC A), and certain multiple character sequences (e.g. <ESC> =,S,R and T - in alpha mode). This is done to allow certain of the sequences to be typed on a keyboard attached to the SVC where the keyboard characters are read by the host system and then echoed back to the SVC. This is illustrated below:-

Characters transfered To SVC	Characters transfered From SVC	Action
ESC		Escape sequence starts.
K		Get a keyboard character.
	ESC	Escape is typed on the keyboard and returned to the Host. (ESC K terminates)
ESC		Host echoes character to SVC. (Escape sequence starts)
ESC		Host sends start of Keyboard input request. Previous ESC is stacked, new one starts
K		Get a keyboard character
	A	A is typed on the keyboard and returned to the Host. ESC K terminates and previous ESC reinstated.
A		Host echoes character to SVC. Alternate PCG now default, ESC A now terminates.

However if an escape sequence is actually started then another ESC character will not be recognised until the sequence is completed. (See below).

9.3. Escape sequences and BASIC

This section highlights a problem that could occur if the special features of the SVC are used from application programs under a high level language. It is unlikely that this problem will be met by anyone writing programs in assembly language, but it could occur in BASIC as it is easy to forget exactly what is happening at the lower systems level.

As mentioned above (in 9.2), once one of the various multiple escape sequences has started the SVC cannot recognise another escape character until the current sequence has finished. The reason for this is that the character <ESC> may well occur in the data being passed to the SVC and so no further checks for any control characters are made until the requested transfer is complete. The problem that could occur with Basic is illustrated below:-

```
1000 REM Define a striped character
1010 PRINT CHR$(27);"C\";   :REM Redefine character "\"
1020 FOR I=1 TO 16 : PRINT CHR$(176); : NEXT I
```

This would not work because between lines 1010 and 1020 the BASIC interpreter would poll the keyboard to see if the user had typed a Control/C to interrupt the program. As a result what the system would attempt to send to the SVC would be:-

```
   ESC  C \     ESC    k      176 176 176 ...
(line 1010) (Poll for ^C)   (line 1020)
```

Presented with this sequence the SVC would interpret ESC k as the first two bytes of the sixteen bytes that will make up the character definition (as they follow the lead-in ESC C \), and then wait for the next input character. However the Host system's keyboard routine thinks that it has just requested the keyboard status and so waits for a reply from the SVC (having sent the ESC k). The net result is that the system hangs up with the Host and the SVC each waiting for the other to send something. The only way out of this impass is to press "Reset". (Note that the character definition never actually gets sent.)

This problem can be avoided by restructuring the BASIC program so that the entire escape sequence is contained within one Print command:-

Use either
```
1010 PRINT CHR$(27);"C\";CHR$(176);CHR$(176);.....
```
or first put it all into a string:-
```
1010 P$=CHR$(27)+"C\"+CHR$(176)+CHR$(176)+.....
1020 PRINT P$;
```

9.4. System peculiarities

When sending strings of characters to the SVC as part of an escape sequence beware of the operating system/High level language!

For example the CONOUT BDOS function in CP/M always polls the keyboard (looking for a ^S) on every character that is output. To avoid this either include a version of PUTVID in your program, or use function 6 - direct console I/O.

With Microsoft BASIC it is advisable to use the WIDTH 255 statement to set the screen width. If this is not done you will find that BASIC automatically inserts the carriage return line feed pair (0DH 0AH) at the most inopportune moments. (A well known law states that it will be in the middle of an escape sequence, and at a point guaranteed to lock the system up). Also you will find that BASIC will expand the character 09 (Ascii TAB) to multiple spaces, and will replace 08 (Ascii Backspace) by the three-byte sequence 08H 20H 08H!

10. Keyboards

The SVC includes an I/O port to which a keyboard may be attached. The software supports two variants of keyboard, normal or extended. In both cases the keyboards are expected to produce 7-bit ASCII codes, but in the case of the extended keyboard a range of double-byte codes are used to distinguish the programmable function keys from the normal alphanumeric keys. An on-board switch is used to indicate which type of keyboard is connected. (See section 5.)

The hardware interface supports either a parallel or serial keyboard. The required specifications and connection information can be found in the optional SVC hardware manual.

10.1. The Gemini GM821 keyboard

The GM821 is a conventional 59-Key Ascii encoded keyboard that connects directly to the parallel SVC keyboard port. In addition to the conventional keys it includes four cursor control keys at the right hand end of the keyboard. Note that on this keyboard Cntrl/^ and Cntrl/| produce the codes for "delete line and scroll up" and "insert line" rather than the Shift/ of the other Gemini keyboards.

10.2. The Gemini GM827 and GM852 keyboards

These keyboards have additional keys in the form of a row of special function keys along the top of the keyboard, (labelled F0-F9 and EDIT), four cursor control keys, and a separate numeric pad to the right of the main keys. These additional keys may be programmed (via the SVC software) to return one or more characters to the host computer every time that they are pressed. In order that the SVC can distinguish these special keys the keyboard returns unique double-byte codes from these keys. The SVC software replaces each double-byte code by a single character or string of characters from an internal table. This table is held in the workspace ram of the SVC, and may be modified at any time, either by program, (using the "ESC f" sequence), or directly from the keyboard. On Reset an initial table is copied out of the SVC-MON EPROM into the RAM. The necessary information is given in appendix D for those able to program 2764 type EPROMs who wish to change the default strings.

The shift key may also be used in conjunction with these keys to produce another set of unique codes.

Each of these keys, with the exception of shift/EDIT can be redefined to produce any character or string of characters required. For example F0 could be set up to hold the string "pip a:=b:*.*[v]<CR>". The key definitions may be set up in two ways:-

a) By the User at the keyboard, and
b) By a program using an escape sequence.

10.2.1. Defining keys from the keyboard.

Typing shift/EDIT on the keyboard will draw the response

*** List/Edit a Function key ***

If a function key is now pressed, the current definition of that key is listed on the screen. All control codes in the string are displayed in the expanded form of ^<character> (e.g. a carriage return would appear as ^M). This is followed by the message:

*** List/Edit complete ***

The SVC monitor has put this information directly onto the screen, NOTHING HAS BEEN SENT TO THE HOST COMPUTER and it is totally unaware of what has happened.

If instead of hitting a function key shift/EDIT is pressed again the following string will appear:

*** Press the function key to be defined, then type in a string ***
*** followed by any function key ***

At this point you can select the function key you wish to redefine. Type it followed by the string you wish to enter. As you type in the string it will be echoed to the screen, once again with control characters being expanded to the form ^<character>. NOTE it is assumed that any character typed is to be part of the string, thus if you hit "backspace" ^H will appear on the screen and the control/H will be entered into the string. If you make a mistake you will have to start again.

The entry of a new definition is ended when any function key is pressed. (No recursive definitions are allowed!.) At this point the following messages will appear:

*** New definition entered ***
*** List/Edit complete ***

If no string was entered the function key will no longer return any characters, and if the key is "listed" the following message will appear:

*** Function key undefined ***
*** List/Edit complete ***

As above, the Host system IS TOTALLY UNAWARE of what is happening, and it is possible to re-define the keys at any time in this manner.

10.2.2. Defining keys by software.

A key may be redefined by software using the following escape sequence within a User program:

ESC f <code> <string> <byte with msb set> .....

where <code> is the unique code identifying a function key. <string> is the string of characters to be returned every time the key is pressed. The new definition is terminated when a byte with the msb set is encountered. If this byte is a legal keycode (81H-0BDH excluding 90H and 9BH - See Appendix C) then a new definition is started, if it is illegal (i.e. > 0BDH) then the escape sequence is terminated.

Two additional features are included in the escape sequence:
<ESC> <f> <d> or <ESC> <f> <D> will reset the key definitions to their default (or power-up) state.
<ESC> <f> <?> will cause the SVC to send to the Host the table of the current function key definitions. The table is terminated with the byte OFFH.
N.B. It is perfectly possible to enter a null string for a key definition and effectively disable it. (It will be ignored until redefined.)

If you get too carried away with your definitions you will see the message:

*** SVC internal error - table overflow ***

This should not normally happen as the table can use up to 512 bytes which gives an average of about 8 characters per key (assuming the numeric pad is redefined as well).

10.2.3. KEYCHAIN

The simple program KEYCHAIN can found on Gemini CP/M disks. (If you do not have a copy on your disk ask your local Gemini dealer for a copy of the .MAC and .COM files.) It is an extension of the earlier distributed program SAVEKEYS.

KEYCHAIN illustrates how a COM file can be set up to hold a particular set of key definitions. When run it invites the user to use the shift/EDIT mode to define all the function keys to his requirements. When this has been done the current function table is read and written away to disk along with a small program which will reload it. The source program format is for Microsoft's M80 assembler. Optionally the program will chain in another file once it has run.

Thus it is possible to easily set up files such as KPEN.COM and KWS.COM that could be executed before running programs such as PEN or WORDSTAR to customise the key settings appropriately.

The syntax for running KEYCHAIN is:

KEYCHAIN <filename> [<chainname>]

where <filename>.COM is the name of the file that the key definitions will be stored under, and <chainname> is an optional name that will be passed across to the CCP once <filename> has run. This will result in <chainname>.COM being loaded and executed (if present).

<chainname> may be omitted if no file is to be chained.

For example if you have a program CAD.COM that requires a set of custom function key definitions you can do the following:

1) Rename CAD.COM to CADPROG.COM i.e. REN CADPROG.COM=CAD.COM
2) Run KEYCHAIN to set up a key file i.e. KEYCHAIN CAD CADPROG

Now when you type CAD, CAD.COM will be executed and so set up the correct key definitions. This will then chain in CADPROG.COM, the original CAD program. As far as the user is concerned he still types in the same name to run the application program, and there is a short delay while the key definitions are set up before the main program is executed. (If, when the program terminates, it sends the sequence <ESC> "f" "D" the function keys will be returned to their normal (default) settings.)

A. Writing your own programs for the SVC

This Appendix is intended to give general guidelines to anyone who wishes to write programs that are intended to execute within the SVC.

The area currently available to user programs is from 0E400H to 0E7FFH, a total of 1k bytes. The user program is downloaded to the SVC by the <ESC> "L".. sequence. The lead in is followed by the size of the program (lo' byte, then hi byte), and then the program itself. This is a similar format to the <ESC> "W" command, but without the offset. The program is loaded into the workspace RAM starting at address 0E400H. Following completion of the load control is passed back to the SVC software. The downloaded program is only executed when the <ESC> "U" sequence is received, at which time a CALL is made to address 0E400H.

Note that this area is also used by the 'graphics fill' routine (see ESC "w"). The fill routine requires a reasonable amount of workspace if it is to handle complex shapes, and so it utilises this 1kbyte USER AREA within the SVC. If you download a program to this area the fill routine automatically adjusts itself to use 3/4 of the remaining space. i.e. if a 512-byte user program is downloaded, then the fill routine will use the last 384-bytes of the user area for its workspace. Due allowance for this should be made by the user program. (N.B. the space is only used if an "ESC w" is received, otherwise it is left untouched by the SVC software.)

A.1. General

On the card the vertical sync output of the CRTC is connected to the NMI line of the Z80B. As a result the processor is interrupted every 20ms. In response to the NMI the SVC software updates the cursor registers of the CRTC and also updates the software clock and various attributes. This interrupt can only be disabled by holding the CRTC permanently reset by writing a 0 to bit 3 of the control latch (address 0C000H). If you wish to leave the display running the following points should be observed:-

The NMI routine requires 8 bytes of stack.
Any routine that wishes to alter the internal registers of the CRTC should first synchronise to an NMI to prevent the loading sequence being corrupted. This is best done by executing a HALT instruction, exit from the Halt state being effected on receipt of an NMI by the Z80B.
The keyboard input runs under interrupts and requires 14 bytes of stack space.

A.2. Mode

The user program can be organised in two ways. One is to be totally independent of the SVC software, in which case memory and registers can be used in an indiscriminate fashion and return to the SVC software has to be via the Reset address of 0. The other is to respect certain registers (detailed below), in which case routines within the SVC software can be called, and a controlled Return can be made to the main program leaving the Screen display intact.

A.3. Stack

On entry to the user program a limited amount of stack space is available (about 10 bytes - due allowance has been made for the NMI routine). The current address at the top of the stack is the correct return address for the routine. So if this amount of stack is adequate the stack pointer can be left alone, and the program terminated by a RET instruction. If not the stack pointer should be saved and a local stack used for the routine, and the stack pointer reset before the final RET.

A.4. Registers

The alternate register set should not be altered. It contains certain values that are used by the Restart routines listed below. Register IY should not be altered. All other registers may be used.

A.5. Utility subroutines

The following subroutines may be called by the user program. They have been retained for compatibility with the IVC although not all are necessary for the SVC:-

RST 08H PUTSCR (Use if IVC compatibility required.)
Puts the character from register A to memory address (DE). In the IVC this was done immediately following a horizontal sync pulse and was used to provide transparent access to the screen memory or the programmable character generator. For the SVC - LD (DE),A - will suffice.

RST 10H GETSCR (Use if IVC compatibility required.)
Gets a character from (DE) and loads it into register A. This is done in a similar fashion to PUTSCR. For the SVC - LD A,(DE) - will suffice.

RST 18H SCAN (Use if IVC compatibility required.)
Scans for a waiting character from the Host system. If one is there it is transfered to the input buffer. (This routine is the one called occasionally by the scrolling routine within the IVC. The SVC does not use this function.)

RST 20H GETCHR
Gets the next character from the Host system. If the buffer is in use it gets the next character from there after adding any waiting one from the interface. The character is returned in the A register.

RST 30H PUTCHR
Transfers the character from the A register to the Host system.

The contents of the following workspace locations may be of interest to the program writer.

0E0D6	Start of Display
0E0DC	Current Cursor position
0E0E0	Screen width
0E0E2	Screen height

B. CRTC Information

Details of the CRTC registers can be located in data sheets on the 6845 display controller. Listed below are the details of the values programmed into the CRTC in response to the <ESC> 1, <ESC> 2, and <ESC> 4 sequences.

Register	80 Wide format	40 Wide format	Graphics format	
	Hex Value			
0	6F	37	37	; Horizontal total characters -1
1	50	28	20	; Horizontal displayed characters
2	58	2E	2B	; Horizontal sync position -1 (in character units)
3	7F	77	45	; Vsync/Hsync width
4	1E	1E	12	; V. character lines total -1
5	02	02	08	; V. scan lines adjust (raster lines)
6	19	19	10	; V. displayed character lines
7	1B	1B	11	; V. sync position
8	40	40	00	; Interlace and skew
9	09	09	0F	; Rasters per character line -1
10	48	48	48	; Cursor type & start raster
11	08	08	08	; Cursor end raster

Note that the horizontal sync width in each case has been set to a larger value than the broadcast standard. This has been done to ensure that a stable display is produced on most monitors (irrespective of quality) when the video on the screen is inverted. (<ESC> I.)

Registers 10 and 11 define the appearance of the cursor as shown below.

```
            msb           lsb
Register 10   . B P R R R R R     B P   Cursor Display Mode
                                  0 0   Non-blink
                                  0 1   Cursor not displayed
                                  1 0   Fast blink (16 field)
                                  1 1   Slow blink (32 field)
                  RRRRR  is the cursor start raster address

            msb           lsb
Register 11   . . . R R R R R

                  RRRRR is the cursor end raster address
```

For example to produce a solid slow-blinking character cell for the cursor the following values should be programmed:-

```
Register 10 set to . 1 1 0 0 0 0 0  ie 60H
Register 11 set to . . . 0 1 0 0 1  ie 09H
```

(The raster lines of a character are numbered from 0 to 9)

NOTE: Although you may program in a new setting for the appearance of the cursor, this may be immediately overwritten by the operating system. (e.g. The standard Gemini CP/M BIOS alters these registers.)

C. Function Key codes

Shown here are the hexadecimal codes associated with the various programmable keys on the GM827 and GM852 keyboards. They are NOT the codes returned to the Host system, but are the codes used in the "<ESC> f" sequence to identify individual keys. The codes are in the range 81H-0BDH. Note that the following codes do not occur and are treated as illegal by the "<ESC> f" sequence: 90H and 9BH

The function keys:

shifted	91	92	93	94	95	96	97	98	99	9A	XX
normal	81	82	83	84	85	86	87	88	89	8A	8B
KEY	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	EDIT

The Cursor control keys:

shifted	9C	9D	9E	9F
normal	8C	8D	8E	8F
KEY	Cursor keys			

The Numeric pad:

shifted	B0	B1	B2	B3
normal	A0	A1	A2	A3
KEY	7	8	9	+
shifted	B4	B5	B6	B7
normal	A4	A5	A6	A7
KEY	4	5	6	-
shifted	B8	B9	BA	AF
normal	A8	A9	AA	AE
KEY	1	2	3	E N
shifted	BB	BC	BD	T
normal	AB	AC	AD	E R
KEY	,	0	.	

D. Changing the default Function key settings

If required the table of default key definitions in the monitor EPROM (SVC-MON) can be changed. In order to do this you must be able to re-program a 2764 type EPROM. The existing EPROM should be copied to the memory of the programmer, and then the end of the program in the EPROM should be located. Currently this is around address 1900H-1A00H. Searching backwards from this point the copyright message "(c) dci software 1984" should be located. The default table starts immediately following this message.

The first four bytes of the table are:-

80 1B 90 1B ..... (In hexadecimal)

ON NO ACCOUNT MUST THESE BE CHANGED otherwise you will find that you have redefined the ESCAPE key (normal and shifted).

The new strings can be entered in a similar manner to those already there. The format is identical to that of the "ESC f..." sequence. (See section 10.2.2.)

E. Block Graphics

The <ESC> G sequence is used to set up a block graphics character set in the PCG. To obtain the block graphics each character cell is divided into six pixels, two units horizontally by three vertically. These pixels are turned on and off by setting and resetting six bits in the character occupying that character cell. To simplify matters the SVC will manipulate the block graphic characters directly in response to commands specifying a particular pixel. (<ESC> S,R, and T).

The block graphics characters are in the range 0C0H to 0FFH and are accessed by enabling the 'alternate character generator' attribute. (See <ESC> "a".) A character cell is divided up as shown below:

```
*---*---*
! 0 ! 3 !
*---*---*
! 1 ! 4 !
*---*---*
! 2 ! 5 !
*---*---*
```

where 0-5 represent the corresponding bit positions in the character:-

11XXXXXX

As the pixels are subdivisions of a standard character cell the resolution available with the block graphics depends upon the currently programmed screen size. It is Width*2 by Height*3. With 25 lines of 80 characters per line the resolution is 75 pixels vertically by 160 horizontally. With 25 lines of 40 characters per line the resolution is 75 pixels vertically by 80 horizontally.

F. PUTVID & GETVID Example

Shown below is a simple example of the use of PUTVID and GETVID.

```
; Simple demonstration program to get a character from
; the keyboard and to echo it to the screen.

escape    equ  1bh

loop:     ld   a,escape       ; Get a character from the keyboard
          call putvid
          ld   a,'K'          ; ESC K gets a character
          call putvid
          call getvid         ; Read the typed character
          cp   03             ; Was it a Control/C?
          jr   z,exit         ; Yes, break out of this loop
          call putvid         ; No, echo to the screen
          jr   loop           ; ...then repeat

;
; PUTVID - Transfer the character in <A> to the SVC
;
putvid:   push af             ; Save the character
pv0:      in   a,(0b2h)       ; Check "ready" flag
          rrca                ; Move flag to carry
          jr   c,pv0          ; Wait if buffer is full
          pop  af             ; Get the character back
          out  (0b1h),a       ; Send it out
          ret                 ; Done
;
; GETVID - Read a character from the SVC to <A>
;
getvid:   in   a,(0b2h)       ; Read the flag register
          rlca                ; Move flag to carry
          jr   c,getvid       ; Wait if the buffer is empty
          in   a,(0b1h)       ; ... else read the character..
          ret                 ; ...and return with it.
;
; Done - return to operating system
;
exit:     ...code to return goes here... (e.g. JP 0 for CP/M)
```
