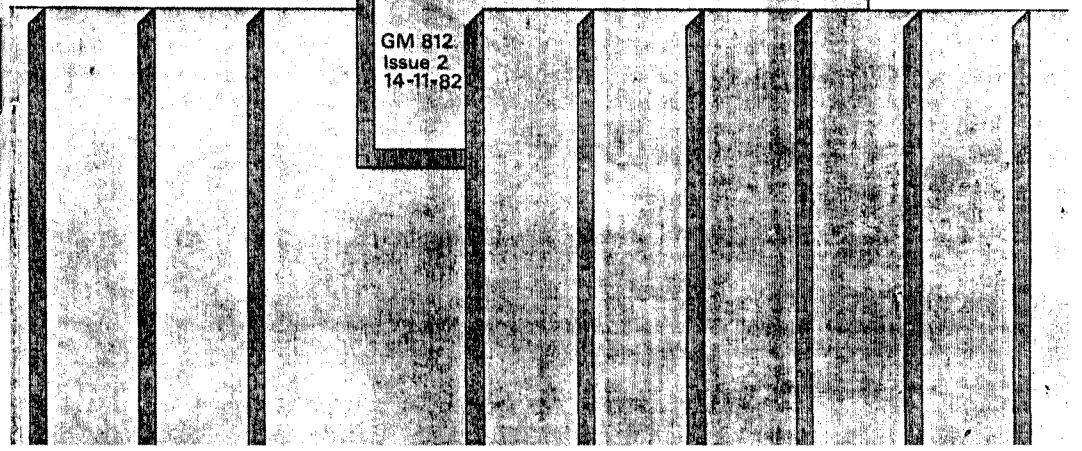
The Gemini MultiBoard Microsystem

Z80 IVC

80-BUS INTELLIGENT VIDEO CONTROLLER CARD

SOFTWARE MANUAL FOR IVC-MON V2

GM 812
Issue 2
14-11-82



CONTENTS

Section		page
	Command Summary	2
1.	General	4
2.	GM812 Hardware/Software	4
3.	On-board links	5
4.	GM812 Host Interface	6
5.	GM812 Control codes	7
5.1	Single byte codes	7
5.2	Escape sequences	9
6.	Caveats	15
6.1	Inadvertent requests	15
6.2	Nested escape sequences	15
6.3	Escape sequences and Basic	15
6.4	System peculiarities	16
7.	Keyboards	17
7.1	GM821 keyboard	17
7.2	GM827 keyboard	17
7.2.1	Defining Function keys from the keyboard	17
7.2.2	Defining Function keys from a program	18
Appendix		
1	User programs	20
2	CRTC programming	22
3	Function key codes	23
4	Changing default settings	24
5	"SAVEKEYS" listing	25
6	Block graphics	31
7	Example	32

Command Summary

NOTE:
1. These are the control codes that the IVC responds to. They are normally sent to the IVC by a User program or the Host's operating system. (See section 4 - PUTVID).
2. Programs that request information from the IVC will have to read the information back from the IVC. (See section 4 - GETVID).
3. It may not be possible to issue commands to the IVC direct from the keyboard attached to a system, as the operating system may modify the characters typed. (eg the standard CP/M line input routine would echo "^[" to the IVC if the ESCAPE key were pressed, rather than the ESCAPE code. - However note that this problem can be solved in Gemi CP/M systems by selecting "EDIT mode", where all characters are echoed to the IVC exactly as typed).
(Also see section 6.2 - Nested Escape sequences).

General

07	^G	Bell
08	^H	Backspace
0A	^J	Linefeed
0D	^M	Carriage return

Cursor movement

1C	^\	Cursor left
1D	^]	Cursor right
1E	^^	Cursor up
1F	^_	Cursor down
1B 3D...	<ESC> "=" RR CC	Cursor addressing

Additional cursor operations

1B 3F	<ESC> "?"	Return cursor coordinates and character
1B 44	<ESC> "D"	Delete cursor
1B 45	<ESC> "E"	Enable cursor
1B 59..	<ESC> "Y"..	Define cursor type

Screen editing

0B	^K	Delete line and scroll up
0E	^N	Insert line
16	^V	Delete character in line
17	^W	Insert character in line
1A	^Z	Clear screen

1B 16	<ESC> ^V	Delete character in screen
1B 17	<ESC> ^W	Insert character in screen
1B 25	<ESC> "%"	Delete to end-of-screen
1B 2A	<ESC> "*"	Delete to end-of-line
1B 5A	<ESC> "Z"	Return contents of current line

Screen format

1B 31	<ESC> "1"	Select 80 wide format
1B 32	<ESC> "2"	Select 48 wide format
1B 33	<ESC> "3"	Select user-defined format
1B 46..	<ESC> "F"..	Define user format
1B 42	<ESC> "B"	Blank screen
1B 56	<ESC> "V"	Video on (unblank screen)
1B 49	<ESC> "I"	Video Invert screen
1B 4A	<ESC> "J"	Video normal screen
1B 41	<ESC> "A"	Alternate character generator is the default
1B 4E	<ESC> "N"	Normal character generator is the default
1B 4D	<ESC> "M"	Memory lock on
1B 4F	<ESC> "O"	Memory lock off

Character set

1B 43..	<ESC> "C"..	Define character
1B 63..	<ESC> "c"..	Define character set
1B 47	<ESC> "G"	Construct block graphics character set
1B 48	<ESC> "H"	Duplicate lower character set in upper but invert
1B 68	<ESC> "h"	Duplicate lower character set in upper c/gen

Block graphics

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

Keyboard

1B 66..	<ESC> "f"	Define Function Key(s)
1B 6B	<ESC> "k"	Test Keyboard status
1B 4B	<ESC> "K"	Get Keyboard character
1B 58	<ESC> "X"	Get one line of input

Miscellaneous

1B 57..	<ESC> "W"..	High speed Write to display
1B 4C..	<ESC> "L"..	Load user program
1B 55	<ESC> "U"	Execute user program
1B 50	<ESC> "P"	Return light pen coordinates
1B 76	<ESC> "v"	Return version number

1. GENERAL

The Gemini Intelligent Video Controller (IVC) is an 80-Bus compatible 8"x8" card that handles the character display for a Gemini (or Nascom) computer system. The card contains its own on-board processor (a Z80A) and communicates to the host system via three I/O ports. As well as handling the video display, the IVC can optionally support a keyboard as well, providing full buffering and permitting "type ahead". In conjunction with the Gemini GM827 keyboard it also supports programmable function keys.

By using its own onboard processor the IVC 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.

The IVC 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 IVC.

2. GM812 Hardware/Software

In order to provide an interference free display the control software has been written to only access the display memory in the line blanking and frame blanking intervals. (ie Only when the CRT controller is not displaying characters on the screen). There is no real penalty involved in taking this approach when considering the character rate to the screen (one character every 64us - equivalent to about 156k baud), but there is a penalty in the time it takes to scroll the screen. (The Z80 block move instruction LDIR cannot be used). However to ensure that the Host system is not unduly slowed down the Video card incorporates an internal buffer, and when a relatively long process (such as scrolling) is in progress, characters are still accepted from the Host system and are queued in this buffer. Only when the buffer is full will the Host system have to wait. (The buffer size is 64 bytes).

The character move rate in scrolling has been maximised as far as possible, and the figures used are based on the Video card Z80A running at 4MHz. If the Video card is run at a lower clock rate then some interference can be expected to occur down the left side of the display.

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 1.

The Video card will also support a keyboard. With the Keyboard option enabled, the attached keyboard is checked once per displayed frame. (ie once every 20ms). If any key has been pressed the 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 is scanned once per frame 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 when it requests them.

The IVC supports two character generators. As supplied one of the character generators is an EPROM, the other a RAM. The character set held by the RAM has to be set up every time the card is powered up and can be easily changed at any time under software control, and so it is referred to as a programmable character generator or PCG. The EPROM occupies the lower half of the character generator address space and generates the characters for codes in the range 00-7F (hexadecimal). The PCG occupies the upper half of the address space and generates the characters for the codes 80-FF. Thus the value of the most significant bit of a character governs which character generator is used to display it. Several commands are provided to support the use of the PCG.

3. On-board Links

Two user links are provided on the card. These are used to set up the Video card on Power-up.

Link 2 : If present the IVC supports a standard Ascii keyboard such as the Gemini GM821 59-key keyboard.
If absent the IVC supports the Gemini GM827 87-key keyboard with programmable function keys. See section 7 for details.

Link 3 : If absent the on-board Keyboard port is ignored.
If present the on-board Keyboard port is enabled.

The other links on the board should be correctly set. The following are assumed:-

A — B

Vsync connected to Z80A NMI input (Link four)
Z80A clock set to 4MHz (Link six) B — C

A separate Hardware manual on GM812 is available. It is not essential for the user, but for those interested in the internal workings of the IVC it is available through your supplier.

LINK 5 PRESENT.

4. GM812 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 Z80A 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).
NB The latter assumes that the Host's software does not reset the Video board while re-initialising the host system.

Port	Dir.	Function
OB1H	R/W	Data transfer to/from Video card.
OB2H	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.
OB3H	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 IVC. (See also Appendix 7).

```
                ;
                ;  PUTVID - Transfer the character in A to the Video card
                ;
F5              PUTVID: PUSH   AF          ;Save character
DB B2           PVO:    IN     A,(OB2H)    ;Read flags
OF                      RRCA               ;Flag to carry
38 FB                   JR     C,PVO       ;Loop if buffer still full
F1                      POP    AF          ;Get character back
D3 B1                   OUT    (OB1H),A    ;Put in buffer
C9                      RET                ;Done

                ;
                ;  GETVID - Read a character from the Video card
                ;           to the A register
                ;
DB B2           GETVID: IN     A,(OB2H)    ;Read flags
07                      RLCA               ;Flag to carry
38 FB                   JR     C,GETVID    ;Loop if buffer empty
DB B1                   IN     A,(OB1H)    ;Read character
C9                      RET                ;Return with it
```


5. GM812 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 IVC responds to. They are normally sent to the IVC by a User program or the Host's operating system. (See section 4 - PUTVID).
2. Programs that request information from the IVC will have to read the information back from the IVC. (See section 4 - GETVID).
3. It may not be possible to issue commands to the IVC direct from the keyboard attached to a system, as the operating system may modify the characters typed. (eg the standard CP/M line input routine would echo "^[" to the IVC 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 IVC exactly as typed).
   (Also see section 6.2 - Nested Escape sequences).

5.1 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 backspace (Hex code 08) is generated by typing control/H which is shown as ^H. The cursor movement codes correspond to those generated by the Gemini GM821 and GM827 keyboards. The insert/delete line and insert/delete character codes correspond to combinations of shift and control 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	Kybd	Function
07	^G	Bell. Bit 4 on the control port (IC25 pin 19) is set and then cleared. This signal may be used to trigger an audible alarm of some kind, or may be ignored. (See Hardware manual).

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 [LINEFEED - GM821 only]
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 [Control/↑ on GM821, Shift/↑ on GM827]
Delete Line and Scroll up. The line in which the cursor is currently positioned is deleted, and the following lines are scrolled up the screen, with a blank line appearing at the bottom. If the cursor is already 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 [Control/↓ on GM821, Shift/↓ on GM827]
Insert line. The line currently 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.

5.2 Escape Sequences

Code sequence	Function
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 48 wide format. The inbuilt 48-character wide screen format is selected. Note that for a readable display the variable dot clock (RV1 see Hardware manual) must be set appropriately.
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 3D..... [<ESC> "=" RR CC]
Cursor addressing. The cursor is positioned to row RR and column CC. RR and CC are offset by 20H. ie 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 3F [<ESC> "?"]
Cursor coordinates. The current X,Y position of the cursor is returned via the IVC 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 IVC 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 PCG. 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 character position in the top most character generator. However if both character generators are programmable, then setting the msb of the character (byte XX above), will result in it being loaded to the lower character generator. (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 2 for appropriate values) and ZZ is the byte that selects either the crystal oscillator or the variable oscillator for the dot clock. If ZZ is OFFH then the crystal oscillator is selected. If it is 00 then the variable oscillator is selected. 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 PCG. The block-graphics character set is constructed in the upper character generator at character addresses 0C0H-0FFH. (See Appendix 6)

1B 48 [<ESC> "H"]
Copy the complement of the contents of the lower character generator to the upper 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. If the keyboard is not enabled a byte of 0 will be returned. (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 1).

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

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

1B 55 [<ESC> "U"]
Execute User program. The program previously loaded by the <ESC> "L" command is executed. (See appendix 1 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, and, if required, without waiting for display blanking before writing to the screen. 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), signifies if the load is to be 'transparent' (ie characters are only moved to the display during the blanking intervals), or 'direct'. If the byte MM is equal to 'T' (54H) or 't' (74H) the load will be done transparently, and as a consequence will not be as fast. Any other value results in the load being immediate, and as a result interference will occur on the screen unless the display is blanked.

1B 58 [<ESC> "X"]
Keyboard line input. Obtain one line of input from the keyboard connected to the IVC. 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, (ie On-screen editing can be used), and it is only when the code for "carriage return" (ODH) is detected that the mode terminates. If the keyboard is not enabled a single byte of ODH is returned.

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 3)

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 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 upper or lower one. If GG=0 then the character set is loaded to the upper character generator, if it is non-zero then the load is to the lower 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 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 7.

1B 68 [<ESC> "h"]
Copy the contents of the lower character generator to the upper character generator. As a result setting bit 7 of a character will have no visible effect. However this allows small changes to be easily made to the standard character set by modifying a few characters (see <ESC> "C"), and using <ESC> "A" to select the new set as the default character set.

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 OFFH is returned. The actual characters themselves are obtained by using the sequence <ESC> "K".

1B 76 [<ESC> "v"]
Return version number. The card returns the version number of the IVC software. A single byteis returned, with 10H representing version 1.0, 11H representing 1.1, ... 20H representing version 2.0 and so on.

6. Caveats

6.1 Inadvertent requests

Some of the escape sequences request data from the Video card. (eg <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.

6.2 Nested escape sequences

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

Characters transfered To IVC	Characters transfered From IVC	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 IVC. (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 IVC. 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).

6.3 Escape sequences and BASIC

This section highlights a problem that could occur if the IVC is used in conjunction with software that assumes the IVC also supports a keyboard. 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 6.2), once one of the escape sequences C,c,F,L,W,Y has started the IVC 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 IVC 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 IVC would be:-

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

Presented with this sequence the IVC 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 IVC (having sent the ESC k). The net result is that the system hangs up with the Host and the IVC 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$;
```

6.4 System peculiarities

When sending strings of characters to the IVC 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 (ODH OAH) 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!

7. KEYBOARDS

The IVC includes an 8-bit parallel port to which a keyboard may be attached. The interface is for an Ascii encoded keyboard that presents 7-bits of data, together with a strobe pulse. The software will handle two keyboard variants, the selection of the appropriate one being determined by the state of link 2 (See section 2).

7.1 The Cherry Keyboard (Part no GM821)

The Cherry Keyboard is a conventional 59-Key Ascii encoded keyboard that connects directly to the IVC keyboard port. In addition to the conventional keys it includes four cursor control keys at the right hand end of the keyboard.

7.2 The Rotec Keyboard (Part no GM827)

The Rotec keyboard has 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 IVC software) to return one or more characters to the host computer every time that they are pressed. In order that the IVC can distinguish these special keys the Rotec keyboard returns unique double-byte codes from these keys. The IVC 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 IVC, 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 IVCMON EPROM into the ram. The necessary information is given in appendix 4 for those able to program 2732 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 re-defined 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 program using an escape sequence.

7.2.1 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> (eg a carriage return would appear as ^M). This is followed by the message:

*** List/Edit complete ***

The IVC 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 andthe 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.

7.2.2 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-OBDH excluding 90H and 9BH - See Appendix 3) then a new definition is started, if it is illegal (ie > OBDH) 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 IVC to send to the Host the table of the current function key definitions. The table is terminated with the byte OFFH.

NB 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:

*** IVC 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).

The simple routine SAVEKEYS is supplied (see appendix 5). This is for setting up COM files holding particular sets of key definitions. 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.

APPENDIX 1
Writing your own programs for the IVC

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

The area currently available to user programs is from 0E400H to 0E7FFH, a total of 1k bytes. The user program is downloaded to the IVC by the <ESCAPE> "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 IVC software. The downloaded program is only executed when the <ESC> "U" sequence is received, at which time a CALL is made to address 0E400H.

General

On the card the vertical sync output of the CRTC is connected to the NMI line of the Z80A. As a result the processor is interrupted every 20ms. In response to the NMI the IVC software updates the cursor registers of the CRTC and also scans the Keyboard port for any characters. 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 6 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 Z80A.

Mode

The user program can be organised in two ways. One is to be totally independent of the IVC software, in which case memory and registers can be used in an indiscriminate fashion and return to the IVC 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 IVC software can be called, and a controlled Return can be made to the main program leaving the Screen display intact.

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.

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.

Utility subroutines
The following subroutines may be called by the user program:-

RST 08H PUTSCR
Puts the character from register A to memory address (DE). This is done immediately following a horizontal sync pulse and is used to provide transparent access to the screen memory or the programmable character generator.

RST 10H GETSCR
Gets a character from (DE) and loads it into register A. This is done in a similar fashion to PUTSCR.

RST 18H SCAN
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).

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

APPENDIX 2
CRTC Information

Details of the CRTC registers are given in the GM812 Hardware manual. Listed below are the details of the values programmed into the CRTC in response to the <ESC> 1 and <ESC> 2 sequences.

Register	80 Wide format	48 Wide format	
0	7F	4C	; Horizontal total characters -1
1	50	30	; Horizontal displayed characters
2	63	3C	; Horizontal sync position -1 (in character units)
3	7F	79	; Vsync/Hsync width
4	1E	1E	; V. character lines total -1
5	02	02	; V. scan lines adjust (raster lines)
6	19	19	; V. displayed character lines
7	1B	1B	; V. sync position
8	A0	A0	; Interlace and skew
9	09	09	; Rasters per character line -1
10	48	48	; Cursor type & start raster
11	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)

APPENDIX 3
Function Key codes

Shown here are the hexadecimal codes associated with the various programmable keys on the GM827 keyboard. 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**	**ENTER**	
shifted	BB	BC	BD		
normal	AB	AC	AD		
KEY	**,**	**0**	**.**		

APPENDIX 4
Changing the default Function key settings

If required the table of default key definitions in the monitor EPROM can be changed. In order to do this you must be able to re-program a 2732_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 0C00H. Searching backwards from this point the copyright message "(c) dci software 1982" 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 7.2.2)

APPENDIX 5
Savekeys

Given here is the source of the SAVEKEYS program that allows a user to save and reload a specific setting of the function keys. 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 disc along with a small program which will reload it. The program format is for Microsoft's M80 assembler.

If you have a Gemini Galaxy with the expanded keyboard this program and its source may be found on the CP/M system disk.

```
        .z80
        title SAVEKEYS program for the IVC & ROTEC keyboard
        page    62
;       This program enables people to set up a custom set
;       of strings for the programmable function keys on the
;       ROTEC keyboard.
;
;       Invoked by "savekeys <name>" where <name>.com will be a
;       an executable program that will automatically set up
;       the Key definitions.

clear   equ     1ah
escape  equ     1bh
cr      equ     0dh
lf      equ     0ah

bdos    equ     5
deffcb  equ     5ch

openf   equ     15
close   equ     16
delfil  equ     19
write   equ     21
create  equ     22
setdma  equ     26

signon: defb    " Function Key File Set-up program Version 0.1",cr,lf,lf
        defb    "$"
go:     defb    " Use the EDIT key on the Keyboard to set up the wanted"
        defb    cr,lf
        defb    " key definitions. When you have every key defined "
        defb    "in the",cr,lf
        defb    " way you want hit the RETURN key"
crlf$:  defb    cr,lf,lf,'$'
done:   defb    'Key file written$'
noname: defb    'You have forgotten to include the name of the file'
        defb    ' that you',cr,lf
        defb    'wish to save the function key settings in$'
ambig:  defb    'Ambiguous file names are not allowed$'
exists: defb    'The file already exists - shall I delete it?$'
cantd:  defb    'I cannot delete the file$'
cantcr: defb    'I cannot create the file$'
cantcl: defb    'I cannot close the file$'
werr:   defb    'Disc is full or a Write error occurred$'

savesp: defs    2
        defs    32
stack:

;       CR LF

crlf:   ld      de,crlf$

;       Print a string
```



```
print:  ld      c,9
        jp      5

;       Getvid & Putvid

getvid: in      a,(0b2h)
        rlca
        jr      c,getvid
        in      a,(0b1h)
        ret

putvid: push    af
pv0:    in      a,(0b2h)
        rrca
        jr      c,pv0
        pop     af
        out     (0b1h),a
        ret

;       Start point - Sign on

start:  ld      (savesp),sp     ;Save the stack pointer
        ld      sp,stack
        ld      de,signon       ;Sign on
        call    print

;       Now check that a file has been specified & make .COM type

        ld      hl,deffcb+1
        ld      a,(hl)          ;Look for a name
        cp      ' '
        ld      de,noname       ;(In case none)
        jp      z,abort         ;Abort if so
        ld      hl,deffcb+9     ;Force extension to..
        ld      (hl),'C'        ;..COM
        inc     hl
        ld      (hl),'O'
        inc     hl
        ld      (hl),'M'
        inc     hl
        ld      (hl),0
```

```
;       Check the name is not ambiguous

        ld      hl,deffcb
        ld      bc,8
        ld      a,'?'
        cpir
        ld      de,ambig
        jp      z,abort         ;Abort if ambiguous

;       See if the file already exists

        ld      de,deffcb
        ld      c,openf         ;Try to open it
        call    bdos
        inc     a               ;There?
        jr      z,nofile        ;No,skip
        ld      de,exists
        call    print           ;Delete it?
wkey:   ld      c,6
        ld      e,0ffh
        call    bdos            ;Get reply
        and     5fh
        jr      z,wkey          ;Wait if none
        cp      'Y'             ;Yes?
        jp      nz,exit         ;No, stop
        call    crlf
        ld      de,deffcb
        ld      c,delfil
        call    bdos            ;Delete it
        inc     a               ;succesful?
        ld      de,cantd        ;(In case not)
        jp      z,abort

;       No file - try to Create it

nofile: ld      de,deffcb
        ld      c,create
        call    bdos
        inc     a               ;Ok?
        ld      de,cantcr
        jp      z,abort

;       Get the keyboard bit done

        ld      de,go
        call    print
wait:   ld      e,0ffh
        ld      c,6
        call    bdos
        cp      3               ;Abort?
        jp      z,exit
        cp      cr
        jr      nz,wait
```



```
;       Now read the full table out into memory

        ld      a,escape
        call    putvid
        ld      a,'f'           ;Function
        call    putvid
        ld      a,'?'           ;Send table
        call    putvid
        ld      hl,ktable       ;Store here
rloop:  call    getvid          ;Get the byte
        ld      (hl),a          ;Put in the table
        inc     hl
        cp      0ffh            ;was it the end?
        jr      nz,rloop        ;no, carry on
        ld      de,keyprog-1    ;Compute length to save
        or      a
        sbc     hl,de
        add     hl,hl           ;H now holds no. of sectors
        ld      b,h             ;Move to B
        inc     b               ;Adjust

;       Now try to Write the file out

        xor     a               ;Clear NR
        ld      (deffcb+32),a
        ld      de,keyprog      ;start here
wloop:  push    bc
        push    de
        ld      c,setdma
        call    bdos
        ld      de,deffcb
        ld      c,write
        call    bdos
        pop     hl              ;Get address back
        pop     bc
        or      a               ;Successful write?
        ld      de,werr
        jp      nz,abort
        ld      de,128          ;Update address
        add     hl,de
        ex      de,hl
        djnz    wloop           ;Loop if more

;       File written - now close it

        ld      de,deffcb
        ld      c,close
        call    bdos
        inc     a               ;Any error?
        ld      de,cantcl
        jr      nz,exit         ;No,done

        ld      de,done
abort:  call    print           ;Print the error message
exit:   call    crlf
```

```
        ld      sp,(savesp)
        ret

keyprog:-
; Program-to be written to disc
        .phase  100h    ;Loads to the TPA
        ld      a,escape        ;Load the new table.
        call    chrout
        ld      a,'f'
        call    chrout
        ld      hl,table
wrloop: ld      a,(hl)
        call    chrout
        inc     hl
        cp      0ffh
        jr      nz,wrloop
        ret

;       Character o/p via direct console I/O

chrout: push    af
        push    hl
        cp      0ffh            ;can't o/p 0ffh
        jr      nz,ch
        ld      a,0e0h
ch:     ld      e,a
        ld      c,6
        call    5
        pop     hl
        pop     af
        ret

table:
        .dephase
ktable:

;       SAVEKEYS will read the existing table in to here

        end     start
```


APPENDIX 6
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 IVC 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. 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 48 characters per line the resolution is 75 pixels vertically by 96 horizontally.

APPENDIX 7
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 IVC
;
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 IVC 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... (eg JP 0 for CP/M)
```