SUPPLEMENTAL TECHNICAL REFERENCE MATERIAL

APPLICATION NOTE: ØØ2

Revision Ø

COPYRIGHT

For information contact:

VICTOR Publications
38Ø El Pueblo Road
Scotts Valley, CA 95Ø66
(4Ø8) 438-668Ø

TRADEMARKS

NOTICE

VICTOR makes no representations or warranties of any kind whatsoever with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. VICTOR shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this publication or its contents.

VICTOR reserves the right to revise this publication from time to time and to make changes in the content hereof without obligation to notify any person of such revision or changes.

First VICTOR printing March 1983.

CONTENTS

CONTENTS
continued

CHAPTER 1

Victor 9000 System Overview

1.1 Computer

The Victor 9000 computer is based upon the Intel 8088 16-bit microprocessor. This processor chip is directly related to the Intel 8086 16-bit microprocessor, but with two subtle differences:

8088	8086
8-bit data bus	16-bit data bus
4 instruction look-ahead	6 instruction look-ahead

The major difference, the 8-bit data bus, has some effect on the relative abilities of the two chips; the main difference is that while the 8086 can load an entire 16-bit word of data directly, the 8088 has to load two 8-bit bytes to achieve the same result - the outcome of which being that the 8088 processor is a little slower than the 8086. The loss of speed, however, is balanced by the fact that the cost of the main circuit board and add-on boards are lower than for the wider 8086 requirement. This means that the end-user will have the best cost/performance ratio for a 16-bit computer.

1.2 Memory

The Victor 9000 has a maximum memory capacity of 896 kilobytes of **R**andom **A**ccess **M**emory or "RAM" (a measure of a computer's internal storage capacity; a "kilobyte" is 1,024 bytes). A byte is able to store one character of data - thus the Victor 9000, with full 896k memory capacity is able to hold, internally, nearly 1 million characters - compare this figure with the older Z80 or 6502 computers that have a maximum memory capacity of less than 70,000 characters or 64k bytes of RAM.

1.3 Disk System

The Victor 9ØØØ has several integral disk configurations available; these are:

- Twin single-sided 6ØØk bytes per drive 5 1/4-inch minifloppies, giving a total capacity of 1.2Mbytes (1,2ØØkbytes) available on-line.

- Twin double-sided 1.2M bytes per drive 5 1/4-inch minifloppies, giving a total capacity of 2.4Mbytes (2,4ØØkbytes) available on-line.

- Single 1ØM byte hard disk (Winchester) plus a single double-sided 1.2M byte 5 1/4-inch minifloppy, giving a total capacity of 11.2Mbytes (11,2ØØkbytes) available on-line.

Future disk systems will include an external 1ØMbyte hard disk (Winchester) that will allow expansion of any of the above systems by a further 1Ø,ØØØk bytes.

Although the Victor 9ØØØ uses 5 1/4-inch minifloppies of a similar type to those used in other computers, the floppy disks themselves are not readable on other machines, nor can the Victor 9ØØØ read a disk from another manufacturers machine. The Victor 9ØØØ uses a unique recording method to allow the data to be packed as densely as 6ØØkbytes on a single-sided single-density minifloppy; this recording method involves the regulation of the speed at which the floppy rotates, explaining the fact that the noise from the drive sometimes changes frequency.

1.4 Display System

The display unit swivels and tilts to permit optimum adjustment of the viewing angle, and the unit incorporates a 12-inch antiglare screen to prevent eye strain. The display, in normal mode, is 25 lines, each line having 8Ø columns. Characters are formed, in normal mode, in a 1Ø-x-16 font cell, providing a highly-readable display. The screen may be used in high-resolution mode, providing a bit-mapped screen with 8ØØ-x-4ØØ dot matrix resolution. The high-resolution mode is available only under software control, there is no means of simply "switching" in to high-resolution. Victor Technologies has provided software to allow full use of the screen in high-resolution mode in the Graphics Tool Kit.

Character sets are "soft" - that is they may be substituted for alternative character sets of the users choice, or creation. Only one 256-character character set may be displayed on the screen at one time - multiple character sets cannot, currently, be displayed simultaneously - but this feature may well become available in the future. Character set manipulation software is available in both the Graphics and Programmers Tool Kits.

1.5 Keyboard

Several different types of keyboards are offered. Each keyboard is a separate, low-profile module with an optional palm rest for ease of use. Every key is programmable, permitting the offering of a National keyboard in each country in which it is marketed. As a result, the keyboard can be customized to satisfy the requirements of foreign languages and so that striking a key enters a character or predetermined set of commands.

Keyboards are as soft as the character sets - this allows a keyboard to be generated to match a newly created or special character set. Each key on the keyboard has three potential states; the unshifted, shifted and alternate. The unshifted mode is accessed when the shift key is not depressed along with the desired key; the shifted mode is accessed when the shift key is depressed along with the desired key; and the alternate mode is accessed when the ALT key is depressed along with the desired key. Keyboard manipulation software is available in both the Graphics and Programmers Tool Kits.

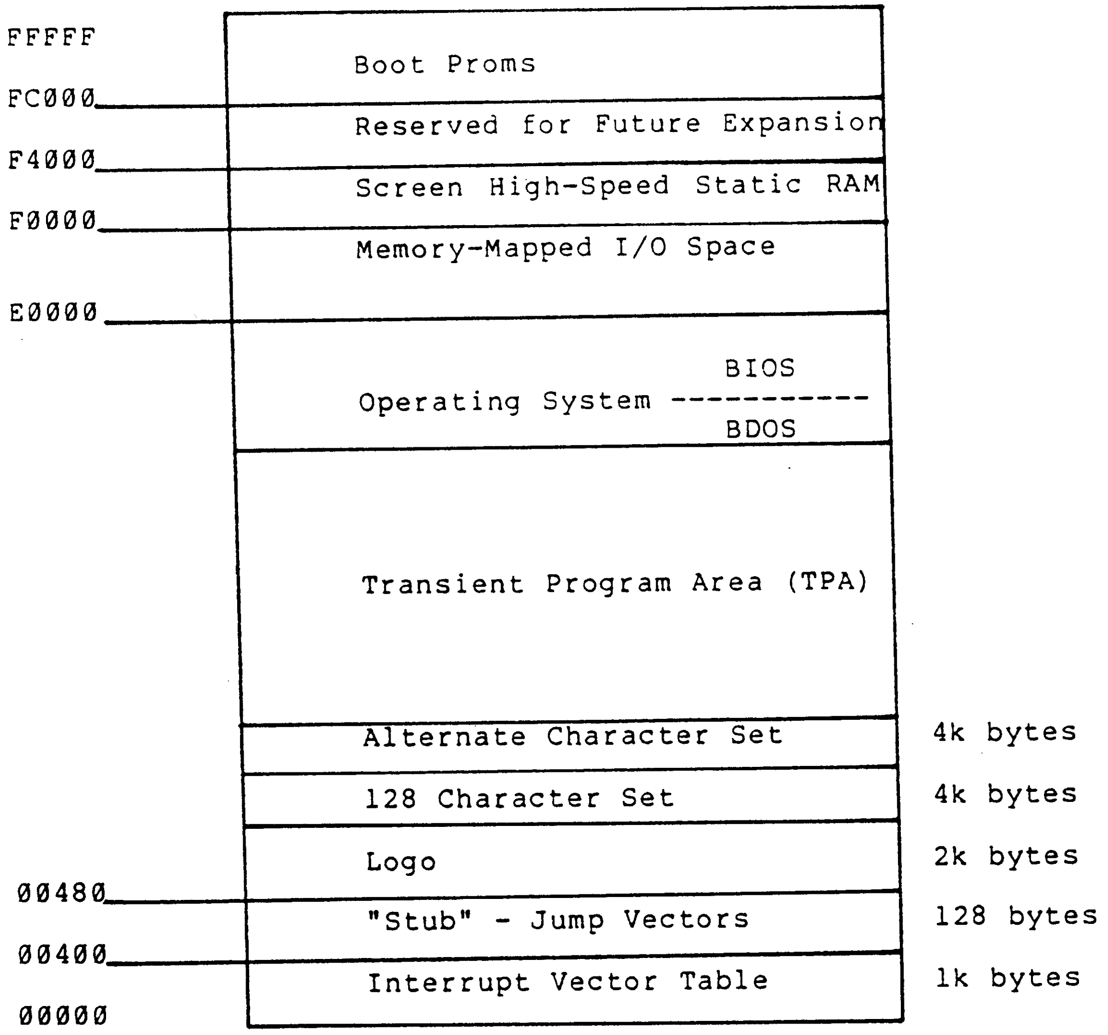
1.6 Memory Map

The Victor 9ØØØ is currently supplied with two major disk operating systems; CP/M-86 from Digital Research, and MS-DOS from Microsoft. Athough these two operating systems appear superficially similar, they are quite different in their operation, program interfacing techniques, and their memory structure. The following diagrams are the memory maps for CP/M-86 and MS-DOS; you will notice that some aspects of the machine never change, such as the screen RAM and interrupt vector locations, these areas are hardware defined, and as such never alter. The memory maps for MS-DOS and CP/M-86 are not fixed in the Victor 9ØØØ, thus some of the elements of the map will not be specific; this is not to be deliberately vague, but improvements to the performance aspects of the software do take place forcing the diagrams to be unspecific to some degree.

1.6.1 Memory Map -- MS-DOS Operating System

FFFFF		
	Boot Proms	
FC000		
	Reserved for Future Expansion	
F4000		
	Screen High-Speed Static RAM	
F0000		
	Memory-Mapped I/O Space	
E0000		
etc. 256k=3FFF0 128k=1FFF0	Operating System BIOS ------------ MS-DOS	
	Command - Resident Portion	
	Command - Transient Portion	
	Transient Program Area (TPA)	
	Alternate Character Set	4k bytes
	128 Character Set	4k bytes
	Logo	2k bytes
00480		
	"Stub" - Jump Vectors	128 bytes
00400		
	Interrupt Vector Table	1k bytes
00000		

1.6.2 Memory Map -- CP/M-86 Operating System



CHAPTER 2

Display Driver Specifications

2.1 Overview

The display system in the Victor 9000 is, like so much of the machine, soft. The operating system BIOS contains the Zenith H-19 video terminal emulator, which is an enhanced control set of the DEC VT52 crt. The BIOS takes all ASCII characters received and either displays them or uses their control characteristics. The control characters ØØhex (ØØdecimal) thru 1Fhex (31decimal) and 7Fhex (127decimal) are not displayed under normal circumstances. The non-display characters previously discussed, plus those characters having the high-bit set, being 8Øhex (128decimal) through FFhex (255decimal), may be displayed on the screen under program control, but extensive use of these characters is easier with the Victor Technologies character graphics utilities.

Most of the control characters act by themselves; for example, the TAB key (Control I, Ø9hex, Ø9decimal) will cause the cursor to move to the right to the next tab position. For more complex cursor/screen control the multiple character escape sequences should be used. The control characters, and the escape sequences are fully described below.

2.2 Screen Control Sequences

Single Control Characters

Bell (Control G, Ø7hex, Ø7decimal - ASCII BEL)
This ASCII character is not truly a displaying character, but causes the loudspeaker to make a beep.

Backspace (Control H, Ø8hex, Ø8decimal - ASCII BS)
Causes the cursor to be positioned one column to the left of its current position. If at column 1, it causes the cursor to be placed at column 8Ø of the previous line; if the cursor is at column 1, line 1, then the cursor moves to column 8Ø of line 1.

Horizontal Tab (Control I, Ø9hex, Ø9decimal - ASCII HT)
Positions the cursor at the next tab stop to the right. Tab stops are fixed, and are at columns 9, 17, 25, 33, 41, 49, 57, 65, and 72 through 8Ø. If the cursor is at column 8Ø, it remains there.

Line Feed (Control J, ØAhex, 1Ødecimal - ASCII LF)
Positions the cursor down one line. If at line 24, then the display scrolls up one line. This key may be treated as a carriage return -- see ESC x9.

Carriage Return (Control M, ØDhex, 13decimal - ASCII CR)
Positions the cursor at column 1 of the current line. This key may be treated as a line feed -- see ESC x8.

Shift Out (Control N, ØEhex, 14decimal - ASCII SO)
Shift out of the standard system character set, and shift into the alternative system character set (Character set 1, G1). This gives the ability to access and display those characters having the high-bit set - being those characters from 8Øhex (128decimal) through FFhex (255decimal).

Shift In (Control O, ØFhex, 15decimal - ASCII SI)
Shift into the standard system character set (Character set Ø, GØ). This gives the ability to access and display the standard ASCII character set - being those characters from ØØhex (ØØdecimal) through 7Fhex (127decimal).

2.3 Multi-Character Escape Sequences

2.3.1 Cursor Functions

Escape Sequence/Function	ASCII Code	Performed Function
ESC A	1B, 41hex 27, 65dec	Move cursor up one line without changing column.
ESC B	1B, 42hex 27, 66dec	Move cursor down one line without changing column.
ESC C	1B, 43hex 27, 67dec	Move cursor forward one character position.
ESC D	1B, 44hex 27, 68dec	Move cursor backward one character position.
ESC H	1B, 48hex 27, 72dec	Move cursor to the home position. Cursor moves to line 1, column 1.
ESC I	1B, 49hex 27, 73dec	Reverse index. Move cursor up to previous line at current column position.
ESC Y l c	1B, 59hex 27, 89dec	Moves the cursor via direct (absolute) addressing to the line and column location described by 'l' and 'c'. The line ('l') and column ('c') coordinates are binary values offset from 2Øhex (32decimal). (For further information on the use of direct addressing see section 2.4).
ESC j	1B, 6Ahex 27, 1Ø6dec	Store the current cursor position. The cursor location is saved for later restoration (see ESC k).
ESC k	1B, 6Bhex 27, 1Ø7dec	Returns cursor to the previously saved location (see ESC j).
ESC n	1B, 6Ehex 27, 11Ødec	Return the current cursor position. The current cursor location is returned as line and column, offset from 2Øhex (32decimal), in the next character input request.

2.3.2 Editing Functions

Escape Sequence/Function	ASCII Code	Performed Function
ESC @	1B, 4Øhex 27, 64dec	Enter the character insert mode. Characters may be added at the current cursor position, as each new character is added, the character at the end of the line is lost.
ESC E	1B, 45hex 27, 69dec	Erase the entire screen.
ESC J	1B, 4Ahex 27, 74dec	Erase from the current cursor position to the to the end of the screen.
ESC K	1B, 4Bhex 27, 75dec	Erase the screen from the current cursor position to the end of the line.
ESC L	1B, 4Chex 27, 76dec	Insert a blank line on the current cursor line. The current line, and all following lines are moved down one, and the cursor is placed at the beginning of the blank line.
ESC M	1B, 4Dhex 27, 77dec	Delete the line containing the cursor, place the cursor at the start of the line, and move all following lines up one - a blank line is inserted at line 24.
ESC N	1B, 4Ehex 27, 78dec	Delete the character at the cursor position, and move all other characters on the line after the cursor to the left one character position.
ESC O	1B, 4Fhex 27, 79dec	Exit from the character insert mode (see ESC @).
ESC b	1B, 62hex 27, 98dec	Erase the screen from the start of the screen up to, and including, the current cursor position.

2.3.2 Editing Functions -- continued

Escape Sequence/Function	ASCII Code	Performed Function
ESC l	1B, 6Chex 27, 1Ø8dec	Erase entire current cursor line.
ESC o	1B, 6Fhex 27, 111dec	Erase the beginning of the line up to, and including, the current cursor position.

2.3.3 Configuration Functions

Escape Sequence/Function	ASCII Code	Performed Function	
ESC x Ps	1B, 78hex 27, 12Ødec	Sets mode(s) as follows:	
		Ps	Mode
	31hex, 49dec	1	Enable 25th line
	33hex, 51dec	3	Hold screen mode on
	34hex, 52dec	4	Block cursor
	35hex, 53dec	5	Cursor off
	38hex, 56dec	8	Auto line feed on receipt of a carriage return.
	39hex, 57dec	9	Auto carriage return on receipt of line feed
	41hex, 65dec	A	Increase audio volume
	42hex, 66dec	B	Increase CRT brightness
	43hex, 67dec	C	Increase CRT contrast
ESC y Ps	1B, 79hex 27, 12Ødec	Resets mode(s) as follows:	
		Ps	Mode
	31hex, 49dec	1	Disable 25th line
	33hex, 51dec	3	Hold screen mode off
	34hex, 52dec	4	Underscore cursor
	35hex, 53dec	5	Cursor off
	38hex, 56dec	8	No auto line feed on receipt of a carriage return.
	39hex, 57dec	9	No auto carriage return on receipt of line feed
	41hex, 65dec	A	Decrease audio volume
	42hex, 66dec	B	Decrease CRT brightness
	43hex, 67dec	C	Decrease CRT contrast
ESC [	1B, 5Bhex 27, 91dec	Set hold mode	
ESC \	1B, 5Chex 27, 92dec	Clear hold mode	
ESC ^	1B, 5Ehex 27, 94dec	Toggle hold mode on/off.	

2.3.4 Operation Mode Functions

Escape Sequence/Function	ASCII Code	Performed Function
ESC (	1B, 28hex 27, 40dec	Enter high intensity mode. All characters displayed after this point will be displayed in high-intensity.
ESC)	1B, 29hex 1B, 41dec	Exit high intensity mode.
ESC Ø	1B, 3Øhex 27, 48dec	Enter underline mode. All characters displayed after this point will be underlined.
ESC 1	1B, 31hex 27, 49dec	Exit underline mode.
ESC p	1B, 7Øhex 27, 112dec	Enter reverse video mode. All characters displayed after this point will be displayed in reverse video.
ESC q	1B, 71hex 27, 113dec	Exit reverse video mode.

2.3.5 Special Functions

Escape Sequence/Function	ASCII Code	Performed Function
ESC #	1B, 23hex 27, 35dec	Return the current contents of the page. The entire contents of the screen are made available at the next character input request(s). (For further information on the use of this function, see section 2.5).
ESC $	1B, 24hex 27, 36dec	Return the value of the character at the current cursor position. The character is returned in the next character input request.
ESC +	1B, 2Bhex 27, 43dec	Clear the foreground. Clear all high-intensity displayed characters.
ESC 2	1B, 32hex 27, 5Ødec	Make cursor blink.
ESC 3	1B, 33hex 27, 51dec	Stop cursor blink.
ESC 8	1B, 38hex 27, 56dec	Set the text (literally) mode for the next single character. This allows the display of characters from Ø1hex (Ø1dec) thru 1Fhex (31dec) on the screen. Thus the BELL character (Ø7hex, Ø7dec) will not cause the bleep, but a character will appear on the screen.
ESC Z	1B, 5Ahex 27, 9Ødec	Identify terminal type. The VT52 emulator will return ESC\Z in the next character input request.
ESC]	1B, 5Dhex 27, 93dec	Return the value of the 25th line. The next series of character input requests will receive the current contents of the 25th line.

2.3.5 Special Functions -- continued

Escape Sequence/Function	ASCII Code	Performed Function
ESC v	1B, 76hex 27, 118dec	Enable wrap-around at the end of each screen line. A character placed after column 8Ø of a line will be placed on the next line at column 1.
ESC w	1B, 77hex 27, 119dec	Disable wrap-around at the end of each line.
ESC z	1B, 7Ahex 27, 122dec	Reset terminal emulator to the power-on state. This clears all user selected modes, clears the screen, and homes the cursor.
ESC {	1B, 7Bhex 27, 123dec	Enable keyboard input. (see ESC }).
ESC }	1B, 7Dhex 27, 125dec	Disable keyboard input. This locks the keyboard. Any character(s) typed are ignored until an ESC { is issued.
ESC i Ps	1B, 69hex 27, 1Ø5dec	Displays banner as follows:

	Ps	Mode
3Øhex, 48dec	Ø	Display entire banner
31hex, 49dec	1	Display company logo
32hex, 5Ødec	2	Display operating system
33hex, 51dec	3	Display configuration

2.4 Direct Cursor Addressing -- Examples of Use

The direct cursor addressing function is accessed by sending the ESC Y l c sequence to the screen (see section 2.3.1). "l" is the line number required, whose valid coordinates are between 1 and 24. An offset of 1Fhex (31decimal) must be added to the location required in order to correctly locate the cursor. "c" is the column number required, whose valid coordinates are between 1 and 8Ø. An offset of 1Fhex (31decimal) must be added to the location required in order to correctly locate the cursor.

Note that the true offset requirement of 2Øhex (32decimal) for line and column may only be used accurately when the line number is viewed Ø to 23, and the column number Ø to 79.

The line/column number requested must be handled as a binary digit, examples of this follow:

2.4.1 Microsoft MS-BASIC -- Direct Cursor Positioning

The following method uses offsets from line 1, column 1:

```
10 PRINT CHR$(27)+"E"    :REM CLEAR THE SCREEN
2Ø DEF FNM$(LIN,COL)=CHR$(27)+"Y"+CHR$(31+LIN)+CHR$(31+COL)
3Ø PRINT "Enter line (1-24) and column (1-8Ø), as LINE,COL ";
4Ø INPUT LIN, COL
5Ø PRINT FNM$(LIN,COL);
6Ø FOR I = 1 TO 1ØØØ    :REM PAUSE BEFORE OK MESSAGE DISPLAYED
7Ø NEXT I
```

The alternative method, using offsets from zero is shown below:

```
1Ø PRINT CHR$(27)+"E"    :REM CLEAR THE SCREEN
2Ø DEF FNM$(LIN,COL)=CHR$(27)+"Y"+CHR$(32+LIN)+CHR$(32+COL)
3Ø PRINT "Enter line (Ø-23) and column (Ø-79), as LINE,COL ";
4Ø INPUT LIN, COL
5Ø PRINT FNM$(LIN,COL);
6Ø FOR I = 1 TO 1ØØØ    :REM PAUSE BEFORE OK MESSAGE DISPLAYED
7Ø NEXT I
```

2.4.2 Microsoft MACRO-86 Assembler -- Direct Cursor Positioning

```
line_off    equ  20h                        ;line position offset from 0
col_off     equ  20h                        ;column position offset from 0
esc         equ  1bh                        ;escape character
msdos       equ  21h                        ;interrupt to MS-DOS

clear_screen          db   esc,'E$'  ;clear screen request
dir_cur_pos_lead      db   esc,'Y$'  ;cursor positioning lead-in

; the cursor position required is handed down in BX
;    where BH = line (0-23 binary), BL = column (0-79 binary)

clear_and_locate:
     mov  ah,9h                             ;string output up to $
     mov  dx,offset clear_screen            ;get the clear screen string
     int  msdos                             ;and output it up to the $
;
; the cursor position required is in BX
;
     add  bh,line_off                       ;normalize line for output
     add  bl,col_off                        ;normalize column for output
;
; send the direct cursor positioning lead-in
;
     mov  ah,9h                             ;select screen output up to $
     mov  dx,offset dir_cur_pos_lead ;select the lead in ESC Y
     int  msdos                             ;and output it up to $
;
; now the contents of BX must be sent to the terminal emulator
;
     mov  dl,bh                             ;ready the line number
     mov  ah,6h                             ;direct console output of DL
     int  msdos                             ;output the line coordinate
;
     mov  dl,bl                             ;ready the column number
     mov  ah,6h                             ;direct console output of DL
     int  msdos                             ;send the column coordinate
;
; the cursor is now at the location selected in BX
```

2.4.3 Microsoft Pascal Compiler -- Direct Cursor Positioning

```
program position (input,output);
{This method uses offsets from line Ø, column Ø.}

const
   clear_screen = chr(27) * chr(69);

var
   result : array[1..4] of char;
   i, line, column : integer.
   row, col : char;

begin
   result[1] := chr(27);          {RESULT = ESC}
   result[2] := chr(89);          {RESULT = "Y"}
   write (clear_screen);
   write ('  Enter line (Ø-23) and column (Ø-79), as LINE COLUMN: ');
   readln (line, column);
   writeln (clear_screen);
   row := chr(32 + line);
   col := chr(32 + column);
   result[3] := row;              {RESULT = ROW}
   result[4] := col;              {RESULT = COL}
   for i := 1 to 4 do
       write (result[i]);         {PRINT CURSOR TO SCREEN}
   for i := 1 to 32ØØØ do         {PAUSE}
end.
```

2.5 Transmit Page -- Examples of Use

The transmit page function is accessed by sending the ESC # sequence to the screen (see section 2.3.5). The result of this sequence is that all characters on the screen, as well as the cursor positioning sequences required to re-create the screen, are sent to the keyboard buffer. Reading the keyboard via a normal keyboard input request will return the entire screen of data to the program. The screen buffer within the program should be at least 1920decimal bytes long to accomodate the entire screen - the program will need to perform 1920 single character inputs to empty the keyboard buffer. Note that the character input requests must be done rapidly to prevent the keyboard buffer overflowing and causing loss of data - note, too, that on a keyboard buffer overflow, the bell sounds.
The following sample programs demonstrate the use for this function request:

2.5.1 Microsoft MS-BASIC -- Transmit Page

```
10 DIM A$(1920)
20 PRINT CHR$(27)+"#";
30 FOR I = 1 TO 1920
40 A$(I)=INKEY$
50 NEXT I
60 PRINT CHR$(27)+"E";
70 FOR I = 1 TO 1920
80 PRINT A$(I);
90 NEXT I
```

2.5.2 Microsoft MACRO-86 Assembler -- Transmit Page

```
coniof        equ    6h                  ;direct console i/o function
conin         equ    Øffh                ;console input request
printf        equ    9h                  ;screen o/p up to $
msdos         equ    21h                 ;interrupt operating system
buffer_length    equ    192Ø             ;entire screen count

read_screen  db     1bh,'#$'             ;read entire screen
clear_screen db     1bh,'E$'             ;clear screen/home cursor
buffer       db     buffer_length dup (?) ;main buffer region

     mov     ax,DS                       ;get buffer data segment
     mov     ES,ax                       ;ready for store
     mov     di,offset buffer            ;get storage buffer
     mov     si,di                       ;init for later use
     mov     dx,offset read_screen       ;read entire screen string
     mov     ah,printf                   ;o/p it up to $
     int     msdos                       ;call the OS
;
; now read entire screen in to BUFFER
;
     mov     ah,coniof                   ;read from keyboard buffer
     mov     dl,conin                    ;ready to read
     mov     cx,buffer_length            ;count of chars to read
;
in_loop:
     int     msdos                       ;get a char in AL
     stosb                               ;save the char in BUFFER
     loop    in_loop                     ; and loop til buffer full
;
     mov     ah,printf                   ;ready to clear the screen
     mov     dx,offset clear_screen      ;get the string
     int     msdos                       ; and o/p it up to $
;
; now replace the screen data
;
     mov     cx,buffer_length            ;get the count
     mov     ah,coniof                   ;get the o/p char function
;
out_loop:
     lodsb                               ;get a char
     mov     dl,al                       ;  ready to go
     int     msdos                       ;o/p it
     loop    out_loop                    ;loop til buffer empty
     ret                                 ;
```

2.5.3 Microsoft Pascal Compiler -- Transmit Page

```
PROGRAM Scrnbuf;

  CONST
    clear_screen   = CHR(27)*CHR(69)*CHR(36);
    transmit_page  = CHR(27)*CHR(35)*CHR(36);
    err_msg        = 'ERROR$';
    direct_conio   = #6;
    conin          = #ØFF;
    print_string   = #9;

  VAR
    screen_dump : ARRAY [1..192Ø] OF CHAR;
    ch : CHAR;
    i : INTEGER;
    param : WORD;
    status : BYTE;

FUNCTION DOSXQQ( command, parameter : WORD ) : BYTE; EXTERNAL;

BEGIN
  EVAL(DOSXQQ(print_string,WRD(ADR(transmit_page) ) ) );
  param:= BYWORD( Ø, conin );
  status:= DOSXQQ( direct_conio, param );
  IF status <> Ø THEN
    BEGIN
      i:= 1;
      WHILE status <> Ø DO
        BEGIN
          ch:= CHR(status);
          screen_dump[i]:= ch;
          i:= i + 1;
          status:= DOSXQQ( direct_conio, param );
        END;
      i:= i - 1;
      EVAL(DOSXQQ(print_string,WRD(ADR(clear_screen) ) ) );
      FOR VAR J:= 1 TO i DO
        EVAL(DOSXQQ( direct_conio, WRD(screen_dump[J]) ) );
    END
   ELSE
     EVAL(DOSXQQ(print_string,WRD(ADR(err_msg) ) ) );
END.
```

CHAPTER 3

Victor 9ØØØ Input/Output Port Specification

3.1 Device Connection

There are 5 ports available on the Victor 9ØØØ - they are as follows:

2 x Serial (RS232C) - Ports A and B
1 x Parallel (Centronics)
2 x Parallel (control - located on CPU board)

The ports are located on the rear of the Victor 9ØØØ as shown in the following diagram:

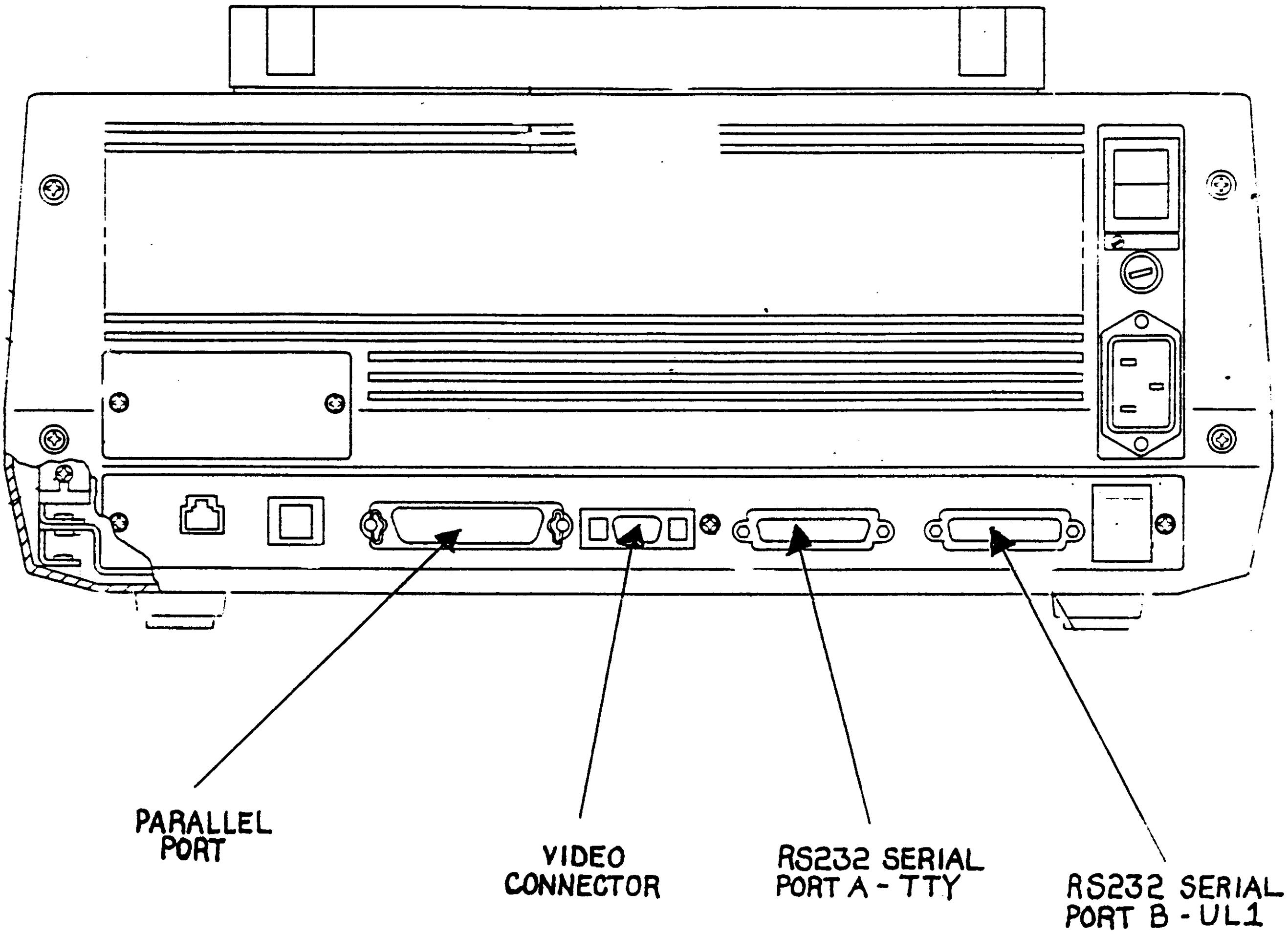


Figure 1
Victor 9ØØØ Parallel and Serial Ports

3.2 Parallel Printer Connection

To connect a parallel printer to the Victor 9000, a suitable cable is required - if the printer is supplied by Victor Technologies, then it will be a matter of plugging the cable into both machines; cables should be attached as follows:

1) Disconnect power from both the computer and printer.
2) Disconnect the Victor video connector (see 3.1)
3) Attach interface cable to Victor and printer
4) Re-attach the video connector
5) Set the printer dip-switches as required

3.3 Parallel Cable Requirements

If a suitable parallel cable is not available, you will need to make one - use the guidelines that follow to create your own cable:

> You will need a male centronics-compatible Amphenol 57-30360 type connector for the Victor 9000 end of the cable; use the type of connector suggested by the printer manufacturer for the printer end, in general, another male centronics-compatible Amphenol 57-30360 type connector will be required. You will also require a length of 12-core cable (10 feet maximum length).

Refer to the port layout in your printer handbook - compare this with the Victor 9000 parallel port layout (see C.1). If the pin numbers and signal requirements are the same, then construct the cable as follows:

```
 1 -------------------- 1
 2 -------------------- 2
 3 -------------------- 3
 4 -------------------- 4
 5 -------------------- 5
 6 -------------------- 6
 7 -------------------- 7
 8 -------------------- 8
 9 -------------------- 9
10 -------------------- 10
11 -------------------- 11
16 -------------------- 16
```

It does not matter which end of the cable is connected to the printer or the computer.

3.6 Operating System Port Utilities

Victor Technologies supplies a selection of programs under both CP/M-86 and MS-DOS to allow the temporary selection of both baud rate and list device port. If you attach a printer to your system you may be required to perform some of the following steps in order to utilize the printer. Before you use any of the utilities discussed you need to be aware of the port the printer is attached to; Port A, B or Parallel. You will also need to know, except in the case of a parallel printer, what the baud rate, stop-bits and parity your printer is set up at. Note that many printers will start to lose data at baud rates above 4800, you must, therefore, select a baud rate that your printer can handle.

3.6.1 SETIO - MS-DOS List Device Selection Utility

To select the correct port for the list device you have attached, the SETIO program has been provided. This program is used as follows:

```
SETIO LST = TTY    - printer is attached to port A
SETIO LST = UL1    - printer is attached to port B
SETIO LST = LPT    - printer is attached to parallel port
```

It is recommended that your printer be attached to either port B or the parallel port.

Once SETIO has executed, it displays a map of the ports, with the ones you selected highlighted on the screen - if this is not corrcet, repeat the process.

3.6.2 STAT - CP/M-86 List Device Selection Utility

To select the correct port for the list device you have attached, the STAT program has been provided. This program is used as follows:

```
STAT LST:=TTY:   - printer is attached to port A
STAT LST:=UL1:   - printer is attached to port B
STAT LST:=LPT:   - printer is attached to parallel port
```

It is recommended that your printer be attached to either port B or the parallel port.

3.6.3 PORTSET - MS-DOS Baud Rate Selection Utility

To select the correct baud rate for ports A or B (but this is not applicable to the parallel port), the PORTSET program is provided. This program is menu driven, and is used as follows:

To the prompt type PORTSET, the screen will display a choice of three ports:

1) Port A (RS232C)
2) Centronics/Parallel Port
3) Port B (RS232C)

Type either 1,2 or 3. If you type 1 or 3, the next menu screen is displayed - this screen has baud-rate choices labelled A through N - select one of the baud-rates.

3.6.4 PORTCONF - CP/M-86 Baud Rate Selection Utility

This program is used in exactly the same manner as PORTSET (see 3.6.3).

3.7 Serial Input/Ouput Ports

The two serial input/output ports are memory mapped ports located in the memory segment EØØØhex; and they are mapped as follows:

EØØØ:4Ø	-	port A data (input/output)
EØØØ:41	-	port B data (input/output)
EØØØ:42	-	port A control (read/write)
EØØØ:43	-	port B control (read/write)

The following information is available in each port's control register:

bit Ø	-	rx character available
bit 1	-	not used
bit 2	-	tx buffer empty
bit 3	-	DCD
bit 4	-	not used
bit 5	-	CTS
bit 6	-	not used
bit 7	-	not used

See Appendix C.2 for information on each port's pinouts.

Note that writing a 1Øhex to the relevent control register allows the resensing of the modem leads (i.e. DCD and CTS) with their current values being updated in the port's control register.
Since the Victor 9ØØØ configures the NEC 72Ø1 chip to operate in auto-enable mode, DCD (pin 8 on the port connector) must be ON, and CTS (pin 5 on the port connector) must be ON to enable the 72Ø1's receiver and trasmitter respectively. RTS and DTR are always ON as a convenient source for an RS-232C control ON (+11 volts).

3.8 Baud Rate and Data Input/Output - Sample Programs

The means of establishing the baud rates, receiving and transmitting data are discussed in the following programs. The serial port's control register are discussed in 3.7 - the means of accessing them is better described with the programming examples that follow.

The following programs provide information on how to set up the baud rates on the serial ports (A and B) - they also demonstrate how to send and receive data from these ports.

3.8.1 Microsoft MS-BASIC -- Baud Rate and Data Input/Output

The following program may be used in place of PORTSET or PORTCONF if you omit the lines 500 through 740 inclusive.

```
10 DIM RATE(14)
20 REM Select the data port
30 PRINT CHR$(27)+"E"; : REM Clear the screen
40 PRINT : PRINT : PRINT : PRINT
50 PRINT "The serial ports are:" : PRINT
60 PRINT ,"          A - Serial Port TTY - left hand on back"
70 PRINT ,"          B - Serial Port UL1 - right hand on back"
80 PRINT : PRINT
90 PRINT ,"Select the port you want to use, A or B ";
100 PORT$ = INPUT$(1)
110 PRINT PORT$
120 IF PORT$ = "a" THEN STATIO=2 : DATIO=0 : GOTO 210
130 IF PORT$ = "A" THEN STATIO=2 : DATIO=0 : GOTO 210
140 IF PORT$ = "b" THEN STATIO=3 : DATIO=1 : GOTO 210
150 IF PORT$ = "B" THEN STATIO=3 : DATIO=1 : GOTO 210
160 GOTO 30
200 REM Set the baud rate
210 PRINT CHR$(27)+"E"; : REM Clear the screen
220 PRINT : PRINT : PRINT : PRINT
230 PRINT "The available baud rates are as follows:" : PRINT
240 PRINT ," 1 =     300 baud"
250 PRINT ," 2 =     600 baud"
260 PRINT ," 3 =    1200 baud"
270 PRINT ," 4 =    2400 baud"
280 PRINT ," 5 =    4800 baud"
290 PRINT ," 6 =    9600 baud"
300 PRINT ," 7 =   19200 baud"
310 PRINT : PRINT : PRINT
320 PRINT "Select one of the above baud rates: ";
330 RATE$ = INPUT$(1)
340 IF RATE$ > "7" THEN 210
350 IF RATE$ < "1" THEN 210
360 PRINT RATE$
400 REM Now set the baud rate in the port selected
410 DEF SEG = &HE002
420 IF DATIO = 0 THEN POKE 3,54 : IF DATIO = 1 THEN POKE 3,118
430 FOR I = 1 TO 14
440 READ RATE(I) : REM Set the baud rate matrix
450 NEXT I
460 NODE = (VAL(RATE$)-1)*2+1
470 POKE DATIO,RATE(NODE)
480 POKE DATIO,RATE(NODE+1)
```

-- Listing Continued on Next Page --

```
500 REM Now data may be entered and sent down line
510 PRINT CHR$(27)+"E"; : REM Clear the screen
520 PRINT : PRINT ,"Baud rate established"
530 PRINT : PRINT : PRINT
540 DEF SEG = &HE004
550 PRINT ,"Enter data to be sent down line with return to end"
560 PRINT ,"or just press return to receive data -"
570 PRINT
580 TEXT$=INKEY$
590 IF TEXT$="" THEN 630
600 IF TEXT$=CHR$(13) THEN PRINT TEXT$ :TEXT$=CHR$(126) :GOTO 620
610 PRINT TEXT$;
620 GOSUB 650
630 GOSUB 690
640 GOTO 580
650 STATUS=PEEK (STATIO) : STATUS=STATUS AND 4
660 IF STATUS = 0 THEN 650 :REM Waiting to send char
670 POKE DATIO, ASC(TEXT$)
680 RETURN
690 STATUS = PEEK(STATIO) :STATUS = STATUS AND 1
700 IF STATUS = 0 THEN RETURN : REM No char available
710 DATUM = PEEK (DATIO) : DATUM = DATUM AND 127
720 IF DATUM = 126 THEN PRINT CHR$(13) : RETURN
730 PRINT CHR$(DATUM); :REM Show char from line
740 RETURN
1000 DATA 0,1,&H80,0,&H40,0,&H20,0,&H10,0,8,0,4,0
```

3.8.2 MACRO-86 Assembler -- Baud Rate and Data Input/Output

The following assembler modules may be included in a program and called with the stated parameters. The character input and output modules will need re-coding if your program requires status return rather than looping for good status.

```
rates   db      0h,1h,80h,0h      ;baud rate conversion table
        db      40h,0h,20h,0h
        db      10h,0h,8h,0h
        db      4h,0h

;**********************************************************
;
; Routine:        BAUD_SET
;
; Function:       To set Port A or B baud rate
;
; Entries:        AL = 0=PortA, 1=PortB
;                 DX = 0=300 baud,  1=600 baud,  2=1200 baud
;                      3=2400 baud, 4=4800 baud, 5=9600 baud
;                      6=19200 baud
;
; Returns:        None
;
; Corruptions: ES, AX, BX, CX, DX
;
;**********************************************************

baud_set:
        mov     cx,0e002h               ;get the segment
        mov     ES,cx                   ;init the segment register
        mov     bx,3                    ;point to counter control
        or      al,al                   ;see if Port A or B to be set
        jnz     set_B                   ;AL > 0, so set Port B counter
;
        mov     byte ptr ES:[bx],36h    ;set it for port A
        jmp     short set_rate          ;  and input the Baud rate
;
set_B:
        mov     byte ptr ES:[bx],76h    ;set port B counter
;
set_rate:
        mov     bx,offset rates         ;get the baud rate table
        shl     dx,1                    ;DX = DX * 2 for words
        add     bx,dx                   ;point to baud rate entry
        mov     dx,[bx]                 ;get the baud rate
        xor     bh,bh                   ;BH=0
        mov     bl,al                   ;get the required port
        mov     byte ptr ES:[bx],dl     ;send first byte
        mov     byte ptr ES:[bx],dh     ; and last byte of rate
        ret                             ;baud rate established
```

3.8.2 Baud Rate and Data Input/Output -- continued

```
;***********************************************************
;
; Routine:      SEND_CHAR
;
; Function:     To output a character to a serial port
;
; Entries:      AL = 0=PortA, 1=PortB
;               AH = Character to send
;
; Returns:      None
;
; Corruptions: ES, AX, BX
;
;***********************************************************

send_char:
        mov     bx,0e004h               ;get the port segment
        mov     ES,bx                   ;set the segment
        xor     bh,bh                   ;BH=0
        mov     bl,al                   ;get the required port
        add     bl,2                    ;required port status
;
in_status_loop:
        mov     al,ES:[bx]              ;get the status
        and     al,4h                   ;mask for TX empty
        jnz     in_status_loop          ;not ready - loop
;
        sub     bl,2                    ;point to data
        mov     ES:[bx],ah              ;character gone
        ret
```

3.8.2 Baud Rate and Data Input/Output -- continued

```
;**************************************************************
;
; Routine:       GET_CHAR
;
; Function:      To input a character from a serial port
;
; Entries:       AL = Ø=PortA, 1=PortB
;
; Returns:       AL = character
;
; Corruptions: ES, AX, BX
;
;**************************************************************

get_char:
        mov     bx,Øe ØØ4h                ;get the port segment
        mov     ES,bx                    ;set the segment
        xor     bh,bh                    ;BH=Ø
        mov     bl,al                    ;get the required port
        add     bl,2                     ;required port status
;
out_status_loop:
        mov     al,ES:[bx]               ;get the status
        and     al,1h                    ;mask for RX character avail
        jnz     out_status_loop          ;not ready - loop
;
        sub     bl,2                     ;point to data
        mov     al,ES:[bx]               ;character received
        ret
```

APPENDIX A

A.1 ASCII Codes Used in the Victor 9ØØØ Computer

The American Standard Codes for Information Interchange (ASCII) has been defined to allow data communication between computers, their peripherals, and other computers. The other major code standard is the Extended Binary Coded-Decimal Interchange Code (EBCDIC) used on some mainframe computers. The Victor 9ØØØ computer is designed to function in ASCII, but communication software is available that allows the Victor 9ØØØ to receive EBCDIC data and have it translated into ASCII, and vice versa.

The following table contains the 7-ASCII codes and their meanings. It is called 7-ASCII as only 7-bits of the potential 8-bits are used to carry data; the "spare" bit is utilized in the Victor 9ØØØ computer to support characters not otherwise available in the 7-ASCII set.

An Eight Bit Byte is pictured as follows:

```
[ 7 ][ 6 ][ 5 ][ 4 ][ 3 ][ 2 ][ 1 ][ Ø ]
```

the bits are numbered Ø through 7 (which adds up to eight bits), and it is the 8th bit (bit 7 in computer jargon) which is not used in 7-ASCII.

A.2 ASCII / HEXADECIMAL / DECIMAL Character Set

ASCII	Hex	Dec	ASCII	Hex	Dec	ASCII	Hex	Dec	ASCII	Hex	Dec
NUL	ØØ	ØØ	space	2Ø	32	@	4Ø	64	`	6Ø	96
SOH	Ø1	Ø1	!	21	33	A	41	65	a	61	97
STX	Ø2	Ø2	"	22	34	B	42	66	b	62	98
ETX	Ø3	Ø3	#	23	35	C	43	67	c	63	99
EOT	Ø4	Ø4	$	24	36	D	44	68	d	64	1ØØ
ENQ	Ø5	Ø5	%	25	37	E	45	69	e	65	1Ø1
ACK	Ø6	Ø6	&	26	38	F	46	7Ø	f	66	1Ø2
BEL	Ø7	Ø7	'	27	39	G	47	71	g	67	1Ø3
BS	Ø8	Ø8	(	28	4Ø	H	48	72	h	68	1Ø4
HT	Ø9	Ø9	)	29	41	I	49	73	i	69	1Ø5
LF	ØA	1Ø	*	2A	42	J	4A	74	j	6A	1Ø6
VT	ØB	11	+	2B	43	K	4B	75	k	6B	1Ø7
FF	ØC	12	,	2C	44	L	4C	76	l	6C	1Ø8
CR	ØD	13	-	2D	45	M	4D	77	m	6D	1Ø9
SO	ØE	14	.	2E	46	N	4E	78	n	6E	11Ø
SI	ØF	15	/	2F	47	O	4F	79	o	6F	111
DLE	1Ø	16	Ø	3Ø	48	P	5Ø	8Ø	p	7Ø	112
DC1	11	17	1	31	49	Q	51	81	q	71	113
DC2	12	18	2	32	5Ø	R	52	82	r	72	114
DC3	13	19	3	33	51	S	53	83	s	73	115
DC4	14	2Ø	4	34	52	T	54	84	t	74	116
NAK	15	21	5	35	53	U	55	85	u	75	117
SYN	16	22	6	36	54	V	56	86	v	76	118
ETB	17	23	7	37	55	W	57	87	w	77	119
CAN	18	24	8	38	56	X	58	88	x	78	12Ø
EM	19	25	9	39	57	Y	59	89	y	79	121
SUB	1A	26	:	3A	58	Z	5A	9Ø	z	7A	122
ESC	1B	27	;	3B	59	[	5B	91	{	7B	123
FS	1C	28	<	3C	6Ø	\	5C	92	|	7C	124
GS	1D	29	=	3D	61	]	5D	93	}	7D	125
RS	1E	3Ø	>	3E	62	^	5E	94	~	7E	126
US	1F	31	?	3F	63	_	5F	95	DEL	7F	127

APPENDIX B

B.1 Victor 9ØØØ Keyboard Layout

Legend:

Shaded region indicates unused key switch

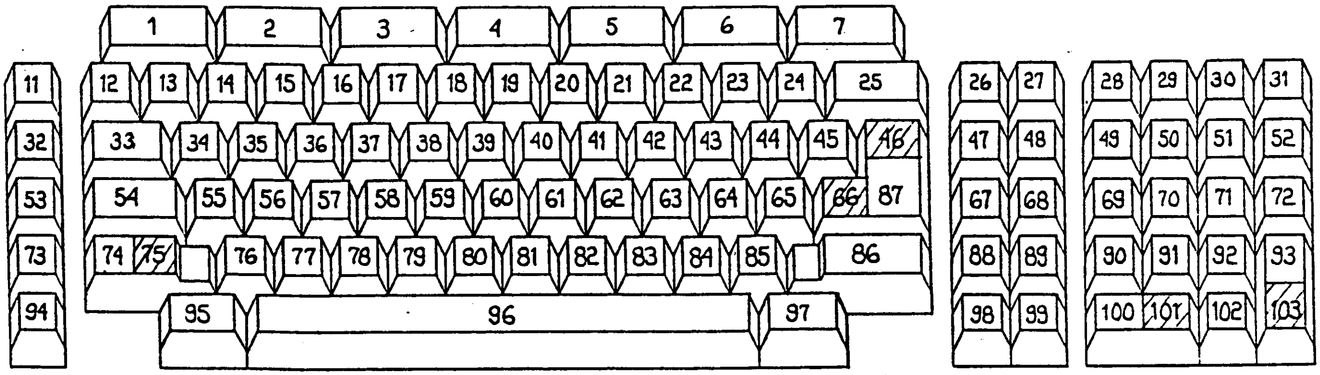


Figure 2
Victor 9ØØØ Keyboard Configuration with Key Switch Positions and Logical Key Numbers

APPENDIX C

C.1 Victor 9ØØØ Parallel (Centronics) Port

Pin Number	Signal
1	Data Strobe
2	Data 1
3	Data 2
4	Data 3
5	Data 4
6	Data 5
7	Data 6
8	Data 7
9	Data 8
1Ø	ACK
11	Busy
17	Pshield
12,18,3Ø,31	Not connected
Remaining	GND

C.3 Victor 9ØØØ IEEE-488 Port

The Victor 9ØØØ IEEE-488 cable attaches to the parallel port - the pin number refers to the actual computer port connector; the IEEE-488 pin number refers to the standard IEEE-488 pin-out as they must attach to the parallel port.

The IEEE pin numbers referred to with the (**z) are wires that are to be bound together as twisted pairs.

Pin Number	IEEE Signal	IEEE Pin Number	
1	DAV	6	(**a)
19	GND	18	(**a)
2	DIO1	1	
3	DIO2	2	
4	DIO3	3	
5	DIO4	4	
6	DIO5	13	
7	DIO6	14	
8	DIO7	15	
9	DIO8	16	
1Ø	NRFD	7	(**b)
28	GND	19	(**b)
11	SRQ	1Ø	(**c)
29	GND	22	(**c)
13	NDAC	8	(**d)
33	GND	2Ø	(**d)
15	EOI	5	
17	shield	12	
34	REN	17	
35	ATN	11	(**e)
16	GND	23	(**e)
36	IFC	9	(**f)
27	GND	21	(**f)
2Ø	GND	24	

C.4 Victor 9ØØØ Control Port

Pin Number	Signal
1	-12V
2	-12V
3	Not connected
4	Not connected
5	+12V
6	+12V
7	+5V
8	+5V
9	Not connected
1Ø	Light Pen
11	GND
12	CA1
13	GND
14	CA2
15	GND
16	PAØ
17	GND
18	PA1
19	GND
2Ø	PA2
21	GND
22	PA3
23	GND
24	PA4
25	GND
26	PA5
27	GND
28	PA6
29	GND
3Ø	PA7
31	GND
32	PBØ
33	GND
34	PB1
35	GND
36	PB2
37	GND
38	PB3
39	GND
4Ø	PB4
41	GND
42	PB5
43	GND
44	PB6
45	GND
46	PB7 / CODEC Clock Output
47	GND
48	CB1
49	GND
5Ø	CB2

APPENDIX D

D.1 Example Assembler Shell Program for MS-DOS Interfacing

The Microsoft MACRO-86 assembler follows closely the Intel ASM-86 specifications. The operating system interfacing technique is via a straightforward interrupt (INT 21Hex), with the required operational parameter in the AH register. MS-DOS does not corrupt any registers other than the ones used for the sending or receiving of data. An example of the running and exiting program technique, plus the required assembler directives, follows. The program example is for the small memory model; but it will apply equally well to the compact or large memory model. The 8Ø8Ø memory model is not recommended as it results in poor usage of the potential of the 8Ø86/8Ø88 processor. At link time, this programming example will generate an .EXE file - the header information on this file type will be found in E.1.

```
title   Example of MS-DOS/MACRO-86 Assembly Programming

dgroup  group   data
cgroup  group   code

msdos   equ     ØØØ21h                  ;interrupt to operating system

data    segment public    'data'
;######  insert your data here ######
data    ends

code    segment public    'code'
        assume  CS: cgroup, DS: dgroup

example proc    near                    ;origin of code

begin:
        push    ES                      ;save return segment address
        call    run_module              ;run the program
;
; run ends - select close down
;
exit    proc    far                     ;close down code
        xor     ax,ax                   ;zero for PSP:Ø
        push    ax                      ;save for far return
        ret                             ;and close down
exit    endp                            ;close down code ends

run_module:
        mov     ax,DATA                 ;get the data segment origin
        mov     DS,ax                   ; and initialize the segment
;##### insert your code at this point ######
        ret                             ;return to exit module

        example endp
        code    ends
        end
```

D.2 Example Assembler Shell Program for CP/M-86 Interfacing

The Digital Research ASM-86 assembler does not follow the standard Intel ASM-86 structure - this makes for a more complex task when transferring assembler programs between the CP/M-86 and the MS-DOS operating systems. The operating system interfacing technique is via a straightforward interrupt (INT EØHex), with the required operational parameter in the CL register. CP/M-86 corrupts all registers, excepting the CS and IP - it is, therefore, recommended that all registers be pushed prior to the INT EØHex being issued. An example of the running and exiting program technique, plus the required assembly directives, follows. The program example follows that of the MS-DOS MACRO-86 example. At GENCMD time, this programming example will generate a .CMD file - the header information on this file type is shown in the System Guide for CP/M-86.

```
title   'Example of CP/M-86/ASM-86 Programming'


reset   equu    ØØØØØh                  ;system reset function
cpm     equ     ØØØeØh                  ;interrupt to operating system

        cseg

begin:
        call    run_module              ;run the program
;
; run ends - select close down
;
        mov     cl,reset                ;select system reset
        mov     dl,ØØh                  ;select memory recovery
        int     cpm                     ;return to operating system
;
run_module:
;##### insert your code at this point ######
        ret                             ;return to exit module

        dseg
;##### insert your data here #####
        end
```

E.1 MS-DOS -- EXE File Header Structure

The Microsoft linker outputs .EXE files in a relocatable format, suitable for quick loading into memory and relocation. EXE files consist of the following parts:

- o Fixed length header
- o Relocation table
- o Memory image of resident program

A run file is loaded in the following manner:

- o Read into RAM at any paragraph (16 byte) boundary
- o Relocation is then applied to all words described by the relocation table.

The resulting relocated program is then executable. Typically, programs using the PL/M small memory model have little or no relocation; programs using larger memory models have relocation for long calls, jumps, static long pointers, etc.

The following is a detailed description of the format of an EXE file:

Microsoft .EXE File Main Header

Byte	Name	Function
Ø+1	wSignature	Must contain 4D5Ahex.
2+3	cbLastp	Number of bytes in the memory image modulo 512. If this is Ø then the last page is full, else it is the number of bytes in the last page. This is useful in reading overlays.
4+5	cpnRes	Number of 512 byte pages of memory needed to load the resident and the end of the EXE file header.
6+7	irleMax	Number of relocation entries in the table.
8+9	cparDirectory	Number of paragraphs in EXE file header.
A+B	cparMinAlloc	Minimum number of 16-byte paragraphs required above the end of the loaded program.
C+D	cparMaxAlloc	Maximum number of 16-byte paragraphs required above the end of the loaded program. ØFFFFh means that the program is located as low as possible into memory.
E+F	saStack	Initial value to be loaded into SS before starting program execution.
1Ø+11	raStackInit	Initial value to be loaded into SP before starting program execution.
12+13	wchksum	Negative of the sum of all the words in the run file.
14+15	raStart	Initial value to be loaded into IP before starting program execution.
16+17	saStart	Initial value to be loaded into CS before starting program execution.
18+19	rbrgrle	Relative byte offset from beginning of run file to the relocation table.
1A+1B	iov	Number of the overlay as generated by LINK-86. The resident part of a program will have iov = Ø.

The relocation table follows the fixed portion of the run file header and contains irleMax entries of type rleType, defined by:

rleType	bytes Ø+1 ra
	bytes 2+3 sa

Taken together, the ra and sa fields are an 8Ø86/8Ø88 long pointer to a word in the EXE file to which the relocation factor is to be added. The relocation factor is expressed as the physical address of the first byte of the resident divided by 16. Note that the sa portion of an rle must first be relocated by the relocation factor before it in turn points to the actual word requiring relocation. For overlays, the rle is a long pointer from the beginning of the resident into the overlay area.

The resident begins at the first 512 byte boundary following the end of the relocation table.

The layout of the EXE file is:

28-byte Header

Relocation Table

padding (<200hex bytes)

memory image

F.1 Victor 9000 Technical Specification

Processor

- o Intel 8088 16-bit microprocessor
- o 128k bytes RAM internally upgradeable to 896k bytes
- o 4k bytes Auto-boot ROM (read only memory)
- o 4 internal expansion slots for plug-in card options
- o 2 x RS232C serial communications ports
- o 1 x Parallel (Centronics) or IEEE-488 port
- o 2 x Parallel user port (5Ø-way KK Connector on CPU board)

Display System

- o 25 line x 8Ø column screen / 5Ø line x 132 column screen
- o 12" CRT, Green p39 phosphor
- o Adjustable horizontal viewing angle (± 45 degree swivel)
- o Adjustable vertical viewing angle (Ø deg to 11 deg tilt)

Floppy Drives

- o Standard 5 1/4-inch, single-sided 96 TPI dual disk drives, with a maximum capacity of 6ØØk bytes per drive.
- o Optional 5 1/4-inch, double-sided 96 TPI dual disk drives, with a maximum capacity of 12ØØk bytes per drive.
- o Optional single 1Ø,ØØØk byte Hard Disk - non-removable; with single 5 1/4-inch, double sided 96 TPI disk drive with a maximum capacity of 12ØØk bytes.

Single-sided floppy drive offers 8Ø tracks at 96 TPI
Double-sided floppy drive offers 16Ø tracks at 96 TPI
Floppy drives have 512 byte sectors; utilising a GCR, 1Ø-bit recording technique.

Floppy access times:

2 micro-second per bit data transfer rate, with an interleave factor of 3. Average seek time is approximately 9Ø milli-seconds.

Hard Disk access times:

Ø.2 micro-second per bit data transfer rate, with an interleave factor of 5. Average seek time is approximately 1ØØ milli-seconds.

F.2 Victor 9ØØØ Physical Specifications

Mainframe Assembly

Height	Width	Depth	Weight (approx)
178 mm	422 mm	356 mm	12.6 kg
7 in	16.6 in	14 in	281 lbs

Display Assembly

Height	Width	Depth	Weight (approx)
264 mm	326 mm	339 mm	8.1 kg
1Ø.4 in	12.9 in	13.4 in	18 lbs

Keyboard Assembly

Height	Width	Depth	Weight (approx)
45 mm	483 mm	2Ø3 mm	1.5 kg
1.8 in	19 in	6.4 in	3 lbs

System Assembly

Height	Width	Depth	Weight (approx)
457 mm	483 mm	559 mm	22.2 kg
18 in	19 in	2Ø.4 in	49 lbs

Width without the keyboard module is 396 mm / 15.6 in

128K Memory Configurations

Figure 1 shows the 1 Megabyte memory space partitioned into 128K segments. Switch settings (SW) are shown for all possible memory configurations. Figure 2 shows physical location of Switches on the 128K memory board.

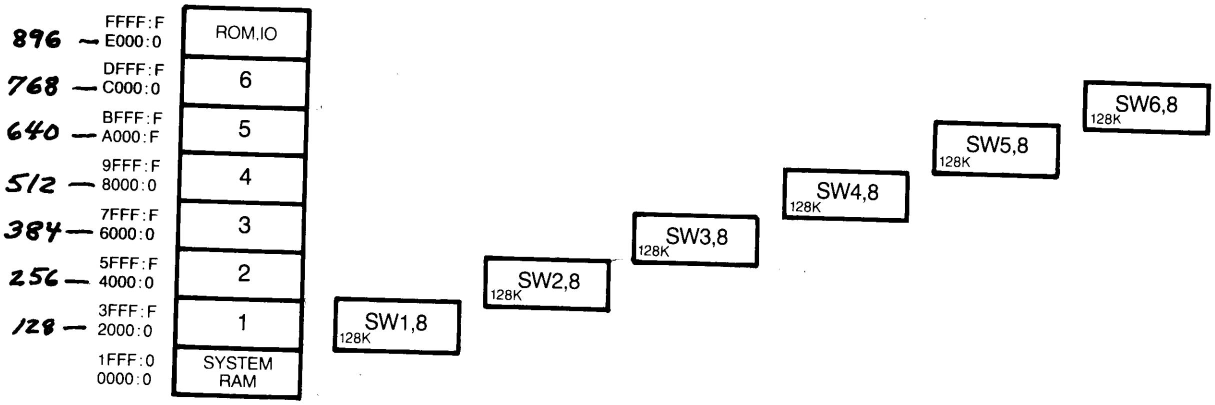


Figure 1: Memory Space Segments

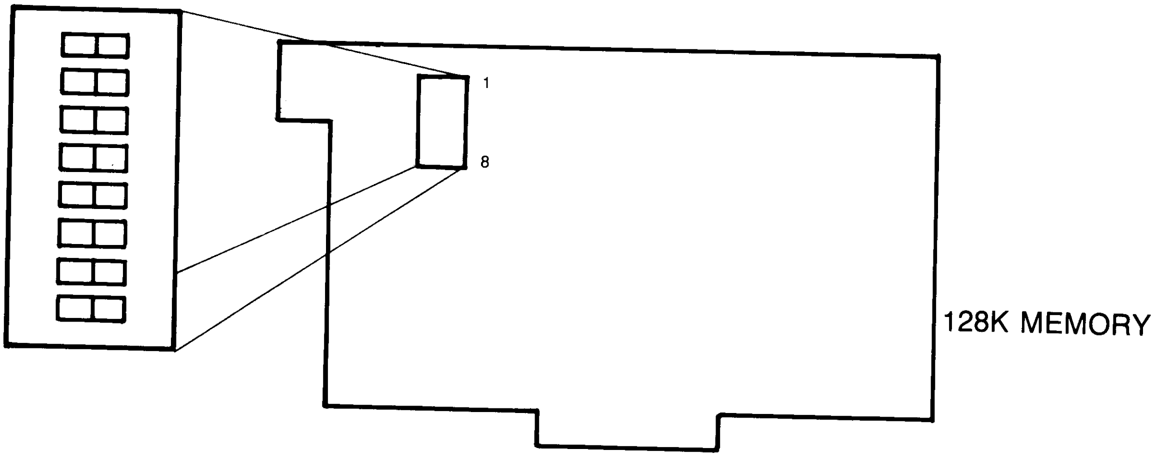


Figure 2: Memory Configuration Switches

VICTOR® 380 El Pueblo Rd., Scotts Valley, CA 95066 USA Telephone 408/438-6680

Part Number 102545-01

DATA SHEET

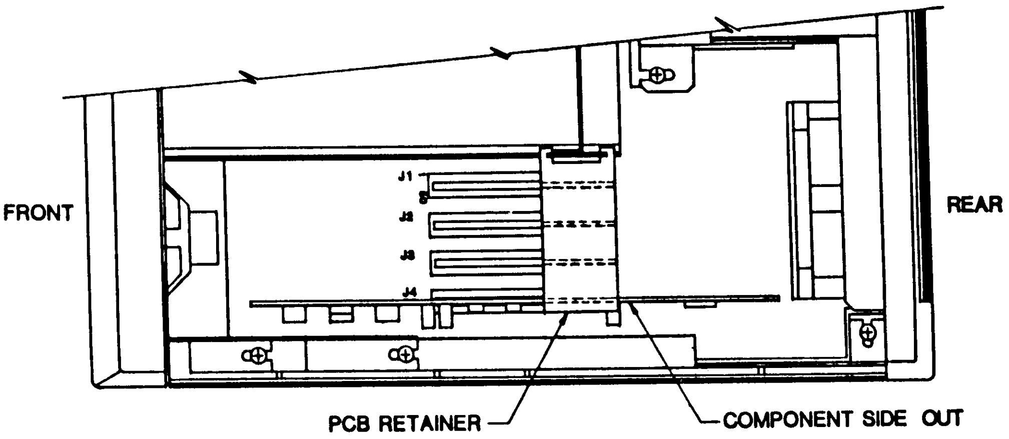
Page 1 of 1

Auxiliary PCB Installation Instructions

The following steps are required to install auxiliary PC boards in the processor. The auxiliary PC board connectors are located on the right side of the processor unit between the speaker and the fan (see Figure 1).

1. Remove power from the system.
2. Disconnect and carefully remove the CRT and keyboard.
3. Remove the rear panel cover (4 screws).
4. Slide the top cover back and out of the front cover.
5. Remove the auxiliary PC board retainer (see Figure 1).
6. Insert the auxiliary PC board into the socket **WITH THE COMPONENT SIDE OUT.**
7. Reinstall the auxiliary PC board retainer.
8. Reinstall the top cover under the front cover.
9. Reinstall the rear panel cover (4 screws removed in 3).
10. Carefully install and connect the CRT and keyboard.
11. Connect power to the system.

AUX PCB INSTALLATION INSTRUCTIONS



MAINFRAME WITH COVER REMOVED

Figure 1: Processor Unit with Cover Removed

Part Number 102849-01

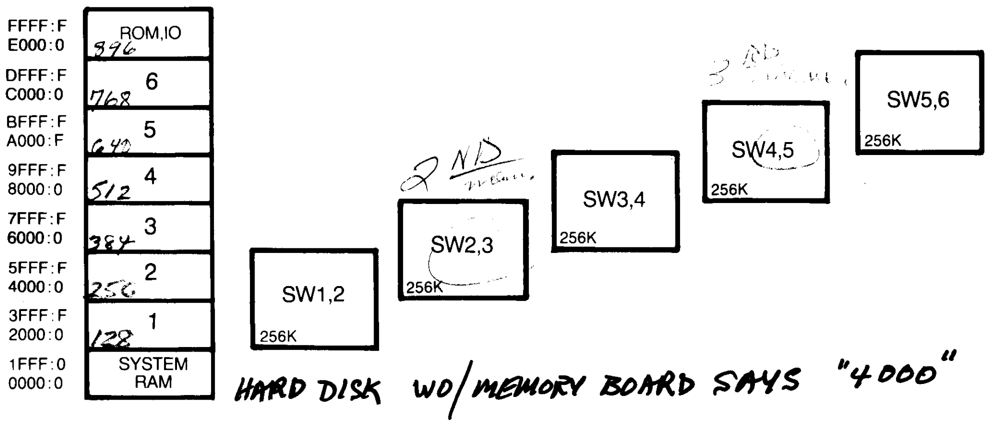
VICTOR®

Telephone (408) 438-6680
380 El Pueblo Road
Scotts Valley, CA 95066 USA

DATA SHEET

256K Memory Configurations

Figure 1 shows the 1 Megabyte memory space partitioned into 128K segments. Switch settings (SW) are shown for possible memory configurations. Figure 2 shows physical locations of switches on the 256K memory board.



HARD DISK WO/MEMORY BOARD SAYS "4000"

Figure 1: Memory Space Segments

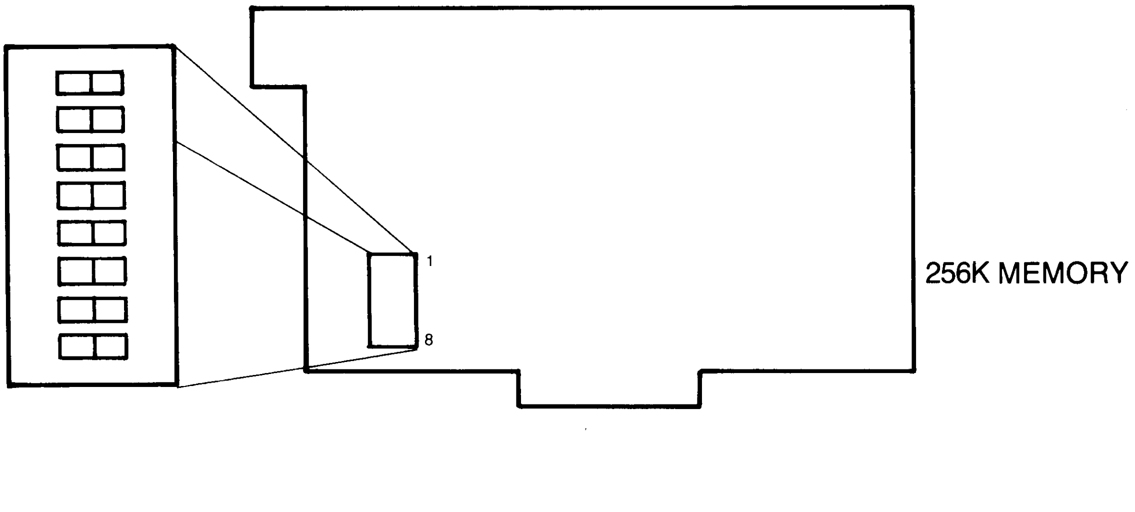


Figure 2: Memory Configuration Switches