Channel 9000

THE INDEPENDENT NEWSLETTER OF THE VICTOR 9000/SIRIUS 1 COMPUTER

VOL. 1 NO. 1 January 1983

PREMIER ISSUE

FROM THE EDITOR:

I have been involved in microprocessor hardware and software design for over 12 years as an engineer, manager, and consultant. Consequently, I have seen many systems of differing shapes, sizes, and capabilities. Recent years have brought a proliferation of microcomputer systems due to the drastic lowering of memory and cpu prices. While some of these systems have been quite nice, I had not, until recently, seen one that enthused me.

The introduction of the IBM/PC last year, however, sparked my interest. The fact that it carried the name 'IBM' held promise of more sophistication and flexibility. Unfortunately, the IBM/PC turned out not to be what I would call technologically innovative, although it has caused a surge in the development of 16-bit software which is sorely needed.

Then, about six months ago, one of my clients showed me the Victor 9000. It was not just another run-of-the-mill micro system. I could tell by examining the 9000's circuitry and running the minimal demonstration software that came with it that considerable thought and design effort went into its development. It is technologically innovative: 600k bytes on one side of an 80 track 5 1/4" diskette, a 800 by 400 pixel graphics monitor standard, and a codec for sound and voice. I finally found a system worthy of enthusiasm.

As I began working with the 9000, converting my clients programs which had been running on a 64k Z80 system, it became clear that there was a severe lack of availability of technical information. More disturbing was the apparent lack of technical knowledge about the 9000 by Victor personnel. This made it virtually impossible to adapt the 9000 to our needs because we needed detailed system information in order to convert the existing Z80 programs to the 9000.

To make a long story short, after about three months we finally received a very preliminary copy of the Victor 9000 technical manual (which, although fairly complete, contained absolutely no schematics) and a diskette with the CP/M BIOS program source code on it. The BIOS source was of limited use because it is all written in Intel assembly language and PL/M and cannot be compiled on the 9000 itself. Apparently all the system software development was done with an Intel development system, which makes it very difficult for the average dealer or user to modify the BIOS program to add such code as special I/O drivers.

The delays in obtaining technical information from Victor is understandable in the light of the 9000 being a new product just in the process of being released. It is further understandable that there might be a technical communication gap with Victor because they did not develop the 9000 themselves. The 9000 was developed by Sirius Systems Technology, Inc. of Scotts Valley, CA. Victor has the exclusive marketing rights for the 9000 in the United States while Sirius markets the 9000 out of the country.

While Victor is making a good effort at organizing their efforts in bringing the 9000 to the marketplace, there seems to be a definite need for an independent forum for information concerning the Victor 9000. Also, Victor does not appear to desire to become involved in any type of specialty hardware or software which brings a great opportunity to those of us who make our living providing those types of items that the OEMs don't supply. To do this, however, we need some method of disseminating the news about what we are doing. There is also a need for education about the ins and outs of the 9000 in order that the capabilities of the 9000 can be more fully utilized.

All of this brings me to the reason that this newsletter is being started. I think the Victor 9000 can be a serious (Sirius ??) contender in the 16-bit microcomputer marketplace, but only the users and independent dealers can make this happen, not Victor. (IBM was fully aware of this as evidenced by the way they chose to market their Personal Computer.) I, and I am sure others, have gleened much information concerning the Victor 9000 that we are willing to share, but we need a place to share it. Here is the place.

We plan to keep this newsletter primarily technical in nature. We do not want a gossip column. Also, anyone interested is invited to submit articles for publication. We plan to offer in each issue of this newsletter the following:

1. The current status of the Victor 9000 product: the state of the hardware and software, expected release dates of upcoming products, technical advisories, etc.

2. News briefs and new product releases for 9000 compatible hardware and software from independent suppliers.

3. A tutorial in each issue covering a technical topic about the 9000. In addition, we will probably be offering, at reasonable prices, utility software and hardware that we have developed relating to the topics presented. We do not intend that the tutorials be a sales gimmick to sell products. Instead, we intend that the products be labor saving conveniences (i.e. we will offer the source code of a program printed in the newsletter on diskette).

4. Comments, letters and/or announcements from dealers, users, user groups, etc. as space permits.

5. Whatever else deemed of interest to Victor 9000 users.

The subscription rate for **Channel 9000** is $30.00 for six issues (approximately one year). We would like to publish more often than bimonthly, but given the in-depth coverage we wish to provide for each issue's tutorial, it may not be possible.

Finally, let me reiterate my feelings that if the Victor 9000 is to be a success there must be an unselfish, open sharing of information and discussion of ideas concerning the 9000. To that end this newsletter is dedicated.

James M. Lesher, Editor

POSTSCRIPT:
As this issue was being prepared, it was announced that Sirius Systems Technology and Victor Business Products were being merged into one company, probably to be called Victor Incorporated, with Kidde Inc. maintaining a significant interest. It was also learned that the management of Sirius would essentially become the management of the new entity. This appears to be a giant step toward correcting the problems mentioned above.

In talking with some Sirius/Victor people, there seems to be much enthusiasm resulting from this reorganization move. There is promise of improved communication and support for dealers and users. In addition, I have been told that this newsletter is a welcome aid and that we would be given whatever help we need in providing information about the Victor 9000.

J.M.L.

COMMING ATTRACTIONS:
In the next issue of **Channel 9000** we will present a potpourri of useful tips and information concerning the Victor 9000 hardware and software.

CHANNEL 9000

9742 Marcus Lane, Tujunga, CA 91042
(213) 352-6443
James M. Lesher, Editor
Subscription rate: $30.00 for 6 issues
Published Bimonthly

KEYBOARD TUTORIAL

This issue's tutorial covers the operation of the Victor 9000 keyboard.

Victor currently offers three keyboards: Standard, Word Processing and Programming. Contrary to what you might think, the differences between the three keyboards are not the placement or legending of the keys but just the total number of keys on the keyboard. The standard keyboard has 91 keys, the word processing keyboard has 97 keys and the programming keyboard has 99 keys.

I expected the programming keyboard to have the same key legending and placement as most crt's. Unfortunately, it is the same as the other two keyboards except that the number row keys have the common characters used in programming on the front faces of the key caps. In order to type these characters, you must depress the ALT key (what Victor calls the CONTROL key) while depressing the key with the desired character on its front face. This is very inconvenient for someone who for years has been using a standard crt keyboard.

There are some other minor irritations such as the shifted and unshifted condition of the square bracket key being reversed and there being no separate ESC or LINE-FEED key. Fortunatly, the keyboard design and BIOS driver software make it fairly easy to correct these deficiencies, as we will discuss later. Victor, unlike IBM, evedently decided that most of the people that would be using the 9000 would not be familliar with anything other than a standard typewriter keyboard layout, but I think that they should provide those of us that are used to the standard crt keyboard the option of buying one.

The Victor keyboard is manufactured by Keytronic Corporation of Spokane, Wa. It is a capacitive type keyboard and uses an on-board single-chip microprocessor (an Intel 8051) for scanning the keys and communicating with the cpu. The keyboard has a maximum capacity of 104 keys, but as stated above, the most currently available are 99.

The Victor keyboard is different from most keyboards in that it is what is termed an event processing keyboard. In other words, it not only sends a message when a key is depressed, but also sends a message when that key is released. This makes for an extremely flexible keyboard because all the keys behave exactly alike; no key has a restricted function. (For example, it is possible with this type of keyboard to make any key the SHIFT key or any key the REPEAT key.)

When a key is depressed or released, the keyboard microprocessor detects the event and stores the key number (the keys are not encoded in any particular manner) and whether it was a closure or opening. This event and any subsequent events are saved until they can be send to the cpu. Each event is represented as an 8-bit number with the least significant 7 bits being the key number and the most significant bit indicating a key open (MSB = 0) or key closed (MSB = 1).

There are two reserved codes used by the keyboard to indicate special circumstances. The first is FE hex, which indicates an overflow of the keyboard event buffer and loss of data. The second is FF hex, which indicates that the keyboard is dead or not connected, although is is not clear to me how the keyboard can send this code to the cpu if it is dead or not connected.

The event data is sent to the cpu in a bit serial manner with full handshaking. The 8 bits are sent LSB first and are followed by a 9th bit which is always zero (a stop bit). The keyboard uses three signals for data transmission: a ready (RDY) signal to the cpu, an acknowledge (ACK) signal from the cpu, and a signal for the data (DATA). In the idle state the RDY, ACK, and DATA signals are all high. To initiate a transmission, the keyboard sets the DATA signal to the value of the LSB of the event data and brings the RDY signal low. At this time, the cpu should sample the DATA signal and respond by lowering the ACK signal. The negative transition of the ACK signal tells the keyboard that a bit has been accepted and causes the keyboard to raise the RDY signal and prepare for transmitting the next bit. When the ACK line returns high, the keyboard processor sets the DATA signal to the value of the next bit and again sets the RDY signal low. This process continues until all 9 bits have been sent.

In order to insure that data is sent properly to the cpu, the keyboard starts a timer counting when it lowers the RDY singal. If there is no response from the cpu within 250 milliseconds, the keyboard raises the RDY signal and restarts the event transmission from the beginning. This allows for the cpu to get back into syncronization with the keyboard if a data bit is missed.

Once the data is recieved by the cpu, the cpu must somehow determine what function the key event is to represent. The designers of the 9000 came up with a very flexible table method for determining the meaning of each keyboard event. There are two groups of three tables with 104 entries in each table (one entry for every possible key on the keyboard). The first group of tables are 8-bit data values or codes corresponding to each key, while the second group of tables are 16-bit attribute codes for each key. The three tables in each group contain the data and attributes for each key in its unshifted, shifted, and alternate state respectively. There is also one other table which is used to store the data for keys that produce multiple character responses.

ATTRIBUTE TABLES

The attribute tables determine how a key event is processed. The attribute word for each key is defined as follows:

Bits	Usage
15-14	Type code
13	Auto repeat
12	Local data
11	Caps lock
10	Shift lock
9-0	Legend code

AUTO REPEAT
The auto repeat bit, if set, causes the depressed key to repeatedly generate its code(s) at a preset interval as long as the key remains depressed.

LOCAL DATA
The local data bit, if set, causes the data generated by the key to be sent directly, and only, to the console output routine. (This bit is set on keys such as the crt intensity controls.)

CAPS LOCK
The caps lock bit, if set, causes the shifted data table entry to be used if the keyboard is in the caps lock mode (see below). This bit is meaningful only in the shifted attribute table.

SHIFT LOCK
The shift lock bit, if set, causes the shifted data table entry to be used if the keyboard is in the shift lock mode (see below). This bit is meaningful only in the shifted attribute table.

LEGEND CODE
The legend code is an as yet unimplimented method of locating specific keys on the keyboard, for possibly modifying the data the key produces. A key that may be in different positions on various keyboards can be located by giving it a unique legend value. A program that wishes to locate the key searches the attribute tables until it finds the proper legend code. Once the table position of the key is determined, the entries for the key may be interrogated or modified as desired.

TYPE CODE
The type code is the basic determinate of how a key is processed. There are four possible types:

Code	Type
00	Special function
01	Single character data
10	Multiple character data
11	Reserved for future use

The **SPECIAL FUNCTION** type is used for functions such as shift or caps lock, and also to generate single character escape codes. If the data table entry for a key defined as a special function has a hex value of 20 or greater, then a two character sequence is generated consisting of the escape character (1B hex) and the data table entry. If the data table entry for the key is less than 20 hex, then the key event is used to activate or deactivate the following modes depending on whether the event is a key closure or opening respectively:

Data value	Mode
0	Dead key
1	Caps lock
2	Shift lock
3	Left shift key
4	Right shift key
5	Alternate
6	Control
7	Repeat
8	Clear keyboard
9	Advance line
10	Advance page
11-31	Reserved

Dead Key - the special function mode data value of zero is used for what is termed dead positions, that is, positions that generate no data (see below).

Caps Lock - causes the shifted data table value to be used for any key with the caps lock atribute bit set.

Shift Lock - causes the shifted data table value to be used for any key with the shift lock atribute bit set.

Left Shift Key / Right Shift Key - causes the shifted data table value to be used.

Alternate - causes the alternate data table value to be used.

Control - causes any single character data generated by a key event that is in the range of hex 40 through hex 7F to be converted to its corresponding ASCII control character. (There is currently no key designated for the control function on the Victor keyboards.)

Repeat - causes the repetition of the last active (key still depressed) key event that generated character data.

Clear Keyboard - causes the flushing of the cpu's keyboard event buffer and the initializing of various internal parameters.

Advance Line - is used in conjunction with the HOLD mode for console output.

Advance Page - is used in conjunction with the HOLD mode for console output.

Note - Control mode has precidence over alternate mode which has precidence over shift (or shift lock) mode. Also, if the alternate mode table entry is 'dead', then the shift mode table entry is used. If the shift mode table entry is 'dead', then the unshifted table entry is used. If the unshifted table entry is 'dead' then the key event generates no data

The **SINGLE CHARACTER DATA** type is used to generate a single character of data subject to the above described modes.

If a key event is a **MULTIPLE CHARACTER DATA** type, then the data table entry for that event is used as an index into the multiple character data table. The first byte pointed to by the index is the number of characters to be generated by this event. This byte is followed by the characters themselves.

KEYBOARD TABLE GENERATION

The Victor SYSELECT system configuration program lets the user select one of the predefined keyboard files supplied by Victor for insertion into the CP/M or MS-DOS BIOS. A source listing for one of the files (the Domestic Programming keyboard) is given in Figure 2. Figure 1 is a diagram of the full 9000 keyboard including the keyboard table position for each key. To modify the function of a key, first find the table entry number for the key from Figure 1. Next, change the source file entries to whatever function you wish for the key. Finally, generate the keyboard object file by performing the following steps (assume the file is named VICT03.A86):

```
A>ASM86 VICT03 $ SZ PZ
A>GENCMD VICT03
A>DDT86
-RVICT03.CMD
```

(After a read, DDT86 gives the load addresses for the file. We will use XXX: to indicate the segment address shown by DDT86.)

```
-WVICT03.KB,XXX:80,5FF
```

Now you can use SYSELECT to include the new keyboard file in your BIOS.

Here is an example of modifying a keyboard file. The MS-DOS editor, EDLIN, uses the character sequence ESC-S to copy one character from the template to the new line. Suppose we want to have function key 1 generate the ESC-S sequence under all conditions (unshifted, shifted, and alternate). Also, we wish to have the key auto-repeat.

From Figure 2 we find that special function key 1 is entry one in the keyboard tables. To generate an ESC followed by a single character, we use the Special Function Type Code with the data value being the ASCII code for S (53 hex). We make entry one in the unshifted data table

```
DB      53H
```

and for entry one in the unshifted attribute data table

```
DW      SPCL+REPEAT
```

In order to force the unshifted table entries to be used at all times, we must 'kill' the shifted and alternate table entries for key one by entering in the data tables

```
DB      00H
```

and for the attribute tables

```
DW      SPCL
```

Finally, we assemble the new file as described above. After assembly, we reboot the 9000 with an MS-DOS diskette and using RDCPM, we copy the new keyboard table from the CP/M diskette and use the MS-DOS version of SYSELECT to incorporate it into the BIOS.

KEYGEN PROGRAM

Victor has a keyboard modification program called KEYGEN which may be used to selectively modify existing keyboard tables. The keys of the keyboard are identified as above by the entry position in the keyboard table so that the layout in Figure 1 may be used with KEYGEN also.

KEYGEN is most easily used for making minor changes to existing keyboard tables. When making major changes or creating a table from scratch, it is probably easier to use the assembly method. I leave it to the user to decide which is more convenient.

ESCAPE SEQUENCE ALTERATIONS

The console output routine of the BIOS provides a method of changing the SINGLE CHARACTER DATA value of any key in the keyboard table in ram. This is accomplished by a special five character ESCAPE sequence:

ESC 4 <mode> <key number> <new data>

The mode character is "1" for unshifted, "2" for shifted and "3" for the alternate table entry. The key number character is the table position of the key to be changed (see Figure 1). New data is the new byte value to be generated by the key whenever it is depressed. Changing a key value forces the key to be a SINGLE CHARACTER DATA, non-local key regardless of the previous characteristic of the key. None of the other attributes of the key are (or can be) changed.

As an example, let us change the the period key (key number 84) so that it produces a 'greater-than' symbol (">") in shifted mode. We can do this from BASIC by executing the following print statement:

```
PRINT CHR$(27);"42";CHR$(84);">"
```

UTILITY PROGRAMS

For those who wish to do keyboard modifications, we are offering a diskette which contains the following:

1. Assembly language text files of the standard keyboard tables.

2. BASIC program to dump a keyboard table file to a text file.

3. Program to load a new keyboard table into ram (CPM & MSDOS).

4. Program to change keyboard table on system disk without using SYSELECT (CPM & MSDOS).

The price of this diskette is $15.00 postpaid. California residents please add 6.5% sales tax ($15.98).

Figure 1. Keyboard / Entry Number Diagram.

```
;*********************************************;
;  header information used by                 ;
;  SYSELECT configuration program             ;
;*********************************************;
        org     00h

;  type of file - K indicates keyboard file
;  1 character field
        db      'K'

;  file format version number  -  1 character
        db      '0'

;  display classification  -  12 characters
        db      'VT52  ASCII '

;  name of keyboard type  -  8 characters
;  displayed as part of banner at boot time
        db      'Domestic'

;  space filler  -  1 character
        db      0

;  keyboard version number  -  3 characters
;  displayed as part of banner at boot time
        db      '03 '

;  space filler  -  1 character
        db      0

;  comment  -  35 characters
        db      '703     '
        db      'Victor Programming     '
        db      '    '

;  origin of file  -  16 characters
        db      'Victor Eng      '

;  date of file  -  8 characters
        db      '82/04/23'

;  length  -  4 characters
        db      ' 10'

;*********************************************;
;  actual keyboard tables begin               ;
;*********************************************;
        ORG     80H

;  table length data required by bios
;  bootstrap loader

        DW      ENDKBD-OFFSET $-2;length of
                        ;keyboard object data
        DW      0       ;reserved location

;  attribute flag definitions

SPCL    EQU     0000H   ;special function
SGL     EQU     4000H   ;single character data
MULT    EQU     8000H   ;multiple character data

REPEAT  EQU     2000H   ;auto repeat this key
LOCAL   EQU     1000H   ;send data only to
                        ;display driver
CAPS    EQU     0800H   ;key affected by
                        ;caps lock mode
SHIFT   EQU     0400H   ;key affected by
                        ;shift mode

;*********************************************;
;  key data table  -  unshifted mode          ;
;*********************************************;

        DB      0B8H    ;   0 - function key 8
        DB      0B1H    ;   1 - function key 1
        DB      0B2H    ;   2 - function key 2
        DB      0B3H    ;   3 - function key 3
        DB      0B4H    ;   4 - function key 4
        DB      0B5H    ;   5 - function key 5
        DB      0B6H    ;   6 - function key 6
        DB      0B7H    ;   7 - function key 7
        DB      0B8H    ;   8 - no key
        DB      0B9H    ;   9 - function key 9
        DB      0BAH    ;  10 - function key 10
        DB      048H    ;  11 - 'H' - clear/home
        DB      025H    ;  12 - '%'
        DB      031H    ;  13 - '1'
        DB      032H    ;  14 - '2'
        DB      033H    ;  15 - '3'
        DB      034H    ;  16 - '4'
        DB      035H    ;  17 - '5'
        DB      036H    ;  18 - '6'
        DB      037H    ;  19 - '7'
        DB      038H    ;  20 - '8'
        DB      039H    ;  21 - '9'
        DB      030H    ;  22 - '0'
        DB      02DH    ;  23 - '-'
        DB      03DH    ;  24 - '='
        DB      008H    ;  25 - backspace
        DB      0BBH    ;  26 - ins
        DB      07FH    ;  27 - del
        DB      03DH    ;  28 - '='
        DB      025H    ;  29 - '%'
        DB      02FH    ;  30 - '/'
        DB      02AH    ;  31 - '*'
        DB      0BDH    ;  32 - scrl
        DB      009H    ;  33 - tab
        DB      071H    ;  34 - 'q'
        DB      077H    ;  35 - 'w'
        DB      065H    ;  36 - 'e'
        DB      072H    ;  37 - 'r'
        DB      074H    ;  38 - 't'
        DB      079H    ;  39 - 'y'
        DB      075H    ;  40 - 'u'
        DB      069H    ;  41 - 'i'
        DB      06FH    ;  42 - 'o'
        DB      070H    ;  43 - 'p'
        DB      03CH    ;  44 - '<'  (1/2 1/4 key)
        DB      05DH    ;  45 - ']'
        DB      000H    ;  46 - no key
        DB      0BEH    ;  47 - erase eol
        DB      0BFH    ;  48 - req can
        DB      037H    ;  49 - '7'
        DB      038H    ;  50 - '8'
        DB      039H    ;  51 - '9'
        DB      02DH    ;  52 - '-'
        DB      070H    ;  53 - rvs
        DB      002H    ;  54 - lock
        DB      061H    ;  55 - 'a'
        DB      073H    ;  56 - 's'
        DB      064H    ;  57 - 'd'
        DB      066H    ;  58 - 'f'
        DB      067H    ;  59 - 'g'
        DB      068H    ;  60 - 'h'
        DB      06AH    ;  61 - 'j'
```

Figure 2. VICT03.A86 listing.

```
        DB      06BH    ;  62 - 'k'
        DB      06CH    ;  63 - 'l'
        DB      03BH    ;  64 - ';'
        DB      027H    ;  65 - '''
        DB      000H    ;  66 - no key
        DB      0E4H    ;  67 - word left
        DB      0E5H    ;  68 - word right
        DB      034H    ;  69 - '4'
        DB      035H    ;  70 - '5'
        DB      036H    ;  71 - '6'
        DB      02BH    ;  72 - '+'
        DB      030H    ;  73 - undl
        DB      003H    ;  74 - left shift
        DB      000H    ;  75 - no key
        DB      07AH    ;  76 - 'z'
        DB      078H    ;  77 - 'x'
        DB      063H    ;  78 - 'c'
        DB      076H    ;  79 - 'v'
        DB      062H    ;  80 - 'b'
        DB      06EH    ;  81 - 'n'
        DB      06DH    ;  82 - 'm'
        DB      02CH    ;  83 - ','
        DB      02EH    ;  84 - '.'
        DB      02FH    ;  85 - '/'
        DB      004H    ;  86 - right shift
        DB      00DH    ;  87 - return
        DB      041H    ;  88 - 'A'  up arrow
        DB      042H    ;  89 - 'B'  down arrow
        DB      031H    ;  90 - '1'
        DB      032H    ;  91 - '2'
        DB      033H    ;  92 - '3'
        DB      00DH    ;  93 - enter
        DB      007H    ;  94 - repeat
        DB      005H    ;  95 - alt
        DB      020H    ;  96 - ' ' (space bar)
        DB      013H    ;  97 - pause/cont
        DB      044H    ;  98 - 'D'  left arrow
        DB      043H    ;  99 - 'C'  right arrow
        DB      030H    ; 100 - '0'
        DB      000H    ; 101 - double/triple zero
        DB      02EH    ; 102 - '.'
        DB      000H    ; 103 - no key

;*********************************************;
;  key data table  -  shifted mode            ;
;*********************************************;

        DB      0C8H    ;   0
        DB      0C1H    ;   1
        DB      0C2H    ;   2
        DB      0C3H    ;   3
        DB      0C4H    ;   4
        DB      0C5H    ;   5
        DB      0C6H    ;   6
        DB      0C7H    ;   7
        DB      0C8H    ;   8
        DB      0C9H    ;   9
        DB      0CAH    ;  10
        DB      045H    ;  11 - 'E'
        DB      060H    ;  12 - '~'
        DB      021H    ;  13 - '!'
        DB      040H    ;  14 - '@'
        DB      023H    ;  15 - '#'
        DB      024H    ;  16 - '$'
        DB      025H    ;  17 - '%'
        DB      020H    ;  18 - ' '
        DB      026H    ;  19 - '&'
        DB      02AH    ;  20 - '*'
        DB      028H    ;  21 - '('
        DB      029H    ;  22 - ')'
        DB      05FH    ;  23 - '_'
        DB      02BH    ;  24 - '+'
        DB      000H    ;  25
        DB      0CBH    ;  26
        DB      0CCH    ;  27
        DB      0FDH    ;  28
        DB      000H    ;  29
        DB      000H    ;  30
        DB      000H    ;  31
        DB      0CDH    ;  32
        DB      000H    ;  33
        DB      051H    ;  34 - 'Q'
        DB      057H    ;  35 - 'W'
        DB      045H    ;  36 - 'E'
        DB      052H    ;  37 - 'R'
        DB      054H    ;  38 - 'T'
        DB      059H    ;  39 - 'Y'
        DB      055H    ;  40 - 'U'
        DB      049H    ;  41 - 'I'
        DB      04FH    ;  42 - 'O'
        DB      050H    ;  43 - 'P'
        DB      03EH    ;  44 - '>'
        DB      05BH    ;  45 - '['
        DB      000H    ;  46
        DB      0CEH    ;  47
        DB      0CFH    ;  48
        DB      000H    ;  49
        DB      000H    ;  50
        DB      000H    ;  51
        DB      000H    ;  52
        DB      071H    ;  53 - 'q'
        DB      002H    ;  54
        DB      041H    ;  55 - 'A'
        DB      053H    ;  56 - 'S'
        DB      044H    ;  57 - 'D'
        DB      046H    ;  58 - 'F'
        DB      047H    ;  59 - 'G'
        DB      048H    ;  60 - 'H'
        DB      04AH    ;  61 - 'J'
        DB      04BH    ;  62 - 'K'
        DB      04CH    ;  63 - 'L'
        DB      03AH    ;  64 - ':'
        DB      022H    ;  65 - '"'
        DB      000H    ;  66
        DB      0E6H    ;  67
        DB      0E7H    ;  68
        DB      000H    ;  69
        DB      000H    ;  70
        DB      000H    ;  71
        DB      000H    ;  72
        DB      031H    ;  73 - '1'
        DB      003H    ;  74
        DB      000H    ;  75
        DB      05AH    ;  76 - 'Z'
        DB      058H    ;  77 - 'X'
        DB      043H    ;  78 - 'C'
        DB      056H    ;  79 - 'V'
        DB      042H    ;  80 - 'B'
        DB      04EH    ;  81 - 'N'
        DB      04DH    ;  82 - 'M'
        DB      000H    ;  83
        DB      000H    ;  84
        DB      03FH    ;  85 - '?'
        DB      004H    ;  86
        DB      000H    ;  87
        DB      000H    ;  88
```

Figure 2, continued ....

```
        DB      000H    ;  89
        DB      000H    ;  90
        DB      000H    ;  91
        DB      000H    ;  92
        DB      000H    ;  93
        DB      007H    ;  94
        DB      005H    ;  95
        DB      000H    ;  96
        DB      013H    ;  97
        DB      000H    ;  98
        DB      000H    ;  99
        DB      000H    ; 100
        DB      000H    ; 101
        DB      000H    ; 102
        DB      000H    ; 103

;*********************************************;
;  key data table  -  alternate mode          ;
;*********************************************;

        DB      0D8H    ;   0
        DB      0D1H    ;   1
        DB      0D2H    ;   2
        DB      0D3H    ;   3
        DB      0D4H    ;   4
        DB      0D5H    ;   5
        DB      0D6H    ;   6
        DB      0D7H    ;   7
        DB      0D8H    ;   8
        DB      0D9H    ;   9
        DB      0DAH    ;  10
        DB      045H    ;  11 - 'E'
        DB      021H    ;  12 - '!'
        DB      07CH    ;  13 - '|'
        DB      03CH    ;  14 - '<'
        DB      03EH    ;  15 - '>'
        DB      022H    ;  16 - '"'
        DB      049H    ;  17 - 'I'
        DB      02AH    ;  18 - '*'
        DB      05EH    ;  19 - '^'
        DB      060H    ;  20 - '`'
        DB      07BH    ;  21 - '{'
        DB      07DH    ;  22 - '}'
        DB      07EH    ;  23 - '~'
        DB      05CH    ;  24 - '\'
        DB      000H    ;  25
        DB      0DBH    ;  26
        DB      0DCH    ;  27
        DB      03DH    ;  28 - '='
        DB      000H    ;  29
        DB      000H    ;  30
        DB      000H    ;  31
        DB      0DDH    ;  32
        DB      0E9H    ;  33
        DB      011H    ;  34
        DB      017H    ;  35
        DB      005H    ;  36
        DB      012H    ;  37
        DB      014H    ;  38
        DB      019H    ;  39
        DB      015H    ;  40
        DB      009H    ;  41
        DB      00FH    ;  42
        DB      010H    ;  43
        DB      000H    ;  44
        DB      01BH    ;  45
        DB      000H    ;  46
        DB      0DEH    ;  47
        DB      0DFH    ;  48
        DB      000H    ;  49
        DB      000H    ;  50
        DB      000H    ;  51
        DB      000H    ;  52
        DB      01BH    ;  53
        DB      001H    ;  54
        DB      001H    ;  55
        DB      013H    ;  56
        DB      004H    ;  57
        DB      006H    ;  58
        DB      007H    ;  59
        DB      008H    ;  60
        DB      00AH    ;  61
        DB      00BH    ;  62
        DB      00CH    ;  63
        DB      01EH    ;  64
        DB      01FH    ;  65
        DB      000H    ;  66
        DB      003H    ;  67
        DB      007H    ;  68
        DB      000H    ;  69
        DB      000H    ;  70
        DB      000H    ;  71
        DB      000H    ;  72
        DB      000H    ;  73
        DB      003H    ;  74
        DB      000H    ;  75
        DB      01AH    ;  76
        DB      018H    ;  77
        DB      003H    ;  78
        DB      016H    ;  79
        DB      002H    ;  80
        DB      00EH    ;  81
        DB      00DH    ;  82
        DB      01CH    ;  83
        DB      01DH    ;  84
        DB      02FH    ;  85 - '/'
        DB      004H    ;  86
        DB      00AH    ;  87
        DB      00BH    ;  88
        DB      00FH    ;  89
        DB      000H    ;  90
        DB      000H    ;  91
        DB      000H    ;  92
        DB      00AH    ;  93
        DB      007H    ;  94
        DB      005H    ;  95
        DB      000H    ;  96
        DB      000H    ;  97
        DB      013H    ;  98
        DB      017H    ;  99
        DB      000H    ; 100
        DB      01BH    ; 101
        DB      000H    ; 102
        DB      000H    ; 103

;*********************************************;
;  attribute data table  -  unshifted mode    ;
;*********************************************;

        DW      SGL                     ;  0
        DW      SGL+100H                ;  1
        DW      SGL+100H                ;  2
        DW      SGL+100H                ;  3
        DW      SGL+100H                ;  4
        DW      SGL+100H                ;  5
        DW      SGL+100H                ;  6
```

Figure 2, continued ....

```
        DW      SGL+100H                ;  7
        DW      SGL                     ;  8
        DW      SGL                     ;  9
        DW      SGL                     ; 10
        DW      SPCL+100H               ; 11
        DW      SGL+REPEAT+SHIFT        ; 12
        DW      SGL+REPEAT+SHIFT        ; 13
        DW      SGL+REPEAT+SHIFT        ; 14
        DW      SGL+REPEAT+SHIFT        ; 15
        DW      SGL+REPEAT+SHIFT        ; 16
        DW      SGL+REPEAT+SHIFT        ; 17
        DW      SGL+REPEAT+SHIFT        ; 18
        DW      SGL+REPEAT+SHIFT        ; 19
        DW      SGL+REPEAT+SHIFT        ; 20
        DW      SGL+REPEAT+SHIFT        ; 21
        DW      SGL+REPEAT+SHIFT        ; 22
        DW      SGL+REPEAT+SHIFT        ; 23
        DW      SGL+REPEAT+SHIFT        ; 24
        DW      SGL+REPEAT+100H         ; 25
        DW      SGL                     ; 26
        DW      SGL+REPEAT              ; 27
        DW      SGL+100H                ; 28
        DW      SGL+100H                ; 29
        DW      SGL+100H                ; 30
        DW      SGL+100H                ; 31
        DW      SGL+100H                ; 32
        DW      SGL+100H                ; 33
        DW      SGL+REPEAT+CAPS+SHIFT   ; 34
        DW      SGL+REPEAT+CAPS+SHIFT   ; 35
        DW      SGL+REPEAT+CAPS+SHIFT   ; 36
        DW      SGL+REPEAT+CAPS+SHIFT   ; 37
        DW      SGL+REPEAT+CAPS+SHIFT   ; 38
        DW      SGL+REPEAT+CAPS+SHIFT   ; 39
        DW      SGL+REPEAT+CAPS+SHIFT   ; 40
        DW      SGL+REPEAT+CAPS+SHIFT   ; 41
        DW      SGL+REPEAT+CAPS+SHIFT   ; 42
        DW      SGL+REPEAT+CAPS+SHIFT   ; 43
        DW      SGL+REPEAT+SHIFT        ; 44
        DW      SGL+REPEAT+SHIFT        ; 45
        DW      SPCL                    ; 46
        DW      SGL                     ; 47
        DW      SGL                     ; 48
        DW      SGL+100H                ; 49
        DW      SGL+100H                ; 50
        DW      SGL+100H                ; 51
        DW      SGL+100H                ; 52
        DW      SPCL+100H               ; 53
        DW      SPCL+200H               ; 54
        DW      SGL+REPEAT+CAPS+SHIFT   ; 55
        DW      SGL+REPEAT+CAPS+SHIFT   ; 56
        DW      SGL+REPEAT+CAPS+SHIFT   ; 57
        DW      SGL+REPEAT+CAPS+SHIFT   ; 58
        DW      SGL+REPEAT+CAPS+SHIFT   ; 59
        DW      SGL+REPEAT+CAPS+SHIFT   ; 60
        DW      SGL+REPEAT+CAPS+SHIFT   ; 61
        DW      SGL+REPEAT+CAPS+SHIFT   ; 62
        DW      SGL+REPEAT+CAPS+SHIFT   ; 63
        DW      SGL+REPEAT+SHIFT        ; 64
        DW      SGL+REPEAT+SHIFT        ; 65
        DW      SPCL                    ; 66
        DW      SGL                     ; 67
        DW      SGL                     ; 68
        DW      SGL+100H                ; 69
        DW      SGL+100H                ; 70
        DW      SGL+100H                ; 71
        DW      SGL+100H                ; 72
        DW      SPCL+100H               ; 73
        DW      SPCL+200H               ; 74
        DW      SPCL                    ; 75
        DW      SGL+REPEAT+CAPS+SHIFT   ; 76
        DW      SGL+REPEAT+CAPS+SHIFT   ; 77
        DW      SGL+REPEAT+CAPS+SHIFT   ; 78
        DW      SGL+REPEAT+CAPS+SHIFT   ; 79
        DW      SGL+REPEAT+CAPS+SHIFT   ; 80
        DW      SGL+REPEAT+CAPS+SHIFT   ; 81
        DW      SGL+REPEAT+CAPS+SHIFT   ; 82
        DW      SGL+REPEAT+SHIFT        ; 83
        DW      SGL+REPEAT+SHIFT        ; 84
        DW      SGL+REPEAT+SHIFT        ; 85
        DW      SPCL+200H               ; 86
        DW      SGL+CAPS+SHIFT+100H     ; 87
        DW      SPCL+REPEAT+100H        ; 88
        DW      SPCL+REPEAT+100H        ; 89
        DW      SGL+100H                ; 90
        DW      SGL+100H                ; 91
        DW      SGL+100H                ; 92
        DW      SGL+100H                ; 93
        DW      SPCL+200H               ; 94
        DW      SPCL+200H               ; 95
        DW      SGL+REPEAT+CAPS+SHIFT   ; 96
        DW      SGL+100H                ; 97
        DW      SPCL+REPEAT+100H        ; 98
        DW      SPCL+REPEAT+100H        ; 99
        DW      SGL+100H                ; 100
        DW      MULT                    ; 101
        DW      SGL+100H                ; 102
        DW      SPCL                    ; 103

;***********************************************;
; attribute data table - shifted mode           ;
;***********************************************;

        DW      SGL                     ;  0
        DW      SGL                     ;  1
        DW      SGL                     ;  2
        DW      SGL                     ;  3
        DW      SGL                     ;  4
        DW      SGL                     ;  5
        DW      SGL                     ;  6
        DW      SGL                     ;  7
        DW      SGL                     ;  8
        DW      SGL                     ;  9
        DW      SGL                     ; 10
        DW      SPCL+100H               ; 11
        DW      SGL+REPEAT+SHIFT        ; 12
        DW      SGL+REPEAT+SHIFT        ; 13
        DW      SGL+REPEAT+SHIFT        ; 14
        DW      SGL+REPEAT+SHIFT        ; 15
        DW      SGL+REPEAT+SHIFT        ; 16
        DW      SGL+REPEAT+SHIFT        ; 17
        DW      SGL+REPEAT+SHIFT        ; 18
        DW      SGL+REPEAT+SHIFT        ; 19
        DW      SGL+REPEAT+SHIFT        ; 20
        DW      SGL+REPEAT+SHIFT        ; 21
        DW      SGL+REPEAT+SHIFT        ; 22
        DW      SGL+REPEAT+SHIFT        ; 23
        DW      SGL+REPEAT+SHIFT        ; 24
        DW      SPCL+REPEAT             ; 25
        DW      SGL                     ; 26
        DW      SGL+REPEAT              ; 27
        DW      SGL+100H                ; 28
        DW      SPCL                    ; 29
        DW      SPCL                    ; 30
        DW      SPCL                    ; 31
        DW      SGL+100H                ; 32
        DW      SPCL                    ; 33
```

Figure 2, continued ....

```
DW      SGL+REPEAT+CAPS+SHIFT       ; 34
DW      SGL+REPEAT+CAPS+SHIFT       ; 35
DW      SGL+REPEAT+CAPS+SHIFT       ; 36
DW      SGL+REPEAT+CAPS+SHIFT       ; 37
DW      SGL+REPEAT+CAPS+SHIFT       ; 38
DW      SGL+REPEAT+CAPS+SHIFT       ; 39
DW      SGL+REPEAT+CAPS+SHIFT       ; 40
DW      SGL+REPEAT+CAPS+SHIFT       ; 41
DW      SGL+REPEAT+CAPS+SHIFT       ; 42
DW      SGL+REPEAT+CAPS+SHIFT       ; 43
DW      SGL+REPEAT+SHIFT            ; 44
DW      SGL+REPEAT+SHIFT            ; 45
DW      SPCL                        ; 46
DW      SGL                         ; 47
DW      SGL                         ; 48
DW      SPCL                        ; 49
DW      SPCL                        ; 50
DW      SPCL                        ; 51
DW      SPCL                        ; 52
DW      SPCL+100H                   ; 53
DW      SPCL                        ; 54
DW      SGL+REPEAT+CAPS+SHIFT       ; 55
DW      SGL+REPEAT+CAPS+SHIFT       ; 56
DW      SGL+REPEAT+CAPS+SHIFT       ; 57
DW      SGL+REPEAT+CAPS+SHIFT       ; 58
DW      SGL+REPEAT+CAPS+SHIFT       ; 59
DW      SGL+REPEAT+CAPS+SHIFT       ; 60
DW      SGL+REPEAT+CAPS+SHIFT       ; 61
DW      SGL+REPEAT+CAPS+SHIFT       ; 62
DW      SGL+REPEAT+CAPS+SHIFT       ; 63
DW      SGL+REPEAT+SHIFT            ; 64
DW      SGL+REPEAT+SHIFT            ; 65
DW      SPCL                        ; 66
DW      SGL                         ; 67
DW      SGL                         ; 68
DW      SPCL                        ; 69
DW      SPCL                        ; 70
DW      SPCL                        ; 71
DW      SPCL                        ; 72
DW      SPCL+100H                   ; 73
DW      SPCL                        ; 74
DW      SPCL                        ; 75
DW      SGL+REPEAT+CAPS+SHIFT       ; 76
DW      SGL+REPEAT+CAPS+SHIFT       ; 77
DW      SGL+REPEAT+CAPS+SHIFT       ; 78
DW      SGL+REPEAT+CAPS+SHIFT       ; 79
DW      SGL+REPEAT+CAPS+SHIFT       ; 80
DW      SGL+REPEAT+CAPS+SHIFT       ; 81
DW      SGL+REPEAT+CAPS+SHIFT       ; 82
DW      SPCL+REPEAT+SHIFT           ; 83
DW      SPCL+REPEAT+SHIFT           ; 84
DW      SGL+REPEAT+SHIFT            ; 85
DW      SPCL                        ; 86
DW      SPCL+CAPS+SHIFT             ; 87
DW      SPCL+REPEAT                 ; 88
DW      SPCL+REPEAT                 ; 89
DW      SPCL                        ; 90
DW      SPCL                        ; 91
DW      SPCL                        ; 92
DW      SPCL                        ; 93
DW      SPCL                        ; 94
DW      SPCL                        ; 95
DW      SPCL+REPEAT+CAPS+SHIFT      ; 96
DW      SGL+100H                    ; 97
DW      SPCL+REPEAT                 ; 98
DW      SPCL+REPEAT                 ; 99
DW      SPCL                        ; 100
DW      SPCL                        ; 101
DW      SPCL                        ; 102
DW      SPCL                        ; 103

;*********************************************;
;  attribute data table  -  alternate mode    ;
;*********************************************;

DW      SGL                         ;  0
DW      SGL                         ;  1
DW      SGL                         ;  2
DW      SGL                         ;  3
DW      SGL                         ;  4
DW      SGL                         ;  5
DW      SGL                         ;  6
DW      SGL                         ;  7
DW      SGL                         ;  8
DW      SGL                         ;  9
DW      SGL                         ; 10
DW      SPCL+LOCAL                  ; 11
DW      SGL+REPEAT+SHIFT            ; 12
DW      SGL+REPEAT+SHIFT            ; 13
DW      SGL+REPEAT+SHIFT            ; 14
DW      SGL+REPEAT+SHIFT            ; 15
DW      SGL+REPEAT+SHIFT            ; 16
DW      SGL+REPEAT+SHIFT            ; 17
DW      SGL+REPEAT+SHIFT            ; 18
DW      SGL+REPEAT+SHIFT            ; 19
DW      SGL+REPEAT+SHIFT            ; 20
DW      SGL+REPEAT+SHIFT            ; 21
DW      SGL+REPEAT+SHIFT            ; 22
DW      SGL+REPEAT+SHIFT            ; 23
DW      SGL+REPEAT+SHIFT            ; 24
DW      SPCL+REPEAT                 ; 25
DW      SGL                         ; 26
DW      SGL+REPEAT                  ; 27
DW      SGL                         ; 28
DW      SPCL                        ; 29
DW      SPCL                        ; 30
DW      SPCL                        ; 31
DW      SGL+100H                    ; 32
DW      SGL+100H                    ; 33
DW      SGL+CAPS+SHIFT              ; 34
DW      SGL+CAPS+SHIFT              ; 35
DW      SGL+CAPS+SHIFT              ; 36
DW      SGL+CAPS+SHIFT              ; 37
DW      SGL+CAPS+SHIFT              ; 38
DW      SGL+CAPS+SHIFT              ; 39
DW      SGL+CAPS+SHIFT              ; 40
DW      SGL+CAPS+SHIFT              ; 41
DW      SGL+CAPS+SHIFT              ; 42
DW      SGL+CAPS+SHIFT              ; 43
DW      SPCL+SHIFT                  ; 44
DW      SGL+SHIFT                   ; 45
DW      SPCL                        ; 46
DW      SGL                         ; 47
DW      SGL                         ; 48
DW      SPCL                        ; 49
DW      SPCL                        ; 50
DW      SPCL                        ; 51
DW      SPCL                        ; 52
DW      SGL                         ; 53
DW      SPCL                        ; 54
DW      SGL+CAPS+SHIFT              ; 55
DW      SGL+CAPS+SHIFT              ; 56
DW      SGL+CAPS+SHIFT              ; 57
DW      SGL+CAPS+SHIFT              ; 58
DW      SGL+CAPS+SHIFT              ; 59
DW      SGL+CAPS+SHIFT              ; 60
```

Figure 2, continued ....

```
        DW      SGL+CAPS+SHIFT          ;  61
        DW      SGL+CAPS+SHIFT          ;  62
        DW      SGL+CAPS+SHIFT          ;  63
        DW      SGL+SHIFT               ;  64
        DW      SGL+SHIFT               ;  65
        DW      SPCL                    ;  66
        DW      MULT+LOCAL              ;  67
        DW      MULT+LOCAL              ;  68
        DW      SPCL                    ;  69
        DW      SPCL                    ;  70
        DW      SPCL                    ;  71
        DW      SPCL                    ;  72
        DW      SPCL                    ;  73
        DW      SPCL                    ;  74
        DW      SPCL                    ;  75
        DW      SGL+CAPS+SHIFT          ;  76
        DW      SGL+CAPS+SHIFT          ;  77
        DW      SGL+CAPS+SHIFT          ;  78
        DW      SGL+CAPS+SHIFT          ;  79
        DW      SGL+CAPS+SHIFT          ;  80
        DW      SGL+CAPS+SHIFT          ;  81
        DW      SGL+CAPS+SHIFT          ;  82
        DW      SGL+SHIFT               ;  83
        DW      SGL+SHIFT               ;  84
        DW      SGL+SHIFT               ;  85
        DW      SPCL                    ;  86
        DW      SGL+CAPS+SHIFT          ;  87
        DW      MULT+LOCAL+100H         ;  88
        DW      MULT+LOCAL+100H         ;  89
        DW      SPCL                    ;  90
        DW      SPCL                    ;  91
        DW      SPCL                    ;  92
        DW      SGL                     ;  93
        DW      SPCL                    ;  94
        DW      SPCL                    ;  95
        DW      SPCL+REPEAT+CAPS+SHIFT  ;  96
        DW      SPCL                    ;  97
        DW      MULT+LOCAL+100H         ;  98
        DW      MULT+LOCAL+100H         ;  99
        DW      SPCL                    ; 100
        DW      MULT                    ; 101
        DW      SPCL                    ; 102
        DW      SPCL                    ; 103

;***********************************************;
;  multiple key data tables                     ;
;***********************************************;

;  this table has a length of 256 bytes

MKTBL   EQU     OFFSET $

MK0     EQU     OFFSET $-MKTBL
        DB      2,'00'          ;double zero

MK1     EQU     OFFSET $-MKTBL
        DB      3,01BH,'xA'

MK2     EQU     OFFSET $-MKTBL
        DB      3,01BH,'yA'

MK3     EQU     OFFSET $-MKTBL
        DB      3,01BH,'xB'

MK4     EQU     OFFSET $-MKTBL
        DB      3,01BH,'yB'

MK5     EQU     OFFSET $-MKTBL
        DB      3,01BH,'xC'

MK6     EQU     OFFSET $-MKTBL
        DB      3,01BH,'yC'

MK7     EQU     OFFSET $-MKTBL
        DB      3,'000'         ;triple zero

;  allocate remainder of table (unused)

        RB      255-(OFFSET $-MKTBL)
        DB      0               ;genemd kludge

;***********************************************;
;  end of keyboard file                         ;
;***********************************************;

ENDKBD  EQU     OFFSET $

        END
```

Figure 2, continued ....

Channel 9000

THE INDEPENDENT NEWSLETTER OF THE VICTOR 9000/SIRIUS 1 COMPUTER

VOL. 1 NO. 2 March 1983

From the Editor:
Thanks! The response to our premier issue has been truly gratifying. We have received much encouragement and many offers of support from dealers and users as well as the people at Victor Technologies. We plan to enhance our efforts to provide as much as we can, of needed and timely information. This issue's tutorial, "The In's and Out's of I/O", is in fact due to requests from our subscribers for information on this subject.

I also want to reiterate my statement in our first issue, that we desire this newsletter to be a two-way communication channel. We are very interested in hearing about what Victor dealers and users are doing with the 9000. The more solid information we have as to what is happening with 9000 hardware and software, the more successful we all can be.

Dateline: Scotts Valley
I had an opportunity, in February, to pay a brief visit to Victor Technologies' offices in Scotts Valley, California. What a beehive of activity. When I arrived, it took me five minutes just to find a parking spot. Scotts Valley is a small resort town in the mountains between San Jose and Santa Cruz, and I was amazed at all the activity.

In assuming the duties of support and marketing, as well as those of manufacturing, Victor, in Scotts Valley, has undergone a very rapid expansion. They have expanded into three buildings, and are currently constructing more space. While all of this cannot help but be somewhat disruptive, Victor has planned ahead for it and appears to be keeping close pace.

The Victor philosophy seems to be to do as much as possible with as few people as possible. This is meant to insure good intra-company communication and avoid the bureaucratic quagmire. While this is a desirable goal, I am sure that it has had some effect on the delivery of product and the dispensing of information. However, Victor is approaching its goal of getting information to its dealers in a timely manner. The Victor Software Status binder (which is hot off the press) is an example of their strategy. This binder, along with its monthly updates, will keep dealers current on the status of Victor software products.

The districts are proceeding as rapidly as possible to staff their local technical support positions. This problem is largely one of finding qualified people (we all know how difficult that is). Prospective applicants are brought to Scotts Valley for extensive interviewing, after which a recommendation is made to the district, as to the applicants qualifications. Again, the plan is sound, but takes time.

In order to maintain the future competence of dealers and staff in the face of rapidly changing technology and applications, Victor will establish, this year, a training institute located in San Louis Obispo, California. The idea is to provide a way to keep staff, dealers, and possibly users, and third party vendors, up to date technically.

There are also plans to set up a "PC conversion" school. This school will provide third party software vendors with the necessary training, so that they can convert programs originally designed for the IBM/PC, expanding them to fully utilize the unique features of the Victor 9000.

The 9000 was designed as a "soft" tooled computer, so that it could meet the varied needs of the international marketplace. It is Victor's desire that software, converted or designed to run on the 9000, be apropos for international distribution. This requires intimate knowledge of the 9000 as well as foreign computer usage methodologies. It is the aim of the above schools to provide this knowledge.

All in all, I was much encouraged by my visit to Scotts Valley. I was treated as an ally, not an interloper. There seems to be a genuine willingness to provide us with whatever information we feel is needed by our readers. They realize that we all aspire to the same goal --success for the Victor 9000.

Victor has chosen to bite-the-bullet in the short term, to insure long term success in its support areas. This has led to a feeling of abandonment by some of those not close to the situation. It is not the case. The work being done by the people at Scotts Valley will be coming to fruition in the near future, and I am sure that we will all be satisfied with the results. Additionally, they realize that it can never be a go-it-alone situation. There must always be feedback. This volition to listen and assist, will only bring us more quickly to our common goal.

James M. Lesher

Third Party Page

Computer Aided Drafting System

While I was in northern California, I also paid a visit to the Sun-Flex Company — the people who make the anti-glare screen for the Victor 9000 as well as most other CRT's. They are the world's leading manufacturer of that type of equipment.

The reason I stopped by was to see a demonstration of the new high resolution graphics package they have specially designed for use with the 9000. The package consists of two components. The first is the TOUCHPEN System. It comprises a transparent, on-screen digitizing mesh that replaces the normal anti-glare screen on the 9000's CRT, a pointing stylus equipped with a replaceable elastomer tip, and a microprocessor driven, intelligent controller card, that plugs into one of the 9000's internal expansion slots. The system works by alternately establishing a horizontal and vertical electric field across the surface of the mesh and then measuring the voltage at the point of contact with the stylus. The controller uses this measurement to interpolate an actual X,Y coordinate pair, compensating automatically for any minor screen distortions. The controller generates 15 X,Y pairs per second. The system has the capability of resolving 8000 points in both the X and Y directions, more than enough to resolve one pixel on the screen.

The second component of the graphics package is AutoCAD, a two-dimensional computer-aided drafting software package. AutoCAD is a well proven product originally developed by AutoDesk, Incorporated, for Z-80 based computers. It acts like a word-processor for drawings. The user can make drawings from simple components such as lines (of any width), circles, arcs, and solid-filled areas. These drawings can be stored on disk and in turn may be used as components of other drawings. This feature gives the AutoCAD user the ability to create, and then select from, customized parts libraries of the items most often used in his drawings. Custom menus can also be created to further simplify the drawing process.

AutoCAD also has a full bidirectional zoom capability, with a ratio of over one million to one between the largest and smallest objects. Objects may be aligned to grid boundaries, or forced to run veritcally or horizontally only. Up to 127 layers and colors may be used, allowing selective viewing or plotting of drawings, as if on transparent overlays. A variety of plotters are supported. The package requires a Victor 9000 with 256K of ram.

As can probably be surmised, I was very impressed with the Sun-Flex package. But I haven't mentioned one of its most impressive points: a suggested retail price of less than $1800.00. The price-performance ratio of this product makes it one of the best.

Sun-Flex plans to provide dealer training and sales support for the package. More information can be obtained by writing or calling:

Sun-Flex Company, Inc.
20 Pimentel Court
Novato, CA 94947
(415) 883-1221

Statistics Package

We received information in the mail concerning a comprehensive statistics package soon to be available for the Victor 9000 running under CP/M-86. The program incorporates an interpretive command processor that allows the user to interactively perform such procedures as analysis of variance, chi-square correlations, cross tabulations, multiple regressions, and more. All functions can also be performed in batch mode, making it easy to do repeated analyses on different data sets. Further information may be obtained from:

Anderson-Bell
5336 S. Crocker St.
Littleton, CO, 80120.
(303) 794-7509

Memory Boards

Computer Technology Innovations, suppliers of add-on products for the IBM/PC, are now offering a series of memory expansion boards designed specifically for the Victor 9000.

VSM-128C	128K Complete memory board
VSM-128X	128K Memory board (expandable to 384K)
VSM-256X	256K Memory board (expandable to 384K)
VSM-384C	384K Complete memory board

Prices for these boards are as follows:

Quantity:	1	2-10	11-50
VSM-128C	315.00	299.00	265.00
VSM-128X	350.00	325.00	299.00
VSM-256X	650.00	575.00	499.00
VSM-384C	899.00	799.00	699.00

These boards are currently in stock at:

Computer Technology Innovations
1037 North Fairoaks Avenue
Sunnyvale, CA 94086
(408) 745-0180

CHANNEL 9000

9742 Marcus Lane, Tujunga, CA 91042
(213) 352-6443
James M. Lesher, Editor
Subscription rate: $30.00 for 6 issues
Published Bimonthly

In's and Out's of I/O

The Victor 9000 has a good mix of input/output ports, providing for a wide range of interface methods and protocols. In this issues tutorial, we discuss the basic hardware features and the more common interface methods. We also discuss some of the software aspects of the various input/output methods available.

Serial Ports

The 9000's two serial ports are configured at the connectors as RS232C DTE type circuits. RS232C refers to a standard published by the Electronic Industries Association (EIA) relating to the interconnection of equipment that communicates serially. DTE stands for Data Terminal Equipment, and generally refers to CRT terminals, printers, and in our case, the Victor 9000. DTE devices are usually connected to Data Communication Equipment (DCE), such as modems. To facilitate these interconnections, the standard defines the circuits such that a DTE device is connected to a DCE device by using a cable where there is a one-to-one correspondence between the pins on either end.

While the EIA standard does not specify a particular type of connector to be used for interconnection, the defacto industry standard has become what is referred to as a subminiature "D" type connector. There is no strong conformity among equipment manufacturers, however, as to the usage of male or female connectors, or even as to which devices use DTE or DCE interfaces.

The following table lists the signals and pins for the 9000's serial port A and B DTE connectors (shown connected to an arbitrary DCE device) which conform to the RS232C standard:

DTE		DCE	Signal
1	<---------->	1	Protective Ground
2	---------->	2	Transmitted Data
3	<----------	3	Received Data
4	---------->	4	Request to Send
5	<----------	5	Clear to Send
6	<----------	6	Data Set Ready
7	<---------->	7	Signal Ground
8	<----------	8	Recvd.Line Signal Det.
15	<----------	15	Xmission Signal Timing
17	<----------	17	Receiver Signal Timing
20	---------->	20	Data Terminal Ready
22	<----------	22	Ring Indicator
24	---------->	24	Transmit Signal Timing

Multi-Protocol Serial Controller

Serial communication is controlled by a device called a Multi-Protocol Serial Controller (MPSC). This device is an Intel 8274 or a NEC uPD-7201. It is beyond the scope of this newsletter (and space available) to give a complete description of the MPSC. The interested user is referred to the NEC and Intel data sheets and application notes. We will, however, mention a few items which may not be contained or evident in the data sheets.

The MPSC is mapped into the memory space rather than the I/O space of the 8088 processor (as are all of the 9000's peripheral devices). The register addresses of the MPSC are as follows:

Register	Address
Port A data	E000:0040
Port B data	E000:0041
Port A command	E000:0042
Port B command	E000:0043

Because the interrupt output of the MPSC is connected to a 8259A type interrupt controller, Write Register 2 (WR2) of Port A should be programmed using the following bit pattern (X's indicate "don't care" values):

```
Msb                 Lsb
 X  0  0  1  0  X  X  X
```

If interrupt operation is used with the MPSC, the MPSC should be programmed for the Status-Affects-Vector mode. The interrupt service routine would then examine Read Register 2 (RR2) of Port B to determine what MPSC condition caused the interrupt. After the interrupt is serviced, the End-of-Interrupt (EOI) command should be written to the MPSC's Port A command register, regardless of which port caused the interrupt.

The interrupt output of the MPSC is connected to Interrupt Request 1 (IR1) of the 8259A interrupt controller. (The base address of the 8259A is E000:0000.) The interrupt vector for IR1 is at address 0000:0104 for MS-DOS, and at address 0000:0044 for CP/M. The IR1 vector is currently not used or initialized by either MS-DOS or CP/M.

Asynchronous Communication

For asynchronous communication (which is how the MPSC is configured by the 9000's BIOS on reset), RS232C connector pins 1 thru 8 and pin 20 are the pins that are generally used. When ports A and B are initialized by the BIOS, pin 5 is enabled to control transmission of data on pin 2, and pin 8 controls reception of data on pin 3. This means that pin 5 must have a positive voltage on it or no data will be sent from the 9000 on pin 2. Likewise, pin 8 must have a positive voltage on it or no data will be received on pin 3. This allows relatively slow devices such as printers to "throttle" the 9000 so that data is not lost.

Auxiliary Signals

Not all of the RS232C signal inputs are available by reading the MPSC registers. The auxiliary signals are brought in through I/O register A on one of the 6522 VIAs (VIA2). VIA2 is also memory mapped and has its base address at E800:0040. Its port A register is at address E800:0041. In addition to being the source of the auxiliary signals, two bits of this register are used to control the selection of the internal or external baud rate clocks. The bit assignments for the VIA port are:

Bit	Signal
0	Serial A Clock Select (0=internal)
1	Serial B Clock Select (0=internal)
2	Port A Ring Indicator
3	Port A Data Set Ready
4	Port B Ring Indicator
5	Port B Data Set Ready
6	Keyboard Data Input
7	Vertical Signal Input

Baud Rates

The baud rate clock for the MPSC can be either internal or external to the 9000. The internal baud rate clock is an 8253 Programmable Interval Timer. The external baud rate clocks come from the RS232C connectors.

For the internal baud rate clocks, counter 0 of the 8253 is used for the Port A baud rate clock and counter 1 is used for the Port B baud rate clock. The input to counters 0 and 1 is a 1.25 MHz signal which is divided down by the counters to give the appropriate clock rate. The 8253 is programmed for Mode 3 operation (square wave output). The 8253 registers are memory mapped at these locations:

Register	Address
Counter 0	E000:0020
Counter 1	E000:0021
Counter 2	E000:0022
Mode Word	E000:0023

The divisors for the common baud rates are given in the following table:

Baud Rate	Low Byte	High Byte
300	04h	01h
600	82h	00h
1200	41h	00h
2400	21h	00h
4800	10h	00h
9600	08h	00h
19200	04h	00h

To change the baud rate for port A to 9600 baud, you would execute the following instruction sequence:

```
MOV  AX,0E000H        ;set I/O base address
MOV  ES,AX
MOV  BX,0020H         ;set 8253 offset
MOV  BYTE PTR ES:3[BX],36H ;load ctr 0 mode 3
MOV  BYTE PTR ES:0[BX],08H ;least sig. byte
MOV  BYTE PTR ES:0[BX],00H ;most sig. byte
```

The external baud rate clocks come from pins 15 and 17 on the RS232C connectors. Pin 15 is external transmit baud rate clock and pin 17 is the external receive baud rate clock. These clock inputs are generally used only with high-speed synchronous modems.

The switching between internal and external baud rate clocks is done through the setting of two output signals of a 6522 VIA, VIA2 (see **Auxiliary Signals**). The current state of these bit settings can be determined by reading the VIA register. For example, if you wanted to change only port A to external clocks, you would execute the following instruction sequence:

```
MOV  AX,0E800H        ;load I/O base address
MOV  ES,AX
MOV  BX,0041H         ;load register offset
MOV  AL,ES:[BX]       ;get current state
OR   AL,01H           ;force bit 0 to 1
MOV  ES:[BX],AL       ;output new state
```

The above sequence maintains register bit 1's previous state. Also, storing to bits that are programmed as inputs has no effect.

External Parallel Port

The parallel port on the back of the 9000 was designed to be used as either a Centronics or IEEE-488 type port. Since it was probably assumed that the Centronics usage would be more common, the connector and pin-out were chosen to conform directly to that type of interface. For IEEE-488 usage, a special adaptor cable must be used.

The port is controlled primarily by one 6522 VIA (VIA1, base address E800:0020), with a few signals going to another 6522 (VIA2). The port signals are buffered using bus transceivers designed for IEEE-488 compatibility (75160 & 75161). The transceivers are wired in such a way that, in the IEEE-488 mode, the 9000 must be the controller (and the only controller) on the buss.

The direction of most of the signals on the connector are controlled by the state of the VIA2-PB2 output. The table below gives, for each pin of the connector, the source and direction with regard to the output of VIA1-PB2. It should be noted that for a VIA2-PB2 low output, the direction of pin 15 is controlled by the output of VIA1-PB3 (pin 35). If the output of VIA1-PB3 is low, pin 15 is an output, otherwise it is an input.

	VIA2-PB2		
Pin	L	H	Signal Source
1	I	O	VIA1-PB0,VIA2-CA2
2	I	O	VIA1-PA0
3	I	O	VIA1-PA1
4	I	O	VIA1-PA2
5	I	O	VIA1-PA3
6	I	O	VIA1-PA4
7	I	O	VIA1-PA5
8	I	O	VIA1-PA6
9	I	O	VIA1-PA7
10	O	I	VIA1-PB6,VIA1-CA1
11	I	O	VIA1-PB5,VIA2-CA2
12	-	-	N/C
13	O	I	VIA1-PB7,VIA1-CA2
14	-	-	N/C or GND
15	I/O	O	VIA1-PB1
16	-	-	GND
17	-	-	CHASSIS GND
18	-	-	N/C
19-29	-	-	GND
30-31	-	-	N/C
32	-	-	TIED TO PIN 13
33	-	-	GND
34	O	O	VIA1-PB2
35	O	O	VIA1-PB3
36	O	O	VIA1-PB4

Care should be taken when configuring the port, to make sure that there are no driver conflicts between the VIAs and the transceivers. In other words, if signals are to be changed from outputs to inputs, change the VIA's direction before changing the transceivers direction. If signals are to be changed from inputs to outputs, change the transceivers direction first, then change the VIA's direction.

The connector on the back of the 9000 is what is termed a ribbon-style connector. This name leads to a lot of confusion. One usually regards the word "ribbon" to refer to the flat,

multi-conductor cable -- ribbon cable. In this case, however, the word ribbon refers to the pins of the connector which are formed from flat "ribbons" of metal. The confusion reaches its height when one discovers that there are ribbon-style connectors made to terminate to ribbon cable.

Centronics Interface

The Centronics interface, as implemented on the 9000, is a very simple, 8-bit, parallel, data transfer protocol. Essentially, all that is done to send a byte is to make sure the printer is on-line and not busy, place the data on the interface data lines, and strobe the printer. The BUSY and ON-LINE signals are high true. The DATA STROBE is a low going pulse.

The current 9000 BIOS programs configure the VIA-PB0 output high and use only the following pins for data transfer:

Pin	Signal
1	Data Strobe (Output)
2	Data 1 (LSB)
3	Data 2
4	Data 3
5	Data 4 (All data are outputs)
6	Data 5
7	Data 6
8	Data 7
9	Data 8 (MSB)
11	Printer Busy (Input)
13	Printer On-Line (Input)

IEEE-488 (GPIB) Interface

The operation of IEEE-488 is far too complex to discuss in the context of this newsletter. We will, however, discuss some items relating to Victor's pending release of its IEEE-488 driver.

The interface connector on the back of the 9000 is not configured for standard IEEE-488 interconnection. The standard IEEE-488 connector is a 24 pin, ribbon-style connector. The table below shows how to wire an adaptor cable to allow connection of IEEE-488 devices to the 9000. In addition, pins 18 - 24 on the IEEE-488 connector should be wired to the GND pins on the Victor 9000 connector.

9000	IEEE-488	Signal
1	6	DAV
2	1	DIO 1
3	2	DIO 2
4	3	DIO 3
5	4	DIO 4
6	13	DIO 5
7	14	DIO 6
8	15	DIO 7
9	16	DIO 8
10	7	NRFD
11	10	SRQ
13	8	NDAC
15	5	EOI
17	12	SHIELD
34	17	REN
35	11	ATN
36	9	IFC

The IEEE-488 driver will be incorporated into the BIOS at system generation time, instead of the Centronics driver. The driver functions will be controlled by sending the appropriate ESCAPE sequences to the driver. As mentioned above, the 9000 must be the only controller on the buss, and will not pass control to any other device.

User Port

The Victor 9000 provides an internal parallel port accessible through a 50 pin ribbon cable type connector. The connector is wired to a 6522 VIA (VIA3) that is memory mapped with a base address of E800:0080. All of the functions of VIA3 are available with the exception of Timer 1 and output PB7, which are used for the CODEC clock (audio output). The configuration of the port connector is given below:

Pin	Signal	Pin	Signal
1	-12 VOLTS	2	-12 VOLTS
3	N/C	4	N/C
5	+12 VOLTS	6	+12 VOLTS
7	+5 VOLTS	8	+5 VOLTS
9	N/C	10	LIGHT PEN IN
11	GND	12	CA1
13	GND	14	CA2
15	GND	16	PA0
17	GND	18	PA1
19	GND	20	PA2
21	GND	22	PA3
23	GND	24	PA4
25	GND	26	PA5
27	GND	28	PA6
29	GND	30	PA7
31	GND	32	PB0
33	GND	34	PB1
35	GND	36	PB2
37	GND	38	PB3
39	GND	40	PB4
41	GND	42	PB5
43	GND	44	PB6
45	GND	46	PB7 (Codec)
47	GND	48	CB1
49	GND	50	CB2

While the usual method of attaching to the user port is by using a ribbon cable and connector, I have found another way that allows the plugging of a small circuit board directly into the connector. AP Products Inc. of Mentor, Ohio, makes a connector (part number 922576-50) intended to be used to attach two ribbon cables to one header. However, this connector can be soldered to a circuit board and used to connect the circuit board to the user port. Care must be taken with the size of the circuit board to insure that it does not hit the disk unit support bracket directly over the user port connector. This limits board height to about 1.9 inches.

I used this method for the design of a board to connect a daisy-wheel printer with a parallel interface to the 9000. On a board 1.9 inches high and 2.7 inches wide, I was able to get four 20-pin ICs, a resistor pack, a 50 pin ribbon cable connector, and the AP Products connector. Designing the interface this way was considerably less expensive than using an expansion buss board.

Device Drivers

Now that we have discussed the hardware aspects of I/O, is seems appropriate to give attention to the software aspects, i.e. device drivers. But first we should define the concept of logical and physical devices.

A logical device is a generalization of a physical mechanism that performs a particular function. Printers, which perform the function of transferring electronic data to paper, are commonly referred to as listing devices. By our definition, a listing device is a logical device, since it doesn't refer to a particular type of printer. A Diablo 630, on the other hand, is a physical device.

MS-DOS and CP/M define four logical devices that they directly support. The first is the CONSOLE device, the principal interactive device for communication with the operator. Second is the LIST device, the device to which hard copy output is usually sent. The last two are the AUXILIARY INPUT and AUXILIARY OUTPUT devices which are non-descript input and output devices that are usually user defined. (An example of an AUXILIARY OUTPUT device would be a second printer.)

Hardware independent software running under MS-DOS or CP/M (meaning software that does not depend on the specific features of a given computer) performs all I/O via these logical devices. The program does not know, or care, about the actual source or destination of its communications. The computer operator connects the desired physical devices to the computer and then tells the operating system (MS-DOS or CP/M) which physical device to associate with each logical device.

However, manufacturers do not know for certain what make, or even what type, of physical device will be connected to their computers. Since they do know what provisions they have made to connect external devices to their computers, meaning I/O ports, they have taken to referring to these I/O ports as the physical devices, rather than the mechanisms to which they are attached.

In addition, the software that drives the I/O port, the device driver, is considered to be a part of the physical device. Some manufacturers provide more than one physical device that all actually use the same I/O port, but incorporate different driver software. This is useful in the situation, for example, where sometimes a printer that requires hardware handshaking is used on a port, and at other times a printer that requires software handshaking is attached to the same port. By changing the logical to physical device assignment, the software that drives the port is changed. (This concept is not currently implemented with MS-DOS or CP/M for the Victor 9000.)

IOBYTE

Each of the four logical devices mentioned above, can be associated with one of four physical devices. Not all physical devices can be assigned to a given logical device. The table below gives the names of the physical devices that can be associated with each logical device. Where the CP/M names differ from the MS-DOS names, the CP/M names are given in parentheses.

Logical Device	Physical Devices			
CON	TTY	CRT	BAT	UC1
AUXIN (AXI or RDR)	TTY	PTR	UR1	UR2
AUXOUT (AXO or PUN)	TTY	PTP	UP1	UP2
LST	TTY	CRT	LPT	UL1

The following table describes the physical devices that are currently implemented on the 9000.

Physical Device	Description
CRT	Keyboard input / CRT display output
TTY	Serial port A
LPT	External parallel port (Centronics)
UL1	Serial port B
BAT	Input from AUXIN / output to LST

The current state of the logical to physical device assignments is kept in a byte in system memory called the IOBYTE. The IOBYTE is divided into four two-bit fields, one field for each logical device. The value of a field (0-3) indicates which physical device is assigned to its corresponding logical device. The field definitions are outlined below.

CON (Bits 0,1)	AUXIN (Bits 2,3)
0 - TTY	0 - TTY
1 - CRT	1 - PTR
2 - BAT	2 - UR1
3 - UC1	3 - UR2

AUXOUT (Bits 4,5)	LST (Bits 6,7)
0 - TTY	0 - TTY
1 - PTP	1 - CRT
2 - UP1	2 - LPT
3 - UP2	3 - UL1

It is possible to examine and set the IOBYTE by using the STAT utility under CP/M or the SETIO utility under MS-DOS. It is also possible to examine and set the IOBYTE from a program by using one of the following sequences.

```
; CP/M read IOBYTE into register al

     MOV  DX,OFFSET PB1     ;load parameter offset
     MOV  CL,50             ;direct bios call fn
     INT  224               ;CP/M bios interrupt
-------------------------
PB1  DB   19    ;get IOBYTE function
PB2  DW   0     ;cx register
PB3  DW   0     ;dx register


; CP/M set IOBYTE from register al

     MOV  AH,0              ;zero upper byte of wd
     MOV  PB2,AX            ;set IOBYTE in parm blk
     MOV  DX,OFFSET PB1     ;load parameter offset
     MOV  CL,50             ;direct bios call fn
     INT  224               ;CP/M bios interrupt
-------------------------
PB1  DB   20    ;set IOBYTE function
PB2  DW   0     ;cx register
PB3  DW   0     ;dx register
```

```
; MS-DOS get IOBYTE to register al

     MOV  AX,DS      ;set ES to data segment
     MOV  ES,AX
     LEA  BX,PB1     ;get offset of param block
     MOV  AX,16      ;get IOBYTE function
     INT  223        ;Victor super bios interrupt
     MOV  AX,PB1     ;get IOBYTE to al (0 to ah)
-------------------------------------
PB1  DW   0     ;parameter passing area

; MS-DOS set IOBYTE from register al

     MOV  AH,0       ;set upper byte to 0
     MOV  PB1,AX     ;set IOBYTE in param block
     MOV  AX,DS      ;set ES to data segment
     MOV  ES,AX
     LEA  BX,PB1     ;get offset of param block
     MOV  AX,17      ;set IOBYTE function
     INT  223        ;Victor super bios interrupt
-------------------------------------
PB1  DW   0     ;parameter passing area
```

Custom Device Drivers

Notice:
In preparing the following section, it was discovered that some of the features mentioned did not work, or contained bugs. Specifically, function 5 for the List Vector and Console Vector calls does not work because the BX register is clobbered by the BIOS code, and the "pass-on" feature passes on one character too many (the extra character is meaningless). Victor has stated that since these are undocumented features of the BIOS, they are not directly supported except as they pertain to Victor programs. Therefore, the likelyhood of the bugs being fixed is small. It is also probable that the implementation of these features will change in future operating system releases, but the basic concepts will remain. We decided to include the discussion of these features anyway, in case the bugs are fixed, and as information to users who might have wanted to use these features.

At sometime it may be required to attach the 9000 to a device that is not compatible with the standard device drivers supplied by Victor. This was the case when I plugged my old Diablo 1620 into one of the 9000's serial ports. My Diablo does not support hardware handshaking, and using the Victor driver, I was forced to run at 300 baud or lose data. Because of this, I decided to write my own device driver that would support the Diablo's ETX/ACK software protocol.

It turns out that there are two ways of incorporating a different device driver into the 9000 system. The first, which is only viable for MS-DOS, involves writing a replacement module for the standard one supplied by Victor as part of their system generation program. The sysgen program uses MS-LINK to combine pre-assembled modules to form the MS-DOS system file. By writing a new module and changing the linker command file to cause the linker to load the new module instead of the standard one, it is possible to create a system disk incorporating the new driver. This task requires an intimate knowledge of the 9000 BIOS and should only be attempted by experienced system programmers.

Fortunately, the second method is somewhat easier. The 9000 BIOS routes all calls to the LST device through a long vector stored in memory. The BIOS also provides a simple method by which to examine or change this vector. A device driver written to use this second method is a COM or CMD file that consists of two parts. The first part is the initialization section, which configures the port hardware, changes the list vector to point to the new list driver entry point, and then exits to the system, leaving the module permanently resident in memory. The second part is the device driver itself. The module will remain as the new device driver until the system is reset.

The second method is the method I used to implement my Diablo driver. The driver for MS-DOS is shown in Listing 1. The CP/M driver is similar to the MS-DOS driver except for initialization and differences in the assemblers. The initialization part of the driver for CP/M is shown in Listing 2. The CP/M driver has a somewhat more complicated initialization section because there is no direct way to make a program permanent in memory. Even though a program terminates, leaving itself resident, a CONTROL-C entered by the operator can cause the program's memory to be deallocated.

List Vector
Before doing an indirect long call through the list vector, the BIOS sets the AX register to indicate the function to be performed.

AX	Function
0	Get data from list device
1	Send data to list device
2	Get list device input status
3	Get list device output status
4	Get list device input vector
5	Set list device input vector

Data to be output is passed in the CL register and input in the AL register. Status is returned in the AL register (00h is busy or no data ready, FFh is not busy or data waiting).

The list device input vector (functions 4 and 5) is a long vector containing the address of the list device input routine. The standard BIOS does not provide an input routine for the list driver and this vector points to a null subroutine that returns zero in the AX, BX, and ES registers. This vector can be used when it is desired to add an input routine for the list device, accessible through the standard list vector, without changing the list device output routine. The list input vector is set from and returned in registers ES (segment) and BX (offset).

Output to the LST device from MS-DOS or CP/M produces calls through the main list vector for functions 1 and 3 only (see above). There is no way to get input from the LST device using standard system calls. In order to call the other functions, the user must make a BIOS call to get the list vector and then do a long indirect call using the address obtained. An example of getting the list input status is given below:

```
        MOV   AX,DS       ;set ES to data segment
        MOV   ES,AX
        LEA   BX,PB1      ;set offset of param block
        MOV   AX,14       ;get device vector function
        INT   223         ;Victor super bios interrupt

        MOV   AX,2        ;get input status function
        CALL  DWORD PTR LSTV ;call through vector
                          ;the input status is in AL
        ---------------
PB1   DW    2     ;list vector function
LSTV  DD    0     ;list vector is returned here
                  ;first word is offset
                  ;second word is segment
```

The following code sequence is used to change the list vector:

```
        MOV   AX,DS       ;set ES to data segment
        MOV   ES,AX
        LEA   BX,PB1      ;get offset of param block
        MOV   AX,15       ;set device vector function
        INT   223         ;Victor super bios interrupt
        ---------------
PB1   DW    2           ;list vector function
      DW    LSTOFF   ;new list vector offset
      DW    LSTSEG   ;new list vector segment
```

Console Vector

The 9000's BIOS also provides a vector scheme for the console device (CON), similar to that for the list device outlined above. This scheme is very useful for adding functions to the keyboard or display. For example, it is used by Victor to implement the calculator program and graphics package. The console vector may be examined or set using the following routines:

```
;  get console vector

        MOV   AX,DS       ;set ES to data segment
        MOV   ES,AX
        LEA   BX,PB1      ;get param block offset
        MOV   AX,14       ;get device vector function
        INT   223         ;Victor super BIOS interrupt
        ---------------
PB1   DW    1           ;console vector function
CONV  DD    0           ;console vector returned here
                        ;first word is offset
                        ;second word is segment


;  set console vector

        MOV   AX,DS       ;set ES to data segment
        MOV   ES,AX
        LEA   BX,PB1      ;get param block offset
        MOV   AX,15       ;set device vector function
        INT   223         ;Victor super BIOS interrupt
        ---------------
PB1   DW    1           ;console vector function
      DW    CONOFF   ;new console driver offset
      DW    CONSEG   ;new console driver segment
```

As with the list driver, the AX register is used to indicate the function to be performed by the console driver.

AX	Function
0	Get data from console (i.e. keyboard)
1	Send data to console (i.e. display)
2	Get console input status
3	Get console output status
4	Get user defined console vector
5	Set user defined console vector

Data to be output is passed in the CL register and input in the AL register. Status is returned in the AL register (00h is busy or no data ready, FFh is not busy or data waiting).

The user defined console (UDC) vector (see functions 4 & 5) is provided to allow a simple method of switching between two the standard console drivers (TTY and CRT) and a custom driver. Setting the IOBYTE (see above) to map the CON device to the UC1 driver causes all console I/O to be routed through the UDC vector. This reduces switching between standard and custom driver to a simple and familiar process that can be done in command mode or under program control.

There is an additional feature provided in conjunction with the use of the UDC vector. This feature is known as the pass-on feature. It allows for the "passing on" of ESCAPE sequences that are not part of the standard CRT driver. In other words, if the CRT driver determines that an ESCAPE sequence it has received is not one that it was designed to recognize, the CRT driver will pass it on through the UDC vector to a secondary, or background, driver.

Furthermore, the UDC driver can cause a flag to be set so that it directly receives all data subsequent to the ESCAPE sequence, preventing the data from going to the CRT driver. When the UDC driver decides it does not need further data it causes the flag to be reset, thus resuming its background status. The flag is set by the UDC driver returning a non-zero value in the AL register whenever it is called. The flag is reset by the UDC driver returning zero in the AL register.

```
; replacement diablo driver using etx/ack protocol

ioseg   equ     0E000h          ;i/o base address
sioda   equ     40h             ;MPSC port offsets
siodb   equ     41h
sioca   equ     42h
siocb   equ     43h
ctc0    equ     20h             ;8253 port offsets
ctc1    equ     21h
ctc2    equ     22h
ctcctl  equ     23h

cseg    segment public
        assume  cs:cseg,ds:cseg

        org     05Ch
fcb     db      1 dup (?)       ;use file name for port spec
        org     100h            ;this org for COM file

        public  dbdinit

dbdinit proc    near            ;init entry point
        mov     ax,cs
        mov     ds,ax
        mov     es,ax
        mov     byte ptr portf,0        ;assume port A
        mov     al,fcb+1        ;get port selection
        and     al,0DFh         ;make upper case
        mov     okmsg1,al       ;put port in message
        cmp     al,'A'
        jz      s0              ;port A
        cmp     al,'B'
        jz      s               ;skip if valid port
        jmp     s9              ;port selection error

s:      mov     byte ptr portf,0FFh     ;select port B

s0:     mov     bx,offset oldvec
        mov     ax,14           ;get print vector
        int     223             ;super bios call
        mov     ax,6            ;test for alt driver
        call    dword ptr prtvec
        cmp     ax,01ABh        ;diablo drive installed ?
        jnz     s1              ;not installed, continue
        mov     dx,offset msg1  ;driver already installed
        mov     ah,9
        int     33
        xor     ax,ax           ;exit to system
        int     33

s1:     mov     ax,cs           ;store segment in parameter
        mov     vector+4,ax     ;passing block
        mov     es,ax           ;and in ES for call to bios
        mov     bx,offset vector        ;get parameter blk address
        mov     ax,15           ;set new device vector
        int     223             ;super bios call
        mov     bx,ioseg        ;set print port to 1200 baud
        mov     es,bx
        cmp     byte ptr portf,0        ;which port
        jnz     s2

;  initialize for port A

        mov     byte ptr es:ctcctl,36h  ;init port A to 1200 baud
        mov     byte ptr es:ctc0,41h
        mov     byte ptr es:ctc0,00h

        mov     byte ptr es:sioca,18h   ;reset channel
        nop                             ;wait awhile
        nop
        nop
        nop
        mov     byte ptr es:sioca,4     ;init 7201
        mov     byte ptr es:sioca,047h  ;16x clk,1 stop,even parity
        mov     byte ptr es:sioca,3
        mov     byte ptr es:sioca,041h  ;rx enable 7 bits
        mov     byte ptr es:sioca,5
        mov     byte ptr es:sioca,0AAh  ;tx enable 7 bits
        mov     byte ptr es:sioca,2
        mov     byte ptr es:sioca,010h
        mov     byte ptr es:sioca,1
        mov     byte ptr es:sioca,000h  ;no interrupts
        jmp     s3

;  initialize for port B

s2:     mov     byte ptr es:ctcctl,76h  ;init port B to 1200 baud
        mov     byte ptr es:ctc1,41h
        mov     byte ptr es:ctc1,00h

        mov     byte ptr es:siocb,18h
        nop
        nop
        nop
        nop
        mov     byte ptr es:siocb,4     ;init 7201
        mov     byte ptr es:siocb,047h
        mov     byte ptr es:siocb,3
        mov     byte ptr es:siocb,041h
        mov     byte ptr es:siocb,5
        mov     byte ptr es:siocb,0AAh
        mov     byte ptr es:sioca,2
        mov     byte ptr es:sioca,010h
        mov     byte ptr es:siocb,1
```

Listing 1.

```
        mov     byte ptr es:siocb,000h  ;no interrupts

s3:     mov     ah,9                    ;all ok message
        lea     dx,okmsg
        int     33
        mov     dx,offset endall
        int     39                      ;exit but remain resident

s9:     mov     ah,9                    ;invalid spec
        lea     dx,inmsg
        int     33

        xor     ax,ax
        int     33                      ;exit to system

dbdinit endp

;  list call entry point

dbdent  proc    far

dbde1:  mov     bx,ioseg                ;point to i/o base
        mov     es,bx
        cmp     al,3                    ;test function number
        jz      dbdosx                  ;status check
        cmp     al,1
        jz      dbd                     ;print char
        cmp     al,6                    ;installation check?
        jnz     dbde2
        mov     ax,0D1ABh               ;return driver ID message
        ret

dbde2:  xor     al,al                   ;return to caller
        dec     al
        ret

; see if we can output a byte and if so, do it

dbd:    call    dbdos                   ;can we send ?
        or      al,al
        jz      dbd                     ;loop if not
        cmp     byte ptr cs:portf,0     ;which port ?
        jnz     dbd0                    ;skip if port B
        mov     es:sioda,cl             ;send char to port A
        jmp     dbd1

dbd0:   mov     es:siodb,cl             ;send char to port B

dbd1:   lea     bx,cs:hscnt             ;check char count
        dec     byte ptr cs:[bx]
        cmp     cl,01Bh                 ;sent esc ?
        mov     al,cs:[bx]
```

Listing 1, continued ...

```
        jnz     dbd3                    ;skip if not esc
        cmp     al,2                    ;force 2 chars after esc
        jnb     dbd2                    ;ok, more than 2 to go
        mov     byte ptr cs:[bx],2      ;make count 2

dbd2:   ret

dbd3:   or      al,al                   ;rdy for handshake ?
        jnz     dbd2                    ;exit if not
        mov     byte ptr cs:dbdcts,1    ;flag time for shake
        ret

;  status check and handshake routine

dbdosx: call    dbdos                   ;call status routine
        ret

dbdent  endp

;  output status near procedure

dbdos   proc    near
        cmp     byte ptr cs:dbdcts,2    ;waiting for ack ?
        jnz     dbdos1                  ;skip if not

;  must wait for ack, check for incomming char

        call    dbdsts                  ;get port status
        and     al,01h                  ;char rcvd ?
        jz      dbdosb                  ;exit if not
        call    dbddta                  ;get char
        and     al,07Fh                 ;strip parity
        sub     al,06h                  ;sub ack
        jnz     dbdosb                  ;not ack, exit
        mov     cs:dbdcts,al            ;clear flag
        mov     byte ptr cs:hscnt,32    ;reset char counter

dbdos1: call    dbdsts                  ;get status
        and     al,04h                  ;can we send ?
        jz      dbdosb                  ;no, exit
        mov     al,byte ptr cs:dbdcts   ;time to shake ?
        or      al,al
        jnz     dbdos2                  ;yep, shake it
        dec     al                      ;indicate ready to send
        ret

dbdos2: inc     al                      ;inc flag indicating etx sent
        mov     cs:dbdcts,al
        cmp     byte ptr cs:portf,0     ;which port ?
        jnz     dbdos3                  ;skip if port B
        mov     byte ptr es:sioda,03h   ;send etx to port A
        jmp     dbdosb                  ;exit
```

```
dbdos3: mov     byte ptr es:siodb,03h   ;send etx to port B

dbdosb: xor     al,al                   ;indicate busy
        ret

;  get port status to al

dbdsts: cmp     byte ptr cs:portf,0     ;which port ?
        jnz     dbdst1                  ;skip if port B
        mov     al,es:sioca             ;get port A status
        ret

dbdst1: mov     al,es:siocb             ;get port B status
        ret

;  get port data to al

dbddta: cmp     byte ptr cs:portf,0     ;which port ?
        jnz     dbddt1                  ;skip if port B
        mov     al,es:sioda             ;get port A data
        ret

dbddt1: mov     al,es:siodb             ;get port B data
        ret

dbdos   endp

;  data storage

vector  dw      2               ;new driver vector block
        dw      dbdent
        dw      0

oldvec  dw      2               ;old driver vector block
prtvec  label   dword
prtoff  dw      0
prtseg  dw      0

hscnt   db      32              ;handshake counter
dbdcts  db      0               ;handshake flag
portf   db      0               ;port A/B flag

okmsg   db      'Diablo ETX/ACK protocol established '
        db      'on port '
okmsg1  db      'B at 1200 baud.',0dh,0ah,'$'
msg1    db      'Diablo driver already installed.',07h,0dh,0ah,'$'
inmsg   db      'Invalid port specification.',07h,0dh,0ah,'$'

endall  equ     $
cseg    ends
        end     dbdinit
```

Listing 1, continued ...

```
;  replacement diablo driver using etx/ack protocol

ioseg   equ     0E000h          ;i/o base address
sioda   equ     40h             ;2701 port offsets
siodb   equ     41h
sioca   equ     42h
siocb   equ     43h
ctc0    equ     20h             ;8253 port offsets
ctc1    equ     21h
ctc2    equ     22h
ctcctl  equ     23h
fcb     equ     5Ch             ;file name offset

        cseg                    ;small system model
        org     100h

dbdinit:
        mov     byte ptr portf,0        ;assume port A
        mov     al,.fcb+1               ;get port selection
        and     al,0DFh                 ;make upper case
        mov     okmsg1,al               ;put port in message
        cmp     al,'A'
        jz      s0                      ;port A
        cmp     al,'B'
        jz      s                       ;skip if valid port
        jmp     s9                      ;port selection error

s:      mov     byte ptr portf,0FFh     ;select port B

s0:     mov     bx,offset oldvec        ;parameter block
        mov     ax,14                   ;get print vector fn
        int     223                     ;super bios call

        mov     ax,6                    ;test for alt driver
        callf   dword ptr prtvec        ;call current driver
        cmp     ax,0D1ABh               ;diablo drive installed ?
        jnz     s1                      ;not installed, continue
        mov     dx,offset msg1          ;driver already installed
        mov     cl,9                    ;send msg
        int     224                     ;CP/M call
        xor     ax,ax                   ;exit to system
        int     224                     ;CP/M call (no return)

;  in CP/M, programs cannot be made permanently resident as in MS-DOS
;  Entering ^C causes the previously allocated memory area to be
;  deallocated. Therefore, we load our program into high memory and
;  then fool CP/M into thinking there is less memory, so that it does
;  not overwrite our program. (See CP/M-Operating System System Guide,
;  pages 48-51.

s1:     mov     cl,27                   ;get bios segment to ES
        int     224                     ;CP/M call
        push    es
        mov     dx,offset pblk          ;direct bios call
        mov     cl,50                   ;to get MRT pointer
        int     224                     ;CP/M call

        pop     es                      ;restore bios segment
        mov     ax,cs                   ;get current segment
        sub     ax,es:1[bx]             ;sub start seg from current
        mov     es:3[bx],ax             ;store new region length

        mov     ax,cs                   ;store segment in parameter
        mov     vector+4,ax             ;passing block
        mov     es,ax                   ;and in ES for call to bios
        mov     bx,offset vector        ;get parameter blk address
        mov     ax,15                   ;set new device vector
        int     223                     ;super bios call

;  set printer port to 1200 baud

        mov     bx,ioseg                ;get i/o base to ES
        mov     es,bx
        cmp     byte ptr portf,0        ;which port
        jnz     s2                      ;jmp if not A

;  from here on, the program is the same as MS-DOS
```

Listing 2.

Channel 9000

THE INDEPENDENT NEWSLETTER OF THE VICTOR 9000/SIRIUS 1 COMPUTER

VOL. 1 NO. 3 May 1983

From the Editor:
Victor has come a long way since last year, when all that was available in the way of an operating system was CP/M-86, and available application software, at best, was limited. At the National Computer Conference (NCC) held this May, Victor showed that it is quickly approaching a position of software parity with Big Blue. Demonstrated at NCC were such items as Lotus' 1-2-3, UNIX, and Networking.

In this issue, we discuss the various operating systems and networks that are, or will shortly be, available for the Victor 9000. As you will see, there is as wide a range of systems that run on the 9000 as anyone could possibly use. Besides MS-DOS 1.25 and CP/M-86 (which come standard), you can get the UCSD P-system, FORTH, BOSS (a single or multi-user system from the UK), and within the next few months, MD-DOS 2.0, UNIX System 3, and XENIX.

Operating Systems
Unfortunately, having so many choices presents us with the dilemma of selecting the most desirable system for our use. As for the standard operating systems, there is little operational difference between CP/M-86 and MS-DOS 1.25. The selection of one over the other is largely a matter of personal preference. However, if you don't have a strong preference, there are a few facts that might influence your choice.

MS-DOS is the most widely used (over 90 percent) operating system for 16-bit microcomputer systems. Because of this, there is more system and application software available for MS-DOS 1.25 than for CP/M-86. Another significant point for MS-DOS 1.25 is that Victor has stated that it plans to support MS-DOS, as part of its product line, more fully than CP/M-86. This is evidenced by Victor's hard-disk system only running MS-DOS, although it comes with a CP/M-86 emulator that allows you to run most CP/M-86 programs.

There is, however, no truth to the published reports that Victor had dropped CP/M-86 from its offering. In fact, CP/M-86 does have one point in its favor worth mentioning. It is that programs compiled under CBASIC using CP/M-80 will usually run, without recompiling, on the 9000, using CRUN86. For those who have spent a lot of time and money developing application software using CBASIC, this could well be the deciding factor in the choice of an operating system.

Networks
I want to interject a few words on Victor's philosophy relating to muli-user operating systems. Victor believes, and I tend to agree, that most microcomputer configurations do not provide sufficient processing and storage capabilities to adequitely support multiple users. Also, in the case of a multi-user 9000 system, it would require having essentially diskless 9000s for use as terminals, if one wanted to have the same display and keyboard features as in the main unit. For this reason, Victor feels that a better solution to the multi-user problem is networking.

Networking is the interconnection of multiple processors through communication channels that are either public (i.e. the telephone system) or private (i.e. a wire). The purpose of networking is to provide shared access to resources that are not economical or practical to duplicate at each work station. These shared resources include items such as large data bases and high-speed printers.

While networks can span miles, or even states, the usual network consists of a group of processors in the same office or building. These local networks can be as simple as two processors attached to the same hard disk controller, to as complex as a system of multiple work stations and network servers.

The two most sophisticated newtorks running on the Victor are covered in this issue.

Wordstar
I have received many requests for information on how to patch Wordstar, in order to implement features that are not accessible through the standard install program. Presented in this issue is a "tutorial" on the inner workings of Wordstar. Most of the relevant (to the Victor 9000) patch areas are listed, along with a brief description of the effects that they control.

In our next issue, we will present a number of examples of special effects that may be "patched" into Wordstar. We will also give a method for providing a "front-end" that will allow you to select the initial state Wordstar from a list of formats (i.e. letters, reports, etc.), eliminating the necessity of keeping separate versions of Wordstar for each format used.

James M. Lesher

Operating Systems

UCSD p-System

The UCSD p-System was developed about six years ago as part of a project thats goal was to create a software system that would support machine independence, standardized programming methods (and languages), and simplicity of operation ("user friendliness"), without sacrificing capability. These goals were achieved, and the UCSD p-System is available for most computer systems, including the Victor 9000.

The p-System is an integrated operating system that allows access to all system functions thru single character commands from a one-line menu at the top of the screen. The menu changes to reflect the area that you are working in at the moment; a different menu for the editor, the file handler, etc. While simple in use, the menus provide quick access to many powerful system commands.

The standard components of the p-System give the user a very complete and capable system. The main component is the System Foundation, which includes the editor, file system, debugger, etc. Next is the UCSD Pascal Compiler, considered to be one of the best available. The implimentation of the p-System on the Victor 9000 includes what is called Extended Memory, which allows the usage of up to 128K of memory for program and data storage. Last, but definitely not least, is Turtlegraphics. Turtlegraphics is a simple but very powerful graphics interface which supports the full resolution of the 9000's screen. It is callable from Pascal and is compatible with a large existing base of packages.

Optional packages available for the p-System are a Fortran 77 compiler and a BASIC compiler, both of which produce modules which can be linked with Pascal. Also available is a more sophisticated editor and a CP/M file transfer utility program.

The UCSD p-System for the Victor 9000 is distributed by:

TDI Systems, Inc.
620 Hungerford Drive, Suite 33
Rockville, MD 20850
(301) 340-8700

In Canada, their address is:

66 Twenty-Third Street
Toronto, Ontario M8V 3N2
(416) 259-5081

BOS - Business Operating System

The BOS operating system is type of environment one would expect to see running only on a medium-scale minicomputer. The features it contains make it an excellent system for implementing complex commercial applications. BOS is available in both single-user (BOS) and multi-user (MBOS) configurations. On the Victor 9000, MBOS can support the main unit, two terminals and one printer, or the main unit, one terminal and two printers.

MBOS is currently the ONLY multi-user, multi-tasking operating system available for the 9000.

BOS has been marketed since 1981, following five years of development by one of England's largest software houses. BOS was designed to be a computer-independent operating system, and is now available on over 50 different micro and mini computers.

All of BOS's sophistication does not mean that it is difficult for the average person to use or understand. BOS provides a full menu building mechanism and system customization utilities that make for an easy to use system. These utilities, along with features such as program and file password protection, automatic logical unit assignments, volume ID checking, and record locking, make BOS a system that is difficult to get into trouble using.

Programming with BOS is done through the use of the Microcobol compiler. This compiler incorporates the major features of mainframe ANS COBOL and provides for interactive operation with formatted displays and system configuration inquiry. Also supported is record locking on shared files, six file access methods including ISAM and RSAM, overlays, and an assembly language interface.

A number of application packages are available that run under BOS. These include word processing (AUTOWRITER), a full accounting package, and data base management type programs (AUTOCLERK and AUTOINDEX). In addition, because BOS was designed for computer independence and has been supported on many different computers for some time, there is a great deal of third-party application software available that is ready to run on the 9000.

Names of local distributors or dealers of BOS may be obtained by contacting:

BOS National Division
I-Concepts, Inc.
2560 Royal Lane, Suite 228
Dallas, Texas 75229
(214) 484-2717

CHANNEL 9000

9742 Marcus Lane, Tujunga, CA 91042
(213) 352-6443
James M. Lesher, Editor
Subscription rate: $30.00 (US) for 6 issues
Overseas Air Mail Rate: $37.50 (US) for 6 issues
Published Bimonthly

Forth

FORTH is an operating system / programming language developed to be a simple interface between the user and his computer. The FORTH system consists of a built in dictionary of procedures called 'words'. Any of these words can be executed simply by typing in the word at the system prompt.

New procedures can be developed and run interactively, using the predefined words and FORTH's reverse polish command syntax. These new procedures can be given names and be stored in the dictionary, thus creating new words. Words entered into the dictionary can be 'forgotten', once they are no longer needed for a particular session, making FORTH very versitile and unlimited in scope. Entire application and turnkey systems can be created using these features.

Words are stored in the dictionary in compiled format, so that execution of FORTH procedures is very fast. The source code for words is stored in what are called 'screens'. Screens can be created, modified, and deleted using FORTH's built in editing features.

FORTH for the Victor 9000 is available in both beginners and professional versions. The beginners version includes a screen editor, 8086/8088 assembler, graphic interface, sound generation, math extentions, and games. The professional version has, in addition, on-line documentation, a decompiler, a debugger, a resident operating system file handler, and many performance improvements.

FORTH is available to run under CP/M-86, MS-DOS, or in a stand alone version. Information can be obtained by contacting:

Dai-E Systems, Inc.
11001 S. W. Barnes Road
Portland, Oregon 97255
(503) 646-6159

Comming Attractions

The following operating systems were running at NCC and should be ready for release in a few months. MS-DOS 2.0 was running on a hard disk system which was being used as the file server of Victor's networking system. Victor's UNIX is an adaptation of System III, a commercial version first released by Western Electic in 1981.

MS-DOS 2.0

MS-DOS 2.0 is the next step in Microsofts goal of producing a machine independent operating system, capable of supporting the rapidly expanding features found in todays microcomputers.

MS-DOS 2.0 is a step upward from MS-DOS 1.25. All of the features contained in version 1.25 are in version 2.0. This means that users upgrading to version 2.0 are still able to work in a familiar environment, using just the MS-DOS 1.25 commands, until they are ready to learn the new features.

The new features found in MS-DOS 2.0 are intended to provide the support needed to manage a system with large amounts of disk storage. They also solve the problem of adding additional disk units from other manufactures. With MS-DOS 1.25, installing an external hard disk means modifying the operating system, since all I/O is handled directly by the BIOS. MS-DOS 2.0 includes a feature called installable device drivers. This mechanism allows the user to let the system know, through entries in a file called CONFIG.SYS, that another disk unit is attached to the system. When MS-DOS 2.0 boots, it uses the CONFIG.SYS file to determine what additional devices are connected to the computer, and loads the drivers for those devices as part of the operating system. These new drivers can suppliment or replace the default drivers.

Since MS-DOS 2.0 has the capability of accessing virtually unlimited amounts of mass storage, the file system has also been enhanced to make keeping track of things more manageable. The file system is modeled after that of UNIX (or XENIX, Microsoft's offering of UNIX), which allows for multiple directories on each logical storage unit.

The directories are arranged in what is called a tree structure. The highest level directory, the root directory, can contain names of files and also names of other directories. In turn, each sub-directory can contain more file names and further sub-directories. By grouping logically related files and programs under related directories, the management of the large amounts of data usually found on hard disk units becomes simpler.

Another feature of version 2.0 is input and output redirection. This allows the input that would normally come from the keyboard, and the output that normally goes to the display, to be redirected to come from, or go to, another device or a file. For example, the command DIR >DIRLIST would send the list of files produced by the DIR command to the file DIRLIST. One could then use the command SORT <DIRLIST to produce a sorted listing of the files on the display (the statement <DIRLIST tells SORT to get its input from the file DIRLIST instead of the keyboard).

An extension of the redirection feature is piping. Piping allows the output of one command to be sent directly to the input of another command, without the necessity of specifying an intermediate file. By using a pipe, the operation of producing a sorted directory listing, as shown above, can be reduced to the command DIR|SORT. The vertical bar between the two commands indicates that the output of the first is to be saved temporarily and used as the input for the second.

There are many more useful features in MS-DOS 2.0 than there is space to discuss. It is enough now to say that version 2.0 will be a significant step forward in providing the Victor 9000 the means by which its inherent capability can be utilzed.

UNIX

UNIX is a powerful and flexible system intended primarilly for programmers. It is multi-tasking and multi-user (although the multi-user aspect is not heavily supported by Victor). UNIX is more than just an operating system. It is a collection of systems and software that are designed to

increase programmer productivity. This software is divided into six catagories: the operating system, languages, text processors, information handlers, graphics, and miscellaneous utilities.

The file system is a hierarcical, tree-structured system, similar to the MS-DOS 2.0 system described above. Features such as I/O redirection, foreground and background program execution, concurrent processes, batch control files, and chaining of programs are all supported.

The UNIX system includes a C compiler and a number of supporting utilities. These utilities include a program verifier and a program beautifier. There is an interactive text editor, and other programs such as a spelling checker, a key-word indexing utility, and file encrypter and decrypter. There is also an elaborate text formatting program that is capable of driving a phototypesetter. In all, there are over 200 subsystems included as part of the UNIX System III package.

XENIX

The Unidot Company is currently supporting an effort to bring Microsoft's multi-user XENIX operating system (another unix derivative) up on the Victor 9000. It is not known at this time how the system will be distributed, but it should be available later this year. For more information contact:

UNIDOT
568 Weddell Drive, Suite 4
Sunnyvale, California 94086
(408) 745-0505

Networks

VICTOR Server Network

The Victor network offering is a local network based on Omninet. Up to 54 work stations and 10 network servers can be connected to the network. The Victor network is termed a linear contention network, because all stations and servers are connected together using one cable that runs from unit to unit. All communications between work stations and network servers occur over this one cable, and the different stations must 'contend' for use of the cable.

Connection to the network is accomplished by installing an interface card, called a transporter, in one of the expantion slots in the Victor 9000, and hooking it up to the network cable. Any model Victor 9000 with at least 256K of ram can be connected to the network. The network servers, however, must be hard disk units, and cannot be used simultainously as work stations.

Each network server manages its own disk and printer resources, providing for private and public files, for file and record locking on shared files, and for spooling of files for printing.

Work stations logged in to the network have access to up to 15 disk volumes and 4 printers at a time. Each disk volume and printer has its own logical name so that the user need not be concerned about the physical location of the resource. In fact, the work station operates just as if it were a stand-alone computer, except that it has more resources available.

Interaction with the network is through a few simple commands. These commands allow the user to log on or off the network, spool files for printing, check the status of the network servers, display the active users on the network, and protect, reserve, or release shared files. Two additional commands provide for formatting and configuring the network servers and for adding or deleting users.

ShareNet

ShareNet is a somewhat smaller and simpler network, but it is by no means less capable than the Victor Server Network. A ShareNet system consists of one network processor and up to 24 work stations and 5 printers. The network processor supports up to 120 megabytes of disk storage.

The basic topology of the ShareNet system is refered to as a 'star' network, because each work station is connected directly and independently to the network processor. This approach, while generally requiring more cabling, is significantly simpler and inherently more reliable than the linear contention networks. The drawback to this type of network is that there is only one network processor, and if it goes down, the entire network is down.

Work stations are attached to the ShareNet by installing a network interface card in the computer to be used as the work station, and running a cable to the network processor. ShareNet currently supports interfaces for the Victor 9000 under MS-DOS and CP/M-86, the IBM-PC under PC-DOS, CP/M-86 and the p-System, and numerous Z-80 based microcomputers running CP/M-80.

The ShareNet network processor is a Motorola 68000 based microcomputer, running a sophisticated operating system that administers the shared resources for the network. The network processor works in a vendor-independent manner that allows different operating systems, running on the various work stations, to concurrently share the same directories and data files. The network directories appear as logical units under each work station's operating system. Security for the network is handled on the file, directory, group, and user levels.

ShareNet supports procedures for file and record locking, transaction processing, and deadlock avoidance. Also supported are broadcast, and station to station messages. Options include electronic mail and a sophisticated data base management system running on the network processor.

For more information on ShareNet, contact:

Novell Data Systems, Inc.
1170 N. Industrial Park Drive
Orem, Utah 84057
(800) 453-1267

In describing the following patches, the offsets given (in hex) are offsets from the start of the program when it is loaded for execution. If you are patching the MS-DOS version of Wordstar using the DEBUG program, you can use the offsets as given for the patches. If you are patching the CP/M version of Wordstar using DDT86, you must add 180 hex to the offsets given. This is because .CMD files have a header area that tells CP/M-86 how to load them.

There is, however, an easier way of patching Wordstar. The INSTALL program that comes with Wordstar has a menu item (F) that allows the user to enter patches during configuration. Using this feature is somewhat safer than patching with a debugger, in that INSTALL prevents you from accidently modifying Wordstar outside of the "legal" patch area. The addresses given below can be used directly with the INSTALL patch facility.

For some of the patch items, the standard values are listed to enable you to be sure you are in the right place.

Screen Patches

Custom patching of some of the screen routines can be done through the use of the Wordstar INSTALL program. However, most patch areas are listed just in case you have a need to know where they are.

Screen Size

These two bytes set the screen dimensions.

HITE	248h	Screen height in lines
WID	249h	Screen width in columns

Highlighting

These two strings are sent to enable and disable highlighting. The first string is sent to enable highlighting. The second string is sent to disable highlighting. Highlighting is enabled for menu display and for block identification.

IVON	284h	7 bytes, first is length
IVOFF	28Bh	7 bytes, first is length

Initialization

The next two strings are used to set the screen to known conditions at the start and end of the Wordstar session. The first string is sent when Wordstar is invoked. The second string is sent when Wordstar is exited.

TRMINI	292h	9 bytes, first is length
TRMUNI	29Bh	9 bytes, first is length

Initialization Subroutines

For initialization sequences that cannot be performed by using the standard patches, Wordstar provides two "hooks" that can be used to call user supplied subroutines. To install custom subroutines, the locations specified should be patched with a jump instruction to your routine (which should end with a RET). The first subroutine is called before the TRMINI string is sent to the screen. The second subroutine is called after the TRMUNI string has been sent to the screen.

INISUB	2A4h	3 bytes in length
UNISUB	2A7h	3 bytes in length

Delay Times

The following six bytes control the delay times used by Wordstar to determine how long to wait before continuing with certain functions. These bytes can be set to any value from 1h (for minimum delay) to 7Fh (for maximum delay).

CRBLIV	2B5h	Cursor Blink Enable
DEL1	2CFh	Short Delay
DEL2	2D0h	Medium Short Delay
DEL3	2D1h	Medium Long Delay
DEL4	2D2h	Long Delay
DEL5	2D3h	Redisplay Delay

The Blink Enable byte, when set to FFh, tells Wordstar to blink the cursor whenever it is resting on an inverse video character. For no blink, it is set to 0h.

The Short Delay controls the ON portion of the cursor blink rate when the cursor is resting on a highlighted character and CRBLIV flag is FFh. It also controls the flashing rate between certain messages such as "REPLACE Y/N" and the cursor (it determines the time spent in the display area).

The Medium-Short Delay controls the alternate cycle of the rates controlled by the Short Delay (above). These are the cursor blink OFF time, and the time spent displaying a message in the status line.

The Medium-Long Delay controls the waiting time from hitting a prefix character (such as ^K or ^O), until the display of the related help menu. It also controls the delay at such non-edit mode messages such as "FILE NAME?" until the special character menu is displayed.

The Long Delay controls such things as the time the signon message, the "NEW FILE" message, and the "ABANDON" message remain on the screen.

The Redisplay Delay controls the waiting time until full screen redisplay during horizontal scrolling. If a line you are typing becomes longer than the width of the screen, that line is immediately moved over so that you can see what you are typing. Wordstar waits the length of time specified by the Redisplay Delay before redisplaying the entire screen in the shifted position.

Default Wordstar Program Disk

The contents of this byte determines which disk drive Wordstar looks at to find its overlay and message files (WSOVLY1 and WSMSGS), if they are not on the currently logged drive. This byte must be changed if you want to put the Wordstar programs on a disk drive other than A:, and then edit files on a different drive. This byte is set to 1h for A:, 2h for B:, 3h for C:, etc.

DEFDSK	2DCh	1h

After setting this byte, you must also "NOP" out an instruction in the body of Wordstar to make this modification operate. This patch must be done using DEBUG or DDT86 as the Wordstar modification

program will not let you change these addresses.

Address	Contents	Change To
1E04h	B1h	90h
1E05h	01h	90h

Horizontal Scroll Distance
This value is the number of columns the text is moved, left or right, each time you move off the screen.

SCRLSIZ 2DDh 14h

Initial Help Level - ^JH
This byte determines the help level in effect at program start-up. Setting it to 3 provides maximum help. A value of 2 eliminates the main editing menu and provides more display area for the file being edited. A value of 1 also causes suppression of the prefix key help menus. Setting it to 0 disables all help.

ITHELP 360h 3h

Maximum Help Message
If this byte is 0 then the message "FOR MAXIMUM HELP TYPE ^JH3" is displayed at the beginning of the first editing session, if the value of ITHELP is 1 or 2. A value of FFh disables the message.

NITHLF 361h FFh

Initial Insert Mode - ^V
If this byte is FFh, Wordstar starts out with insert mode on. if it is 0h, insert mode will be off.

ITITOG 362h FFh

Initial Directory Display - F
This value determines whether or not a directory listing of the logged drive is given when not in edit mode. A value of 0h inhibits the directory display, FFh enables it.

ITDSDR 363h FFh

Page Format - Dot Commands
The following group of values determine the format of the page break display during editing, and also the format for pagination during printing. The default values shown can be overridden, both in edit mode and during printing, by use of the dot commands. Most parameters in this table are defined twice. Once in terms of lines and once in multiples of 1/48 inches. The specification for the line height is repeated a number of times, and is listed separately.

For clarity, all values listed for the page parameters are in decimal.

Line Height (all values must be the same)

366h	DB	8	Line Ht. in 1/48"
36Ah	DB	8	
36Eh	DB	8	
372h	DB	8	
376h	DB	8	
37Ah	DB	8	

Paper Height (.PL)

367h	DB	66	Paper Ht. in Lines
368h	DW	8*66	Paper Ht. in 1/48"

Margin at Top of Page (.MT)

36Bh	DB	3	Top Margin in Lines
36Ch	DW	8*3	Top Margin in 1/48"

Heading Margin (.HM)

36Fh	DB	2	Hdg. Margin in Lines
370h	DW	8*2	Hdg. Margin in 1/48"

Bottom Margin (.MB)

373h	DB	8	Bot. Margin in Lines
374h	DW	8*8	Bot. Margin in 1/48"

Footing Margin (.FM)

377h	DB	2	Ft. Margin in Lines
378h	DW	8*2	Ft. Margin in 1/48"

Standard Character Width (.CW)

37Ch	DB	12	10 ch/inch in 1/120"

Alternate Character Width (.CW)

37Dh	DB	10	12 ch/inch in 1/120"

Page Offset (Printing only) (.PO)

37Eh	DB	8	8 chars from left

Screen Margins - ^OL and ^OR
These two values determine the initial values used for word wrap (^OW) and paragraph reform (^B) functions. The values are specified as the column number minus 1. The legal values for the left margin are 0, to the value of the right margin column minus 3. Legal values for the right margin are 2, to the value of the screen width minus 4.

INITLM	37Fh	0h	Left Margin
INITRM	380h	40h (64)	Right Margin

Subscript / Superscript Roll - .SR
This value is the distance above or below the line that superscripts of subscripts are printed. The value is in 1/48 inches. This applies only to printers that are capable of incremental movement, such as daisy wheel printers.

INITSR 381h 3h

Word Processing Flags
The following group of flags determine the initial states of the various word processing toggles. Except for the Line Spacing value, 0h means OFF and FFh means ON for each flag. All of the flags except the Page Break Enable flag can be changed during editing. If the Page Break Enable flag is set to 0h, then the Page Break Display flag is ignored and page breaks are not shown. The Line Spacing value is set to 1 for single spacing, 2 for double spacing, etc.

385h	FFh	Word Wrap - ^OW
386h	FFh	Justify - ^OJ
387h	FFh	Variable tabs - ^OV
388h	0h	Soft Hyphen - ^OS

389h	FFh	Hyphen Help - ^OH
38Ah	FFh	Display Ctrl Chars - ^OD
38Bh	FFh	Ruler Line Display - ^OT
38Ch	FFh	Page Break Enable
38Dh	FFh	Page Break Display - ^OP
38Eh	1h	Line Spacing - ^OS
38Fh	0h	Block Move - ^KN

Non-Document Mode
If a file name is specified at the time Wordstar is invoked, the non-editing menu is bypassed and edit mode is immediately entered. The value of the Non-Document Flag determines whether edit mode is entered in Document or Non-Document mode. A value of 0h enables Document mode, FFh enables Non-Document mode.

NONDOC 392h 0h

Decimal Tab Character
This byte defines the character that terminates the "decimal" alignment function (^O^I). This may be changed to allow column alignment about any desired character.

DECCHR 393h 2Eh ('.')

Dot Command Character
This byte defines the character that indicates the start of a dot command, if found in column 1. It may be changed to any desirable character.

DOTCHR 395h 2Eh ('.')

Non-Break Space
This character prints as a space, but Wordstar does not use it as a place to break a word for word wrapping, or as a place to add spacing for justification. May be changed to any desirable character.

BLNCHR 396h 0Fh (Ctrl-O)

Dot Command Enable
This flag enables or disables the dynamic interpretation of dot commands during editing. If disabled, all dot commands are ignored during editing, and the display (at the end of the line) of a question mark, for unrecognized commands, is inhibited. A value of FFh enables interpretation of dot commands, 0h disables interpretation.

DOTSON 397h FFh

Hyphenation Zone
This value determines how many spaces, short of being full, a line can be before hyphenation help occurs (if hyphenation help is turned on). This means that if the last full word that will fit completely on the line is less than this number of columns from the end of the line, the user will be prompted to hyphenate the following word. This number will dynamically vary, depending on the number of words on the line and the internal hyphenation rules used.

HZONE 39Ah 4h

Flag Characters
The following is the list of characters that indicate special conditions during editing. The first 9 characters display in the right-hand column of the display during editing. These indicate the status of each line of text. The last two characters occur in the text. SOFHYC is the character used to indicate the presence of a soft hyphen. Its high order bit is set to cause it to display in a highlighted mode. PAGFIL is the character used to fill the line at the end of a page (if page breaks are enabled). Any of these characters may be changed to whatever suits the user.

EOFCHR	3ADh	2Eh ('.')	End of File
BOFCHR	3AEh	3Ah (':')	Beginning of File
CONCHR	3AFh	2Bh ('+')	Line Continued
OVPCHR	3B0h	2Dh ('-')	Next Line Overprints
LFCHR	3B1h	4Ah ('J')	Line Feed w/o CR
PAGCHR	3B2h	50h ('P')	Last Line of Page
SOFTCR	3B3h	20h (' ')	Soft CR (Non Break)
HARDCR	3B4h	3Ch ('<')	Hard CR (Break)
PDTCHR	3B5h	4Dh ('M')	MergePrint Cmd Line
SOFHYC	3B6h	ADh ('-'+80h)	Soft Hyphen
PAGFIL	3B7h	2Dh ('-')	End of Page Marker

Place Marker Display Characters
The following characters are used to indicate the placement of markers in the text. The first two characters are used to delineate the start and end of a block of text for subsequent block commands (^K). The remaining 10 characters indicate the positions of user defined placemarks.

3BAh	42h ('B')	Beginning of Block
3BBh	4Bh ('K')	End of Block
3BFh	30h ('0')	User Placemark 0
3C0h	30h ('1')	User Placemark 1
3C1h	30h ('2')	User Placemark 2
3C2h	30h ('3')	User Placemark 3
3C3h	30h ('4')	User Placemark 4
3C4h	30h ('5')	User Placemark 5
3C5h	30h ('6')	User Placemark 6
3C6h	30h ('7')	User Placemark 7
3C7h	30h ('8')	User Placemark 8
3C8h	30h ('9')	User Placemark 9

Print Questions Defaults
The next group of flags control the default responses to the questions asked at the start of printing. Setting a flag to 0h causes a NO default, setting it to FFh causes a YES default. Setting the last flag to FFh causes suppression of "Use Form Feeds?" message altogether.

3CAh	0h	Disk File Output
3CBh	0h	Use Form Feeds
3CCh	0h	Suppress Page Formatting
3CDh	0h	Pause Between Pages
3D1h	0h	Suppress FF Message

Dot Command Defaults
These three flags determine the initial settings of certain dot commands. Setting them to FFh enables the command, setting them to 0h disables the command.

ITPOPN	3D3h	0h	Omit Page #'s - .OP/.PN
ITMIJ	3D4h	FFh	Micro Justification - .UJ
ITBIP	3D5h	FFh	Bidirectional Print - .BP

File Names
The last three of the four filenames given below are the names Wordstar uses to access its overlays. The first filename is the name of the

main Wordstar command file (which contains these tables). This name is used when the "R" command (Run external program) is given so that Wordstar can get back to itself. If you rename the Wordstar command file, or make a copy of it with a different name, this filename should be changed to match that of the file that contains it.

```
3E7h    DB   'WS      CMD'
3F3h    DB   'WSMSGS  OVR'
3FFh    DB   'WSOVLY1 OVR'
408h    DB   'MAILMRGEOVR'
```

Dispatch Tables

Wordstar uses what are called dispatch tables to determine what to do with each key entered from the keyboard. There are two of these tables. The first table is used to interpret commands entered while at the "NO FILE" menu. The second table is used to interpret commands entered during the editing of a file.

Each time a keyboard entry is received, Wordstar scans the appropriate table, looking for a matching entry. If at the "NO FILE" menu and no match is found, the entry is ignored. If no match is found while editing a file, the character is entered into the text. In either case, if a match is found, then the command function whose address is given in the table is called.

Command sequences may be changed to suit the user. However, it is recommended that, if possible, the old command sequence be left as is, and that a new command sequence be added to the table. This provides compatability for those who may not be aware of the changes.

All table entries have the following format:

```
DB   FIRSTCHAR,SECONDCHAR
DW   OFFSET(CMDADDRESS)
```

Commands may be comprised of either one or two characters. For a one character command, FIRSTCHAR is set the the desired command character, and SECONDCHAR is set to zero. For two character commands, SECONDCHAR is set to the second character of the command sequence.

There are a few restrictions on the choice of the command characters. FIRSTCHAR should be a control character in the range 01h to 3Fh. This is because only those characters not in the dispatch table can be entered into the text.

SECONDCHAR may be any character execept a lower case letter. While searching the table for a matching entry for the second command character, Wordstar treats letters and control characters as equivalent (i.e. I is the same as ^I). Because of this Wordstar also converts lower case letters to upper case before performing the search.

CMDADDRESS is the address of the routine that performs the command's function. This value should not be changed. If you wish to make a duplicate entry for a given command, then the CMDADDRESS for the new command sequence should be carefully copied from the CMDADDRESS of the existing table entry.

If CMDADDRESS is a value less than 256, then instead of being an address of a command routine, CMDADDRESS is taken to be an index, into WSMSGS.OVR, of a help message (see MSGINDEX, below). The message indexed by CMDADDRESS is displayed on the screen. The entries for these help messages are usually placed at the end of the dispatch table so as not to increase the search time for regular commands.

The editing mode dispatch table also contains entries for help messages for certain command prefixes. These messages are automatically displayed if the second command character is not typed within a certain time period (see Delay Times above). The entries for these messages are, and must be, at the beginning of the table. They have the following format:

```
DB   PREFIXCHAR,0FFh
DB   MSGINDEX
DB   0
```

PREFIXCHAR is the first character of a two character command sequence for which a help menu exists. MSGINDEX is an index into the pointer table at the beginning of the WSMSGS.OVR file. This pointer table contains offsets into the message file where the various messages are to be found.

The format of the WSMSGS.OVR file is as follows:

File Offset	Contents
000h - 07Fh	Wordstar Copyright Msg.
080h - 1FFh	Message Pointer Table
200h - END	Messages

There is space in the pointer table for 192 entries. The last message in the file is a dummy (or error) message, and any pointer table entry that points to this message can be used for the addition of a new message.

All messages are displayed starting at the same position on the screen (line 2, column 1). Control characters should not be used in the message text except that a new line (CR-LF) is indicated by a 0Eh character and the end of the message is indicated by a 00h character. Highlighting of parts of the message can be done by setting the most significant bit of the character to be highlighted (i.e. 'A'+80h causes the A to be highlighted).

Note:
The message offsets in the pointer table are 100h greater than the actual offset of the message in the file. If you use DEBUG to patch the file, then these offsets will be the actual location of the message in memory, because DEBUG loads files into its segment at an offset of 100h. However, the offsets of the parts of the file in memory will be 100h down from those listed in the file offsets table above. If you use DDT86 to patch the file, then the file offsets given above are the actual offsets in memory, but the offsets in the pointer table will be 100h more than the actual position of the message in memory.

The following tables list the addresses for the command entries in the dispatch tables for an

unmodified Wordstar file.

NO-FILE Dispatch Table

Addr	Cmd	Function
430h	D	Edit file in DOCUMENT mode
434h	N	Edit file in NON-DOCUMENT mode
438h	H	Set help level
43Ch	X	Exit to system
440h	P	Print file
444h	M	Run MailMerge
448h	Y	Delete file
44Ch	F	Display directory
450h	Z	Scroll directory up
454h	W	Scroll directory down
458h	L	Change logged drive
45Ch	R	Execute external program
460h	O	Copy file
454h	E	Rename file
468h	V	DEBUG command
46Ch	S	Run SpellStar
470h		Unused
474h		Unused
478h		Unused
47Ch		Unused

EDIT MODE Dispatch Table

Addr	Cmd	Function
481h		^Q Help display
485h		^K Help display
489h		^O Help display
48Dh		^P Help display
491h	^JH	Set help level
495h	^S	Cursor left 1 character
499h	^H	Cursor left 1 character
49Dh	^D	Cursor right 1 character
4A1h	^A	Cursor left 1 word
4A5h	^F	Cursor right 1 word
4A9h	^X	Cursor down 1 line
4ADh	^E	Cursor up 1 line
4B1h	^QS	Cursor to start of line
4B5h	^QD	Cursor to end of line
4B9h	^QX	Cursor to bottom of screen
4BDh	^QE	Cursor to top of screen
4C1h	^QB	Cursor to beginning of block
4C5h	^QK	Cursor to end of block
4C9h	^QP	Cursor to last command posn
4CDh	^QV	Cursor to last find/repl posn
4D1h	^Q0	Cursor to place marker 0
4D5h	^Q1	Cursor to place marker 1
4D9h	^Q2	Cursor to place marker 2
4DDh	^Q3	Cursor to place marker 3
4E1h	^Q4	Cursor to place marker 4
4E5h	^Q5	Cursor to place marker 5
4E9h	^Q6	Cursor to place marker 6
4EDh	^Q7	Cursor to place marker 7
4F1h	^Q8	Cursor to place marker 8
4F5h	^Q9	Cursor to place marker 9
4F9h	^QR	Cursor to beginning of file
4FDh	^QC	Cursor to end of file
501h	^QF	Find
505h	^QA	Replace
509h	^QL	SpellStar command
50Dh	^L	Repeat last find/replace
511h	^QW	Scroll continuous down
515h	^QZ	Scroll continuous up
519h	^Z	Scroll up 1 line
51Dh	^W	Scroll down 1 line
521h	^R	Scroll up 1 screen
525h	^C	Scroll down 1 screen
529h	Del	Delete 1 character left
52Dh	^G	Delete 1 character right
531h	^T	Delete 1 word right
535h	^Y	Delete line
539h	^QDel	Delete to beginning of line
53Dh	^Q^Y	Delete to beginning of line
541h	^QY	Delete to end of line
545h	^T	Delete 1 word right
549h	^V	Compliment insert mode
54Dh	^B	Reform paragraph
551h	^QQ	Repeat next command
555h	^N	Insert CR-LF
559h	^I	Tab
55Dh	^M	Carriage return
561h	^P	Literal character entry
565h	^KH	Hide block marker
569h	^KB	Mark beginning of block
56Dh	^KK	Mark end of block
571h	^K0	Set place marker 0
575h	^K1	Set place marker 1
579h	^K2	Set place marker 2
57Dh	^K3	Set place marker 3
581h	^K4	Set place marker 4
585h	^K5	Set place marker 5
589h	^K6	Set place marker 6
58Dh	^K7	Set place marker 7
591h	^K8	Set place marker 8
595h	^K9	Set place marker 9
599h	^KV	Move block
59Dh	^KC	Copy block
5A1h	^KY	Delete block
5A5h	^KN	Column / block mode toggle
5A9h	^KZ	DEBUG command '
5ADh	^U	Stop command
5B1h	^KX	End edit and exit
5B5h	^KD	End edit
5B9h	^KS	Save file and restart
5BDh	^KQ	Abandon edit
5C1h	^KR	Insert file
5C5h	^KW	Write block
5C9h	^KJ	Delete file
5CDh	^KF	File directory on/off
5D1h	^KP	Print file
5D5h	^KL	Select drive
5D9h	^KO	Copy file
5DDh	^KE	Rename file
5E1h	^OL	Set left margin
5E5h	^OR	Set right margin
5E9h	^OI	Set tab stop
5EDh	^ON	Clear tab stop(s)
5F1h	^OF	Set margins from line
5F5h	^OW	Word wrap toggle
5F9h	^OJ	Justify toggle
5FDh	^OV	Variable tabs toggle
601h	^OD	Dot command display toggle
605h	^OT	Ruler line display toggle
609h	^OP	Page break display toggle
60Dh	^OE	Soft hyphen toggle
611h	^OH	Hyphen help toggle
615h	^OG	Set temp left margin
619h	^OX	Release margins
61Dh	^OC	Center line
621h	^OS	Set line spacing
625h	^JD	Dot command help
629h	^JS	Status line help
62Dh	^JF	Flag character help
631h	^JP	Place marker help
635h	^JB	Paragraph reform help
639h	^JM	Margins help
63Dh	^JI	Command help
641h	^JV	Move text help
645h	^JR	Ruler line help
649h		Unused

64Dh	Unused
651h	Unused
655h	Unused
659h	Unused
65Dh	Unused
661h	Unused
665h	Unused
669h	Unused

Printer Control

The commands sent to your printer to control its various functions are defined in the following areas. For this discussion, any standard values given for these areas are those used when Wordstar is configured for a Diablo 630 printer.

POSMTH
The most important flag in this group is POSMTH. It stands for Printer Overstrike Method. This defines the basic capabilities of your printer mechanism and how (if possible) certain printing actions are performed. The usage of many of the printer commands is determined by POSMTH.

For a printer that can only overprint a line by performing a carriage return without a line-feed, POSMTH should be set to FFh.

If the printer has the capability of backspacing and overprinting, then POSMTH should be set to 00h.

If the printer is capable of incremental movement, and can print a character without moving the carriage (or equivalent) then POSMTH should be set to 01h. This is the setting for most "daisy wheel" type printers.

POSMTH	746h	01h

Boldface Intensity - ^PB
This value is the number of times a character is printed, offset by a small distance, to give a bold appearance. This value may range from 2, on up. However, it is stated that for a daisy wheel printer, only the value 2 should be used.

BLDSTR	747h	02h

Double Strike - ^PD
This is the number of times a character is printed in the same position in order to give different style of bold appearance. It may be set to any desired value.

DBLSTR	748h	02h

Non-Daisy Wheel Control Strings

The following control string definitions are used only when the value of POSMTH is FFh or 00h. If POSMTH is 01h, then these strings are ignored.

Next Line
This string is sent to the printer to advance it to the next line and return the carriage to the left-most position.

PSCRLF	74Ch	11 bytes, first is length

Overprint Line
This string is sent to the printer to return it to the left-most position in order to overprint the current line.

PSCR	757h	7 bytes, first is length

Half-Line Feed
This string is sent to the printer to return it to the left-most position and advance one-half line downward. If POSMTH is FFh or 00h, and ROLUP (see below) is non-zero, this string is used for printing superscripts and subscripts at half-line intervals.

PSHALF	75Eh	7 bytes, first is length

Backspace
This string is sent to the printer to move its carriage left one character position. It is used for backspacing, underlining, and double strike, if POSMTH is 00h.

PBACKS	765h	6 bytes, first is length

Alternate Widths - ^PA ^PN
The first string is sent to the printer to enable printing at a different width. The second string is sent to restore printing to the standard width. This feature should only be installed if your printer is capable of supporting it.

PALT	76Bh	5 bytes, first is length
PSTD	770h	5 bytes, first is length

Roll Up / Down
These two strings are sent to the printer to roll the carriage up or down without moving it left or right. They are used for printing superscripts and subscripts. The roll up distance must match the roll down distance. This feature should only be installed if your printer is capable of supporting it.

ROLUP	775h	5 bytes, first is length
ROLDOW	77Ah	5 bytes, first is length

Combination Control Strings

The following strings are used to control all types of printers.

User Print Patches - ^PQ ^PW ^PE ^PR
These four strings define the functions performed during printing, when the corresponding control sequence is encountered. These may be set to implement any feature desired. Lines containing these commands are always printed in the forward direction.

USR1	77Fh	5 bytes, first is length
USR2	784h	5 bytes, first is length
USR3	789h	5 bytes, first is length
USR4	78Eh	5 bytes, first is length

Ribbon Color - ^PY
The next two strings are used to change the color of the ribbon, on printers that support that feature. The first string sets the printer to the alternate color. The second string restores the printer to the original color.

RIBBON	793h	5 bytes, first is length
RIBOFF	798h	5 bytes, first is length

Initialize Printer
These two strings are used to restore the printer to known conditions. The first string is sent to the printer at the start of printing, the second is sent at the conclusion of printing. These strings may be changed to suit your particular printer, or additional items may be patched in after the standard ones.

PSINIT	79Dh	17 bytes, first is length
PSFINI	7AEh	17 bytes, first is length

Additional Patches

Strikeout Character - ^PX
This character is used for the "strikeout" print feature.

SOCHR	7C1h	2Dh ('-')

Underscore Character - ^PS
This character is used for the "underscore" print feature.

ULCHR	7C2h	5Fh ('_')

Initialization Subroutines
For printers that require initialization sequences than cannot be accomidated through use of the standard patches, Wordstar provides two "hooks" to user supplied routines. If used, the contents of the specified locations should be replaced with jump instructions to the custom routines. The first routine is called before the PSINIT string is sent to the printer. The second routine is called after the PSFINI string has been sent.

PRINIT	7C3h	3 bytes in length
PRFINI	7C6h	3 bytes in length

Printer Drivers

Wordstar makes provision for the selection of one of five printer drivers to control I/O to your printer. While Wordstar predefines the usage of each driver, except for the OEM driver (CSWTCH=3), there is really no logical difference between them as far as calling sequence and function performed. The OEM driver is a special driver and is not discussed here.

Which of the five drivers is to be used is controlled by the value in CSWTCH. These values and their corresponding drivers are listed below.

CSWTCH	7C9h	0h

CSWTCH	Driver
0	Standard list device
1	INSTALL patchable port driver
2	User defined driver
3	OEM daisy wheel printer driver
4	Alternate console printer driver

There are three entry points for each driver. These are as follows:

Busy Status
The busy status entry tells Wordstar if the printer is ready to accept a character for printing. If it is ready, the routine returns with the carry flag set to zero (CY=0). If the printer is not ready, the routine returns with the carry flag set to one (CY=1). If there is no way to determine if the printer is busy other than by trying to send a character, then the status routine should return with the carry flag set to zero (CY=0), just as if the printer were ready.

There is a flag that Wordstar examines to determine whether the printer driver has a meaningful busy status routine, or just a dummy one. This flag is HAVBSY. Wordstar uses this flag to improve its performance during concurrent editing and printing. If this flag is non-zero, it indicates that the driver busy routine is not a dummy routine.

HAVBSY	7CAh	0h

Send Character
The send character routine outputs the character in the AL register to the printer. If the printer is busy, then this routine must wait until the character can be sent.

Receive Character
The receive character routine checks for a character from the printer. If there is one, it gets it and returns it in the AL register with the carry flag set to zero (CY=0). If no character is available, it returns with the carry flag set to one (CY=1). This routine is used only when a handshaking protocol that requires receiving data from the printer is used (i.e. ETX/ACK or XON/XOFF).

The entry points for each of the drivers is given below, along with the amount of space available for the routine. If more room is needed for a routine, the space allocated to the unused routines may be used, or the additional code can be placed in the other extra patch areas.

CSWTCH = 0h			
LIBSY	7CCh	15 bytes	Printer status
LISEND	7DDh	16 bytes	Send character
LISINP	7EDh	3 bytes	Receive char

CSWTCH = 1h			
POBSY	7F0h	13 bytes	Printer status
POSEND	7FDh	4 bytes	Send character
POINP	801h	15 bytes	Receive char

CSWTCH = 2h			
PUBSY	811h	3 bytes	Printer status
PUSEND	814h	3 bytes	Send character
PUINP	817h	3 bytes	Receive char

CSWTCH = 4h			
ACBSY	81Ah	19 bytes	Printer status
ACSEND	82Dh	13 bytes	Send character
ACINP	83Ah	63 bytes	Receive char

Handshake Protocol
Wordstar supports two software handshaking protocols. These are the ETX/ACK protocol and the XON/XOFF protocol. If it is desired that one of these protocols be used, then the value at PROTCL

must be set to indicate which protocol to use. To use the ETX/ACK protocol, PROTCL must be set to 1. To use the XON/XOFF protocol, PROTCL must be set to 2. Any other protocol must be implemented by the user, and PROTCL must be set to 0.

PROTCL	879h	0h

If the ETX/ACK protocol is selected, then Wordstar needs to know how many characters it can send to the printer before waiting for a handshake. This value is one-half the printer's buffer size, and must be stored at EAKBSZ.

EAKBSZ	87Ah	7Fh

Daisy Printer Control Strings

The following control strings and definitions apply only when POSMTH is 01h. These are mainly escape sequences for controlling daisy wheel printer movements. However, these may be used for any printer capable of incremental movement that can support these commands.

Vertical Motion Index
This string is the lead-in string used to set the vertical spacing of the printer, in 1/48" increments. The character that sets the spacing is computed by Wordstar and sent immediately after this lead-in string.

The two bytes following the string define the minimum value and the range of values for the character sent after the lead-in. The minimum value is the value to be sent to set the spacing index to 0. The range value is the maximum spacing value (in 1/48") that can be set, plus one.

DVMILE	87Ch	5 bytes, first is length
DVMMIN	881h	Minimum value
DVMRNG	882h	Range

Horizontal Motion Index
This string is the lead-in string used to set the horizontal spacing of the printer in 1/120" increments. The character that sets the spacing is computed by Wordstar and sent immediately after this lead-in string.

The two bytes following the lead-in string define the minimum value and the range of values for the character sent after the lead-in. The minimum value is the value to be sent to set the spacing index to 0. The range value is the maximum spacing value (in 1/120") that can be set, plus one.

DHMILE	87Ch	5 bytes, first is length
DHMIN	881h	Minimum value
DHRNG	882h	Range

Forward Print
This string is sent to the printer to set the printer to forward print mode.

DFWD	88Fh	5 bytes, first is length

Reverse Print
This string is sent to the printer to put it in reverse printing mode.

DBAK	894h	5 bytes, first is length

Forward Space
This string is sent to the printer to cause it to space forward one unit of horizontal motion, as set with the horizontal motion index (see above).

DSP	899h	5 bytes, first is length

Backspace
This string is sent to the printer to cause it to space backward on unit of horizontal motion, as set with the horizontal motion index (see above).

DBS	89Eh	5 bytes, first is length

Line Feed
This string is sent to the printer to cause it to space downward on unit of vertical motion, as set with the vertical motion index (see above).

DLF	8A3h	5 bytes, first is length

Reverse Line Feed
This string is sent to the printer to cause it to space upward on unit of vertical motion, as set with the vertical motion index (see above).

DRLF	8A8h	5 bytes, first is length

Phantom Space
This string is sent to the printer to cause it to print the character on the print wheel associated with the space character. This is usually the cents character.

DPHSPC	8ADh	5 bytes, first is length

Phantom Rubout
This string is sent to the printer to cause it to print the character on the print wheel associated with the rubout character. This is usually the logical not character.

DPHRUB	8B1h	5 bytes, first is length

Proportional Spacing
The following flag determines whether Wordstar attempts proportional spacing while printing. If this flag is 0h, then proportional spacing is allowed. If it is FFh, then proportional spacing is suppressed.

DNPROS	8Bfh	0h

Justification Algorithm
This flag determines the algorithm used by Wordstar for microjustification. If the flag is FFh then more emphasis is placed on expanding spaces between words than on spreading out characters. To emphasize the opposite effect, the flag is set to 0h.

DMJWB	8C0h	0h

Extra Patch Areas

Wordstar provides an area that may be used for patch routines that do not fit into the standard space alloted for them. This extra space is from 2E0h to 35Fh, inclusive.

J. David Sekercan

Channel 9000

THE INDEPENDENT NEWSLETTER OF THE VICTOR 9000/SIRIUS 1 COMPUTER

VOL. 1 NO. 4 July 1983

From The Editor:
Channel 9000 was fortunate to finally be mentioned in the Clubs and Newsletters column of the July, 1983 issue of BYTE Magazine. We are getting many inquiries from end users, hungry for information on the Victor 9000. Hopefully, Channel 9000 and the other Victor related publications described in this issue, can fill the information gap which still exists.

I am disturbed by some of the stories that have been related to me by end users, regarding their attempts to find out about software availablilty, more detailed information about certain features of the 9000, and just trying to get some general help. While there is probably not a great deal that any one person can do to improve this situation (with the possible exception of publishing a newsletter), I have an idea I would like to try.

I would like to hear about the problems that both dealers and users are having in obtaining help and information from Victor. If we can compile enough information to pinpoint the specific areas of need, I believe Victor will be responsive. A signed letter, outlining your problem in as much detail as possible, should be sent to Channel 9000, at our standard mailing address.

Local User Groups
Another way for users (and dealers) to get help, is to help each other. One way to do this is through the formation of local user groups. I have been "volunteered" to assist in the formation of the Los Angeles user group. Since I am becomming involved in this activity, I would also like to hear from other user groups throughout the country, or from anyone interested in starting or joining one.

Victor has stated that they are very supportive of the idea of user groups. I am sure that we get their assistance in coordinating the formation of these groups.

National User Group
I would also like to propose the formation of a National Victor 9000 User Group. This idea has been discussed informally in various circles for some time. I believe the time is right to start serious discussion..

There are many services that a national group can offer that may be impractical for local groups. One service is the cataloguing, and distribution, at low cost, of public domain software tailored for the Victor 9000. Another posibility is the establishment of a Remote Bullitin Board System, so that users can dial up and download software and information, and also leave messages regarding questions or problems.

There are many other possibilities, and we welcome your comments and suggestions.

L. A. District Bulletin Board
The Victor Los Angeles District Headquarters is pioneering an information bulletin board system (IBBS) that should be operational shortly. The system is designed to provide bulletins, outgoing messages, and information about hardware and software products, to four classes of users. These classes are dealers, end users, software vendors, and Victor employees. Each user class receives only information relating to that class. However, general announcements to all classes are possible.

Each District Branch Office will maintain a separate IBBS to eliminate the necessity of long distance phone charges. While use of the system is not restricted, each user is required to have his own password for access, along with a special communication program. This is being done to ensure reasonable responsiveness from the IBBS.

More information about the IBBS can be obtained by contacting:

Claude Coleman
Victor Los Angeles District Headquarters
4685 McArthur Court
Newport Beach, CA 92660

In Closing
I believe that the ideas presented here can be used to significantly narrow the Victor information gap. However, these ideas can do no good unless put into action. I hope that this discussion will lead to that action.

James M. Lesher

Utility Disks Offered

Channel 9000 has a number of utility programs that are available on diskette. These are offered at nominal prices and are intended for non-commercial use only. Anyone interested in distributing these programs on a commercial basis should contact Channel 9000 concerning distribution arrangements.

California residents please add 6.5% sales tax.

Keyboard Utility Disk - $15.00
Three programs, including source code, allow you to load a keyboard table into the BIOS, change the keyboard table on a system diskette, or save a keyboard table from the BIOS to a disk file. Also included is a BASIC program that produces an assembly language text file dump of a keyboard table.

MS-DOS to CP/M Utilities - $30.00
RDMSDOS.CMD allows you to transfer files to CP/M diskettes from MS-DOS diskettes (operates similar to RDCPM.COM). COM2CMD.COM converts COM format files to CMD format allowing you to develop CP/M programs using the MS-DOS assembler and linker (which are much better than what is available for CP/M). Converted programs can be run under CP/M or under the CP/M emulator.

WordStar Format Selection - $35.00
This program, when patched into standard WordStar, allows the user to select from various initialization formats. Any patchable function can be set by this program. The format selections are defined in a standard text file which can be easily modified using any editor. Source code and a sample format file are included. Supplied for both CP/M and MS-DOS.

ETX/ACK & X-ON/X-OFF Printer Drivers - $25.00
Two printer drivers which support the ETX/ACK and the X-ON/X-OFF protocols on either of the Victor 9000's serial ports. Once installed, the driver can be enabled or disabled through simple commands (similar to the way 132 column mode works). Source code is included. Supplied for both CP/M and MS-DOS.

More Victor Related Publications

The popularity of the Victor 9000 is finally being shown, as happened in the case of the IBM/PC, by the emergence of more publications dealing primarily with the specific machine. The following three publications are aimed at various segments of the Victor 9000 marketplace. Each is briefly described, with references to where further information can be obtained.

Sirius Computing
The largest publication exclusivly for the Victor 9000 / Sirius 1 computer (90+ pages per issue) was started the same month as Channel 9000 (January). It is published in England and is distributed mainly by ACT Limited, the English distributor of the Sirius 1. The magazine contains ads, reviews, tutorials, interviews, and a question and answer column. It is published bimonthly. Information may be obtained by writing or calling:

Sirius Computing Sales
Paradox Group Limited
39-41 North Road
London, England N7 9PD
(01) 607-9489

Resource 9000
Primarily intended for the Victor dealer, this magazine was started by a group of attorneys in the Boston area, who, after buying some 9000's, realized the need for more information concerning the product. The magazine contains ads, reviews, and tutorials. The subscription rate is $50.00 per year. Its first issue came out in May, 1983. It is to be published monthly. For more information, contact:

Resource 9000
240 Commercial Street
Boston, MA 02109
(617) 367-6959

Hi-Tek Routines Club Newsletter
Originally billed as the "Victor 9000 Users Club", this group's aim is "to better understand and utilize advanced micro-processors, such as the Victor 9000 and IBM Personal Computer." This newsletter format publication contains a question and answer section and listings of technically oriented programs. The newsletter is published monthly for an annual subscription rate of $35.00. Further information may be obtained by writing:

Hi-Tek Routines Club
310 S.W. 2nd Street
Ft. Lauderdale, FL 33312

CHANNEL 9000

9742 Marcus Lane, Tujunga, CA 91042
(213) 352-6443
James M. Lesher, Editor
Subscription rate: $30.00 (US) for 6 issues
Overseas Air Mail Rate: $37.50 (US) for 6 issues
Published Bimonthly

Printing TAB With BASIC

BASIC has the nasty habit of converting the ASCII TAB character (CHR$(9)) into the appropriate number of spaces. This occurs when using PRINT with the BASIC interpreter, and LPRINT with both the BASIC interpreter and compiler. Apparently, this is an artifact from the days of Teletypes, when many terminals did not support the TAB function. However, with todays printers and terminals, this "feature" can cause much frustration. The Diablo 630, amoung others, requires the TAB character be sent "as is" to perform many of its special functions.

Fortunately, at least when using LPRINT, there is a simple solution to this problem. Because the ASCII character set defines only 128 codes, most printers (especially when using RS232 communication) strip the high-order bit from any character received. Since BASIC allows character codes from CHR$(0) to CHR$(255), it is possible to trick BASIC into ignoring the TAB character by setting the high-order bit before using it in an LPRINT command. This is done as follows:

```
10 LPRINT CHR$(9+128)
```

Unfortunately, the above method does not work if you want to send CHR$(9) to the display, or to a device on the parallel port that does not strip the high-order bit, or if you want to send full 8-bit data using the RS232 ports. If you desire to do this type of output, the only solution is to patch BASIC to prevent it from modifying the TAB character.

To patch the MS-DOS BASIC interpreter (version 5.27), follow the steps below.

```
A>DEBUG MSBASIC.COM<cr>
>E2D30<cr>
XXXX:2D30 75. EB<cr>
>E2DC0<cr>
XXXX:2DC0 75. EB<cr>
>W<cr>
```

To patch the CP/M-86 BASIC interpreter, use the following steps:

```
A>DDT86<cr>
-RBASIC86.CMD<cr>
  START      END
XXXX:0000 XXXX:717F
-S2A87<cr>
XXXX:2A87 75 EB<cr>
XXXX:2A88 15 .<cr>
-S2B17<cr>
XXXX:2B17 75 EB<cr>
XXXX:2B18 14 .<cr>
-WBASIC86<cr>
```

To patch the MS-DOS BASIC runtime file BASRUN.EXE (version 5.32 or 5.33) it is first necessary to rename the file to give it a different extention. This is because DEBUG loads EXE files for execution and they cannot be properly resaved once they have been loaded this way. Changing the extention prevents DEBUG from changing the file when loading.

To determine which version of BASRUN.EXE you have, check the size and date of the file with the DIR command. Version 5.32 has a size of 20608 and a date of 7/21/82. Version 5.33 has a size of 21120 and a date of 10/18/82.

Use the following steps to modify BASRUN.EXE 5.32:

```
A>REN BASRUN.EXE BASRUN.532<cr>
A>DEBUG BASRUN.532<cr>
>E2736<cr>
XXXX:2736 75. EB<cr>
>W<cr>
Writing 5080 bytes.
>Q<cr>
A>REN BASRUN.532 BASRUN.EXE<cr>
```

Use the following steps to modify BASRUN.EXE 5.33:

```
A>REN BASRUN.EXE BASRUN.533<cr>
A>DEBUG BASRUN.533<cr>
>E27B0<cr>
XXXX:27B0 75. EB<cr>
>W<cr>
Writing 5280 bytes.
>Q<cr>
A>REN BASRUN.533 BASRUN.EXE<cr>
```

Communications

Today, with so many people and businesses having computers, an important capability is inter-computer communication. There is always a need to transfer data from one computer to another. Usually this is accomplished by just taking a diskette out the first computer and inserting it into the second. This works ok if the two computers are of the same type, or use the same disk format. Unfortunately, the Victor 9000 has a disk format that is unique in the industry. That means that the only way to exchange data between the 9000 and anything else is over wire, at least until someone designs an add-on disk unit that is compatable with other formats.

While I am sure there are many other communication programs available, I am covering here only the two most popular third-party programs, along with the Victor offerings. If anyone would care to submit a review of other programs, or more detailed discussions of the ones presented here, we would be glad to publish them.

MOVE-IT
Move-It is the simplest, and easiest to use, of the general purpose programs covered here. Although it has a "dumb terminal" mode, Move-It is primarily designed to transfer files between two computers. Once Move-It is running on the two computers that wish to communicate, all commands need be entered at only of them (either end).

Move-It allows you to display a local or remote file directory for any drive on the two systems. File transfers can be done from and to any drive, and Move-It supports "wild-card" filenames. This makes it very easy to transfer an entire diskette full of information.

Move-It transfers both text and object files. All file transfers are done in a blocked manner, with each block being checked for validity before the next one is sent. Any failure of the transmission is reported to the user.

Move-It also has a message command that allows you to send a one-line message to the remote computer. This feature is handy for telling the remote operator to change disks, or that the session is over. When a message is sent, the bell is sounded at the receiving end to alert the operator.

Move-It is available for CP/M-80 computers, the IBM-PC, and the Victor 9000. For more information, contact:

Wolf Software Systems
23842 Archwood Street
Canoga Park, CA 91307
(213) 703-8112

Crosstalk-XVI
The Crosstalk program is the most widely used microcomputer communications package on the market. Crosstalk-XVI is an enhanced version of the original 8-bit version of Crosstalk, completely rewritten to make full use of the capabilities of 16-bit machines. Crosstalk-XVI is currently undergoing testing for release as a Victor Level 2 software product.

In addition to supporting text and object file transfers between computers (with error checking), Crosstalk allows the Victor 9000 to act as a very smart remote terminal. Configurations for different systems (baud rates, parity, etc) can be stored in separate "command" files so that changing communication parameters is done simply by telling Crosstalk to load a new command file.

Log-on response procedures can be stored in "script" files so that signing on to a computer utility such as CompuServe can be done at the "push of a button." The script file system uses its own programming language to make it a very powerful communications controller.

Crosstalk also allows you to store strings of characters that will automatically be sent when the function keys are hit. If you have a "smart" modem, Crosstalk can even be programmed to dial the number for you.

Data received from on-line sources, in addition to being displayed on the 9000's screen, can be selectively captured to memory, or saved on disk, for editing, reviewing, or printing, at a later time. You can also review data captured to memory, on the 9000's screen, while remaining on-line. It is also possible to selectively send incomming data directly to your printer.

Crosstalk is available for all common CP/M-80 systems, for CP/M-86, the IBM-PC, and, of course, the Victor 9000 in either MS-DOS or CP/M-86 versions.

For more information, contact:

Microstuf
1845 The Exchange
Atlanta, GA 30339
(404) 952-0267

Victor Technologies
The following Victor software products are either released for sale, or should be released soon. These products cover a wide range of capabilities. For further information, contact your Victor dealer or local Victor branch office.

PC COMM
PC COMM provides the means for sending text and object files from the IBM-PC to the Victor 9000, only. It consists of two programs. The first runs on the IBM-PC and sends files to the Victor. The second program runs on the Victor and receives the files sent from the IBM-PC. Communication is over a Centronics type cable connected from the Victor parallel printer port to the IBM-PC Parallel Printer Adapter. The PC COMM package includes the cable.

PC COMM is intended primarily for software houses and dealers, providing a low cost method of transporting programs from the IBM-PC to the Victor 9000.

ASYNC
Victors ASYNC program is an inter-computer communication program whose overall capabilities fall between those of MOVE-IT and CROSSTALK, mentioned above. ASYNC provides for asynchronous terminal emulation, data transmission and capture in terminal mode, file transfering with complete error checking in datalink mode, and "smart" modem control.

ASYNC is available only for CP/M-86, although it will run under MS-DOS using the CP/M-86 emulator.

BISYNC 86 - 3270 & 3780
Victor is offering two emulation packages that support the IBM, and IBM compatable, bisynchronous communication protocols.

The 3270 program supports emulation of the following IBM equipment: 3271 controller and 3277 display, 3275 display, 3284 printer, 3286 printer. The 3270 program can be used with the Victor 9000 to perform operator controlled, physical emulation. It is also possible to enhance the 3270 interface through user-defined program additions. These user defined routines can provide for extensive pre and post processing of the communicated data, or any other processing that may be necessary.

Victor's 3780 package is a Remote Batch Entry Station emulation program. It is capable of emulating IBM 2770, 2780, 3741, and 3780 devices. The emulator provides for the transmission of data from the Victor keyboard, or from one or more data files. Received data can go to the screen, the printer, or to one or more print or punch files.

Both the 3270 and 3780 emulators use IBM's bisynchronous communications protocol with baud rates from 1200 to 9600. Both products are available only for CP/M-86, but will run under MS-DOS using the CP/M-86 emulator.

NOTE:
We recently learned that the BETA release of the 3270 package has been temporarilly withdrawn. It will be re-released in the future with a new vendor version.

```
10 DEFINT A-Z                    ' only integers required
20 PRINT "Envelope Addresser - Version 1.0"
30 PRINT
40 BLKMARK=18                    ' use CTRL-R as block marker
50 OFFSET=45                     ' column offset for envelope address
60 ' get letter file name and open it
70 PRINT "Enter Letter Filename: ";
80 LINE INPUT "";L$
90 OPEN "R", #1, L$
100 ' set up to read file one character at a time
110 FIELD #1,128 AS F$           ' read as random file, 128 bytes at a time
120 J=1                          ' initialize character counter
130 R=1                          ' initialize file record counter
140 GET #1,R                     ' get the first record
150 ' skip until first occurance of block marker
160 GOSUB 1000                   ' get one character from file
170 IF C=26 THEN 500             ' exit if end of file encountered
180 IF C<>BLKMARK THEN 160       ' loop back if not block marker
190 ' print all lines (offset by OFFSET) until next block marker
200 LPRINT TAB(OFFSET);          ' tab to address position
210 GOSUB 1000                   ' get one character from file
220 IF C=26 THEN 500             ' exit if end of file
230 IF C=BLKMARK THEN 500        ' exit if end of address block
240 IF C<>13 THEN Z70            ' skip if not CR
250 LPRINT                       ' print CR-LF
260 LPRINT TAB(OFFSET);          ' tab to address position
270 IF C<32 THEN 210             ' skip all other control chars
280 LPRINT C$;                   ' print regular character
290 GOTO 210
500 ' end of address block
510 LPRINT                       ' restore carriage
520 CLOSE                        ' close input file
530 SYSTEM                       ' return to system
1000 ' subroutine to return one char in file in C and C$
1010 IF J>128 THEN 1060          ' skip if not at end of record
1020 C$=MID$(F$,J,1)             ' get next char from record
1030 C=ASC(C$) AND &H7F          ' strip sign bit (Wordstar flag)
1040 J=J+1                       ' increment char counter
1050 RETURN
1060 R=R+1                       ' increment record counter
1070 GET #1,R                    ' read next record from file
1080 J=1                         ' reset character counter
1090 GOTO 1000                   ' get one char from record
```

Listing 1. Envelope Addressing Program.

ETX / ACK Protocol

I have often been asked why the ETX/ACK or the X-ON/X-OFF protocols do not work with the Victor 9000. This "problem" is not one associated only with the 9000. WordStar states, during the installation of one of the above protocols, that in order for the protocol to work, the USER must install a routine in WordStar's patch area to receive characters from the printer. If this receive routine is not installed, then, for the ETX/ACK protocol, a few characters will be printed and then WordStar will "hang", and for the X-ON/X-OFF protocol, characters will be printed (and overprinted) just as if no protocol had been specified. While it is fairly easy to write a receive routine for the port the printer is on, this does not solve the general problem of communicating with a printer that does not use hardware handshaking.

The printer I use with my 9000 is a Diablo 1620, which does not use hardware handshaking. Modifying WordStar to support the ETX/ACK protocol that the 1620 uses solves the handshaking problem for WordStar, but does nothing for all of the other programs that output to the printer. A better solution to the problem is to install a custom device driver in the system that supports your printers particular protocol. This method solves the handshaking problem for all programs, not just WordStar.

Custom device drivers were discussed, with examples included, in Channel 9000, Volume 1, Number 2.

Envelope Addressing

I use WordStar to compose most of the letters that I write. I find it inconvenient to have to duplicate the addressee's address (which is at the beginning of the letter) for the purpose of typing the envelope. To make the addressing of envelopes easier, I have written a BASIC program which finds the address in the letter and prints it in the proper position on the envelope.

To indicate the position of the address in the letter, I precede and follow it with the WordStar ^PR user patch command, which, for my setup, performs no function. Entering this command into the text places a Ctrl-R (12 hex, 18 decimal) in the text file. The BASIC program skips all lines up to the first Ctrl-R, then prints all lines up to the second Ctrl-R.

The BASIC program is given in Listing 1. It may easily be modified to print a return address, for use with envelopes that are not pre-printed. This program can be compiled or converted to assembly language to make execution quicker and easier.

MS-DOS / CP/M-86 Conversion

In researching WordStar in preparation for writing the May, 1983 issue, I discovered that WordStar was designed to run under both CP/M-86 and MS-DOS. There is a one-byte flag in WordStar that indicates which system the program is running under. WordStar uses this flag to decide whether to use CP/M system calls or MS-DOS system calls. All that is required to convert WordStar between CP/M and MS-DOS is changing the system flag, adding or removing the CP/M CMD file header, and changing the system calls in the printer patch area. The following is a description how to convert WordStar from one system to the other.

System Type Flag
This flag indicates to WordStar which system it is running under. It is set to 00h for CP/M-86 and to FFh for MS-DOS.

MSDOS	2D6h	System determination

CP/M Header
In order to provide compatability between CP/M and MS-DOS, WordStar is configured as the 8080 Memory Model under CP/M, and as a COM file under MS-DOS. Both these configurations assume only one segment (the CODE segment) and also assume that the program starts at offset 100h.

CP/M CMD files have a header that tells CP/M how to load the file, how much memory to allocate, and how to initialize the segment registers before executing the program. This header is 128 bytes in length. MS-DOS COM files have no header and are always loaded in a single segment starting at 100h.

The CP/M loader requires that space be allocated in the CMD file for the memory from 0h to FFh, whereas the MS-DOS COM file loader assumes that the data in the COM file starts at 100h. This means that to convert WordStar from CP/M to MS-DOS, 384 bytes must be stripped from the front of the CMD file to make it a COM file. This can be done using either DDT86 under CP/M or by using DEBUG under MS-DOS.

The following procedure converts CP/M WordStar to MS-DOS using DDT86 (the underlined parts are typed by the user):

```
A>DDT86<cr>
-RWS.CMD<cr>
 START      END
XXXX:0000 XXXX:52FF
-S356<cr>
XXXX:0356 00 FP<cr>
XXXX:0357 00 .<cr>
-WWS.COM,180,52FF<cr>
```

The following procedure converts MS-DOS WordStar to CP/M using DEBUG (the underlined parts are typed by the user):

```
A>DEBUG
>NWS.COM<cr>
>L280<cr>
>F100,27F,00<cr>
>E100 01 C7 09 00 00 FF 0F<cr>
>E456 00<cr>
>RCX<cr>
CX 5180
:5300<cr>
>NWS.CMD<cr>
>W<cr>
Writing 5300 bytes
```

Printer Interface

The above procedure converts the editing portion of WordStar between MS-DOS and CP/M. However, it does not convert the printing portion. The printer drivers in WordStar are all in the custom patch areas described in our last issue (Vol. 1, No. 3). Because of this, these drivers must be converted manually.

On the Victor 9000, WordStar is almost always configured to use the Standard List Device for printer output (CSWTCH = 0h). The patch areas for the Standard List Driver are as follows:

LIBSY	7CCh	15 bytes	Printer Status
LISEND	7DDh	16 bytes	Send Character
LISINP	7EDh	3 bytes	Receive Char.

The WordStar INSTALL program puts dummy routines in for the Printer Status (LIBSY) and Receive Character (LISINP) calls. These need not be changed when converting WordStar between CP/M and MS-DOS. The Send Character routine (LISEND) is the operating system call that causes a character to be printed on the current list device. The versions of this routine for CP/M and MS-DOS are given below, both in assembly language and in hex. The hex codes may be used for patching the driver using DEBUG, DDT86, or WordStar's INSTALL program, as described in Volume 1, Number 3.

CP/M Send Character Routine

```
LISEND:  MOV  DL,AL     ;char is in AL
         MOV  CL,5      ;print char fn
         INT  0E0h      ;CP/M bdos call
         CLC            ;clear status flag
         RET            ;return to WS
```

CP/M Send Character Routine in Hex

```
7DDh     8Ah,D0h,B1h,05h,CDh,E0h,F8h,C3h
```

MS-DOS Send Character Routine

```
LISEND:  MOV  DL,AL     ;char is in AL
         MOV  AH,5      ;print char fn
         INT  021h      ;MS-DOS dos call
         CLC            ;clear status flag
         RET            ;return to WS
```

MS-DOS Send Character Routine in Hex

```
7DDh     8Ah,D0h,B4h,05h,CDh,21h,F8h,C3h
```

Control C

There is a problem with WordStar versions prior to version 3.21 that has to do with the proper handling of Control-C when running under MS-DOS. These early versions use MS-DOS function calls that cause termination of the program if a Control-C is detected during input. Because of this, these early versions are not suitable for conversion from CP/M to MS-DOS.

Program Name

Once WordStar has been converted from to run under a different operating system, the file name that WordStar uses to get back to itself, following the execution of the "R" (run external program), must be corrected. This file name is at offset 3E7h (see Volume 1, Number 3).

Overlay Files

There are no modifications required to the WordStar OVR files for running under an alternate operating system.

132 Column Mode

The Victor 9000 is can switch its screen format from 80 columns by 24 lines to 132 columns by 50 lines. This is through the use of the 132 column utility program, 132C, which enables the 9000 to be switched between the two formats. The program 132C-ON switches the 9000 to 132 column mode. The program 132C-OFF puts the 9000 back in 80 column mode.

WordStar is also has the capability of running in 132 column mode. The height of the screen is determined by the HITE patch location (248h) and the width of the screen is determined by the WID patch location (249h). These should be set to 50 (32 hex) and 132 (84 hex), respectively, for 132 column operation.

Automatic Column Mode Switching

WordStar can be patched so that it automatically enables 132 column mode when it is invoked, and restores 80 column mode when it is terminated. After the 132C utility has been run to enable 132 column mode, 132 column mode can be turned on by sending "ESC |" (1Bh,7Ch) to the display. 80 column mode is restored by sending "ESC z" (1Bh,7Ah) to the display. All that is required to have WordStar switch automatically into and out of 132 column mode, is to have it send the appropriate ESC sequence to the display at initializaion and termination. The patch locations in WordStar that are used to accomplish this are TRMINI (292h) and TRMUNI (29Bh).

In the standard version of WordStar for the Victor 9000, there is not enough extra room at the TRMINI patch area to add the ESC sequence for turning on 132 column mode. There is, however, an ESC sequence installed in this patch area that is not generally needed when invoking WordStar. It is the

enable keyboard sequence "ESC (" (1Bh,7Bh). This sequence can be replaced with the 132 column mode on sequence. There is no space problem with the TRMUNI patch area.

The following are the required patches for automatic switching into and out of 132 column mode:

```
TRMINI  292h  07h,1Bh,78h,04h,1Bh,7Ch,1Bh,33h

TRMUNI  29Bh  02h,1Bh,7Ah
```

Highlighting Method

The standard WordStar configuration uses reverse video for highlighting of menus and text blocks. I have found the contrast between the reverse and normal video to be greatly annoying. I prefer the way that Dbase-II performs highlighting by using the foreground and background (high and low intensity) modes available on the 9000's display. The foreground and background method has the additional advantage that the contrast between the two modes is adjustable using the screen contrast and brighness keys (these are the ALT cursor positioning keys).

Although the opposite implimentation works just as well, I chose to have the menus and marked text blocks display at low (background) intensity, and the normal text to display at high (foreground) intensity.

The WordStar patches to change the highlighting method to that described above are as follows:

```
IVON     284h     02h,1Bh,29h

IVOFF    28Bh     02h,1Bh,28h
```

One side effect of making the menus low intensity and the text high intensity is that, after exiting WordStar, the cursor is at a different intensity than the rest of the screen. If this is annoying, then the "ESC)" (1Bh,29h) sequence should be added to the TRMUNI patch area to set the cursor to low intensity mode when WordStar terminates. The following patch accomplishes this:

```
TRMUNI   29Bh     02h,1Bh,29h
```

If the menus are set to display at high intensity, and the text at low intensity, then the above patch is not needed.

WordStar "Front End"

I have three or four standard formats that I repeatedly use in word processing. One of the few things that I don't like about WordStar is the inablity to select its initial state from a predefined list of formats. This means that I either have to manually set up my editing format each time I enter WordStar or I have to keep several copys of WordStar, with each one configured for a particular format.

Not caring for either of the above options, I decided to write a small program that I could patch into WordStar, that would allow me select from a predefined list of formats, and would configure WordStar to the format selected.

The program reads a file named FORMATS.WS to get the menu list of formats that I have defined, along with the patch information required to implement each format. After selecting a format, the program sets the specified patch locations to the given values and then jumps to WordStar's entry point. The FORMATS.WS file is an ASCII file that can be easily changed using a simple editor (like Wordstar, in non-document mode).

The syntax for each menu entry in the FORMATS.WS file is defined as follows:

Title Line -
The first line of each menu item contains the title of the format that is displayed on the screen when WordStar is invoked. The first character on the line must be an asterisk ('*'). Everything after the asterisk, to the end of the line, is displayed on the screen as the title for the format. The following line would cause the title " 1. Inter-office memos" to be displayed on the screen, if it were the first entry in FORMATS.WS.

```
*Inter-office memos
```

Default Drive Line -
The second line of each format item is the default drive specification. This is the drive for which WordStar will display the directory. The syntax for this line is the drive letter followed by a colon (':'). The following line would cause the "B" drive to be the default drive after WordStar starts running.

```
B:
```

Patches -
The third and following lines of each format item are the patches to be made in WordStar prior to its start-up. The syntax for each patch line is the starting address of the patch (in hex) followed by a semi-colon (';'), followed by one or more byte values (in hex), separated by commas (','), that are the values to be placed at the patch address. If more than one byte value is specified, these values are placed at successive addresses after the initial patch address. The following line would cause the value 2h to be stored at 29Bh, 1Bh to be stored at 29Ch, and 29h to be stored at 29Dh.

```
29B;2,1B,29
```

Non-consecutive patches must be entered on separate lines. There is no limit to line length, but consecutive entries must all be on the same line. In other words, the first entry on each patch specification line must be a patch starting address. All patch information is in HEX.

An example FORMATS.WS file is shown below. The "Letters" format sets the left margin to column 1, the right margin to column 65, micro-justification is turned off, hyphen help is turned on, and the page break display is turned off. The "132 Column" format sets the automatic 132 mode (described above), sets the left column to 1 and the right column to 100, turns micro-justification off, turns hyphen help off, and turns the page break display off. For a more complete description of the effect of each of the patches, refer to Volume 1, Number 3.

```
*Letters
B:
37F;0
380;40
386;0
389;FF
38D;0
*132 Column Mode
C:
248;32
249;84
292;7,1B,78,4,1B,33,1B,7C
29B;4,1B,29,1B,7A
37F;0
380;63
386;0
389;0
38D;0
```

This format file produces the following screen display when the modified WordStar is started:

```
+-----------------------------------------+
|                                         |
| WordStar Format Selection Menu - Version 1.0 |
|                                         |
|                                         |
| 1. Letters                              |
| 2. 132 Column Mode                      |
|                                         |
|                                         |
| Enter Selection:                        |
|                                         |
|                                         |
+-----------------------------------------+
```

When the selection is entered, the program then patches WordStar according to the information in FORMATS.WS for the corresponding selection. After the patches are made, the program jumps to WordStar and WordStar continues normally.

The format selection program is given in Listing 2 in CP/M assembler format. The program is the the same for MS-DOS, except for minor assembler differences. To patch the format program into WordStar, you must first assemble it, then make it into a COM or CMD file, and then use DEBUG or DDT86 to incorporate it into WordStar. (When linking the program under MS-DOS, you will get an error message saying that no stack was defined. This is normal for COM files. Ignore the message.)

To make a patched WordStar CMD file named WSM.CMD, using CP/M, follow the steps below:

```
A>ASM86 WSMENU<cr>
A>GENCMD WSMENU 8080<cr>
A>DDT86<cr>
-RWS.CMD<cr>
  START      END
XXXX:0000 XXXX:52FF
-RWSMENU.CMD<cr>
  START      END
YYYY:0000 YYYY:55FF
-MYYYY:5300,55FF,XXXX:5300<cr>
-SXXXX:181<cr>
XXXX:0181 81 7F<cr>
XXXX:0182 38 51<cr>
XXXX:1083 E9 .<cr>
-SXXXX:5300<cr>
XXXX:5300 00 84<cr>
XXXX:5301 00 39<cr>
XXXX:5302 BC .<cr>
-WWSM.CMD,XXXX:0,55FF<cr>
```

To make a patched WordStar COM file named WSM.COM, using MS-DOS, perform the following steps:

```
A>ASM WSMENU;<cr>
A>LINK WSMENU;<cr> (ignore the warning)
A>EXE2BIN WSMENU WSMENU.COM<cr>
A>DEL WSMENU.EXE<cr>
A>DEBUG WSMENU.COM<cr>
>NWS.COM<cr>
>L<cr>
>E101 81 38<cr>
>E5280 84 39<cr>
>RCX<cr>
CX 5180
:5480<cr>
>W<cr>
Writing 5480 bytes
>Q<cr>
```

After you have a patched version of WordStar, create a simple FORMAT.WS file, like the one above, to test the new program. Once you have determined that the new program is working, you can then experiment with more complicated patch setups.

```
;  WordStar Setup Menu Program

progdrv equ     ds:byte ptr .2DCh;WordStar default drive spec
msflag  equ     ds:byte ptr .2D6h;MS-DOS flag

        cseg

        org     100h            ;small model start

start:  jmp     wsmenu

        org     5280h           ;address of end of WS.COM

wsentry dw      0               ;wordstar entry address

wsmenu: mov     sp,offset stack ;set stack pointer
        mov     ax,cs           ;
        mov     ds,ax           ;
        mov     es,ax           ;
        cld                     ;all strings forward

;  set DMA segment if CP/M

        cmp     msflag,0        ;test for MS-DOS version
        jnz     wsm0            ;skip if MS-DOS

        mov     dx,ax           ;move SEG to proper register
        mov     cl,51           ;set DMS segment
        call    bdos            ;

;  read in FORMATS.WS and parse data

wsm0:   mov     al,progdrv      ;get WS program drive spec
        mov     fcb,al          ;store in file block

        mov     dx,offset fcb   ;point to file block
        mov     ah,15           ;open file
        call    bdos            ;call system

        or      al,al           ;check if found
        jz      wsm1            ;skip if yes

        mov     dx,offset ermsg1;open error
        jmp     error           ;quit

wsm1:   mov     ds:byte ptr fcb+32,0;reset record number

        mov     dx,offset formats;set dma address for read

wsm1a:  push    dx              ;save offset
        mov     ah,26           ;set transfer adr
        call    bdos            ;

        mov     dx,offset fcb   ;point to file block
        mov     ah,20           ;sequential read
        call    bdos            ;read entire file

        pop     dx              ;restore read offset
        cmp     al,0            ;check for eof
        jnz     wsm1b           ;skip if eof

        add     dx,128          ;increment offset for next rec
        jmp     wsm1a           ;read some more

;  display format menu and get patch table addresses

wsm1b:  mov     dx,offset menumsg;put up initial menu msg
        call    print           ;

        mov     bx,offset fmttbl;initialize pointer
        mov     si,offset formats;point to start of format file

wsm2:   lods    al              ;get char of title
        cmp     al,'*'          ;start of title ?
        jz      wsm4            ;print title if found
        cmp     al,1Ah          ;end of file ?
        jz      wsm8            ;go on if eof found
        jmp     wsm2            ;keep looking

wsm4:   mov     dx,offset menunum;print format index number
        mov     ah,9            ;print string
        call    bdos            ;

        inc     ds:byte ptr menunum+1;increment index number
        cmp     ds:byte ptr menunum+1,'9';overflow ?
        jbe     wsm5                    ;skip if ok

        mov     ds:byte ptr menunum+1,'0';reset ls digit
        mov     ds:byte ptr menunum,'1' ;set ms digit

wsm5:   lods    al              ;get char of title
        mov     dl,al           ;put in proper register
        mov     ah,2            ;display char
        call    bdos            ;

        cmp     dl,0Ah          ;end of line ?
        jnz     wsm5            ;loop back if not

        call    skipctl         ;skip all ctrl chars

        mov     ds:word ptr [bx],si;save patch address
        add     bx,2            ;point to next table position
        inc     ds:word ptr fmtcnt;increment number of formats
```

Listing 2. WordStar Front End.

```
        jmp     wsm2            ;look for next format

wsm9:   mov     dx,offset menumsg1;put up rest of menu message
        call    print           ;

wsm9:   call    getsel          ;get selection
        cmp     bx,0            ;check validity of selection
        jle     wsm9a           ;invalid, try again

        cmp     bx,ds:fmtcnt    ;check upper limit
        jle     wsm10           ;skip if ok

wsm9a:  mov     dx,offset ermsg3;invalid entry
        mov     ah,9            ;print string
        call    bdos            ;

        jmp     wsm9            ;invalid, try again

wsm10:  dec     bx              ;base at zero instead of one
        shl     bx,1            ;make word index
        mov     si,ds:fmttbl[bx];get setup table address

        lods    al              ;get drive spec
        cmp     al,'A'          ;check minimum
        jb      error0          ; exit if error

        sub     al,'A'          ;make bdos value
        mov     ds:drvspec,al   ;save

        lods    al              ;check for terminator
        cmp     al,':'          ;
        jnz     error0          ; exit if not proper terminator

wsm11:  call    skipctl         ;skip ctrl chars

wsm12:  call    getnum          ;get patch address
        jb      error1          ;exit if syntax error
        cmp     bl,'*'          ;end of patches ?
        jz      wsm20           ; exit if yes
        cmp     bl,1Ah          ;end of file ?
        jz      wsm20           ; exit if yes
        cmp     bl,';'          ;check ending
        jnz     error1          ; exit if error

        mov     di,ax           ;put patch adr in index reg

wsm13:  call    getnum          ;get patch value
        jb      error1          ; exit if syntax error
        stos    al              ;store patch

        cmp     bl,','          ;more to come ?
        jz      wsm13           ;loop if yes
```

Listing 2. WordStar Front End.

```
        jmp     wsm12           ;look for next patch line

;  set default drive

wsm20:  mov     dl,ds:drvspec   ;get default drive spec
        mov     ah,14           ;select disk function
        push    bx              ;save index
        call    bdos            ;call DOS

        jmp     ds:word ptr wsentry;go to wordstar

;  error exits

error0: mov     dx,offset ermsg4;bad drive spec
        jmp     error

error1: mov     dx,offset ermsg5;bad patch table
        jmp     error

error:  mov     ah,9            ;print error msg
        call    bdos            ;

exit:   cmp     msflag,0        ;MS-DOS ?
        jnz     exitms          ;skip if MS-DOS

        xor     ax,ax           ;CP/M exit
        mov     dx,ax           ;
        int     224             ;exit to system

exitms: int     32              ;exit to system

;  universal bdos call

bdos:   cmp     msflag,0        ;MS-DOS ?
        jz      bdos1           ;skip if not

        int     33              ;call MS-DOS
        ret                     ;

bdos1:  mov     cl,ah           ;change for CP/M call
        push    bx              ;
        int     224             ;call CP/M
        pop     bx              ;
        ret                     ;

;  print string system call

print:  mov     ah,9            ;print string system call
        call    bdos            ;
        ret                     ;

;  get number from input file - SI is pointer to current position
```

```
getnum: xor     bx,bx           ;init number to zero

gn1:    lods    al              ;get next char
        cmp     al,1Ah          ;end of file
        jz      gn9             ; exit if eof
        cmp     al,0Dh          ;end of line ?
        jz      gn9             ; exit if eol
        cmp     al,'*'          ;end of patch ?
        jz      gn9             ; exit if eop
        cmp     al,';'          ;end of address ?
        jz      gn9             ; exit if eoa
        cmp     al,','          ;end of byte ?
        jz      gn9             ; exit if eob
        cmp     al,' '          ;space or ctrl
        jbe     gn1             ; skip if space or ctrl

        cmp     al,'0'          ;less than 0 ?
        jb      gn10            ; error exit if so
        cmp     al,'9'          ;over 9 ?
        jbe     gn2             ;skip if not

        and     al,5Fh          ;force upper case
        cmp     al,'A'          ;less than A ?
        jb      gn10            ; error exit if so
        cmp     al,'F'          ;more than F ?
        ja      gn10            ; error exit if so

        add     al,9            ;convert letter

gn2:    and     ax,0Fh          ;assume good hex char
        shl     bx,1            ;shift over prev. result
        shl     bx,1            ;
        shl     bx,1            ;
        shl     bx,1            ;
        add     bx,ax           ;add in next digit (hex)
        jmp     gn1             ;go for more

gn9:    xchg    ax,bx           ;swap
        clc                     ;no error
        ret                     ;

gn10:   stc                     ;flag error
        ret                     ;

;  skip control chars and spaces

skipctl:lods    al              ;get char
        cmp     al,' '          ;test
        jbe     skipctl         ;loop if ctrl or space

        dec     si              ;back up pointer
        ret                     ;exit
```

Listing 2. WordStar Front End.

```
;  get selection

getsel: mov     ah,1            ;get 1 char from kbd
        call    bdos            ;

        sub     al,'0'          ;make binary
        mov     bl,al           ;move to index reg
        mov     bh,0            ;clear index msb
        ret

;  data area

fcb     db      0               ;drive spec
        db      'FORMATS WS '
        rb      27

;  selection menu message

menumsg db      27,69   ;clear screen
        db      'Channel 9000 WordStar Format Selection Menu'
        db      ' - Version 1.0',13,10
        db      10,10,10,10
        db      '$'

menumsg1 db     13,10,10,'Enter Selection: $'

menunum db      ' 1.  $'

ermsg1  db      'Cannot open FORMATS.WS',13,10,'$'
ermsg2  db      'Not enough memory available.',13,10,'$'
ermsg3  db      13,10,'Invalid selection.'
        db      ' Enter selection again: $'
ermsg4  db      'Bad drive specification in '
        db      'patch information file',13,10,'$'
ermsg5  db      'Bad patch format.',13,10,'$'

        rb      128
stack:

drvspec db      0               ;default drive
fmtcnt  dw      0               ;number of formats
fmttbl  rw      15              ;reserve room for 15 adrs

formats db      0               ;end of prog (gencmd kludge)

        end
```

Channel 9000

THE INDEPENDENT NEWSLETTER OF THE VICTOR 9000/SIRIUS 1 COMPUTER

VOL. 1 NO. 5 September 1983

From the Editor:

The biggest news of recent days is the re-reorganization that Victor is undergoing. Last November, Sirius Systems Technology merged with Victor United (and Victor Business Products) to form Victor Technologies, the manufacturing entity, and Victor United, the sales and distribution subsidiary. Subsequently, eight districts were established, each charged with the duties of providing marketing, sales, and technical support for a specific region of the country.

The districts had a great deal of autonomy in terms of what they could do and how they could do it. I think that Victor, seeing the success it was having in Europe, tried to fashion its domestic marketing organization after the European model, in which each country has a single distributor, who determines how the product is to be marketed in its region. Also, there appears to have been a mandate that the districts should put forth an image of success. Apparently, little expense was spared in doing this.

In addition, Victor has had some problems in gearing up for increased production to meet the growing demand. They were still having difficulty getting software out the door. These problems, coupled with typically slow summer European sales, led to a net operating loss last quarter measured in millions of dollars.

All of this has caused Victor, evidently being prodded by its parent, Kidde, Inc., to retrench, and reevaluate its marketing and distribution strategies. While things are still in a state of flux, it appears that Victor is cutting the number of districts to four, and eliminating all but the major branches. Victor has laid off close to 1000 people, some from the main plant, but most from the districts and branches.

While all of this makes the outlook for Victor seem bleak, I don't believe that it is. Most new companies, like infants, make some false steps learning to walk. Victor is far from falling on its face. First of all, the Victor 9000 is what I consider to be the most viable small business computer on the market. Sirius sales in Europe will pick up again as summer comes to a close. And last, but not least, Kidde, Inc. is a well fixed company with an investment to protect.

Although there is no doubt in my mind that Victor will eventually be very successful, how do these recent events affect those of us who are current users, owners, and dealers. It seems to me, at least for the near future, that there is probably not much hope for improved customer support. Fewer people will be available to provide assistance. It still looks like the only way we are going to get any help, is to continue to help ourselves.

Users groups still appear to be the best way to do this. The following is a very brief list of those groups that I have heard from since our last edition.

Arizona, Pheonix
Problem Solvers Unanimous
Joe E. Reid (602) 946-5948

Arizona, Tuscon
Bernard Bell (602) 326-6324

California, Sacramento
Victor Users Group of Sacramento
Guy M. Wong (916) 924-9611

California, San Francisco Bay Area
Victor*Group
Jack Skalicky (415) 921-5884

There are other groups for which I do not have contacts yet. I have also received many letters asking if there are groups in various parts of the country. One of the problems is going to be getting those people who want to start or belong to a group in contact with each other.

There is some good news from Victor concerning this. Sandy Wells, the editor of the **Software Newsletter** for Victor, is doing all she can to support and encourage the formation of users groups. If you have a group, or would like to start or join one, please call or write her at the address below.

Sandy Wells
Victor Technologies, Inc.
380 El Pueblo Road
Scotts Valley, CA 95066
(408) 438-7000

Victor has always stated that they support the idea of users groups, and I am very pleased that they are starting to back this up with action at the main office.

James M. Lesher

Third Party Products

Auxiliary Monitor Interface
Professional Data Systems has developed a device that converts the Victor 9000 monitor signals to composite video. A key feature of the interface is that it does not require internal attachment or modification to the Victor 9000. It is easily connected between the CRT output port on the CPU unit and the 9000's video monitor. Once installed, the interface allows the connection of multiple monitors, or large screen projection systems, making it ideal for group presentations for demos, trade shows, training, etc.

PDS is offering the Auxiliary Monitor Interface in conjunction with a broad selection of quality monitors. Custom configurations are also available.

Professional Data Systems
444 Camino Del Rio South
Suite 130
San Diego, CA 92108
(619) 291-2300

Clock/Calendar Board
A Clock/Calendar board is now available for the Victor 9000. This board is easily fitted into one of the internal expansion slots of the 9000. It features an on-board NI-CAD battery that keeps its popular MSM real-time clock/calendar chip running, even when the system is turned off.

Features include: Selectable 24 or 12 hour format; Automatic leap year recognition; Easy time and date setting; Latched input and output ports. Sample programs are included with purchase. Suggested retail price is $225.00.

Computer Possibilities Unlimited, Inc.
280 North Central Avenue
Suite 114
Hartsdale, NY 10530
(914) 949-0766

External Hard Disk System with Clock/Calendar
Univation, Inc., the manufactures of add on memory boards for the Victor, now are offering an external add-on, 11 Megabyte, hard disk system, that is attractively styled to compliment your Victor computer. The hard disk unit contains its own power supply, and attaches to the 9000 through an interface card that plugs into one of the 9000's internal expansion slots. The interface card also comes standard with a clock/calendar.

The hard disk can be partitioned so that multiple logical units may be defined simultaneously for MS-DOS and CP/M-86. The partitions and/or logical units may be of any size, up to the full capacity of the unit. Single unit pricing for the hard disk system is $1750.00.

Univation, Inc.
1037 North Fair Oaks Avenue
Sunnyvale, CA 94086
(408) 745-0180

VT100 Emulator
EM100 is a VT100 emulator for the Victor 9000. It is capable of emulating all of the major functions of a VT100 terminal with the exception of blink and the double width character set (these may be added later). The program features a user friendly, menu driven setup. An ASCII file transfer utility is included that supports wild card file transfer. Logging to a local floppy is also supported. EM100 is currently being field tested and will be available for final shipment by October 1st. The price of EM100 is $125.00.

Diversified Computer Systems
P.O. Box 7575
Boulder, CO 80306
(303) 443-6522

CHANNEL 9000

9742 Marcus Lane, Tujunga, CA 91042
(213) 352-6443
James M. Lesher, Editor
Subscription rate: $30.00 (US) for 6 issues
Overseas Air Mail Rate: $37.50 (US) for 6 issues
Published Bimonthly

Back Issues Available

Due to their demand, we try to keep reprints of all issues in stock.
Prices are: $5.00/ea for the US and Canada, $6.25 for Overseas.

Topics Covered:
Vol. 1, No. 1: Keyboard Tables
Vol. 1, No. 2: I/O Ports and Drivers
Vol. 1, No. 3: WordStar Patches
Vol. 1, No. 4: More on WordStar
Vol. 1, No. 5: Victor Display System

The Victor 9000 video display is a flexible and unique design, which provides for a wide range of display options. The display has two standard modes of operation: character mode and high resolution mode. In character mode, the screen is configured as a matrix of 2000 cells (or character positions) arranged as 25 rows of 80 cells, where each cell is 10 pixels (dots) wide by 16 pixels high. In high resolution mode, the screen is configured as a matrix of 1250 cells, arranged as 25 rows of 50 cells, where each cell is 16 pixels wide by 16 pixels high. For both modes, the overall resolution of the screen is 800 by 400 pixels.

The display screen has an aspect ratio of 3:2. This means that the pixels are spaced vertically 1-1/2 times farther than they are horizontally. In other words, a vertical line 10 pixels long is the same physical length as a horizontal line 15 pixels long. The aspect ratio must be taken into account when drawing, so that objects are proportioned properly.

In order to implement "soft tooled" character sets, as well as bit-mapped graphics, for the display, Victor has chosen a two level memory scheme. At address F000:0000, there is a 4K byte screen buffer, organized as 2048 16-bit words. Each word in the screen buffer is used to define the contents of a cell (character) to be displayed on the screen, along with its attributes. The format for a screen buffer word is as follows:

```
LSB                                                   MSB
15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
rv li ul nd xx |---Font Cell Pointer (fcp)-----|
```

rv - Reverse Video
li - Low Intensity
ul - Underline
nd - Non-Display
xx - Reserved

The pattern displayed in a cell is determined by the Font Cell Pointer (FCP). The FCP points to a 16 word (32 byte) area, in one of the two 64K blocks of the lower 128K of system memory, which contains the dot matrix pattern for the character to be displayed. Each 16 word area used for defining a cell (character) pattern is called a "font cell".

The 4K byte screen buffer is mapped into physical address space from F000:0000 to F000:0FFF. The hardware is configured so that the addresses in the range F000:1000 to F000:1FFF logically appear to be the same as F000:0000 to F000:0FFF. This enables the display controller to be programmed to start scanning the screen buffer at any address from F000:0000 to F000:1000, and still be able to display a full screen's worth of information. This feature allows the scrolling of the screen by adding 80 (in character mode) to the display controller's start address and blanking the 80 words following the previous end of screen. When doing this, it is necessary to AND the lower 16 bits of the resulting address with 1FFF (hex) to keep everything within the accessible address range.

Character Mode Fonts

The format of a font cell, used in character mode, is given below. The letter "G" is used for the example. Each X indicates that the corresponding bit is set, blank indicates that the bit is cleared. Note that the least significant bit of each word is on the left.

```
     LSB                                   MSB
      0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
00      X X X X X X
01    X X           X X
02    X X             X
03    X X
04    X X
05    X X
06    X X     X X X X
07    X X       X X
09    X X       X X
10    X X     X X X
11      X X X X X   X
12
13
14                                    X
15
```

As stated above, in character mode, only the 10 least significant bits of each word (bits 0 - 9) are used to define the character. The other bits, except for the most significant bit of each word, are ignored. The MSBs of the font cell words are used to determine placement of the underline (or strikeover) in the cell. In the above example, the MSB of word 14 is set, which means that if the underline attribute bit in the screen buffer word is set, then an underline will appear in the 14th row of the character cell when the "G" is displayed. Any or all of the words in the font cell may have their MSB's set. Usually, however, only one MSB in any given font cell word is set.

Character Tables for Character Mode

The dot-matrix patterns used to generate the characters that appear on the display are stored in files with the .CHR extension. The typical contents of a character table file is shown, in Listing 1, in assembly language format.

The system generation program, SYSELECT, provides for the selection of one or two character sets to be included with the system. The console modification program, MODCON, provides for dynamically changing the current character set (in RAM). The graphics kernal, GRAFIX, provides for the display, in graphics mode, of any of the available character sets, simultainously.

Character sets used for the the system display can be either 128 or 256 characters is size. However, the current BIOS software only provides enough

space for 256 characters total. This means that if the primary character set specified in SYSELECT contains 256 characters, the second character set will not be incorporated in the system, even though it was specified to be included. Correspondingly, if the primary character set contains 128 characters, only 128 characters from the secondary character set will be included in the system, even though it may actually contain 256 characters.

There are also some peculiarities in the way the BIOS handles the display of characters corresponding to the hexadecimal codes 80 through FF. If the primary character set contains 256 characters, then the upper 128 characters can be displayed by sending the hex codes 80 through FF to the console. If the primary character set contains 128 characters, then sending the hex codes 80 through FF to the console causes the display of the characters corresponding to the hex codes 00 through 7F.

The results for the secondary set are somewhat different, since the secondary character set can only contain 128 characters. Sending the hex codes 80 through FF to the console, while the secondary character set is enabled, causes the display of the characters corresponding to the hex codes 00 through FF from the primary character set.

Another ideosyncracy occurs when the primary character set contains 256 characters. Since SYSELECT does not allow more than 256 characters, total, for character sets, there is no secondary character set, when the primary character set contains 256 characters. Enabling the secondary character set under these conditions causes the character font pointers in the BIOS to point to that area of memory where programs are loaded. This can cause some strange dot patterns to be displayed on the screen.

The character sets used by the system are accessed directly by the hardware, so they must be designed to display properly under those restraints. The fonts should be designed to display horizontally in a 10 by 16 pixel area, and they should correspond to the ASCII definitions, where applicable. The SYSELECT program only uses that portion of the character set file header up to, and including, the "number of records" entry (see Listing 1).

High Resolution Mode

In high resolution mode, all 16 bits of each font cell word are utilized for display. The MSB's of each word are still active for underlining, but this feature is generally not used in this mode.

Screen access in high resolution mode is usually done using a bit mapped approach, where each pixel on the screen is controlled by a separate bit in memory. While there are many other ways of mapping the screen, this is the simplest method, and is the method used by the GRAFIX program.

Since the screen resolution is 800 by 400, it requires 320,000 bits, or 20,000 words, to represent it in a bit mapped manner. In order to have each pixel controlled by a separate bit, the screen buffer must be set up so that each font cell pointer points to a different 16-word area of memory. For discussion, we will use the address 0000:0000 as the base of our 20,000 word font memory. In actual applications, this wouldn't be done because both MS-DOS and CP/M use this lower area of memory.

Figure 1 shows a mapping scheme for the screen buffer that simplifies the computations required to determine which bit controls which pixel. This method was also chosen because it makes the drawing of vertical lines trivial; setting the same bit in a number of successive words accomplishes this.

All addresses and offsets given in Figure 1 are in hexadecimal. Each box in the figure represents one word (two bytes) of the screen buffer. The number in each box represents a 16 word (32 byte) font cell. The 9000 display driver hardware effectivly multiplies this number by 32 to get the address of the first word of the font cell.

With the mapping scheme given, the following BASIC program will convert the cartesian coordinates (X,Y) to the byte offset and bit number for the pixel. The program assumes that the coordinate of the lower, left pixel is (0,0), and the upper, right pixel is (799,399).

```
 10 INPUT "X";X
 20 INPUT "Y";Y
 30 X.COL = INT(X/16)*400
 40 Y.ROW = INT(399-Y)
 50 WRD = X.COL + Y.ROW
 60 WBIT = X MOD 16
 70 BYTE = WRD * 2
 80 IF WBIT>7 THEN BYTE=BYTE+1
 90 BBIT = WBIT MOD 8
100 PRINT "BYTE = ";BYTE;"  BIT = ";BBIT
```

The above mapping method uses only 1250 of the 2048 words of the screen buffer, and only 40,000 of the 65,536 bytes of font memory that is capable of being addressed. With some minor modification, the above scheme can be made to accomodate a graphic area that is 1024 by 512 dots, with the display screen being a 800 by 400 dot window on that area. In addition, the font memory can be readily switched between the first and second 64K block of memory, allowing for the quick changing of display information.

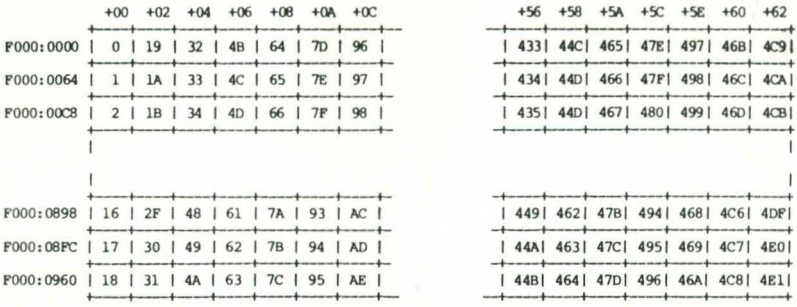

Figure 1. High Resolution Mapping

Display System Hardware
The operation of the display system is controlled by a HD46505 type CRT Controller chip (CRTC). This chip provides all the timing, addressing, and control functions for the display system. The CRTC is memory mapped at the following addresses:

E800:0000	Address Register (AR)
E800:0001	Register File (RF)

The Address Register (AR) is used to tell the chip which of the 18 registers accessible through the Register File (RF) you wish to read or write. To read or write to a register, you first write the number of the register you wish to access to the AR at E800:0000, then you read or write the data through the RF address, E800:0001.

The following is a brief description of the CRTC register functions. All registers are 8-bit. 16 bit parameters are programmed using two separate registers. For a more detailed description, obtain a manufacturer's data sheet (Hitachi).

R0 - **Horizontal Total Register**
This register determines the horizontal frequency sent to the CRT. It is programmed in terms of character times, being the total of displayed and non-displayed character time units minus one. In interlace mode (the mode used by the 9000), the total character times must be even (i.e. the number programmed into R0 must be odd).

R1 - **Horizontal Displayed Register**
This register determines the number of characters displayed on each line. Any value less than that programmed into R0 is legal.

R2 - **Horizontal Sync Position Register**
This register determines the position of the horizontal sync pulse and, therefore, the left-right centering of the displayed lines on the screen. It is programmed in terms of character positions. It should be a value between those of R1 and R0, but any value less than R0 is legal. Increasing the value of R2 moves the displayed lines to the left, decreasing R2 moves the lines to the right.

R3 - **Sync Width Register**
This register determines the widths of the horizontal and vertical sync pulses. The most significant four bits of R3 (bits 7-4) determine the vertical sync pulse width in terms of the horizontal period (i.e. a value of 12 would set the width to 12 horizontal periods). A value of zero sets the vertical sync width to 16 horizontal periods. The least significant four bits of R3 determine the horizontal sync pulse width in terms of character times. The value of zero is illegal.

R4 - **Vertical Total Register**
This register determines the total number of character rows per frame. The value programmed should be one less than the total number of character rows. This register, together with R5, determines the overall frame rate. Values in the range of one to 127 are legal.

R5 - **Vertical Total Adjust**
This register sets the additional number of horizontal time periods used to fine tune the vertical period (see R4). Values in the range of zero to 31 are legal.

R6 - **Vertical Displayed**
This register determines the number of character rows displayed on the screen. Any number smaller than R4 is legal.

R7 - **Vertical Sync Position**
This register determines the position of the vertical sync pulse and, therefore, the up-down centering of the displayed lines on the screen. It is programmed in terms of character row number. It should be a value between those of R6 and R4, but any value less than R4 is legal. Increasing the value of R7 moves the displayed lines upward, decreasing R7 moves the lines downward.

R8 - Mode Control
This register sets the interlace mode and delay timing of two of the CRTC output signals. The two least significant bits (bits 1 & 0) set the interlace mode as follows:

Bit 1	Bit 0	Mode
0	0	Non-interlace mode
1	0	Non-interlace mode
0	1	Interlace sync mode
1	1	Interlace sync & video

The most significant four bits (bits 7 - 4) control delay timing for some of the outputs of the CRTC. For the hardware design of the 9000, these bits are always set to zero.

R9 - Maximum Raster Address Register
This register determines the number of rasters (lines) per character. It is programmed to one less than the number of rasters, for non-interlace and interlace sync mode. For interlace sync and video mode, it is programmed to two less than the number of rasters, and the number of rasters must be even. The difference for interlace sync and video mode is due to difference in the display method used in that mode. For that mode, the odd and even rasters are displayed in alternating frames, in order to improve the display resolution and prevent flicker. If in this case, R9 is programmed to one less than the number of rasters instead of two less, the even frame would display 9 rasters per character, instead of 8, as it should.

R10 - Cursor Start Register
This register determines the cursor starting position in the character cell and the mode of cursor displayed. The five least significant bits (bits 4 - 0) set the starting raster number for the cursor. Bits 6 and 5 set the cursor mode, as follows:

Bit 6	Bit 5	Mode
0	0	Steady cursor
0	1	No cursor
1	0	Blink at 1/16 field rate
1	1	Blink at 1/32 field rate

R11 - Cursor End Register
This register determines the cursor ending position in the character cell. The five least significant bits (bits 4 - 0) set the ending raster number for the cursor.

Cursor Note:
By setting the cursor start and end to the same raster number, an underline cursor can be programmed. By setting the cursor start to zero, and the cursor end to the last raster number, a block cursor can be programmed.

R12 & R13 - Start Address Registers
These registers determine the address in screen memory of the first character of the displayed scan (the character at the top left of the display). For the 9000, only 11 bits of address are used. The least significant 8 bits of the address are programmed into R13. The most significant 3 bits of the address are programmed into the least significant 3 bits of R12. In the Victor 9000, two of the unused address bits of the CRTC are used to enable high-intensity mode, and to control which 64K address space is used for the font memory. Bit 4 of R12 controls the 64K font memory block. A zero sets it to the lower 64K of system memory, a one sets it to the upper 64K. Bit 5 controls the resolution mode of the display. A zero sets character mode, a one sets high resolution mode.

R14 & R15 - Cursor Address Registers
These registers determine the position that the cursor is displayed on the screen. The position is programmed by setting the cursor address to the address, in screen memory, of the character that corresponds to the position at which the cursor is to display. R14 and R15 are programmed in the same manner as R12 and R13, above, including the bits used as mode control.

R16 & R17 - Light Pen Registers
These registers contain the address, in screen memory, of the character that was being displayed when the light pen input was strobed.

CRTC Standard Programming
The Victor BIOS and other Victor programs use two recommended sets of values for initializing the CRTC registers. These values are given in the table below.

Register	Char. Mode	Hi-Res Mode
R0	92 (5C)	58 (3A)
R1	80 (50)	50 (32)
R2	81 (51)	52 (34)
R3	207 (CF)	201 (C9)
R4	25 (19)	25 (19)
R5	6 (06)	6 (06)
R6	25 (19)	25 (19)
R7	25 (19)	25 (19)
R8	3 (03)	3 (03)
R9	14 (0E)	14 (0E)
R10	0 (00)	0 (00)
R11	15 (0F)	15 (0F)
R12	0 (00)	0 (00)
R13	0 (00)	0 (00)
R14	0 (00)	0 (00)
R15	0 (00)	0 (00)

The values for character mode result in an 80 character by 25 line display with a horizontal frequency of 16,129 Hz, and a vertical frequency of 38.2 Hz. The values for high resolution mode result in a 50 character by 25 line display with a horizontal frequency of 15,890 Hz, and a vertical frequency of 37.7 Hz. While these values are close to the television standards of 15,750 Hz for the horizontal frequency, and 30.0 Hz for the vertical frequency, they are far enough off that not all standard monitors would be capable of synchronizing with them. Care should be taken if you wish to use a non-Victor monitor, to make sure that monitor can synchronize with the Victor's outputs.

Video Output

Connection to the external monitor is made through a 9-pin, subminiature "D" connector on the back of the Victor 9000 CPU unit. Connection for the power for the monitor, as well as the video signals, is made through this connector. The pin assignments for the connector are as follows:

Pin #	Signal
1	Video
2	Video Ground
3	+12 Volts (Power to monitor)
4	Ground
5	Brightness
6	Video Shield
7	Vertical Sync
8	Ground
9	Horizontal Sync

Light Pen Input

The 9000 provides a buffered TTL input for the light pen input to the CRTC. This signal is pulled to +5 volts through a 2.2K resistor. The input is activated by grounding it. The signal is available through the User Port connector inside the Victor CPU unit. The light pen input signal is on Pin 10 of the User Port connector, and its corresponding ground is on Pin 11.

Brightness and Contrast

The Victor 9000 display system provides for 8 levels of brightness and 8 levels of contrast for the display. Brightness is the overall intensity of both the high intensity and low intensity modes. Contrast is the difference in intensity between the high intensity and low intensity modes. Decreasing the contrast decreases the difference in intensities.

While the brightness and contrast are normally controlled through the sending of escape sequences to the console, is also possible to control them directly through the hardware. The brightness and contrast are each controlled by three bits of the B port of the VIA2 6522 Peripheral Interface Adapter chip (base address E800:0040).

VIA2 pins	Function
PB2	Brightness 0 (LSB)
PB3	Brightness 1
PB4	Brightness 2 (MSB)
PB5	Contrast 0 (LSB)
PB6	Contrast 1
PB7	Contrast 2 (MSB)

The following assembly language and BASIC routines illustrate the manipulation of the contrast and brightness controls. The assembly routines assume that the ES register contains 0E800H.

```
;increase brightness
INCB:MOV  AL,BYTE PTR ES:40H  ;get current value
     MOV  AH,AL               ;make copy of register
     ADD  AL,04H              ;increment brightness
     AND  AL,1CH              ;mask all but brightness
     CMP  AL,0                ;check for overflow
     JZ   NODO                ;skip if at maximum
     AND  AH,0E3H             ;get original reg values
     OR   AH,AL               ;merge in brightness
     MOV  BYTE PTR ES:40H,AH  ;output new values
NODO:RET                      ;continue on

;decrease brightness
DECB:MOV  AL,BYTE PTR ES:40H  ;get current value
     MOV  AH,AL               ;make copy of register
     SUB  AL,04H              ;decrement brightness
     AND  AL,1CH              ;mask all but brightness
     CMP  AL,1CH              ;check for underflow
     JZ   NOD1                ;skip if at minimum
     AND  AH,0E3H             ;get original reg values
     OR   AH,AL               ;merge in brightness
     MOV  BYTE PTR ES:40H,AH  ;output new values
NOD1:RET                      ;continue on

;increase contrast
INCC:MOV  AL,BYTE PTR ES:40H  ;get current value
     MOV  AH,AL               ;make copy of register
     ADD  AL,20H              ;increment contrast
     AND  AL,0E0H             ;mask all but contrast
     CMP  AL,0                ;check for overflow
     JZ   NOD2                ;skip if at maximum
     AND  AH,1FH              ;get original reg values
     OR   AH,AL               ;merge in contrast
     MOV  BYTE PTR ES:40H,AH  ;output new values
NOD2:RET                      ;continue on

;decrease contrast
DECC:MOV  AL,BYTE PTR ES:40H  ;get current value
     MOV  AH,AL               ;make copy of register
     ADD  AL,20H              ;increment contrast
     AND  AL,0E0H             ;mask all but contrast
     CMP  AL,0E0H             ;check for underflow
     JZ   NOD3                ;skip if at minimum
     AND  AH,1FH              ;get original reg values
     OR   AH,AL               ;merge in contrast
     MOV  BYTE PTR ES:40H,AH  ;output new values
NOD3:RET                      ;continue on
```

```
 10 DEFINT V
 20 DEF SEG=&HE800
 30 REM INCREASE BRIGHTNESS
 40 VIA2=PEEK(&H0040)
 50 IF (VIA2 AND &H001C)<>&H001C
       THEN VIA2=VIA2+4 : POKE &H0040,VIA2
 60 RETURN
 70 REM DECREASE BRIGHTNESS
 80 VIA2=PEEK(&H0040)
 90 IF (VIA2 AND &H001C)<>0
       THEN VIA2=VIA2-4 : POKE &H0040,VIA2
100 RETURN
110 REM INCREASE CONTRAST
120 VIA2=PEEK(&H0040)
130 IF (VIA2 AND &H00E0)<>&H00E0
       THEN VIA2=VIA2+&H20 : POKE &H0040,VIA2
140 RETURN
150 REM DECREASE CONTRAST
160 VIA2=PEEK(&H0040)
170 IF (VIA2 AND &H00E0)<>0
       THEN VIA2=VIA2-&H20 : POKE &H0040,VIA2
180 RETURN
```

High-Resolution Graphics Programming Notes

Programming for high resolution mode requires that some careful consideration be given to the memory areas used for graphic functions, along with those used for the program itself. There are restrictions as to where the font memory can reside, and there are areas within the possible font memory locations that are reserved for use by the CP/M and MS-DOS operating systems.

"Reserved" Operating System Areas

The lowest 11K bytes of memory (slightly more for MS-DOS) are used by the operating system. The first 1K bytes are hardware and software interrupt vectors. For MS-DOS, the next 128 bytes are used for jump tables, which are the link between the MS-DOS BDOS and the Victor BIOS. Following that, for both CP/M and MS-DOS, are 10K bytes that are used for the dot patterns for the sign-on logo and the resident character sets. Immediately following these areas is the start of the Transient Program Area (TPA), the lowest address available for program usage. The tables below show the (hex) addresses of these areas.

MS-DOS Memory Map

Addresses	Usage
0000:0000 - 0000:03FF	Interrupt Vectors
0000:0400 - 0000:047F	MS-DOS Jump Table
0000:0480 - 0000:0C7F	Logo Dot Patterns
0000:0C80 - 0000:1C7F	First 128 Char Set
0000:1C80 - 0000:2C7F	Second 128 Char Set
0000:2C80 -	Start of TPA

CP/M-86 Memory Map

Addresses	Usage
0000:0000 - 0000:03FF	Interrupt Vectors
0000:0400 - 0000:0BFF	Logo Dot Patterns
0000:0C00 - 0000:1BFF	First 128 Char Set
0000:1C00 - 0000:2BFF	Second 128 Char Set
0000:2C00 -	Start of TPA

It is generally not desirable to require that the system be rebooted when a program is terminated. This requires that those memory areas up to the beginning of the TPA either not be modified, or else be restored to their original contents when the program ends.

Placement of Graphics Memory

As discussed earlier, 40,000 bytes are required for the screen representation for bit mapped graphics. These 40,000 bytes are also required, by the hardware, to be entirely within the first 64K block of memory (0000:0000 - 0000:FFFF), or entirely within the second 64K block of memory (1000:0000 - 1000:FFFF). The beginning of available memory in the first 64K is at the start of the TPA. For MS-DOS, there are 54,144 bytes (D380 hex) available. For CP/M, there are 54,272 bytes (D400 hex) available. For both systems there is sufficient memory available in the first 64K bytes for hi-res graphics.

The availability of sufficient memory in the second 64K block depends on the total memory available in a given Victor 9000 configuration. The CP/M and MS-DOS operating systems both reside in the high end of available memory. CP/M (version 2.4) requires 32,480 bytes. MS-DOS, configured as a floppy only system (version 2.6), requires 32,720 bytes, and configured as a hard disk system (version 2.61), requires a minimum of 39,856 bytes. For systems with only 128K of memory, neither operating system leaves enough memory space in the second 64K to accommodate bit mapped graphics. In systems with 256K, or more, of available memory, the entire second 64K block can be utilized for graphics.

System Character Font Location

Since there is a possibility that the absolute address of the system character font table will change with new releases of the operating systems, a programmatic method of determining its location is needed. This can be done simply by first clearing the screen (using the <ESC> E sequence), and then reading the first word of the screen memory (at F000:0000) to get the font cell pointer. The font cell pointer will point the the cell for the ASCII SPACE character (hex 20), so it must be adjusted to point to the start of the table. The following BASIC and assembly programs determine the starting segment address for the system character set.

```
10 DEFINT A-Z
20 DEF SEG = &HF000
30 PRINT CHR$(27);"E";
40 A=PEEK(0)+((PEEK(1) AND &H07) * 256)
50 A=(A*2)-64
```

```
MOV  DL,1BH          ;Clear screen
MOV  AH,2            ;
INT  33              ;
MOV  DL,'E'          ;
MOV  AH,2            ;
INT  33              ;
MOV  AX,0F000H       ;Point to screen ram
MOV  ES,AX           ; using ES
MOV  AX,WORD PTR ES:0;Get font pointer
AND  AX,7FFH         ;Mask out pointer part
SHL  AX,1            ;Make segment address
SUB  AX,64           ;Point to table start
```

Since there are 8K bytes allocated for system character sets, adding 256 to the value obtained above yields the lowest segment address that should be used for graphics memory.

System Character Font Usage

If the system character font is left undisturbed, it can be used for displaying text or special characters in high resolution mode. The dot matrix patterns for characters have the same proportion in high resolution mode as in character mode. It is also possible to use the other special character sets (vertical, proportional, etc.), but these must be loaded separately by the application.

Displaying a character in high resolution mode requires a lot of bit manipulation because the standard character cells are 10 by 16 pixels, and the cells used in high resolution mode are 16 by 16 pixels. This means that it is not possible to display a character just by copying it to the graphics memory. Quite a few calculations must be performed to determine the word and bit offsets for each dot of the character being displayed. In addition, the calculations depend on how the font pointers in the screen memory are ordered (that is, the mapping scheme used).

Graphics Program Placement

The CP/M-86 operating system loads all programs at the highest possible memory location (unless the program was generated to specifically run at a given absolute address). In this case, no consideration concerning placement is needed. Only the size of the program needs to be checked to ensure that it does not encroach on the graphics memory.

Placement under MS-DOS is not so simple. MS-DOS COM files are always loaded at the base TPA address, as are most EXE files. For 128K systems, this can present some problems, since the second 64K is not available for graphics memory use.

For COM files, one solution is to keep the program less than 14,000 bytes long, thus leaving enough space for graphics. Another is to reserve 40,000 bytes near the beginning of the program for graphics memory (this, however, makes the program file on disk 40,000 bytes bigger). Still another solution is to build into the program a small relocation routine that moves the program after it is loaded.

These solutions are valid only for COM programs you are writing, not for existing programs that you may possibly wish to use to do some high resolution graphics programming, like the BASIC interpreter. It would be an advantage to be able to use the interpreter to develop and debug a graphics program (a game, for example), with the intention of compiling it, when finished, to speed its execution.

To enable us to use the BASIC interpreter to do this, BASIC must be made to load at least 40,000 bytes higher than it normally does. A short assembly language program is all that is required to do this. The program consists only of an INT 39 instruction with the DX register containing 40,000. The INT 39 instruction causes an exit to the system that allows the program to remain resident, but telling the system to assign to it 40,000 bytes of memory (determined by the contents of the DX register when the INT 39 is executed). Once this program is executed, all subsequent COM programs will be loaded 40,000 bytes higher than normal, until the system is rebooted. The graphics memory is accessed in BASIC by using the DEF SEG statement along with PEEKs and POKEs. DEF SEG, PEEK, and POKE would also be used to initialize the CRTC and other memory mapped I/O devices for high resolution mode.

The loading area for EXE files can be somewhat determined through the use of the appropriate switches when linking. The linker normally allocates memory starting as low as possible, ordering segments (by Group and Class) as they are encountered in the object file. For assembly language programs, forcing loading into high memory is fairly simple. For compiled BASIC programs, it is trickier.

Compiled BASIC programs that are to be loaded into high memory must be compiled using the /O option. This forces linking with the BASCOM.LIB library so that the BASRUN.EXE runtime module is not required. This restriction is because when the BASIC program is loaded into high memory, there is no room for the BASRUN.EXE file, which must be loaded above the BASIC program. Once the program is compiled, it must be linked using the /H and /S options. The /H option forces loading into high memory. The /S option forces allocation of additional stack space for strings. The value used with the /S option must be empirically determined. I start with a value of 60000 and decrease it until I get no linking errors. The following is an example of compiler and linker commands:

```
BASCOM progname/O;
LINK progname/H/S:60000;
```

CRTC Notes

First, something should be said concerning the aesthetics of graphics programming. I'm sure you have noticed the "jump" that the screen does when switching between 80 column mode and 132 column mode, or when switching in and out of graphics mode when using GRAFIX. This jump is due to the slightly different video synchronization frequencies between the two modes. To me, this jump is annoying. For most applications, there is a simple way around the problem.

All that is necessary to prevent the jump from being seen is to have the display memory blank, or force the intensity to zero, when changing the CRTC parameters. While the jump is still present, it can't be seen if nothing is on the screen. Setting the screen intensity to zero is the easiest solution because is doesn't require the clearing of the 40,000 bytes of graphics memory, which you may wish to leave in tact while switching between modes. I much prefer the screen going blank for a short period to the "jump" of the display.

A programming note concerning the CRTC should also be mentioned. This note can also apply to other memory mapped I/O devices. The CRTC is always programmed by first writing a register file address byte to E800:0000, and then writing the corresponding data byte to E000:0001. Since the 8088 processor does word writes by storing the low address byte first, followed by the high address byte, a CRTC register can be set by doing one write word operation instead of two write byte operations. This saves both program space and execution time. The following example stores the value 5C hex to register 1.

```
      MOV  AX,0E800H       ;Point to I/O space
      MOV  ES,AX           ; using ES
      MOV  WORD PTR ES:0,5C01H; Set register 1
```

It is also possible to use the string instructions of the 8088 to quickly initialize the CRTC, or other devices. The following code sets the CRTC to high resolution mode.

```
      ; assume DS = CS for this example
      MOV  AX,0E800H       ;Point to I/O space
      MOV  ES,AX           ; using ES
      MOV  SI,OFFSET TBL   ;Point to data table
      MOV  CX,TBLLEN       ;Get length of data
      CLD                  ;Increment pointers
LP:   LODSW                ;Get word of data
      MOV  WORD PTR ES:0,AX;Store to CRTC
      LOOP LP              ;Go til done
      etc...


TBL       DW   3A00H       ;Data for register 0
          DW   3201H       ;Data for register 1
          DW   3402H       ;Etc, etc, etc
          DW  0C903H
          DW   1904H
          DW   0605H
          DW   1906H
          DW   1907H
          DW   0308H
          DW   0E09H
          DW   200AH
          DW   0F0BH
          DW   000CH
          DW   000DH
          DW   000EH
          DW   000FH

TBLLEN    EQU  ($ - TBL)/2     ;Table size in words
```

Debugging Graphics Programs

The debugging of graphics programs can present some interesting logistical problems. One such problem occurs when using a debugger such as DEBUG or DDT86. The problem is that if you set a breakpoint in your program, when that breakpoint is hit, the debugger tries to display information about that breakpoint. If you have changed to high resolution mode, what is displayed, if anything, will be garbled. What I have done to get around this problem is to write a short routine that restores character mode, and then essentially does an INT 3 (INT 3 is what both debuggers use for breakpoints). I pick some otherwise unused software interrupt to use in place of INT 3 (INT 254, for example) and put the address of the routine in that interrupt's vector location (0000:03F8 for INT 254). Setting my breakpoint, unfortunately, must be done manually, as must resetting it. It is set by storing CD FE (hex) at the point in the program at which I wish to break. The breakpoint routine is given in Listing 1. It saves all registers, restores character mode, and clears the screen. It then subtracts one from the address offset that was stored on the stack when the interrupt occurred. This is done because INT 3 used by the debugger is a one byte instruction, and all other interrupts are two byte instructions. Adjusting the offset compensates for this difference and corrects the instruction pointer (IP) display. Finally, the routine restores all registers, and does a long jump through location 0000:000C, the vector for INT 3. The routine is completely relocatable. It can be placed in any segment at any offset.

Other methods of performing breakpoints are possible. What is done depends on the specific requirements you have while debugging.

```
; File: brkpnt.asm

; graphics mode debugging breakpoint routine

; assemble as follows:

;       asm brkpnt;                 OR          asm brkpnt;
;       link brkpnt;                            link brkpnt;
;       exe2bin brkpnt brkpnt.com   .           sysloc brkpnt brkpnt.com

; load in debug as follows:

;       Nbrkpnt.com
;       LXXXX:YYYY      (XXXX:YYYY is desired loading location)

; then set desired interrupt vector to XXXX:YYYY
```

Listing 1. Debugging breakpoint.

```
cseg    segment
        assume cs:cseg,ds:cseg

int254: push    bp                      ;save base pointer
        mov     bp,sp                   ;keep copy of stack pointer
        push    es                      ;save rest
        push    ds                      ;
        push    ax                      ;
        push    bx                      ;
        push    cx                      ;
        push    dx                      ;
        push    si                      ;
        push    di                      ;
        mov     dl,1Bh                  ;clear screen
        mov     ah,2                    ;
        int     33                      ;MS-DOS call
        mov     dl,'E'                  ;
        mov     ah,2                    ;
        int     33                      ;
        call    next                    ;this is to get the IP value
next:   pop     dx                      ; for this instruction
        mov     ax,cs                   ;set DS to current segment
        mov     ds,ax                   ;
        mov     ax,0E800h               ;point to I/O space
        mov     es,ax                   ; using ES
        mov     si,offset (tbl - next)          ;offset to table from next
        add     si,dx                   ;adjust for position in segment
        mov     cx,tbllen               ;get length of table
lp:     lodsw                           ;get data
        mov     word ptr es:0,ax        ;send to CRTC
        loop    lp                      ;go til done
        dec     word ptr [bp+2]         ;correct address offset on stack
        mov     bx,offset (exit - next) ;get offset to exit
        add     bx,dx                   ;adjust for position in segment
        xor     ax,ax                   ;point to interrupt tables
        mov     es,ax                   ;
        mov     ax,word ptr es:0Ch      ;get INT 3 offset
        mov     word ptr [bx+1],ax      ;offset of INT 3
        mov     ax,word ptr es:0Eh      ;get INT 3 segment
        mov     word ptr [bx+3],ax      ;segment of INT 3
        pop     di                      ;restore registers
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        pop     ds
        pop     es
        pop     bp
exit    db      0EAh            ;jmp far instruction
        dw      0000h           ;offset for jump
        dw      0000h           ;segment for jump

tbl     dw      5C00H,5001H,5102H,0CF03H,1904H,0605H,1906H,1907H
        dw      0308H,0E09H,000AH,0F0BH,000CH,000DH,000EH,000FH
tbllen  equ     ($ - tbl)/2

cseg    ends
        end     int254
```

Listing 2. Debugging breakpoint, continued.

```
        ORG     0H

; set type (1 char)
        DB      'C'
; format version (1 char)
        DB      '0'
; display class (12 chars)
        DB      'Victor Int'1'
; banner name (8 chars)
        DB      'Int'1   '
; filler (1 char)
        DB      ' '
; banner version (3 chars)
        DB      '01 '
; filler (1 char)
        DB      ' '
; comment (35 chars)
        DB      'Victor International                '
; originator (16 chars)
        DB      'Sirius S/W      '
; date created (8 chars)
        DB      '82/02/20'
; number of records (4 chars per rec)
        DB      '64  '
; filler
        DB      0,0
; character orientation and dimensions
;   Bit  7   = 0 if horizontal, 1 if vertical
;   Bits 4-6 = up/down shift lines for super/sub script
;   Bits 0-5 = height of char in lines minus 1
        DB      7FH
; user/system & stock/special flags
;   Bit 0 = 0 if user, 1 if system
;   Bit 1 = 0 if stock, 1 if special
        DB      0H
; proportional flag & character width
;   Bits 4-7 = 0 if not proportional, F if proportional
;   Bits 0-3 = width of char set (in dots) minus one
        DB      09H

        ORG     80H

; Character 0 (hex)

        DW      0000000000000000B
        DW      0000000000000000B
        DW      0000000000000000B
        DW      0000000000000000B
        DW      0000000000000000B
        DW      0000000000000000B
        DW      0000000000000000B
        DW      0000000000000000B
        DW      0000000000000000B
        DW      0000000000000000B
        DW      0000000000000000B
        DW      0000000000000000B
        DW      0000000000000000B
        DW      0000000000000000B
        DW      1000000000000000B
        DW      0000000000000000B

; Character 1 (hex)

        DW      0000000000000000B
        DW      0000000000000000B
        DW      0000000011111100B
        DW      0000000100000010B
        DW      0000000101001010B
        DW      0000000100000010B
        DW      0000000100110010B
        DW      0000000100000010B
        DW      0000000101111010B
        DW      0000000100110010B
        DW      0000000100000010B
        DW      0000000011111100B
        DW      0000000000000000B
        DW      0000000000000000B
        DW      1000000000000000B
        DW      0000000000000000B

; Character 2 (hex)

        DW      0000000000000000B
        DW      0000000000000000B
        DW      0000000011111100B
        DW      0000000111111110B
        DW      0000000110110110B
        DW      0000000111111110B
        DW      0000000111001110B
        DW      0000000111111110B
        DW      0000000110000110B
        DW      0000000111001110B
        DW      0000000111111110B
        DW      0000000011111100B
        DW      0000000000000000B
        DW      0000000000000000B
        DW      1000000000000000B
        DW      0000000000000000B

; Character 3 (hex)

        DW      0000000000000000B
        DW      0000000000000000B
        DW      0000000000000000B
        DW      0000000111011100B
        DW      0000001111111110B
        DW      0000001111111110B
        DW      0000001111111110B
        DW      0000001111111110B
        DW      0000000111111100B
        DW      0000000011111000B
        DW      0000000000110000B
        DW      0000000000100000B
        DW      0000000000000000B
        DW      0000000000000000B
        DW      1000000000000000B
        DW      0000000000000000B

; The remaining characters are omitted
; due to space limitations
;
; If the char set is proportional,
; then at the end of the fonts, there
; is a table of widths, one for each
; character. Each width occupies 1/2,
; byte, and is the width of the char
; (in dots) minus one.
```

Listing 2. Character Font File