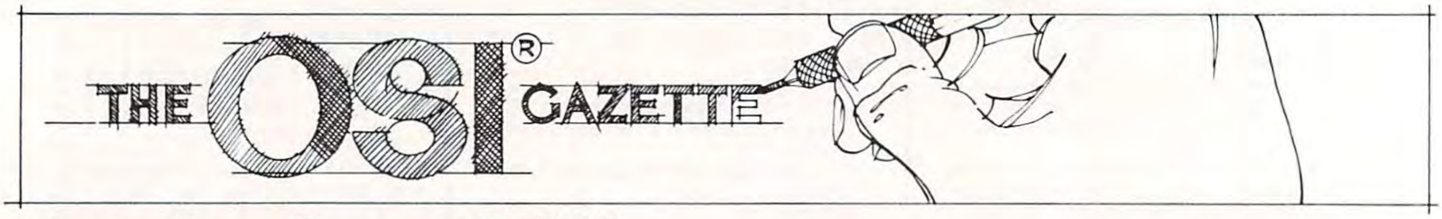

# Exploring OSI's Video Routine

Kerry Lourash
Decatur, IL

Welcome to the BASIC-in-ROM Explorers' Club! On our journey through the Fill-the-Buffer routine, we had to bypass a tour of the Video routine at $BF2D. Now we are ready to unravel the mysteries of the routine that makes objects appear and disappear on the screen.

The Video routine (VR) is a section of machine language code located in BASIC-in-ROM at $BF2D-BFFC. Input from the keyboard and the LOAD routine and output from the SAVE, PRINT, LIST, etc. routines are fed to the VR, which displays the information on the screen.

This is what the VR does:

**1.** Prints text on the screen.

**2.** Does automatic carriage return (CR) and line feed (LF) when the end of the video line is reached.

**3.** Scrolls the screen.

**4.** Slows printing rate, if necessary, for compatibility with printers or other slow ipherals.

## Preparing For Our Journey

The format of our map (see Fig. 1) is the same as that of our first trip (**COMPUTE!** #12, p. 90). I've shown subroutines immediately after the point where they are called, instead of in numerical order. Addresses at the left are part of the main routine and indented addresses are subroutines.

The result approximates an outline of the VR. Machine language addresses have been retained so ML readers can pinpoint and disassemble any part of the routine for more information. BASIC-oriented readers should consider the addresses as line numbers. Most assembly language mnemonics have been replaced by explanations of what is happening. The few mnemonics that are used have their BASIC equivalents listed in the heading of the chart.

*All* numbers are hexadecimal unless specified otherwise.

The VR uses several locations in RAM and ROM:

**0200** - Holds address of the video memory location where current character will be printed.

**0201** - Temporary storage for character to be printed.

**0202** - Storage for A register while A, X, and Y are pushed on the stack. Also holds the number of bytes to be scrolled in the last page of video memory.

**0206** - TV delay loop value.

**0207** - 020E Scroll-one-byte subroutine.

**BFFB** - Holds number of last page of video memory for C1P(D3).

**BFFC** - Holds number of last page of video memory for C2P(D7).

**FFE0** - Cursor "home" position; C1P=65, C2P=3F.

**FFE1** - Characters/line-1; C1P=17, C2P=3F.

**FFE2** - Video memory size; 0=1K, 1=2K.

**D000** - D3FF C1P video memory.

**D000** - D7FF C2P video memory.

Both the Fill-the-Buffer routine and the video routine generate an automatic CR/LF, but the two functions shouldn't be confused. Unlike the FTB, whose "terminal width" counter is in RAM,(loc. 0F), the VR has its character/line permanently set in the monitor (loc. FFE1). If you set the terminal width at less than the char./line value, the FTB will tell the VR to do a CR/LF before the VR does one automatically. However, if you set the terminal width greater than the video line length, the VR will still be triggered at 24 or 64 (decimal) characters, and the video line length will not be longer. You may see a CR/LF at seemingly random intervals. The intervals are not random; both FTB and VR are doing CR/LFs independently of each other. Another difference is that the VR doesn't generate nulls after its CR/LF, as the FTB can. A third difference is that the actual CR/LF subroutines are located in the VR. When you hit the RETURN key or the FTB does a CR/LF, the FTB is sending a CR and a LF character to the VR.

I'd also like to clear up the definition of a few terms, such as "high" and "low" bytes and "pages." The address D365 is a two-byte address. D3 is the high byte and 65 is the low byte. A page contains 256 (dec.) or 0100 (hex) bytes. Notice that the high byte is also the page number (0000-00FF is zero

OSI **AARDVARK** OSI

# NOW MEANS BUSINESS!

**WORD PROCESSING THE EASY WAY— WITH MAXI-PROS**

This is a line-oriented word processor designed for the office that doesn't want to send every new girl out for training in how to type a letter.

It has automatic right and left margin justification and lets you vary the width and margins during printing. It has automatic pagination and automatic page numbering. It will print any text single, double or triple spaced and has text centering commands. It will make any number of multiple copies or chain files together to print an entire disk of data at one time.

MAXI-PROS has both global and line edit capability and the polled keyboard versions contain a corrected keyboard routine that make the OSI keyboard decode as a standard typewriter keyboard.

MAXI-PROS also has sophisticated file capabibilities. It can access a file for names and addresses, stop for inputs, and print form letters. It has file merging capabilities so that it can store and combine paragraphs and pages in any order.

Best of all, it is in BASIC (0S65D 51/4" or 8" disk) so that it can be easily adapted to any printer or printing job and so that it can be sold for a measly price.

MAXI-PROS – $39.95

**NEW-NEW–NEW TINY COMPILER**

The easy way to speed in your programs. The tiny compiler lets you write and debug your program in Basic and then automatically compiles a Machine Code version that runs from 50-150 times faster. The tiny compiler generates relocatable, native, transportable machine code that can be run on any 6502 system.

It does have some limitations. It is memory hungry – 8K is the minimum sized system that can run the Compiler. It also handles only a limited subset of Basic – about 20 keywords including FOR, NEXT, IF THEN, GOSUB, GOTO, RETURN, END, STOP, USR(X), PEEK, POKE, -,=,*,/, ⟨,⟩, ⟨⟩, Variable names A-Z, and Integer Numbers from 0-64K.

TINY COMPILER is written in Basic. It can be modified and augmented by the user. It comes with a 20 page manual.

TINY COMPILER – $19.95 on tape or disk

**THE AARDVARK JOURNAL**

**FOR OSI USERS** – This is a bi-monthly tutorial journal running only articles about OSI systems. Every issue contains programs customized for OSI, tutorials on how to use and modify the system, and reviews of OSI related products. In the last two years we have run articles like these!

1) A tutorial on Machine Code for BASIC programmers.
2) Complete listings of two word processors for BASIC IN ROM machines.
3) Moving the Directory off track 12.
4) Listings for 20 game programs for the OSI.
5) How to write high speed BASIC – and lots more –

Vol. 1 (1980) 6 back issues - $9.00
Vol. 2 (1981) 2 back issues and subscription for 4 additional issues - $9.00.

**ACCOUNTS RECEIVABLE** – This program will handle up to 420 open accounts. It will age accounts, print invoices (including payment reminders) and give account totals. It can add automatic interest charges and warnings on late accounts, and can automatically provide and calculate volume discounts.

24K and 0S65D required, dual disks recommended. Specify system.
Accounts Receivable. $99.95

*** * * SPECIAL DEAL – NO LESS! * * ***

A complete business package for OSI small systems – (C1, C2, C4 or C8). Includes MAXI-PROS, GENERAL LEDGER, INVENTORY, PAYROLL AND ACCOUNTS RECEIVABLE – ALL THE PROGRAMS THE SMALL BUSINESS MAN NEEDS. $299.95

P.S. We're so confident of the quality of these programs that the documentation contains the programmer's home phone number!

**SUPERDISK II**

This disk contains a new BEXEC* that boots up with a numbered directory and which allows creation, deletion and renaming of files without calling other programs. It also contains a slight modification to BASIC to allow 14 character file names.

The disk contains a disk manager that contains a disk packer, a hex/dec calculator and several other utilities.

It also has a full screen editor (in machine code on C2P/C4)) that makes corrections a snap. We'll also toss in renumbering and program search programs – and sell the whole thing for –
SUPERDISK II $29.95 ( 5 1/4") $34.95 (8").

**AND FUN, TOO!**

**BOOKKEEPING THE EASY WAY – WITH BUSINESS I**

Our business package 1 is a set of programs designed for the small businessman who does not have and does not need a full time accountant on his payroll.

This package is built around a **GENERAL LEDGER** program which records all transactions and which provides monthly, quarterly, annual, and year-to-date PROFIT AND LOSS statements. GENERAL LEDGER also provides for cash account balancing, provides a BALANCE SHEET and has modules for DEPRECIATION and LOAN ACCOUNT computation.

GENERAL LEDGER (and MODULES) $129.95.

**PAYROLL** is designed to interface with the GENERAL LEDGER. It will handle annual records on 30 employees with as many as 6 deductions per employee.
PAYROLL - $49.95.

**INVENTORY** is also designed to interface with the general ledger. This one will provide instant information on suppliers, initial cost and current value of your inventory. It also keeps track of the order points and date of last shipment.
INVENTORY - $59.95.

**GAMES FOR ALL SYSTEMS**

**GALAXIAN** - 4K - One of the fastest and finest arcade games ever written for the OSI, this one features rows of hard-hitting evasive dogfighting aliens thirsty for your blood. For those who loved (and tired of) Alien Invaders. Specify system – A bargain at $9.95

**NEW – NEW – NEW**

**LABYRINTH** - 8K - This has a display background similar to MINOS as the action takes place in a realistic maze seen from ground level. This is, however, a real time monster hunt as you track down and shoot mobile monsters on foot. Checking out and testing this one was the most fun I've had in years! – $13.95.

**NIGHT RIDER** - You've seen similar games in the arcades. You see a winding twisting road ahead as you try to make time and stay on the road. NIGHT RIDER uses machine code to generate excellent high speed graphics - by the same author as MINOS.
NIGHT RIDER – $12.95 cassette only

**THIEF** - Another machine code goody for the C1P cassette only. You must use mobile cannon to protect the valuable jewels in the middle of the screen from increasingly nasty and trigger happy thiefs. Fast action and fun for one or two players. THIEF $13.95 on C1 cassette only!

**SUPPORT ROMS FOR BASIC IN ROM MACHINES** – C1S/C2S. This ROM adds line edit functions, software selectable scroll windows, bell support, choice of OSI or standard keyboard routines, two callable screen clears, and software support for 32-64 characters per line video. Has one character command to switch model 2 C1P from 24 to 48 character line. When installed in C2 or C4 (C2S) requires installation of additional chip. C1P requires only a jumper change. – $39.95

C1E/C2E similar to above but with extended machine code monitor. – $59.95

*Please specify system on all orders*

**This is only a partial listing of what we have to offer. We now offer over 100 programs, data sheets, ROMS, and boards for OSI systems. Our $1.00 catalog lists it all and contains free program listings and programming hints to boot.**

VISA master charge

**AARDVARK TECHNICAL SERVICES, LTD.**
**2352 S. Commerce, Walled Lake, MI 48088**
**(313) 669-3110**

OSI OSI

page). The C1P video memory has 4 pages (D000-D3FF) and the C2P has 8 (D000-D7FF).

The cursor stays on the "home" line; its address varies by only 24 or 64 (dec.). Because we know it will always be on the same page, the cursor's location can be specified by only one byte (loc. 0200). I call this byte the cursor offset. The locationn of the cursor is found by adding the cursor offset to D300 (C1P) or D700 (C2P).

For those not familiar with machine language, I suggest you think of the A, X, and Y registers as variables. When a value is loaded into the X register, think x = value.

### On Our Way At Last!

We start with the character to be displayed in the A register. This character may come from any one of several routines such as LOAD, LIST, or FTB. At BF2D, the A register is loaded into location 0202. The A, X, and Y registers are saved on the stack. At BF35, the contents of 0202 are put back into the A register. This seemingly meaningless back-and-forth shift is done because the X and Y registers must be transferred to the A register so that they can be pushed onto the stack. At BF38, the character is checked to see if it is a null (00). If so, the routine branches to BF6D where the Y, X, and A registers are pulled from the stack and restored. Then, at BF72, the VR returns to the routine that called it.

If the character is not a null, location 0206 is examined to see if it is greater than zero. If it is greater than zero, the contents of 0206 are used as a counter for the TV delay loop. This timing loop slows the VR to keep it from printing too fast for slower peripherals. If 0206 contains zero, the timing loop is bypassed.

At BF47, the character is checked to see if it is an LF(0A). If the character is other than an LF, it falls through to BF48. If the character is an LF, we go to BF76. For the time being, let's bypass BF76 and see what happens if the character is not an LF.

At BF48, the character is tested to see if it's a CR(0D). If the character is a CR, the routine falls through to BF4F. Again, let's defer exploration of this route and branch to BF55. This is the route all non-control characters travel.

### Stalking The Non-Control Character

At BF55, the character is stored in location 0201. We JSR (jump to subroutine) to BFC2, where the contents of 0201 are printed D300 (C1P) or D700 (C2P) plus the cursor offset. The cursor offset is stored in 0200, which was initialized with the contents of FFE0 when the BREAK key was pressed at system start-up. The contents of FFE0 are 65 (C1P) or 40 (C2P). This means "home" position in the C1P is D365 and D740 in the C2P. After the character is printed, we RTS to BF5B and increment the cursor offset (loc. 0200).

At BF65, the current cursor offset is compared to the maximum cursor offset. If the end of the video line has been reached, an automatic CR/LF is done (JMP BF73). Otherwise, the routine JSRs to BFDE. At BFDE the character at D300 (C1P) or D700(C2P), plus the cursor offset, is stored in 0201. Remember that location 0200 was incremented at BF5B, so the character stored is the one in front of the current character. As far as I can tell, this character is never reused, except when a CR is done. At BFEF, the cursor character (5F) is printed and an RTS to BF6D is done.

At BF6D, the A, X, and Y registers are pulled from the stack and restored. Then BF72 does an RTS back to the routine that called the VR.

Let's go back and see the path a CR character follows. The CR starts at BF4F with a JSR to BFD5. BFD5 does a JSR to BFC2, which prints the character in 0201, the character "underneath" the cursor. This character is invariably a "space" (20). At BFD4 we RTS to BFD8, where the character in the "home" position is stored in 0201. At BFEF, the cursor character (5F) is printed at the "home" position. Now we RTS to BF52, which JMPs to BF6D. The A, X, and Y registers are restored and we RTS to the VR calling routine.

### Spoor Of The Wily LF

Let's review the status of the TV display at the end of the CR. The cursor character has moved from its former position at the end of the home line to the home position. The character that formerly occupied this position is now stored in 0201.

With this in mind, we track the line feed character through the VR. The LF is usually done immediately after a CR. We left the LF at BF76, which JSRs to BFC2 and prints the contents of 0201 at the home position. This restores the first character of the line and erases the cursor. At BFD4, we RTS to BF79, where the cursor offset is ANDed with the hex number E0. This has the effect of rounding the offset to the start of the video line.

The rounded-off number is stored in 0202. Next, a scroll-one-byte routine is copied from BASIC ROM to RAM at 0207-020E. At BF8C, the X register is loaded with D3(C1P) or D7(C2P). The X register will be used later to determine whether or not the routine is scrolling the last page of video memory. Hex 20 is stored in the A register and the line width is put into the Y register. If the line width is greater than 20, which indicates a 2K memory, the A register is doubled (40). At BF99, the A register is used to set the 0207-E subroutine for a 20 (32 dec.) or a 40 (64 dec.) character line length. The Y register is zeroed in preparation for use as an offset counter for the 0207 subroutine.

At BF9E, the actual scroll is started with a JSR to 0207, which gets one byte from video memory and stores it in the next line above. The Y register is incremented and we RTS to BFA1 and check to see if the current page has been completely scrolled.

If the page is not done, we branch to BF9E to scroll another byte. When the page is done, the 0207 subroutine is set to scroll the next page. A check is made to see if the 0207 subroutine is set to the last page of video memory. If the sub is not set to the last page, we go back to BF9E to scroll another page. If we are on the last page, we scroll down to the home line, using the Y register and location 0202 to tell when to stop scrolling. At BFB6, the home line is cleared by storing "space" characters in its memory locations. We JSR to BFDE, which prints the cursor in the home position. Finally, we RTS to BF6D, pull the A, X, and Y registers from the stack and, at BF72, the VR returns to the routine that called it. Our journey is finally over, and I hope it has been an informative one.

Video Routine (BF2D)

**Figure 1.**

```
  JSR          - GOSUB
  RTS          - RETURN
BRANCH, JUMP - GOTO
INCREMENT    - Add one
/Ø2ØØ/       - Contents of loc. Ø2ØØ
AND          - Logical function
All numbers are in hexadecimal.

BF2D Put /A reg./ (char.) in Ø2Ø2
BF3Ø Save A,X,Y registers on stack
BF35 Put /Ø2Ø2/ (char.) in A reg.
BF38 If char. is null, branch to BF6D.
BF3A Load Y reg. with /Ø2Ø6/ (TV delay)
BF3D If Y is zero, branch to BF47.
BF3F TV delay loop
BF47 If char. is a LF, branch to BF76.
BF4B If char is not CR, branch to BF55.
BF4F JSR to BFD5.

 BFD5 JSR BFC2
  BFC2 Load X with /Ø2ØØ/
       (cursor offset)
  BFC5 Load A with  /Ø2Ø1/
       (char to print)
  BFC8 Load Y with /FFE2/
       (video mem size)
  BFCB If Y is not zero, go to BFD1.
  BFCD Store A is D3ØØ+/X/ (C1P)
  BFDØ RTS
  BFD1 Store a in D7ØØ+/X/ (C2P)
  BFD4 RTS
 BFD8 Load A with /FFED/
      (cursor "home" offset)
 BFDB Put /A/in Ø2ØØ (cursor offset)
 BFDE Put /Ø2ØØ/ is X.
 BFE1 Put char at D3ØØ-/X/ in A.
 BFE4 Put /FFE2/ (video mem size) in Y.
 BFE7 If Y is equal to zero, (1K video
      mem.) goto BFEC.
 BFE9 Load A with char. at D7ØØ+/X/.
 BFEC Put A in Ø2Ø1
      (temporary char. storage)
 BFEF Put cursor char. (5F) in A.
 BFF1 Branch always to BFC8.
 BFC8 Load Y with /FFE2/
      (video mem size)
 BFCB If Y is not zero, go to BFD1.
 BFCD Store A is D3ØØ+/X/ (C1P)
 BFDØ RTS
 BFD1 Store a in D7ØØ+/X/ (C2P)
 BFD4 RTS

BF52 JMP BF6D
BF55 Put char. in Ø2Ø1
     (temporary char. storage)
BF58 JSR BFC2

 BFC2 (See BFC2 subroutine above)
 BFD4 RTS

BF5B Increment /Ø2ØØ/ (cursor offset).
BF5E Put /FFE1/ (chars/line-1) in A.
BF62 Add /FFEØ/ (cursor "home" offset)
     to A.
BF65 If A is greater than /Ø2ØØ/
     (cursor offset) JMP BF73.
BF73 JSR BFD8

 BFD8 (See BFD8 subroutine above)
      Put char. in "home" position into
      Ø2Ø1 and print cursor in
      its place.
 BFD4 RTS

BF76 JSR BFC2

 BFC2 (See BFC2 above)/
      Print char. from Ø2Ø1 at home
      position.
 BFD4 RTS

BF79 Put /FFEØ/ (cursor "home" offset)
     in A.
BF7C AND A with number EØ and put
     result in Ø2Ø2.
BF81 Transfer scroll subroutine from
     BFF3-A to RAM. (Ø2Ø7-B)
BF8C Load X with /BFFE/ (D3) or
     /BFFC/ (D7).
BF8F Put 2Ø (line length) in A.
BF91 Put /FFE1/ (chars.line-1) in Y.
BF94 If Y greater than 2Ø, then /A/.
BF99 Use A to set line length in
     scroll subroutine (Ø2Ø7-E).
BF9C Zero Y register (byte counter).
BF9E JSR Ø2Ø7

 Ø2Ø7 Load one byte from video memory
 Ø2ØA Store byte one line above
      previous location
 Ø2ØD Increment Y.
 Ø2ØE RTS

BFA1 If page is not done, loop to BF9E.
BFA3 Increment high byte of video
     addresses in scroll subroutine.
BFA9 If high byte not equal to D3 (C1P)
     or D7 (C2P) then branch to BF9E.
BFB1 Scroll last page down to
     cursor line.
BFB6 Put "spaces" in home line
     (erase line).
BFCØ Branch always to BF6A.
BF6A JSR BFDE (See BFDE sub above.)
     print cursor at home position,
     store char. in Ø2Ø1.
BF6D Pull A,X,Y from stack.
BF72 RTS (Return from calling routine.)
```
©