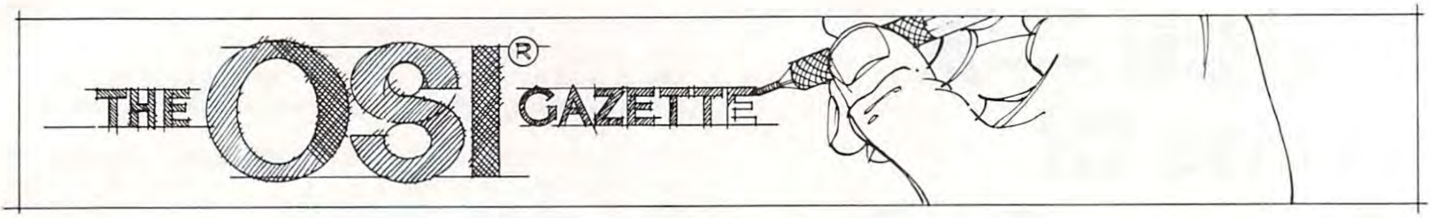


Calling BASIC Commands From Machine Language Routines

William Taylor
Leavittsburg, OH

While working on a tape operating system (TOS) for my OSI C1P and a Stringy Floppy tape drive, many unknown, but desired, features were needed to interface ROM BASIC and the TOS. First, I wanted the TOS to always have command of BASIC's LOAD and SAVE routines. Second, I wanted always to return to the TOS whenever a BASIC program had been loaded into the BASIC workspace. Third, I wanted to go directly from the TOS and RUN a BASIC program that was in the BASIC workspace. In addition, I wished to exit the TOS to the ML Monitor; write a file directory; store the directory on tape; retrieve the directory; and write or load language tapes into the C1P using file marks.

Since the Stringy Floppy tape drives require that all programs stored on tape have file marks or numbers, I needed to free the C1P from ROM BASIC in order to create files on the tape for all programs stored on the tape. The TOS could be written in machine language. The TOS would generate the file numbers under the control of the user, but interfacing the TOS to ROM BASIC was the problem that I faced and pondered for several weeks. How the OSI ROM BASIC and the TOS were interfaced brought several interesting points to light that could be useful in other programming tasks.

Let me summarize. First, calling BASIC commands and executing BASIC programs can be handled from machine language routines. Also, we may LIST, SAVE, LOAD, and exit BASIC to our machine language routines without any USR function call. How these commands can be executed from a machine language routine will become clear with some new knowledge of how BASIC's interpreter works. Let's start with some facts about the BASIC interpreter and how BASIC commands are executed.

Let's look at BASIC's LOAD and SAVE flags and see how they are used to determine if BASIC programs are to be listed to the CRT or to the Cassette port and if the keyboard or the Cassette input port will be the input device.

BASIC's Immediate Mode Commands

BASIC commands are usually executed when input from the keyboard is entered. For example, when you type RUN followed by a carriage return any BASIC program in the workspace will be executed or start to run, starting at the first line of the program. Notice that I said type RUN! This type of command is known as an immediate mode command. If you had typed a number before the RUN command the C1P would have responded with OK. The program would not run but the line of text would have been saved or entered into the program memory. To understand what happens in either the programming mode or the immediate mode we must know how BASIC interprets the code input by the operator. To do this let's look inside BASIC and examine some of what happens during the course of any type of code execution.

At the beginning of system memory is what has become known as zero page. This memory area consists of the first 256 locations of low memory. OSI BASIC uses this area of memory as a scratch pad. OSI BASIC uses page locations $0013 through $005A as what is known as the BASIC Input Line Buffer. What is the Input Line Buffer? This area of low memory is used by BASIC to temporarily store any input code from the user. The code input by the user in the Input Line Buffer will be examined by BASIC to determine what the code's destiny will be. When the user terminates a line of code with a carriage return, the destination of the code input by the operator depends on two factors. First, if the code began with a line number

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 capabilities. 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!

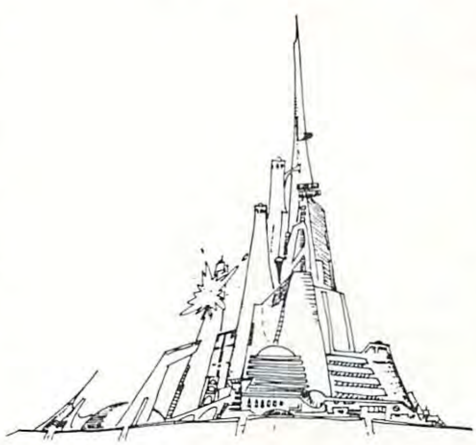
SUPERDISK II

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

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

master charge

OSI OSI

this signals BASIC that the code must be saved as a BASIC program line. Second, if the code in the Input Line Buffer does not start with a numeral, then the code represents a BASIC immediate command or some error that the user made while typing at the keyboard. In either of the latter cases, the code will be immediately executed. If the code was a valid command, the command will be executed. If the input was an error, BASIC will respond with Syntax Error.

To demonstrate and reveal the format of the code placed in the Input Line Buffer, please examine the following example of an input line which will be considered a BASIC line of program text: 10 LIST. On examination of the Input Line Buffer, it would reveal the following code if no carriage return were typed after the line of text. Type in the line of code: 10 LIST. Do not enter a carriage return. BREAK the C1P. Call up Monitor Mode by typing M. Call address mode. Call memory location $0013. You will find that the code listed in the next example will reside at memory locations starting at $0013.

```
0013 31 = ASCII 1
0014 30 = ASCII 0
0015 20 = ASCII space
0016 4C = ASCII L
0017 49 = ASCII I
0018 53 = ASCII S
0019 54 = ASCII T
```

On examination of the code in the Input Line Buffer you will find that all the code will be the hexadecimal ASCII equivalent of the text entered at the keyboard.

The code stored in the Input Line Buffer will have a different appearance if you terminate the line with a carriage return. The code will appear in the Input Line Buffer as in the next example.

```
0013 99
0014 00
0015 20
0016 00
0017 49
0018 53
0019 54
```

Try entering the line 10 LIST (CR). Break the computer. Call $0013 and examine the code in the Input Line Buffer. As you can see BASIC has converted its contents.

Now let's try an Immediate Mode operation and examine the Input Line Buffer. First, clear BASIC workspace. Type NEW (CR). Next type LIST (CR). Break the computer and call Monitor Mode. As before, call $0013 and examine the code stored in the buffer. On examination you should find the following code:

```
0013 99
0014 00
0015 53
0016 00
0017 00
```

This data spells out the LIST command. The byte $99 is a Token for the keyword LIST. What is a Token? It is a single byte that represents a command or keyword. OSI BASIC has a Token for all BASIC keywords. Tokens are used by BASIC in immediate Mode or they are stored in all BASIC programs stored in the BASIC program workspace or BASIC source code table. For the sake of this article let's say that a Token describes to BASIC a keyword. A keyword is an indicator to BASIC as to what function BASIC must perform in the case of $99 (LIST) BASIC is told to LIST all the source code in the BASIC workspace.

The point that we have made with the examples indicates that, for BASIC to know what is expected, the proper code must be in the Input Line Buffer starting at $0013. We can use the facts just presented to make BASIC think an operator has entered an Immediate Mode command, but the command can be initiated from a machine language routine as you will see. We are not ready yet to use our new knowledge about the Input Line Buffer and Tokens as commands called from machine language routines. First we must learn some more facts about BASIC.

How does BASIC execute the code for commands in the Input Line Buffer? The code must be read by the BASIC Interpreter. On examination of a Zero page memory map, you will find a machine language routine which starts at $ 00BC. This routine is called a "PARSER." It is used to read a line of code, character by character, stored in the line buffer or code stored in a program line in the BASIC workspace. The Parser routine at $00BC looks at the first character of code in the buffer to see if the character is an ASCII numeral or not. If the first character were a numeral, the Parser tests each character until a non-numeral is found. If the first character is a numeral, the line of code in the buffer is recognized as a line of source code and will be stored in the source code table. When the Parser detects a non-numeral, the Parser routine hands the code to a routine that "Tokenizes" the line before the line is placed in the source code table or back into the input line buffer. If the first character in the buffer is a non-numeral, the parser determines that the input code must be an immediate mode command. If you recall the earlier examples, we demonstrated the keyword LIST entered as a program source line. First we examined the buffer without a carriage return. It was evident that the code was ASCII. Next, we entered a line of text ending with a carriage return and examined the data in the buffer. At this point, we found that the data was in a Tokenized form. As you can see, the BASIC interpreter had, in fact, converted the ASCII to a condensed (or Tokenized) line of code.

To understand how the parser routine interprets the source code (or the code in the Input

Line Buffer) please refer to Listing 1. The machine language parser routine shown in Listing 1 shows that memory locations $00C2, 00C3, and 00C4 contain an LDA direct instruction or AD 13 00. This instruction causes the 6502 accumulator to be loaded with the code at the first address of the Input Line Buffer. On initialization, (BASIC Cold Start) address $00C2, 00C3, and 00C4 will point to $0013 (the beginning of the Input Line Buffer). If you type RUN in Immediate Mode without a program in the BASIC workspace, address $00C2, 00C3, and 00C4 will contain AD 00 03. As you can see, the Parser now points to the beginning of the BASIC workspace.

At this point, enough knowledge about the Input Line Buffer and the parser routine has been presented to allow us to explore the possibility of implementing and executing BASIC Immediate Mode commands called from outside ROM BASIC using machine language routines.

Let us now experiment with the Input Line Buffer and the parser routine to see if we can actually call a BASIC Immediate Mode command from a machine language program. As I mentioned at the beginning of this article, I needed to call BASIC's LOAD and SAVE commands. Let's begin with these. First, let's try the SAVE command to demonstrate how it can be called from a machine language routine.

To use the SAVE command we must learn yet more facts about how BASIC functions. When the user wishes to save a program that is stored in the BASIC workspace, the SAVE command must be used. What happens when you type SAVE? When the command, SAVE, is entered at the keyboard and ended with a carriage return, the code will, of course, be placed in the Input Line Buffer as ASCII. When the carriage return is entered, BASIC examines the code and recognizes that this is an Immediate Mode command. The code in the Input Line Buffer will be Tokenized and placed back in the Input Line Buffer. The Input Line Buffer would not contain:

$0013 94 = TOKEN FOR SAVE
$0014 00 = NULL
$0015 53
$0016 00 = NULL

Now, on examination of the Parser routine at address $00C2, 00C3, and 00C4, you will find that the Parser has read the code located at address $0013 and found a Token for the keyword SAVE, and that BASIC has executed the comand. When the SAVE command was executed, BASIC performed the task of setting what is called the SAVE flag. This flag tells the computer that any data sent from BASIC will be sent to the cassette port and to the screen. The SAVE flag is located at $0205. If the contents of $0205 are set to $00, then output from BASIC will be listed to the screen.

If the SAVE flag contains $01 then the cassette port along with the screen will be activated.

We may use these facts to call the BASIC SAVE command from a machine language routine. Let me demonstrate with an example. Enter the machine language routine (Listing 2) into the computer. Now write a BASIC program into the computer. This program can be any program that you may have on hand, but a single program line will do for the demonstration. Exit BASIC and call the address of the machine language routine of Listing 2. Run the machine language routine. As you can see, the BASIC program that you entered into the computer was LISTed out to the screen of your monitor. Also, the program will be sent to the cassette port.

On examination of the Assembly Listing, notice that we have loaded the Input Line Buffer at $0013 with the Token for LIST ($94). Also notice that, in the Listing, we are setting the SAVE flag at $0205 to the the value of $01. We have set address $00C3 and $00C4 in the Parser routine to point to the beginning of the input line buffer. Finally, we call a routine in the BASIC interpreter located at $A4B5. This routine is called the LIST routine and will execute the LIST command when called by a BASIC program, Immediate Mode, or by a machine language calling routine. As you can see, we have programmed a SAVE and a LIST

Program 1.

```
 10 0000           ;
 20 0000           ;
 25 0000           ;
 30 0000           ;
 40 0000           ;      PARSER CODE
 50 00BC           *=$BC
 60 00BC E6C3      S0 INC $C3   INCREMENT LOW ADDR. BYTE
 70 00BE D002      BNE S1
 80 00C0 E6C4      INC $C4   INCREMENT HIGH ADDR. BYTE
 90 00C2 ADFFFF    S1 LDA $FFFF   LOAD WITH CODE CHARACTER
100 00C5 C93A      CMP #': CHECK FOR COLON (STATEMENT END)
110 00C7 B00A      BCS S2  IF YES BRANCH TO START NEW LINE
120 00C9 C920      CMP #'  ISIT A SPACE
130 00CB F0EF      BEQ S0  IF YES GET NEW CHARACTER
140 00CD 38        SEC     SET CARRY FLAG
150 00CE E930      SBC #$30  SUBTRACT $30
160 00D0 38        SEC     SET CARRY FLAG
170 00D1 E9D0      SBC #$D0  SET C FLAG FOR ASCII NUMBERS
180 00D3 60        S2 RTS   END ROUTINE.   CHARACTER NOW IN A
```

Program 2.

```
 10 0000           ;
 20 0000           ;
 30 0000           ;
 40 0000           ;
 50 0000           ;  BASIC SAVE COMMAND CALL
 60 0000           ;
 70 0000           ;
 80 0000           ;
 90 0000           ;
100 1000           *=$1000
110 1000 A901      START LDA #$01  VALUE SAVE FLAG=ON
120 1002 8D0502    STA $0205     ST0RE IN SAVE FLAG
130 1005 A999      LDA #$99     TOKEN LIST
140 1007 8513      STA $13     PUT IN LINE BUFFER
150 1009 A900      LDA #$00     NULL
160 100B 8514      STA $14     PUT BUFFER+1
170 100D 8516      STA $16     PUT BUFFER +3
180 100F A953      LDA #$53
190 1011 8515      STA $15     PUT BUFFER+2
200 1013 A914      LDA #$14     PARSER SCAN START LOW BYTE
210 1015 85C3      STA $C3     PUT IN PARSER
230 1017 A900      LDA #$00     PARSER SCAN HIGH BYTE
240 1019 85C4      STA $C4     PUT IN PARSER
250 101B 4CB5A4    JMP $A4B5    GOTO BASIC LIST ROUTINE
```

command into BASIC from outside ROM BASIC and caused its execution.

In a similar manner, let's call and execute a LOAD command from a machine language routine. Enter Listing 3 into the computer. Next bring up BASIC in Warm Start. (Type NEW (CR).) Exit BASIC. Call up the machine language routine for the LOAD command. Place a BASIC program tape into your cassette recorder, execute the machine language routine, and start your recorder on play. Your BASIC program will load into the computer as if called directly under BASIC.

On examination of Listing 3, you will find that the implementation of the LOAD command was very simple. We only need to set the LOAD flag to turn the system on for a BASIC load and jump to the Warm Start of BASIC.

Listing 4 will be used to implement the BASIC RUN command from a machine language program. As before, enter the machine language program into memory and then load a BASIC program into the BASIC workspace. Exit BASIC and call the machine language routine. Start the machine language program. The computer will jump to the BASIC program and run.

On close examination of Listing 4, you will see that we have used the same procedure to force a BASIC RUN command that we used in the SAVE and LOAD routines. We loaded the input line buffer with the Token for RUN, set the Parser scanner to start reading the code in the Input Line Buffer at $0013. With the RUN command it was found that two BASIC interpreter routines were needed to force the computer to execute the RUN command. These were the conversion routine at $A3A6 and the execution routine located at $A5F6.

At the beginning of this article, I said that an executive TOS could be written in machine language that could call BASIC commands. Also, it was mentioned that in order for the TOS to be truly an executive, we must devise some means of exiting BASIC and returning to the TOS. I have shown how BASIC commands could be executed from machine language routines. But, how do we exit BASIC to our machine language routines? At first, it appears that ROM BASIC can only be exited with a BREAK or through a USR function call. This is true unless we can devise some means of patching into BASIC at some point and make BASIC think there is some new form of keyword present in the interpreter. Well, implementing new Keywords is not possible with ROM BASIC, so some other method must be devised.

An article which appeared in *Micro* described interception of BASIC Syntax error codes when printed on the monitor screen. A patch devised to intercept a Syntax error can be utilized to direct an exit from BASIC and force a return to a calling machine language program. The machine language patch routine shown in Listing 5 can be used to force an exit from BASIC during a running BASIC program, and in Immediate Mode or when a BASIC program has finished loading from cassette into the BASIC workspace. Listing 5 is a routine that has been revised for the purpose of exiting BASIC. The routine appeared in an article titled "Stop Those S' Errors" published in the November 1980 issue of Micro Magazine (*Micro*, 30:37).

The patch code for the BASIC exit routine utilizes a vector location in zero page. The vector is located at $03 and $04. Normally, this vector points to the string output routine of the BASIC interpreter at $A8C3. If we replace this jump with a call to our patch routine, we may use the pointer and our patch routine to exit BASIC on command. Listing 5, shows the Exit patch routine that is loaded into memory starting at $0240. To use the patch routine, replace the jump at $03 and $04 with the start of the exit patch routine. That is, load $40 into memory location $03 and $02 into location $04. This can be done in BASIC using the POKE command: POKE 3, 64 : POKE 4, 2. Once the address for the patch code has been loaded into the pointer at $03 and $04 the pointer will not have to be changed unless the computer has been reset.

Program 3.

```
 10 0000             ;
 20 0000             ;
 30 0000             ;
 40 0000             ;
 50 0000             ;          BASIC LOAD COMMAND CALL
 60 0000             ;
 70 0000             ;
 80 0000             ;
 90 0000             ;
100 1100             *=$1100
110 1100 A9FF        START LDA #$FF    VALUE LOAD FLAG =ON
120 1102 8D0302      STA $0203      PUT IN LOAD FLAG
130 1105 4C74A2      JMP $A274      GOTO BASIC WARM START
```

Program 4.

```
 10 0000             ;
 20 0000             ;
 30 0000             ;
 40 0000             ;
 50 0000             ;          ** BASIC RUN COMMAND CALL **
 60 0000             ;
 70 0000             ;
 80 1150             *=$1150
 90 1150             ;
100 1150 A952        LDA #$52         GET RUN TOKEN
110 1152 8513        STA $13         PUT IN LINE BUFFER
120 1154 A900        LDA #$00         NULL
130 1156 8514        STA $14         PUT BUFF+1
140 1158 8516        STA $16         PUT BUFF+3
150 115A 85C4        STA $C4         PUT PARSER HIGH BYTE
160 115C A94E        LDA #$4E
170 115E 8515        STA $15         PUT BUFFER+2
190 1160 A913        LDA #$13         GET PARSER START LOW
200 1162 85C3        STA $C3         PUT PARSER LOW
210 1164 20A6A3      JSR $A3A6         GO BASIC CONVERSION RTN.
220 1167 4CF6A5      JMP $A5F6         GO TO BASIC EXECUTION (RUN)
```

Program 5.

```
 10 0000             ;
 20 0000             ;
 30 0000             ;
 40 0000             ;
 50 0000             ;
 60 0000             ;          BASIC EXIT PATCH ROUTINE
 70 0000             ;
 80 0000             ;
 90 00F0                     *=0240
100 00F0             ;
110 00F0 48          PHA        SAVE PRT CHRACTER IN ACC.
120 00F1 AD65D3      LDA $D365       GET CHRACTER FROM SCREEN
130 00F4 C93F        CMP #$3F       TEST FOR ERROR(?)
140 00F6 D008        BNE OUT       NO NOT ERROR GO PRINT CHR.
150 00F8 A900        LDA #$00       YES ERROR GET READY TO EXIT
160 00FA 8D0302      STA $0203       RESET LOAD FLAG
170 00FD 4CFFFF      JMP $FFFF       RETURN TO CALLER($FFFF DUMMY
180 0100 68          OUT PLA       ADDRERESTORE CHARACTER TO ACC
190 0101 4CC3A8      JMP $A8C3       GO PRINT CHR. RETURN TO BASIC
```

The patch routine at $0240 tests memory location $D365 for a question mark ($3F) for each character printed out to the monitor screen. In the event of an error, such as ? Sn Error, the question mark will be loaded into video RAM at $D365. The routine tests $D365 for $3F. If there should be any type of error, the question mark code will appear at $D365. On detection of the error code, the patch routine will cause an exit to your machine language routine. Under normal program execution, the data to be printed is passed to the string printing routine of BASIC as if the patch routine did not exist.

The exit patch code routine was implemented into my TOS to detect an error at the end of a program loading from tape. My Stringy Floppy tape unit sends $8F when all the program on tape has been sent to the C1P. This hex byte, when seen by BASIC, will send back a Syntax error which will be detected by the patch routine causing an automatic exit to the TOS. While in BASIC, if the user types any key followed by a carriage return. It will cause a Syntax error and force a return to any calling routine. In addition, programming a line of illegal code at the exit point of the BASIC program will force a return to the calling machine language routine. An example line of illegal code could be: 10/ or 10 EXIT etc...

This article has presented some ways of implementing BASIC commands and calling these commands from machine language programs. Through these efforts, I have further expanded the ways in which we may use OSI BASIC and machine language programs as a means of system development. In my case, I have a TOS that functions like a disk operating system (DOS). With the information presented in this article, you may also be inspired to develop new programming techniques. Although this article was developed around OSI 6502 BASIC, the concepts should apply to other systems using similar BASIC such as, PET, and APPLE. Of course, tokens and interpreter routine addresses may need changing but the basic principles still apply.

References:

OSI BASIC In ROM, Edward H. Carlson

"Stop Those S' Errors," *Micro* Magazine, November 1980. ©