A Quick Screen-Clear for the Challenger II

Prevent development of a microcomputer inferiority complex.

Richard Lary
PO Box 234
Kilauea HI 96754

I don't know about you, but I don't like to see my OSI Challenger II outdone by other computer systems. For example, several systems have available a BASIC command that clears the video display in the blink of an eye. OSI owners, who are not so fortunate, must clear the display using a loop containing a PRINT statement. This method works but is rather slow. There is a faster and better method.

Surely you have noticed how quickly the screen is cleared when you press the BREAK key. The code that accomplishes this clearing of the screen is programmed in the system's monitor ROM; unfortunately, the program in ROM is not written as a subroutine and cannot be called by a BASIC program. I have borrowed this code from the monitor and added a return-from-subroutine instruction to the end of the code. By placing this code and the added RTS instruction in RAM, you have a clear-screen subroutine that can be called by a BASIC program.

Program Location

Program 1 shows the assembly listing of this subroutine. Since the program uses relative branching and zero-page addressing, the program can be located anywhere, unchanged in RAM. As shown, Program 1 is for Challenger II systems with the 540 video board. Changing the value of the last-page index from D8 to D4 will allow Challenger I owners to use the program.

Since the clear-screen subroutine will be located in memory along with the BASIC program that calls the subroutine, the memory size allowed for BASIC must be set lower than the starting address of the subroutine. This prevents BASIC from overwriting the subroutine. Although you can place the subroutine anywhere in memory, the best location is in the last 25 bytes of available memory. This way, the remaining lower bytes will be available to BASIC.

```
                        SCNADR = $FE
                        LASTPG = $D8
                        SPACE  = $20

OFE6  A2 D8   BEGIN     LDX #LASTPG     LOAD X REG W/ LAST PAGE HI BYTE
OFE8  A9 DO             LDA #$DO        LOAD ACCUM W/ SCREEN ADDR HI BYTE
OFEA  85 FF             STA SCNADR+1    SAVE ACCUM
OFEC  A9 00             LDA #$00        LOAD ACCUM W/ SCREEN ADDR LO BYTE
OFEE  85 FE             STA SCNADR      SAVE ACCUM
OFFO  A8                TAY             LOAD Y REG W/$00
OFF1  A9 20             LDA #SPACE      LOAD ACCUM W/ ASCII SPACE
OFF3  91 FE   LOOP      STA (SCNADR),Y  SAVE AT SCREEN ADDR LO BYTE + Y REG
OFF5  C8                INY
OFF6  DO FB             BNE LOOP        DONE W/ PAGE? NO,DO NEXT LOCATION
OFF8  E6 FF             INC SCNADR+1    YES,SCREEN ADDR HI BYTE = HI BYTE+1
OFFA  E4 FF             CPX SCNADR+1    SCREEN ADDR HI BYTE = LAST PAGE?
OFFC  DO F5             BNE LOOP        NO, DO NEXT PAGE
OFFE  60                RTS             YES,DONE! RETURN TO BASIC
```

Program 1. Clear-screen subroutine

Memory Size

Once you have decided on the starting address, you can use either of the following methods to set the memory size. One method is to cold-start the system. Before loading the BASIC program, enter the starting address −1 (in decimal) in response to the monitor's prompt: MEMORY SIZE? This method has a disadvantage in that it requires you to remember to set the memory size before you load the program.

A better method is to let your BASIC program set the memory size for you. There are two zero-page memory locations that store the address of the last memory location available to BASIC; the low- and high-order bytes of the last address are stored in decimal locations 133 and 134, respectively. You can use BASIC to POKE the starting address −1 into these two memory locations.

For example, in a 4K system the last address in RAM would be 4095 decimal; the starting address for the subroutine would be 4070 decimal. The memory size for BASIC would be the starting address −1, or 4069 decimal. You then need to convert this decimal value into its hex value, which in this case is = 0F E5.

In order to use POKE statements, you have to convert the high- and low-order bytes of this hex value into their respective decimal values. High-order byte = 0F = 15 decimal and low-order byte = E5 = 229 decimal. Now that you have the decimal values for the high and low-order bytes of the last address available to BASIC, you can use the following POKE statements to set the memory size to 4069: POKE 133,229 and POKE 134,15.

Placing these POKE statements at the beginning of a BASIC program allows the program to set the memory size. Should you ever need to set the memory size after loading a program, you can use these two POKE statements to set the memory size without having to cold-start the system.

The Program

The BASIC program listed in Program 2 contains the code required to set the memory size, load the subroutine and call the subroutine. Line 30 sets the memory size as described and includes the RUN command, which directs program execution to begin with the line number following the line containing the RUN command. If you leave out the line number, the program will get tied up in an endless loop. Also, be sure that any variable assignments are made in lines after the line containing the RUN command; otherwise, the RUN command will set all previously assigned variables to zero. The DATA statements in lines 60–80 contain the decimal values for the machine-language instructions from the assembly listing shown in Program 1. Challenger I owners will have to change 216 in line 60 to 212 in order to run the program. The variable SA in line 100 must be set equal to the subroutine's starting address in decimal. Line 120 READs the machine-

language instructions from the DATA statements and POKEs the instructions into memory.

With the exception of the REM statements, the statements in lines 10-120 would normally be placed at the beginning of the BASIC program, which will be calling the clear-screen subroutine.

To call a machine-language subroutine from BASIC, BASIC requires that the starting address of the subroutine be stored in the USR vector. The USR vector is simply two zero-page memory locations, 11 and 12 decimal, which tell BASIC where the machine-language subroutine starts.

Line 160 sets up the USR vector by POKEing memory locations 11 and 12 decimal with the decimal values of the low-order and high-order bytes of the subroutine-starting address. Line 170 causes BASIC to execute the machine-language subroutine pointed to by the USR vector.

If the clear-screen subroutine is the only machine-language subroutine that your BASIC program will be calling, then you can set up the USR vector once at the beginning of the program and use the statement in line 170 anywhere in your program to call and execute the clear-screen subroutine. If your program will be calling more than one machine-language subroutine, you will have to reset the USR vector each time BASIC calls a machine-language subroutine, in order to call the right subroutine.

```
10  REM--MACHINE LANGUAGE CLEAR SCREEN SUBROUTINE
20  REM--SET MEMORY SIZE TO 4069
30  POKE 133,229 : POKE 134,15:RUN 40
40  REM--DATA STATEMENTS CONTAIN MACHINE LANGUAGE
50  REM--SUBROUTINE INSTRUCTIONS IN DECIMAL
60  DATA 162,216,169,208,133,255,169,0,133,254
70  DATA 168,169,32,145,254,200,208,251,230,255
80  DATA 228,255,208,245,96
90  REM--SET VARIABLE 'SA' EQUAL TO SUBROUTINE STARTING ADDRESS
100 SA = 4070 : REM--4070 = OFE6 HEX
110 REM--READ INSTRUCTIONS AND POKE INTO MEMORY
120 FOR LC = SA TO SA + 24 : READ D : POKE LC,D : NEXT LC
130 REM--SUBROUTINE NOW LOADED AND CALLABLE FROM BASIC
140 REM--BASIC CODE TO CALL AND EXECUTE SUBROUTINE
150 REM--SET UP USR VECTOR, 11 = SA LOW BYTE; 12 = SA HIGH BYTE
160 POKE 11,230 : POKE 12,15
170 X = USR(X) : REM--CALL SUBROUTINE
180 END
```

Program 2. BASIC program.

So, OSI owners, by using the statements from Program 2 in your BASIC programs, you, too, can now clear your screen in the blink of an eye. ■