Hidden 64 Memory

Alan R. and Julie R. Krauss

BASIC programmers can POKE data into the Commodore 64's hidden RAM, but retrieving that data requires switching between blocks of RAM and ROM. The machine language program given here makes it easy to do what BASIC can't do directly—giving you an extra 20K.

The Commodore 64 contains 24 kilobytes of Random Access Memory (RAM) which cannot immediately be used by BASIC. However, that memory is accessible to the VIC-II chip, so it seems an ideal place to store a high-resolution (bit-mapped) screen (as well as other large arrays of data). The catch is that although it is possible to move data into this area using POKE statements, you can't retrieve all of it directly using BASIC. In this article we describe a technique which makes most of this large block of memory available to the BASIC programmer.

How To Get Five Quarts To A Gallon

The microprocessor in most smaller computers, including the Timex 1000, IBM PC, and DEC PDP-11, has 16 or fewer memory address lines. So these computers can address no more than 2^{16}—64K—bytes of memory directly. The more expensive machines *appear* to have a larger addressable memory because their memory-management circuitry and operating systems allow them to switch blocks of memory into and out of their actual address space. The inexpensive Commodore 64 has no special Memory Management Unit, yet it is able to address 20K of Read Only Memory (ROM) and numerous I/O chips plus 64K of RAM. This is like filling a one-gallon pitcher with five quarts of water. It works because the microprocessor can switch between various blocks of ROM and RAM even when they have the same addresses.

In its normal configuration, the first 2K of the 64's memory is used as a work area for the Operating System, and for screen memory. Of the remaining RAM, locations 2048 through 40959 are the programmer's BASIC area. The space above 40959 contains 4K of RAM (addresses 49152–53247) which is not contiguous with BASIC's dedicated area and can be accessed by BASIC only via PEEKs and POKEs; by the ROM BASIC interpreter (40960–49151); by the Kernal Operating System (57344–65535); and by the Input-Output (I/O) circuitry (53248–57343). However, there is another 20K of RAM which is similarly addressed; to be used, it must be switched in and out of the ROM-masked space. This chore is handled by registers at locations 0 and 1.

What The Pointers Mean

Although the extra RAM isn't directly available to BASIC, data may be stored there by using the POKE instruction. However, a PEEK of one of these locations will return the value stored at that address in ROM. In order to have access to the corresponding RAM, it is necessary to set a pointer so that the processor will ignore the ROM. Bits 0, 1, and 2 of location 1 are the pointers. Their functions are as follows:

Bit	Value	Meaning
0	1	Indicates normal BASIC ROM
	0	Indicates noncontiguous RAM (addresses 40960–49151)
1	1	Indicates Kernal ROM
	0	Indicates underlying RAM (addresses 57344–65535)
2	1	Indicates I/O chips
	0	Indicates ROM character tables (addresses 53248–57343)

If we wished to save a variable—call it A—at, say, address 45000, and later retrieve it, we might envision a routine like this:

```
10 POKE 45000,A :REM STORE THE VALUE
20 POKE 1,54 :REM SET POINTER TO RAM
30 A=PEEK(45000):REM GET THE VALUE
40 POKE 1,55 :REM RESTORE POINTER
50 END
```

This, though, amounts to sawing off the branch we're sitting on. The result is a lovely crash.

It's easy to see why. Line 20 tells the processor to look not at the BASIC interpreter in ROM, but somewhere else instead. So when line 30 calls for the BASIC PEEK instruction, it can't be found, and the system hangs. If we change the pointer for Kernal ROM or for I/O, we also crash.

Machine Language Makes It Easy

But the extra memory is too tantalizing to pass up. Since we can't get at it properly from BASIC, we'll try machine language (ML). We'll use a BASIC loader for the routine, as in Program 1.

It works! We can now store data in the formerly unavailable area of masked RAM. The ML routine, which is only 14 bytes long, sets the appropriate pointer (bit 0 in this example) to ignore the ROM (here, the BASIC interpreter). It then puts the byte of data into a location normally accessible to BASIC (since location 251 is in unused zero page space, we chose that) and resets the pointer. A disassembly of the machine language for this routine is shown in Program 2.

Let's take a look at Program 1. Lines 100–120, 340, and 350 are not really part of our routine—lines 100 and 110 give us some data to use in illustration, and line 340 prints the data that the ML routine saved for us, just to prove it really worked. The 4K of RAM beginning at location 49152 is unused, and since it is not contiguous with the BASIC area, it can't be overwritten by BASIC; so we've chosen to put our ML there. Line 130 sets location 49152 as the starting location for the machine language routine.

Lines 140, 160, 170, and 180 determine what value location 1 should contain, and put that value in BL. The numbers in the DATA statements (lines 190 and 200) are the bytes of ML (in decimal). Three of these are 0. The first (shown as 00 for prominence) will hold the block pointer, BL. The second and third, 000 and 0000, will hold the low-order and the high-order bytes, respectively, of the address of the target masked-RAM location.

Lines 210–240 POKE the ML into place. Line 250 inserts the value of the block pointer into byte 2 of the ML routine. Lines 260 and 270 calculate the high- and low-order bytes, respectively, of the target address; lines 280 and 290 POKE them into place. Line 300 disables interrupts so that the keyboard will not be scanned during execution of the machine language routine; this obviates the possibility of the system's hanging should the scan interrupt occur at the wrong moment.

The innocuous-looking line 310, the system call, signals the real action—here is where we branch out to perform our machine language routine. When the subroutine is finished, it returns control to BASIC. Now we can get our data whenever we need it: It has been left in location 251 for us.

Note that the ML could reside virtually anywhere—even in masked RAM. If it is placed within the normal BASIC area, of course, the appropriate BASIC pointers should be altered to protect it. From line 140 on, the routine is perfectly general and may be used to read the value stored at any RAM address within the range 0–65535, except for that lying beneath the I/O area (53248–57343). To see the underlying 4K of RAM in this area would require another technique, since there are three layers of memory here; our routine uncovers the second layer and lets us look at Character ROM directly. This could be useful in programs using custom-character routines, in order to restore portions of the ROM character table selectively.

Finally, we must note two things. First, this routine may be used to read memory locations in which either the BASIC program or the ML routine resides. However, we must not permit a POKE instruction (for example, line 120) to alter the program unless we specifically wish to do so. Also, if we POKE to a location in the I/O area, we may drastically alter our output.

Program 1: ML Access To Hidden RAM

```
100 A=3{15 SPACES}:REM{2 SPACES}PUT DESIR
    ED DATA BYTE IN VARIABLE "A"
110 AD=45000{10 SPACES}:REM{2 SPACES}WE'L
    L SAVE "A" AT LOC. 45000 (IN MASKED R
    AM)
120 POKE AD,A{9 SPACES}:REM{2 SPACES}SAVE
     "A"
130 MS=49152{10 SPACES}:REM{2 SPACES}MACH
    INE CODE WILL BE LOADED STARTING AT L
    OC. 49152
140 IF 40959<AD AND AD<49152 THEN BL=54:
    {SPACE}GO TO 190
145 REM{21 SPACES}LOCATION 1 WILL CONTAIN
     BLOCK POINTER, BL
150 REM{21 SPACES}BL = 54 -- BASIC INTERP
    RETER ROM OUT
160 IF 53247<AD AND AD<57344 THEN BL=51:
    {SPACE}GO TO 190
165 REM{21 SPACES}BL = 51 -- I/O ROUTINES
     OUT
170 IF 57343<AD AND AD<=65535 THEN BL=53:
     GO TO 190
175 REM{21 SPACES}BL = 53 -- KERNAL ROM O
    UT
180 BL = 55{11 SPACES}:REM{2 SPACES}WITHI
    N NORMAL BASIC AREA
190 DATA 162,00,134,1 :REM{2 SPACES}MACHI
    NE LANGUAGE ROUTINE
200 DATA 174,000,0000,134,251,162,55,134,
    1,96
210 FOR I=0 TO 13{5 SPACES}:REM{2 SPACES}
    LOOP FOR BASIC LOADER
```

```
220 READ ML{11 SPACES}:REM{2 SPACES}GET N
    EXT BYTE OF M. L. ROUTINE
230 POKE (MS+I),ML{4 SPACES}:REM
    {2 SPACES}PUT M. L. BYTE INTO PLACE
240 NEXT
250 POKE(MS+1),BL{5 SPACES}:REM{2 SPACES}
    STORE BLOCK POINTER IN 2ND BYTE OF M.
     L. ROUTINE
260 HA=INT(AD/256){4 SPACES}:REM
    {2 SPACES}HIGH-ORDER BYTE OF MASKED-R
    AM ADDRESS
270 LA=AD-256*HA{6 SPACES}:REM{2 SPACES}L
    OW-ORDER{3 SPACES}"{3 SPACES}"
    {4 SPACES}"{5 SPACES}"{4 SPACES}"
280 POKE MS+5,LA{6 SPACES}:REM{2 SPACES}P
    UT ADDRESSES INTO M. L. ROUTINE
290 POKE MS+6,HA
300 POKE 56333,127{4 SPACES}:REM
    {2 SPACES}DISABLE INTERRUPTS
310 SYS(MS){11 SPACES}:REM{2 SPACES}EXECU
    TE M. L. ROUTINE
320 POKE 56333,129{4 SPACES}:REM
    {2 SPACES}RE-ENABLE INTERRUPTS
330 A=PEEK(251){7 SPACES}:REM{2 SPACES}RE
    AD THE DATA BYTE
340 PRINT A
350 END
```

Program 2: Disassembly Of ML Routine

```
LDX #$00   ;BLOCK POINTER TO X REGISTER
STX  $01   ;STORE IN LOCATION 1
LDX  $0000;CONTENTS OF MASKED RAM TO X-REGISTER
STX  $FB   ;STORE IN LOCATION 251 (DECIMAL)
LDX #$37   ;NORMAL VALUE OF POINTER (55 DECIMAL)
STX  $01   ;RESTORE NORMAL VALUE TO LOCATION 1
RTS        ;RETURN TO BASIC
```

reset

```
.
.
.,3102 78       SEI
.,3103 A9 EA    LDA #$EA
.,3105 8D 15 03 STA $0315
.,3108 A9 31    LDA #$31
.,310A 8D 14 03 STA $0314
.,310D A9 00    LDA #$00
.,310F 8D 1A D0 STA $D01A
.,3112 20 6E FF JSR $FF6E
.,3115 A9 1B    LDA #$1B
.,3117 8D 11 D0 STA $D011
.,311A 4C E2 FC JMP $FCE2
.,311D A2 03    LDX #$03
.
```

READY.

```
.
.
.,2000 A2 00    LDX #$00
.,2002 BD 08 30 LDA $3008,X
.,2005 18       CLC
.,2006 6A       ROR
.,2007 90 02    BCC $200B
.,2009 09 80    ORA #$80
.,200B 9D 08 30 STA $3008,X
.,200E E8       INX
.,200F E0 08    CPX #$08
.,2011 D0 EF    BNE $2002
.,2013 AD 08 30 LDA $3008
.,2016 85 FB    STA $FB
.,2018 A2 00    LDX #$00
.,201A BD 09 30 LDA $3009,X
.,201D 9D 08 30 STA $3008,X
.,2020 E8       INX
.,2021 E0 08    CPX #$08
.,2023 D0 F5    BNE $201A
.,2025 A5 FB    LDA $FB
.,2027 8D 0F 30 STA $300F
.,202A 60       RTS
.,202B 00       BRK
.
```

dyer 3008 – 300f

schiebe rol.

02
07
03/04
05/06
07 = 7