# Ramfinder

## *Identify, stash and fetch*

**by Ian Adam**

### Introduction

Adding an external RAM cartridge to a Commodore 64 or 128 can greatly increase its power and speed. For example, program overlays and disk files can be held in RAM, for near-instant access. A word processor or spreadsheet can now handle vastly larger documents or tables, rivalling those on any other personal computer. Another of my favourite uses is to prepare a number of graphics images, either high-resolution or low-res, and stash them in the RAM cartridge. When these are fetched rapidly, some pretty good animation can be created. Many other kinds of programs can use that extra capacity for a variety of different purposes, if only they know it's there.

The speed of the RAM cartridges is truly amazing. The RAM Expansion Controller is a special-purpose Direct Memory Access chip; it has a very limited instruction set, and is optimized for just one purpose - moving data. As a result, the data transfer rate is one byte per clock cycle, or one million bytes per second. This is far higher than with any other method, even much higher than you could achieve with hand-crafted machine language (a maximum of 70,000 cycles per second). Compared to loading data from a 1541 disk drive... well, there's just no comparison. When programming animation with the cartridge, I find that it's actually necessary to introduce delay loops in order to keep the animation down to a reasonable speed! The RAM cartridge can load high-resolution images about twice as fast as the video chip can display them, and four times as fast as the human mind can perceive them.

With all of these capabilities at hand, it follows that the thorough programmer will take the time to write programs in such a way that external RAM is taken advantage of. After all, there's no sense in the user buying a cartridge, if programs for the computer don't make use of the facility. Besides, your programs will look so much more impressive when they use all of the power at hand.

Right away, though, you run into the little problem of finding out how much RAM, if any, you have to work with. The standard Commodore operating system doesn't test for external RAM, and the cartridge itself doesn't go out of its way to tell you that it's present, so you have to devise a way to find out for yourself. What's more, while the cartridge does have a status byte to tell you how big it is, unfortunately two of the three available cartridges can have the same status byte!

That's the bad news. The good news is that all three cartridges use the same ten instruction registers, so they can all be controlled with the same commands. Furthermore, they are all located at the same address in the I/O block, at $df00 to $df0a, regardless of what computer they are installed in. Here are the cartridges Commodore has made available for the 64 and 128:

| Model | Banks | RAM | Status Byte | For | Bank #S |
|---|---|---|---|---|---|
| 1764 | 4 | 256K | xxx1xxxx | C 64 | 0 to 3 |
| 1700 | 2 | 128K | xxx0xxxx | C 128 | 0 and 1 |
| 1750 | 8 | 512K | xxx1xxxx | C 128 | 0 to 7 |

Check the larger accompanying table for further details on the meaning of the various control registers. In theory at least, the status byte (at $df00) should be a sufficient signature to identify the cartridge uniquely, once you know which model of computer it's installed in. After all, there is no duplication of the byte within each computer model. The 64 is not supposed to use a 128-model cartridge, since its meager power supply is barely capable of powering the computer itself, let alone any RAM expansion. The 1764 comes with an upgraded power supply, and so would not be of interest to an owner of a 128.

In the real world, however, you must remember that hardware could be combined in ways that your program might not have anticipated. For example, a Commodore 128 could be running a C64 program in 64 mode, and still have access to either of the 128-model expansion cartridges. You could also encounter a 64-model cartridge being operated in a 128. Thus, there is no guarantee that the cartridge will be the one you expect from its signature byte.

What's more, there still remains the problem of sorting out whether a cartridge is present at all. A genuine status register can take on many different values at different times, as a glance at the table will illustrate. However, if there is no

cartridge present, a read of the address of the non-existent status register gives a random value, which could mimic the status byte of a cartridge. All in all, an interesting programming challenge.

### The *Ramfinder* program

To the rescue rides the *Ramfinder* program. The challenge of detecting RAM isn't all that difficult to deal with, and any experienced programmer could tackle it reasonably well. However, I've always felt that the programmer should be freed to deal with important matters like making his or her program work properly, and not have to spend time and energy worrying about little details like what sort of hardware is attached.

To help out with this, I prepared the *Ramfinder* program, which has several useful advantages. This compact program will run in either the 64 or the 128, with no preference for either. As a further advantage, it is fully relocatable to any available start address (SA), so it will be compatible with just about any program you may want to write. What's more, it has three handy entry points:

```
sys sa      identify RAM cartridge & report
sys sa+4    STASH to expansion RAM
sys sa+7    FETCH from expansion RAM
```

All of this usefulness is packed into just over 100 bytes of machine language.

Of the three entry points, the first entry is the key one, because it will check whether or not a RAM cartridge is present. If none is found, it will return a value of zero. If it succeeds in finding external RAM, then the program will perform a couple of additional tests to identify which cartridge is present. It will return a result of 2, 4, or 8, representing the number of banks of memory available. The result is stored in zero-page memory, where it can be retrieved with a simple **lda $fb**, or a **peek(251)** from BASIC. The result is also held in the accumulator on departure.

The second and third entry points will perform very simple STASH and FETCH operations. Because the 64 and 128 manage their memory in such different fashions, these operations will not deal with subtleties like data in hidden memory banks. However, they are ideal for my favourite task, pulling graphic screens in and out of memory. To use these operations, put the number of the external RAM bank that you want to use in $fb (from BASIC: **poke 251, bank#**. For example, if you have a four-bank cartridge, select a bank number of 0 to 3). Load the microprocessor registers as follows:

| | |
|---|---|
| accumulator | high byte of expansion address |
| X register | high byte of computer address |
| Y register | high byte of length of transfer |
| | (all low bytes will be set to zero) |

If you are working in machine language, this is very straightforward. If you are working in BASIC 2.0 on the 64, just POKE these three values into memory locations 780 through 782, then **sys sa+4**. With BASIC 7.0 on the 128, the values can be transferred directly by the extended SYS command (as an example: **sys sa+4,8,4,4** to stash a low-res screen in the cartridge at $0800), but be sure you are in Bank 15 when you use the program.

If you find you need a more comprehensive STASH and FETCH capability, see Dale Castello's wedge commands for the 64 in *Transactor*, Volume 8 Issue 2, page 38 or use the built-in commands in BASIC 7.0 on the 128.

### Starting *Ramfinder*

How you use the *Ramfinder* program is at least partly dependent on what you want to do. If you are doing machine language programming and want to deal with the expansion cartridge issue painlessly, then type in the source code and add it to your library of useful routines. Again, note that the code is fully relocatable, so you should find it most accommodating in getting along with other routines. Its only requirement is for one byte of space in zero page, at $fb. A JSR to the start of the code will identify the expansion RAM available, and on return the accumulator will contain the number of 64K banks available. You can use the stash and fetch commands if suitable to your needs.

For you non-ML programmers, a BASIC loader is also supplied. Type the program in, being especially careful with the DATA statements at the end. Be sure to save a copy of the program before running it. When you do run the program, it gives a brief description of itself, then asks for the address to load the machine language into. Enter the address of any suitable free RAM (in the 128, you must be in Bank 15, so the load address must be less than 16270 in order to stay in non-banked RAM). If you are unsure, just press Return and the code will be loaded into the cassette buffer automatically. The program will then give further instructions for each of its routines.

If you want to incorporate the routine into other BASIC programs that you write, you have my blessings. Of course, you won't need to include all of the detailed instructions - just the DATA statements and their loader.

### How it works

The only way to detect a RAM cartridge reliably is actually to command it to work, then find out whether it performed as expected. As I mentioned, the status byte should tell you about the cartridge, but unfortunately it cannot be relied upon. Reading this when a cartridge is not installed may yield a phantom random number, leading to the erroneous conclusion that extra RAM is available.

To get around this problem, the program puts a known byte in zero page (the seed value 1 in its storage location at $fb), then commands the cartridge to save the page in expansion RAM. The value in $fb is changed (to #$b5, a convenient alterna-

tive), then the page is fetched back. By checking what value remains, the presence or absence of a cartridge can be deduced. If none, then a value of zero is returned.

With the knowledge that a RAM cartridge is present, the status byte can be read reliably. If bit 4 is clear, then the cartridge must be the 1700, and the task is finished.

Otherwise, there are still two possibilities, so one more test is required. This depends on the characteristic that the bank addresses 'wrap around'; that is to say, access to a bank beyond those in place will be decoded into the existing banks. To make use of this, remember that zero page has already been stashed in bank 0: this page will now be verified against bank 4. In the 1764 (the 256K cartridge) bank 4 is read as bank 0, so the verify operation succeeds. In the 1750, bank 4 is distinct and different from bank 0, so the verify fails. Thus, the detection is complete.

Table of REU Registers

| REGISTER | ADDRESS | TYPE | MEANING | |
|---|---|---|---|---|
| STATUS | $DF00 | Read Only | bits 0-3 | version |
| | | | bit 4 | 'size' |
| | | | bit 5 | 1 = verify error |
| | | | bit 6 | 1 = complete |
| | | | bit 7 | interrupt pending |
| COMMAND | $DF01 | R/W | bits 0,1 | transfer type |
| | | | bit 4 | 0 = $FF00 trigger |
| | | | bit 5 | 1 = reset parameters |
| | | | bit 7 | execute |
| ADDRESS | $DF02 | R/W | low byte, | computer address |
| | $DF03 | R/W | high byte | |
| EXP ADDR | $DF04 | R/W | low byte, | expansion RAM address |
| | $DF05 | R/W | high byte | |
| BANK | $DF06 | R/W | RAM bank #, low bits only | |
| LENGTH | $DF07 | R/W | low byte, | length of transfer |
| | $DF08 | R/W | high byte | |
| IRQ MASK | $DF09 | R/W | bit 5 | IRQ on verify error |
| | | | bit 6 | IRQ on completion |
| | | | bit 7 | enable interrupts |
| INCREMENT | $DF0A | R/W | bit 0 | 0 = increment RAM addrs |
| | | | | 1 = fix RAM address |
| | | | bit 1 | 0 = increment host addrs |
| | | | | 1 = fix host address |

## The benefits are yours

How you use this program is up to you. It is most useful when combined with other programs, whether in BASIC or machine language. *Ramfinder* is compatible with both; its length and transportability make it easy to incorporate with other programs of all types. *[If you've ever plugged in your REU and booted GEOS only to discover that the REU wasn't seated properly and thus was not seen by the system, you'll recognize another use for the program as published. - MO]*

There are two beneficiaries of this process; one is the user, whose investment in an expansion cartridge is rewarded with programs that offer more power and speed. The other beneficiary is you, the programmer - your programs will be slicker and more popular when they take advantage of all the resources available to them. Ultimately, that reflects favourably on your ability as a programmer!

**Listing 1:** *ramfinder.bas*

```
PK  100 print chr$(147):print"** ramfinder **"
PH  110 print:print"(c) ian adam"
PK  120 print"vancouver bc 1988"
GP  130 :
GK  140 print:print"this short program will identify an"
CA  150 print"external ram cartridge attached to"
GG  160 print"the computer, and indicate its size"
JM  170 print"in 64k banks. the program will operate"
OL  180 print"without modification in either"
II  190 print"the 64 or the 128."
MD  200 :
DK  210 print:print"the program is fully relocatable to"
HL  220 print"any start address, for compatibility."
OE  230 print"good locations are 828 in the 64,"
HK  240 print"and 2810 in the 128."
OG  250 :
FE  260 print:input"your start address";a$
IO  270 sa=val(a$):if sa=0 then sa=828 -2000*(peek(46)>27)
MI  280 :
HP  290 for i=sa to sa+117
GJ  300 read a:poke i,a
KD  310 next
EL  320 ;
CF  330 print chr$(147):print"identifying ram:"
CE  340 print:print"sys"sa
JA  350 print:print"this command will locate a ram"
KL  360 print"cartridge and indicate the number of"
EI  370 print"banks in location $00fb (251)."
BP  380 print"a value of 0 means no expansion ram."
GF  390 print"options are 2, 4, or 8 banks of 64k."
HI  400 print:print"number of banks installed now:"
DN  410 sys sa
DF  420 print:print"peek(251) =" peek(251)
CD  430 print:print"press return to continue"
FB  440 input a$
GD  450 :
HP  460 print chr$(147):print"stash and fetch:"
GC  470 print:print"to start, set these parameters;  all"
PF  480 print"others will be set to zero:"
CC  490 print:print"poke 251, external ram bank #"
ND  500 print"accumulator = msb external ram address"
KB  510 print"x register  = msb computer address"
GM  520 print"y register  = msb length to transfer"
GI  530 :
LN  540 print:print"on the 64, poke these three values"
LL  550 print"into locations 780 to 782, then...":print
AA  560 print"sys"sa+4" to stash"
DK  570 print"sys"sa+7" to fetch"
IL  580 :
NK  590 print:print"on the 128, use the extended sys"
CF  600 print"command.  for example, to save this"
IM  610 print"screen at the start of external ram:"
BG  620 print:print"poke 251,0:sys"sa+4",0,4,4"
KO  630 :
AI  640 end
OP  650 :
FG  660 data169,0,240,6,24,144,80,56,176,77,120,162,10,157,0,223,202,208
DO  670 data250,232,142,8,223,134,251,169,180,141,1,223,169,181,133,251,141,1
CG  680 data223,197,251,240,40,173,0,223,41,16,208,4,169,2,208,31,169,4
GA  690 data141,6,223,169,1,133,251,169,183,141,1,223,173,0,223,41,32,208
NE  700 data4,169,4,208,6,169,8,208,2,169,0,133,251,88,96,141,5,223
ON  710 data142,3,223,140,8,223,166,251,142,6,223,169,0,141,2,223,141,4
BH  720 data223,141,7,223,105,180,141,1,223,96
```

**Listing 2:** *ramfinder.src*

```
;******************
;*               *
;*               *
;*  external ram  *
;*   identifier   *
;*               *
;*               *
;*  for the c-64  *
;*   and  c-128   *
;*               *
;*               *
;******************
;
;
;
; (c) ian adam
;     may 1988
; vancouver bc
;
;
;
zpbank = $00fb
rec    = $df00
;
;
;
; 'jump table'
;
; start address = test exram
; sa + 4        = stash
; sa + 7        = fetch
;
;
; dummy start address:
;
* = $2000
;
; code is fully
; relocatable,
; and executes
; on either the 64
; or 128 (bank 15)
;
;
      lda #$00    ;entry to test ram
      beq trial
;
      clc         ;entry for stash
      bcc stash
;
      sec         ;entry for fetch
      bcs stash
;
;
;
; *************
; *           *
; *   trial   *
; *   stash   *
; *           *
; *************
;
; move zero page from computer
; to external ram bank 0, as
; a test of cartridge operation:
;
;
;
trial sei
      ldx #$0a
clear sta rec,x   ;clear registers
      dex
      bne clear
;
      inx
      stx $df08   ;move 1 page
      stx zpbank  ;plant seed
;
      lda #$b4    ;control byte = stash
      sta rec+1   ;execute
;
;
; *********************************
;
; the value 1 was saved as a test.
; if the stash was successful,
; then that seed value will be
; restored when the same page is
; fetched back.  thus, this
; sequence will detect a working
; external ram cartridge:
;
;
;
      lda #$b5    ;control byte = fetch
      sta zpbank
      sta rec+1   ;execute
;
      cmp zpbank  ;check it
      beq noram   ;exit if no exram found
;
; external ram located -
; find out how much:
;
      lda rec
      and #$10    ;check # of banks
      bne more
;
;
;
; if bit 4 is clear, then
; there must be 128k of
; external ram, in 2 banks:
;
      lda #$02
      bne exit
;
;
; *******************************
;
; if bit 4 is set, then there
; are either 4 banks (256k) or
; 8 banks (512k). test for this
; by verifying bank 4.  if there
; are only 4 banks, bank 0 will
; read as bank 4, and verify ok.
; if there are 8 banks, a verify
; error will result:
;
;
;
more  lda #$04
      sta $df06   ;set bank 4
      lda #$01
      sta zpbank
      lda #$b7    ;control byte = verify
;
      sta rec+1   ;execute
;
      lda rec     ;check status
      and #$20    ; for error
      bne most
;
      lda #$04    ;no error, 4 banks
      bne exit
;
most  lda #$08    ;error = 8 banks
      bne exit
;
;
; ***************
; *             *
; *  exit with  *
; *   message   *
; *             *
; ***************
;
;
noram lda #$00
exit  sta zpbank  ;leave message
      cli
      rts
;
; the # of expansion banks will
; be left in zpbank ($00fb).
; 0 banks means no external ram.
; options:  2, 4, or 8 banks.
;
;
; *********************
; *                   *
; *  stash and fetch  *
; *                   *
; *********************
;
;
;
; a = high byte expansion address
; x = high byte computer address
; y = high byte of length
; bank number in zpbank
; all other parameters set to 0
;
;
stash sta $df05   ;external ram address
      stx $df03   ;set computer address
      sty $df08   ;set length
      ldx zpbank
      stx $df06   ;set bank
;
      lda #0      ;set low bytes to 0
      sta $df02
      sta $df04
      sta $df07
;
;
; build control byte and execute:
;
; the carry bit will increment the
; control byte by 1, when a fetch
; was specified in the jump table
;
;
      adc #$b4    ;build control byte
      sta rec+1   ;execute
;
      rts         ;all finished
;
.end
```