Robin Harbron

computer and video games old and new, programming, music, movies, books, news about my family, and whatever else

---

REU Programming

REU Programming
by Robin Harbron.

Note: This article originally appeared in Loadstar Letter #46, published in 1997. Thanks to the folks at Loadstar for granting permission for me to post it on my website.

Want to know how to make use of a Ram Expansion Unit in your own programs? The following BASIC and assembly programs are examples of how to detect an REU, determine its size, and then store, retrieve or swap memory with it.

The REU is accessed through IO2, which is the memory locations from $DF00 to $DFFF. The REU actually has 7 registers, with some registers being multiple bytes, so all your PEEKing and POKEing is done in the addresses $DF00 to $DF0A (57088 to 57098).

How to detect an REU? If the REU is present, addresses $DF02 through $DF05 retain what is stored in them. I don't know of any other device with that property, so the following routines exploit this. In the BASIC version (which only uses one variable, just to be unobtrusive), x will equal 0 if there is no REU, and x=1 if there is one present.

```
10 rem detect
20 forx=2to5:poke57088+x,x:nextx
30 forx=5to2step-1
40 ifpeek(57088+x)<>xthenx=1
50 nextx
60 ifx=0thenprint"no reu"
```

In the assembly version (sys 16384 to start it from BASIC), the accumulator will be 0 if there is no REU, and 1 if there is one. PEEK(780) will tell you the contents of the accumulator if you call this routine from BASIC.

```
         *= $4000
         ; detect REU

         ldx #2
loop1    txa
         sta $df00,x
         inx
         cpx #6
         bne loop1

         ldx #2
loop2    txa
         cmp $df00,x
         bne noreu
```

```
         inx
         cpx #6
         bne loop2

         lda #1
         rts

noreu    lda #0
         rts
```

Now, how to tell how big the REU is? This is my solution, although there seems to be no absolute best way. This routine takes a while to run, in the BASIC version (of course the assembly version takes no noticeable time at all). Simply speaking, the routine writes the bank number into the first byte of every bank (all 256 possible banks, which would be a 16 meg REU!). Banks that are not available seem to be shadows of the existing banks, so what you write into a shadow bank ends up overwriting what we had put in a previous bank. Then we just go through the banks again, and see how many consecutive, increasing values we can find. This number is the number of available banks. This routine is also non-destructive. It reads and stores the original contents of the REU, and then puts them back when it's done. The routine requires a 257 byte buffer for this, which the variable s/label temp points to. After running the BASIC version, the variable A contains the number of banks available. For example, A=8 after running the program on my 512K REU. The ML version returns the number of banks in the accumulator.

```
10 rem size
15 s=49152
20 poke57090,0:poke57091,192
30 poke57092,0:poke57093,0
40 poke57095,1:poke57096,0
50 poke57098,0
60 forb=0to255:poke57094,b
63 pokes,b:poke57089,178
65 pokes+1+b,peek(s):nextb
70 b=0:o=0
80 poke57094,b:poke57089,177
90 n=peek(s)
100 ifn>otheno=n:b=b+1:goto80
110 a=b
120 forb=255to0step-1:poke57094,b
130 pokes,peek(s+1+b):poke57089,176
140 nextb
150 printa
```

```
         *= $4000
temp     = $c000
         ;detect reu size

         lda #0
         sta $df04
         sta $df05
         sta $df08
         sta $df0a
         lda #1
         sta $df07

         lda #<temp
```

```
          sta $df02
          lda #>temp
          sta $df03

          ldx #0
loop1     stx $df06
          stx temp
          lda #178
          sta $df01
          lda temp
          sta temp+1,x
          inx
          bne loop1

          ldy #177
          ldx #0
          stx old
loop2     stx $df06
          sty $df01
          lda temp
          cmp old
          bcc next
          sta old
          inx
          bne loop2
next      stx size
          ldy #176
          ldx #255
loop3     stx $df06
          lda temp+1,x
          sta temp
          sty $df01
          dex
          cpx #255
          bne loop3
          lda size
          rts
old       .byte 0
size      .byte 0
```

Now we get to the primary function of the REU: Storing and retrieving data. The following routine is a model for you to use. It either stashes or retrieves a screen to/from the REU. Here's the breakdown of the registers:

$DF02 & $DF03 (57090 & 57091) point to the base address in the computer's memory that we want to deal with. It's stored in standard lo/hi format. $400 is the address we stash in here to deal with the screen.

$DF04-$DF06 (57092-57094) point to the base address in the REU's memory that we want to use. It's stored in lo/hi/bank format. The bank number can be thought of as the "even higher" byte. We'll just store the info in location $0, bank 0.

$DF07-$DF08 (57095-57096) determine the length in bytes of the transfer. It's stored in lo/hi format as well. Storing a zero in this register causes the REU to transfer 64K at once, which is the limit for one move.

$DF0A allows you to fix either the C64 or REU memory address, to allow you to either write or read one particular memory location many times. If bit 7 is set, the C64 address is fixed, and if bit 6 is set, the REU address is fixed. This has many uses: You could fix the address in the REU (perhaps storing a 0 or 255 there) and then set the length to 8000 bytes. Then you could fill an entire bitmap screen with that byte in just 8000 cycles, by transferring from the REU to the C64! This is at least 4 times faster than doing it with the processor. Or how about sampling a particular location in memory? It's been used to examine the random number generator in the SID chip.

$DF01 (57089) actually executes the move. Setting bit 7 is necessary, to tell the REU to execute the move. The other bits tell the REU how to do that move:

If bit 5 is set, the address and length registers are unchanged after the command is executed. This is useful if you are going to do many consecutive, identical transfers. If the bit is not set, the address registers will point to the address immediately after the last address accessed, and the length register will be set to 1.

If bit 4 is set, the command will be executed immediately. If it is not set, the command will not be executed until you write to location $FF00. This sounds weird, but is useful if you want to do transfers around the IO/ROM areas of memory. This gives you a chance to set up the transfer, then switch IO out (for example), and then execute the transfer. A simple LDA $FF00 : STA $FF00 will do the job.

Bits 1 and 0 define the transfer type. 00 is for a C64 to REU move, 01 is for a REU to C64 move, 10 is to swap between REU and C64 memory (this takes twice as long, as it has double the work to do), and 11 is for a compare between the REU and C64. To use the compare feature, clear bit 5 of $DF00, and execute the compare (just like doing any other transfer, but no bytes are actually moved). Now, if bit 5 of $DF00 has been set, then a difference was found between the REU and C64 memory.

Here's the BASIC version. Just make sure you set x according to the comments in line 60.

```
10 poke57090,0:poke57091,4:rem c64
20 poke57092,0:poke57093,0:poke57094,0:rem reu
30 poke57095,232:poke57096,3:rem length
40 poke57098,0:rem not fixed addresses
50 poke57089,128+16+x
60 rem x=0 c64->reu, x=1 reu->c64, x=2 swap c64 and reu
```

In the assembly version, just set the labels c64 to the C64 base address, reu to the REU base address, bank to the bank number in the REU (I used 2, as my assembler makes use of 0 and 1), length to the number of bytes you want to transfer, and action equal to 0, 1, 2 or 3, corresponding to what type of transfer you'd like.

```
          *= $4000
          ;reu store/swap/get

c64       = $0400
reu       = $00
bank      = $02
length    = 1000
action    = 0

          lda #<c64
          sta $df02
          lda #>c64
          sta $df03
          lda #<reu
          sta $df04
```

```
lda #>reu
sta $df05
lda #bank
sta $df06
lda #<length
sta $df07
lda #>length
sta $df08
lda #0
sta $df0a
lda #144+action
sta $df01
rts
```

-

My other Pages

- Family
- Loadstar Letter Articles
  - REU Programming
  - REU Programming & Border Removal
  - Swiftlink Programming

Categories

- Blogging (8)
- Classic Computers (31)
- Family/Life (35)
- Lego (2)
- My Music (33)
- Other's Music (5)
- Programming (30)
- Travelogue (19)
- Uncategorized (77)
- Video Games (28)
- Writing (10)

Blogroll

- A Beggar at the Table
- Block Upon Block
- David and Erin James
- Disenfranchised Housewife
- Doug Anderson
- It’s No HTA
- Marla
- MCKUCHTAS
- MEF
- mercy in the middle
- Nafcom’s Blog

- papasmurf
- Reppepper's Blog
- Rob O'Hara
- Shroom's World
- Tony's Rants

Forums

- AtariAge
- Commodore 128 Alive!
- Commodore PET Alive!
- commodore.ca
- Denial
- DTV Hacking
- Lemon64
- LUGNET
- Minigame Competition
- S.S. of Commodore Coders
- Slang

My Bands

- Bedford Level Experiment – the band!

Meta

- Log in
- Valid XHTML
- WordPress

Archives

- June 2019
- December 2018
- August 2018
- August 2016
- April 2014
- March 2014
- October 2013
- October 2012
- January 2011
- December 2008
- March 2008
- February 2008
- January 2008
- December 2007
- November 2007
- August 2007
- June 2007
- April 2007

- March 2007
- February 2007
- January 2007
- December 2006
- November 2006
- October 2006
- September 2006
- August 2006
- July 2006
- June 2006
- May 2006
- April 2006
- March 2006
- February 2006
- January 2006
- December 2005
- November 2005
- October 2005
- September 2005
- August 2005
- July 2005
- June 2005
- May 2005
- April 2005
- March 2005
- February 2005
- January 2005
- December 2004
- November 2004
- October 2004
- September 2004
- August 2004
- July 2004
- June 2004
- May 2004
- April 2004
- March 2004
- February 2004

- Search
