REU Programming

Hello all!

I want to contribute to the mailing list on the software side. So I send you a description of the programming of the 17xx RAM expansion units. I sent this already a week ago but it got lost during the problems with the old mailing list address. In the meantime there has already been posted some information about REUs, but my description is a bit different and contains more detailed information, so I thought I nevertheless send it to you.

The following is based on the Commodore 1764 user's manual (german version) and my own experiences programming the 1764 Ram Expansion Unit (REU).

External RAM Access With REUs

The REUs provide additional RAM for the C64/128. Three types of REUs have been produced by Commodore. These are the 1700, 1764 and 1750 with 128, 256 and 512 KBytes built in RAM. However they can be extended up to several MBytes.

The external memory can not be addressed directly by the C64 with it's 16-bit address space. It has to be transferred from an to the main memory of the C64. For that purpose there is a built in RAM Expansion Controller (REC) which transfers memory between the C64 and the REU using Direct Memory Access (DMA). It can also be used for other purposes.

RAM Expansion Controller (REC) Registers

The REU is programmed by accessing it's registers, that appear memory mapped in the I/O-area between $DF00 and $DF0A when a REU is connected through the expansion port of the C64. They can be read and written to like VIC- and SID-registers.

```
$DF00: STATUS REGISTER
       various information can be obtained (read only)

Bit 7:     INTERRUPT PENDING  (1 = interrupt waiting to be served)
           unnecessary
Bit 6:     END OF BLOCK  (1 = transfer complete)
           unnecessary
Bit 5:     FAULT  (1 = block verify error)
           Set if a difference between C64- and REU-memory areas was found
           during a compare-command.
Bit 4:     SIZE  (1 = 256 KB)
           Seems to indicate the size of the RAM-chips. It is set on 1764
           and 1750 and clear on 1700.
Bits 3..0: VERSION
           Contains 0 on my REU.
```

```
$DF01: COMMAND REGISTER
       By writing to this register RAM transfer or comparision can be
       executed.

Bit 7:     EXECUTE  (1 = transfer per current configuration)
           This bit must be set to execute a command.
Bit 6:     reserved  (normally 0)
Bit 5:     LOAD  (1 = enable autoload option)
           With autoload enabled the address and length registers (see
           below) will be unchanged after a command execution.
           Otherwise the address registers will be counted up to the
           address off the last accessed byte of a DMA + 1,
           and the length register will be changed (normally to 1).
Bit 4:     FF00
           If this bit is set command execution starts immediately
           after setting the command register.
           Otherwise command execution is delayed until write access to
           memory position $FF00
Bits 3..2: reserved  (normally 0)
Bits 1..0: TRANSFER TYPE
           00 = transfer C64 -> REU
           01 = transfer REU -> C64
           10 = swap C64 <-> REU
           11 = compare C64 - REU

$DF02..$DF03: C64 BASE ADDRESS
              A 16-bit C64 - base address in low/high order.

$DF04..$DF06: REU BASE ADDRESS
              This is a three byte address consisting of a low and
              high byte and an expansion bank number.
              Normally only bits 2..0 of the expansion bank are valid
              (for a maximum of 512 KByte), the other bits are always
              set. This must be different if more than 512 KByte are
              installed.

$DF07..$DF08: TRANSFER LENGTH
              This is a 16-bit value containing the number of bytes to
              transfer or compare.
              The value 0 stands for 64 Kbytes.
              If the transfer length plus the C64 base address exceeds
              64K the C64 address will overflow and cause C64 memory
              from 0 on to be accessed.
              If the transfer length plus the REU base address exceeds
              512K the REU address will overflow and cause REU memory
              from 0 on to be accessed.

$DF09: INTERRUPT MASK REGISTER
       unnecessary

Bit 7:     INTERRUPT ENABLE  (1 = interrupt enabled)
```

```
Bit 6:     END OF BLOCK MASK  (1 = interrupt on end)
Bit 5:     VERIFY ERROR  (1 = interrupt on verify error)
Bits 4..0: unused (normally all set)

$DF0A: ADDRESS CONTROL REGISTER
       Controlls the address counting during DMA.
       If an address is fixed, not a memory block but always the same
       byte addressed by the base address register is used for DMA.

Bit 7:     C64 ADDRESS CONTROL  (1 = fix C64 address)
Bit 6:     REU ADDRESS CONTROL  (1 = fix REU address)
Bits 5..0: unused (normally all set)
```

To access the REU-registers in assembly language it is convenient to define labels something like this:

```
    status   = $DF00
    command  = $DF01
    c64base  = $DF02
    reubase  = $DF04
    translen = $DF07
    irqmask  = $DF09
    control  = $DF0A
```

How To Recognize The REU

Normally the addresses between $DF00 and $DF0A are unused. So normally if values are stored there they get lost. So if you write e.g. the values 1,2,3,... to $DF02..$DF08 and they don't stay there you can be sure that no REU is connected. However if the values are there it could be because another kind of module is connected that also uses these addresses.

Another problem is the recognition of the number of RAM banks (64 KByte units) installed. The SIZE bit only tells that there are at least 2 (1700) or 4 (1764, 1750) banks installed. By trying to access & verify bytes in as many RAM banks as possible the real size can be determined. This can be seen in the source to “Dynamic memory allocation for the 128” in Commodore Hacking Issue 2.

I personally prefer to let the user choose if and which REU banks shall be used.

Simple RAM Transfer

Very little options of the REU are necessary for the main purposes of RAM expanding.

Just set the base addresses, transfer length and then the command register.

The following code transfers one KByte containing the screen memory ($0400..$07FF) to address 0 in the REU:

```
    lda #0
    sta control ; to make sure both addresses are counted up
```

```
    lda #<$0400
    sta c64base
    lda #>$0400
    sta c64base + 1
    lda #0
    sta reubase
    sta reubase + 1
    sta reubase + 2
    lda #<$0400
    sta translen
    lda #>$0400
    sta translen + 1
    lda #%10010000;  c64 -> REU with immediate execution
    sta command
```

To transfer the memory back to the C64 replace “lda #%10010000” by “lda #%10010001”.

I think that this subset of 17xx functions would be enough for a reasonable RAM expansion. However if full compatibility with 17xx REUs is desired also the more complicated functions have to be implemented.

Additional Features

Swapping Memory

With the swap-command memory between 17xx and C64 is exchanged. The programming is the same as in simple RAM transfer.

Comparing Memory

No RAM is transferred but the number of bytes specified in the transfer length register is compared. If there are differences the FAULT-bit of the status register is set. This bit is cleared by reading the status register which has to be done before comparing to get valid information.

Using All C64 Memory

C64 memory is accessed by the REU normally in the memory configuration existing during writing to the command register. However in order to be able to write to the command register the I/O-area has to be active.

If RAM between $D000 and $DFFF or character ROM shall be used it is possible to delay the execution of the command by storing a command byte with bit 4 (“FF00”) cleared. The command will then be executed by writing any value to address $FF00.

Example:

```
    < Set base addresses and transfer length >
    lda #%10000000 ; transfer C64 RAM -> REU delayed
    sta command
    sei
    lda $01
    and #$30
    sta $01 ; switch on 64 KByte RAM
    lda $FF00 ; to not change the contents of $FF00
    sta $FF00 ; execute DMA
    lda $01
    ora #$37
    sta $01 ; switch on normal configuration
    cli
```

Transfer Speed

During DMA the CPU is halted and the memory access cycles normally available for the CPU are now used to access one byte each. So with screen and sprites switched off in every clock cycle (985248 per second on PAL machines) a byte is transferred. If screen is on or sprites are enabled transfer is a bit slower, as the VIC exclusively accesses RAM sometimes. An exact description of those “missing cycles” can be found in Commodore Hacking Issue 3.

Comparing memory areas is as fast as transfers. (Comparison is stopped once the first difference is found.)

Swapping memory is only half as fast, as for every bytes two C64 memory accesses (read & write) are necessary.

Interrupts

By setting certain bits in the interrupt mask register IRQs at the end of a DMA can be selected. However as the CPU is halted during DMA it will always be finished after the store instruction into the command register or $FF00. So there is no need to check for an “END OF BLOCK” (bit 6 of status register) or to enable an interrupt.

Executing Code In Expanded Memory

Code in external memory has always to be copied into C64 memory to be executed. This is a disadvantage against bank switching systems. However bank switching can be simulated by the SWAP command. This is done e.g. in RAMDOS where only 256 bytes of C64 memory are occupied, the 6 KByte RAM disk driver is swapped in whenever needed. Probably too much swapping is the reason for RAMDOS to be not really fast at sequential file access.

Other Useful Applications Of The REU

The REC is not only useful for RAM transfer and comparison.

One other application (used in GEOS) is to copy C64 RAM areas by first transferring it to the REU and then transferring it back into the desired position in C64 memory. Due to the fast DMA this is about 5 times faster than copying memory with machine language instructions.

Interesting things can be done by fixing base addresses. Large C64 areas can be filled very fast with a single byte value by fixing the REU base address. Thus it is also possible to find the end of an area containing equal bytes very fast e.g. for data compression.

Fixing the C64 base address is interesting if an I/O-port is used, as data can be written out faster than normally possible. It would be possible to use real bitmap graphics in the upper and lower screen border by changing the “magic byte” (highest by the VIC addressed byte) in every clock cycle during the border switched off.

Generally the REC could be used as graphics accelerator e.g. to copy bitmap areas or to copy data fast into the VIC-addressable 16 KByte area.

Comparison Of Bank Switching and DMA

When comparing bank switching and DMA for memory expansion I think DMA is the more comfortable methode to program and also is faster in most cases. The disadvantage with code execution not possible in external memory could be minimized by copying only the necessary parts into C64 memory. Executing the code will take much more time than copying it into C64 memory.

Richard Hable - Richard.Hable@JK.Uni-Linz.AC.AT

Marko Mäkelä - (Marko.Makela@HUT.FI)