The H8-512K RAM add-on Programmer's Manual
DRAFT
04/14/19

The MMU organizes the RAM into 16K "pages", at 16K boundaries. It also divides the 64K CPU address space into 16K "blocks", at 16K boundaries. Any page may be assigned to any block. In addition, separate "page tables" exist for read vs. write, allowing direct copying between different regions of RAM.

The MMU I/O space is divided into 8 Output ports. Writing a 5-bit value to one of these ports will assign a RAM page to the corresponding address block, for either reading or writing. The following table shows port offsets and affected address blocks.

Port Offsets		**CPU Memory Address Block**
Read	**Write**	
0	4	0000-3FFF
1	5	4000-7FFF
2	6	8000-BFFF
3	7	C000-FFFF

Also note that the high bit (bit 7) of the data to these ports must be "1" in order to enable mapping. Any time data is output with bit 7 set to "0" the mapping will revert to the power-on state, where pages 0-3 are assigned blocks 0-3. This map enable feature is not normally used, except during initial switching to banked memory use. Once banked memory is setup, all bank switching should always have bit 7 set to "1". The format of each page table entry is as follows. Note, “MAP” is not actually stored in the page table, but is rather a single, global, bit. The last byte output to any port offset will determine the MAP value.

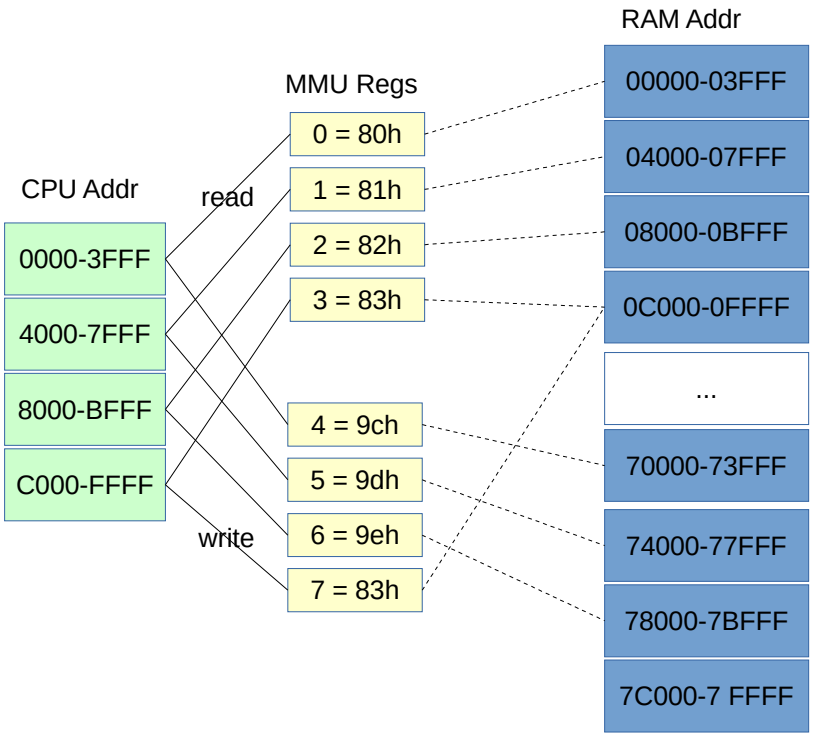
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
MAP			A18	A17	A16	A15	A14

CPU address lines A14 and A15 select which address block is being used, and the contents of the (read or write) page table entry determines address lines A14-A18 sent to the RAM chip(s). CPU address lines A0-A13 are passed to the RAM chip(s) directly. Any 16K page of RAM can be mapped to any 16K block of CPU addresses. 16K alignment is strictly maintained.

As with all banked memory schemes, care must be taken to ensure that the code which is executing is in memory that does not change with the mapping. It is also necessary to consider interrupts and whether the necessary code is available to service an interrupt. For this reason, interrupts are often disabled during critical bank switch operations.

Note that typical bank switching involves setting several page table entries, and such an operation is not atomic. This is another reason to disable interrupts during bank switching.

The following diagram shows how addresses are mapped.



This diagram shows the relationship between CPU addresses and MMU Registers, as well as an example page table mapping to copy data from “bank 0” to “bank 7”. The code doing the copy must reside (execute) in common memory, C000-FFFF. This includes any additional resources such as stack.

Initialization

Care must be taken when initially setting up the MMU after a RESET (or power on). The disabled state is equivalent to a mapping of “0, 1, 2, 3”, **BUT** the MMU registers do not contain this mapping. The MMU must **NOT** be enabled until all mappings have been initialized. A typical sequence to perform initial mappings might be:

```
setup:      di
            mvi   a,0
            out   mmu+0
            out   mmu+4
            inr   a
            out   mmu+1
            out   mmu+5
            inr   a
            out   mmu+2
            out   mmu+6
            inr   a
            out   mmu+3
            ori   080h
            out   mmu+7
            ei
            ret
```

Note that running this initialization code at any other time is likely to cause a crash, as disabling the MMU when non-default mapping is in use will likely cause the code being executed to disappear.

CP/M 3 Banked Memory

Here is a typical CP/M 3 Bank Switching table for the first 208K (four 48K banks plus 16K common) of the RAM:

```
table:      ;    _00_40_80_C0_
bank0:      db     0, 1, 2, 3
bank1:      db     4, 5, 6, 3
bank2:      db     7, 8, 9, 3
bank3:      db    10,11,12, 3 ; not needed?
```

Note how the top address block, C000-FFFF, always contains the page value "3". This establishes "common memory" (this mapping never changes) and is necessary for CP/M 3 to operate. Also note that this table does not show bit 7 (the MAP bit). An implementation might set bit 7 of all bytes in the table only after proper detection/initialization of hardware is done.

Because of the separate page tables for read and write, the CP/M 3 "XMOVE" feature may be implemented. This allows for more flexible placement of buffers and can also improve performance of the Directory HASH Buffers and warm boot reloading of the CCP. XMOVE uses CP/M bank numbers, and so does not alter the page table scheme from what is normally used. This is in contrast to a RAM Disk implementation, which is likely to directly map desired pages to a convenient address block for “I/O”.

The reference implementation (mem512k.asm) demonstrates all of the above. An example bank select routine, where the logical bank number is passed in A, is:

```
bank$sel:
        lxi     h,table ; table of bank schemes
        add     a
        add     a
        mov     c,a     ; logical bank number * 4
```

```
        mvi     b,0     ;
        dad     b       ; index to desired bank
        push    h       ; same mapping for RD and WR
        mvi     b,4
        mvi     c,mmu-1
bnksl1:
        inr     c       ; set RD pages
        outi
        jrnz    bnksl1
        pop     h
        mvi     b,4
bnksl2:
        inr     c       ; set WR pages
        outi
        jrnz    bnksl2
```

This routine updates all 4 read page registers and all 4 write page registers. Both read and write use the same mapping, as required by CP/M 3 (normal execution). If address block C000-FFFF page mapping never changes, then only 3 page bytes need be output for each of RD and WR mappings. For XMOVE, the setup requires selecting a different table entry for RD vs. WR. The RAM Disk also takes advantage of this feature to directly copy to/from the device.

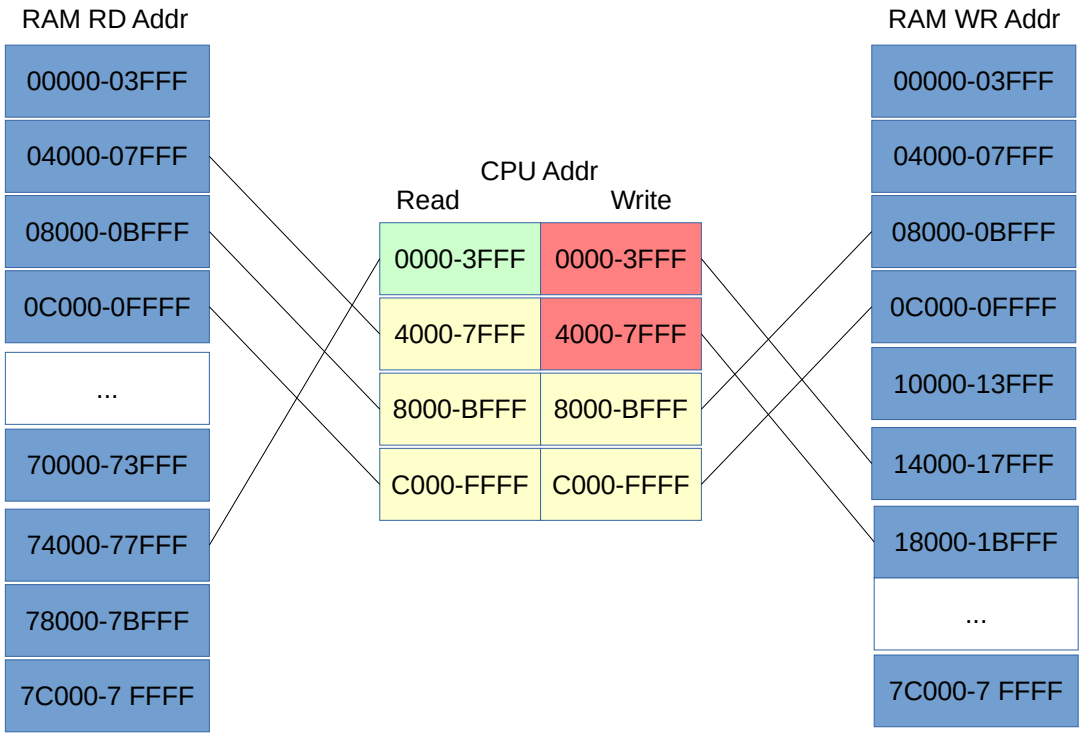
RAM Disk

Since CP/M 3 can rarely take advantage of more that 3 banks (160K), the additional memory beyond what CP/M 3 can use may be organized into a "RAM disk".

Because of the flexible page assignments and separate page tables for read and write, I/O to/from the RAM Disk can be done directly to the CP/M DMA Buffers. This greatly improves performance and simplifies the driver code.

The reference RAM Disk implementation (rd512k.asm) computes the page (13-31) which represents the desired disk data, then creates read and write page selections base on direction of transfer and user's DMA buffer and bank. Note, the DMA buffer may be for BDOS buffers or the user's program space so a specific bank cannot be assumed.

The rd512k Ram Disk driver uses (remaps) CPU Address block 0000-3FFF to move data. Because a CP/M DMA buffer may be on any boundary, it is possible for it to span a 16K block. This requires the driver to also map the subsequent block (4000-7FFF) to the page following the start of the DMA buffer. Since all of CP/M 3 BIOS code resides above 8000, this is not a problem. Because H8/H89 interrupt vectors are kept in page 0 (block 0000-3FFF), interrupts must be disabled during transfers.



This diagram shows an example of a ramdisk READ operation under CP/M 3, where the OS resides in “bank 0” (RAM addresses 00000-0FFFF), the user’s program (i.e. disk buffer) reside in “bank 1” (RAM addresses 10000-1FFFF*), and the ramdisk resides in memory beyond the banks used by CP/M (in this example, the computed ramdisk (sector) address was somewhere in 70000-73FFF). The yellow boxes represent OS mappings, the green box is for ramdisk data (read in this case), and the red boxes represent the user’s buffer (to write, in this case). Note that the transfer (CPU) address must use the low order 14 address bits from the target address, combined with bits 14, 15 of the mapping address.

* The CP/M 3 “TPA” (user program/data) might span the “common memory” boundary, requiring the mapping to replicate the same mapping as the OS would use for that “bank”.

A ramdisk transfer under HDOS might be simpler, since that OS does not employ multiple banks of memory. It may only be necessary to map the ramdisk chunk, leaving the rest of the mapping set to “normal”.