Bank Switching
on the IMSAI 8080

by Bruce E. Hall, W8BH

At some point you are going to hear about banked ROM/RAM, bank switching, or "shadow" ROM, and wonder what the heck that is. It sounds vague and mysterious. I asked a related question to the creator of this beautiful IMSAI 8080 kit. He very politely suggested that I *RTFM*. OK, I can do that. Read on if you are curious what "banking" means and what you can do with it.

The 8080/Z80 microprocessor can directly address 64K of memory. This was a huge amount of memory in the late 1970s. But a few short years later, applications needed more. Bank switching is a technique to extend available memory beyond the 64K limit.

There are *two* different ways in which bank switching is used in the IMSAI. The first works on ROM, effectively removing a ROM from the address space after it is no longer needed. The second method does bank switching on RAM, giving you large banks of extra RAM that can be swapped in and out of the available 64K address space. Both are described below.

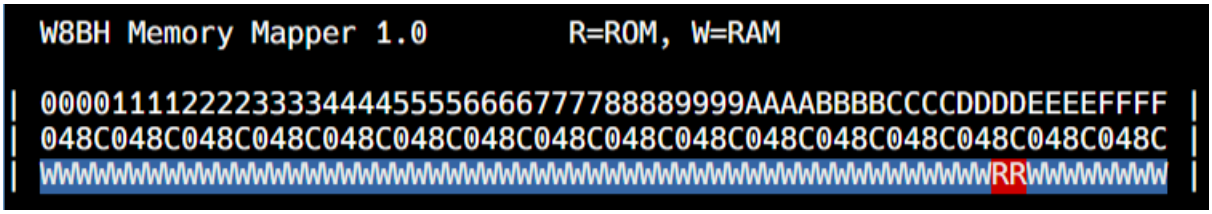
ROM bank switching: basic information.

ROM bank switching lets us remove the startup ROM from the address space, once it completes its task, thus freeing the 2K of address space it occupies. In other words, it changes the memory configuration from 62K RAM/2K ROM to 64K RAM. This method is described in the kit literature as "MPU-B style bank-switched ROM and RAM for boot rom masking".

For the IMSAI 8080 esp, ROM bank switching is enabled by bit 11, NVS_BANK_ROM, in the startup configuration settings. If this bit is 0, no bank switching occurs, and the ROM is always present. If this bit is 1, bank switching can be used, programmatically removing the startup ROM from the address space.

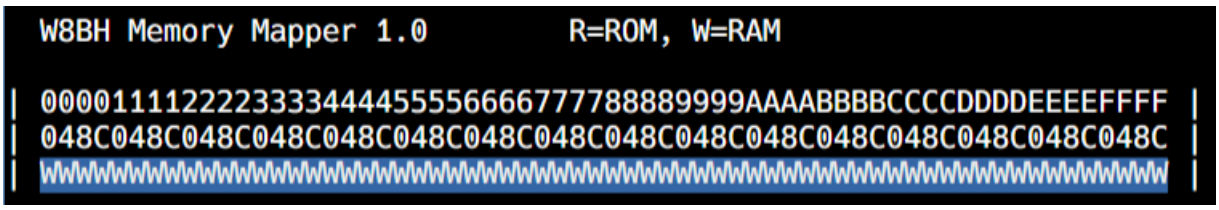
It's time to see it in action. Check your startup configuration now and set it to an appropriate value for CP/M, like **0158**. NVS_BANK_ROM should be off. I also recommend turning NVS_AUTO_RUN off for these exercises.

Start CP/M using the default 54K system. Load and run map.asm, a small memory mapping program. It displays a line of 64 characters, each character representing 1K of



memory. A blue W indicates RAM and a red R indicates ROM. Notice that there is a 2K ROM (two red R's) present at starting address D800. The red block represents the startup ROM for CP/M. The rest of memory is RAM.

Now do the same thing, changing NVS_BANK_ROM to 1. I use configuration setting **0958**. The memory map should look like this:



The ROM is no longer present. It has been removed from the address space, freeing the memory it occupied for other uses.

The boot ROM does not interfere with the 54K CP/M system, which loads itself below D800. But consider the 60K and 62K CP/M systems. They occupy memory in the same region as the ROM.

CP/M component	54K	60K	62K	64K
Start of CCP	BC00	D400	DC00	E400
Start of BDOS	C406	DC06	E406	EC06
Start of BIOS	D200	EA00	F200	FA00
End of BIOS	D7B9	EFB9	F7B9	FFB9

Notice the ROM memory conflicts, highlighted in red. Loading one of these systems without ROM banking will fail. The 54K system was chosen because it does not interfere with the ROMs. Use banked ROM with larger memory CP/M systems. I prefer to use a 62K system with banked ROM enabled.

By now you should understand the value of banked ROM. Continue reading if you want to see implementation and programming details. Otherwise, skip ahead to the section on banked RAM.

ROM bank switching: deeper dive

How is ROM bank switching implemented? Finding stuff in "the manual" is difficult. The manual is not a single document. The IMSAI PCS 80/30 User Manual, on page 11-3, describes operation of the bootstrap ROM:

> *"Under software control, the ROM can be located at 0000 or D800...Power on clear initializes the ROM at 0000 to serve as a power on jump....A custom programmed PROM in this location can initialize the system, load programs in the entire 64K memory space, and then disable the PROM while jumping to the program start location. The PROM can be re-enabled at any time to use code stored in it, either at location 0000 or at location D800. The PROM enable is delayed to permit a jump or call to a routine, even from a program executing directly underneath the PROM location. A system reset automatically re-enables the PRM at 0000."*

Details about bank switching the ROM are found starting on page 11-24:

> *The MPU-B control port is a single I/O mapped port address (F3). It is a write-only port and only two of the eight bits are active. The two bits control whether group 0 or 1 circuits appear in the memory space or not. When they are 0, the circuits appear in the memory space; when they are 1, the circuits do not appear in the memory space and whatever memory is installed in the system at that location then appears.*
>
> *Group 0 control: bit 6*
> *BIT 6=0 2K ROM appears at 0000-07FF*
> *BIT 6=1 System memory appears at 0000-07FF*
>
> *Group 1 control: bit 7*
> *BIT 7=0 2K ROM appears at D800-DFFF*
> *BIT 7=1 System memory appears at D800-DFFF"*

A single I/O port, F3, controls how the ROM is enabled and where it appears in the address space.

Bank switch control port (F3) settings:

Bit	Value=0 (ROM enabled)	Value=1 (ROM disabled)
6 (0000-07FF space)	2K ROM at 0000-07FF	RAM at 0000-07FF
7 (D800-DFFF space)	2K ROM at D800-DFFF	RAM at D800-DFFF

Here are the resulting four port values (00, 40, 80, C0) and their effect on the memory map:

	Bit 7 = 0	Bit 7 = 1
Bit 6 = 0	**00h** (0000 ROM, D800 ROM)	**80h** (0000 ROM, D800 RAM)
Bit 6 = 1	**40h** (0000 RAM, D800 ROM)	**C0h** (0000 RAM, D800 RAM)

To summarize: there is a 2K ROM on the IMSAI MPU-B board. This ROM is enabled at startup and can be enabled/disabled under software control via I/O port F3. If enabled, the ROM can appear at address locations 0000-07FF and/or D800-DFFF, depending on the byte value (00, 40, 80, or C0) sent to port F3. Disabling the ROM lets us reclaim the address space it occupies.

With this knowledge we can modify the mapper program, poking the memory control port F3 with different values, and displaying the resulting memory maps. Here is a snippet of the code:

```
LD   A,40h          ; 40h: 0000 RAM, D800 ROM
OUT  (0F3h),A       ; change memory configuration
CALL map            ; and show it

LD   A,80h          ; 80h: 0000 ROM, D800 RAM
OUT  (0F3h),A       ; change memory configuration
CALL map            ; and show it

LD   A,0C0h         ; C0h: 0000 RAM, D800 RAM
OUT  (0F3h),A       ; change memory configuration
CALL map            ; and show it

LD   A,0            ; 00h: 0000 ROM, D800 ROM
OUT  (0F3h),A       ; change memory configuration
CALL map            ; and show it
```

The complete program is found in map2.asm.

Normally, CP/M loads user programs at 0100H. But address 0100H is in a potential ROM area. As soon as we stuff 00 or 80h into port F3, the program disappears!

Instead, we must load it in an area that is always RAM. I chose 8000H. You can use DDT to load the hex file and run it. ("DDT MAP2.HEX", then "G8000")

Does it work? Absolutely! Remember to enable bank switching with config setting **0958**:

```
 MPU-B Control Byte Test        R=ROM, W=RAM

|      0000111122223333444455556666777788889999AAAABBBBCCCCDDDDEEEEFFFF |
| BYTE 048C048C048C048C048C048C048C048C048C048C048C048C048C048C048C048C |
|  40  WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWRRWWWWWWWW |
|  80  RRWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW |
|  C0  WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW |
|  00  RRWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWRRWWWWWWW |
```

Notice how ROM can be moved and removed programmatically. The first colored line, with a port value of 40h, shows the ROM at D800. The next line, corresponding to a port value of 80h, shows ROM at 0000. The last line, with a port value of 00, shows the ROM at 0000 and D800, which is the default at startup. Rerun the program with NVS_BANK_ROM off, and ROM will be located at D800 for all control byte values.

This program does not use any CP/M calls, so we can also run map2 from the IMSAI monitor. Here are the steps to do it:

1. Eject the system disk from drive A:
2. Press <ext.clr> on the front panel
3. Press the <space> bar: you should see "IMSAI IEEE MONITOR VERS 1.0"
4. Press "H" with no return, and the system will wait for your hex file.
5. Open the hex file with a windows editor (I use Notepad++) and select everything *up to and including the ":00" on the last line.*
6. Paste it *as plain text* (Ctrl-Shift-V) into the TTY window. It should slowly transfer.
7. At the monitor '?' prompt, press "J", then "8000" <enter>

```
:1880000031008421618OCD59803E00D3F3CD2C803E40D3F3CD2C803E93
:1880180080D3F3CD2C803EC0D3F3CD2C80214B81CD598076212181CDBB
:1880300059801100040640210000 4E3EDC77BE71E528052133811803D3
:18804800213F81CD5980E11910E8212481CD5980C97EB7C8D302231865
:18806000F80D0A20204D50552D4220436F6E74726F6C20427974652083
:1880780054657374202020202020202020523D524F4D2C20573D52414D13
:188090000D0A0D0A7C2030303030313131313232323233333333334348E
:1880A80034343535353536363636373737373838383839393939941418A
:1880C000414142424242434343434444444444545454546464646207C3A
:1880D8000D0A7C203034384330343843303438433034384330343843382
:1880F000303438433034384330343843303438433034384330343843433E
:1881080030343843303438433034384330343843303438433207C0D0A51
:18812000007C20001B5B33396D1B5B34396D207C0D0A001B5B34316D11
:188138001B5B39376D52001B5B34346D1B5B39376D57000D0A5379733F
:0C81500074656D2048616C7465642E003D
:00000001FF
```

Interestingly, ROM is enabled during memory reads but NOT during memory writes. Therefore, when ROM is enabled by clearing either bit 6 or 7, writing to a ROM address results in writing to the "underlying" RAM. (You'll have to wait until ROM is disabled to read it, though.) This behavior has unintended consequences for the memory mapping program.

Consider the situation where an operating system is running in RAM, and the memory configuration is changed so that the OS is overlayed with ROM. The mapper program works by writing a test byte to memory and then reading it back. The assumption is that writing to

ROM will fail with no side effects. In the case of our control-port-enabled ROM, however, we are unwittingly writing to and corrupting the underlying RAM. Bottom line: once you run this program, you can't go back to the operating system. You will need to reload CP/M from disk.

The IMSAI 8080 kit documentation says that ROM banking is only compatible with the MPU-A/B ROM images. However, ROM banking also works in memory map 0 (64K RAM, no ROM image). To see it in action, set your configuration to **0858**. Now create a small, front-panel-friendly program to write to the MPU-B control port:

```
0011   8000             .ORG 8000H              ; put this program in RAM
0012   8000 DB FF           IN A,(0FFh)         ; get value from input switches
0013   8002 D3 F3           OUT (0F3h),A        ; send to shadow ROM port
0014   8004 2F              CPL                 ; complement
0015   8005 D3 FF           OUT (0FFh),A        ; send to output LEDs
0016   8008 18 FE      fin: JR fin              ; done (infinite loop)
```

This 9-byte program reads the input switches and copies them to port F3 and the output LEDs. Using the front-panel switches, enter the following bytes at address 8000 (or anywhere in RAM, for that matter): DB FF D3 F3 2F D3 FF 18 FE. Set bits 6 & 7 of the input switches according to the desired control byte, run the code, and start examining memory.

When you send 0 to port F3, enabling ROM, you will see the following pattern:

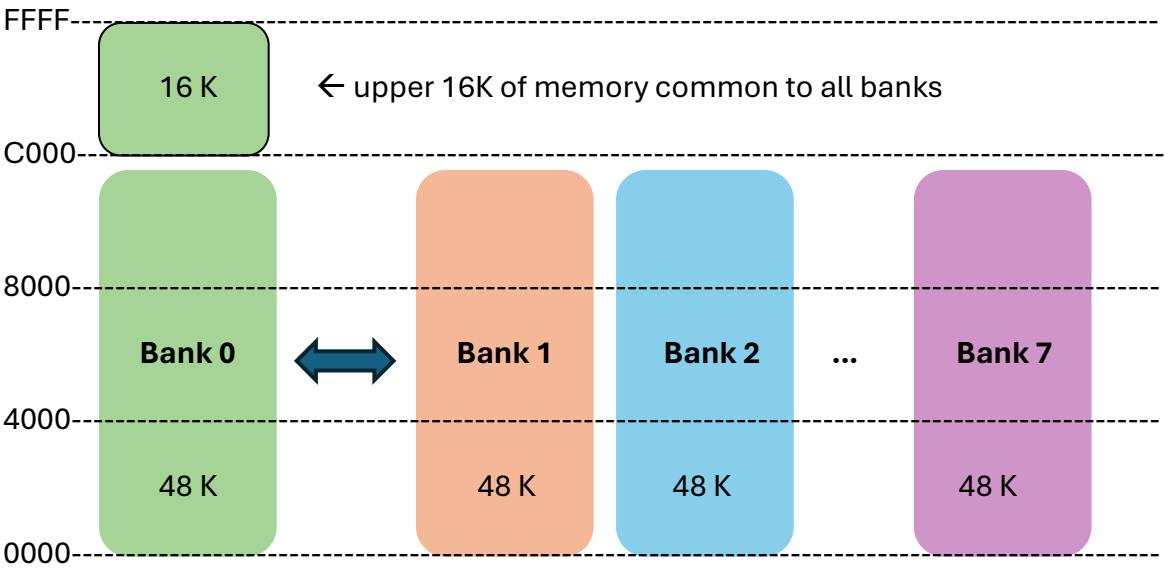
- 0000-7FFF is ROM. <Examine> shows 0. You cannot deposit any values*
- 8000-D7FF is RAM. <Examine> shows data. You can deposit data.
- D800-DFFF is ROM. <Examine> shows 0. You cannot deposit any values*
- E000-FFFF is RAM. <Examine> shows data. You can deposit data.
-

**It isn't quite true that data is not deposited; the data goes to RAM but you can't see it while the ROM is enabled*. Run the code again, disabling ROM with value C0h. After the ROM is disabled, the data you previously entered will miraculously appear.

RAM bank switching: basic information.

The IMSAI 8080's second bank switching method involves RAM rather than ROM. It lets us swap large chunks of RAM in and out of the 64K address space, thereby increasing the amount of available RAM beyond the 64K addressing limit.

In RAM bank switching, there is 16K of RAM in upper address space (C000-FFFF), which is always present, and a series of RAM banks, each 48K in size, which can be switched in and out of lower memory (0000-BFFF). The system memory is Bank 0.



By default, the emulated IMSAI 8080 sees only 64K of address space with Bank 0 in the lower 48K. The additional banks of RAM are not normally accessible. CP/M 2.2 does not use them. But their existence is confirmed by of the kit's listed features: "An MMU with Banked memory management to enable CP/M 3.0 (Plus) with banked memory."

There you have it: the IMSAI 8080 kit provides a memory management unit (MMU) that enables banked RAM. This MMU gives us 7 additional banks of 48K that can be switched into lower memory. CPM 3.0 uses it, but CP/M 2.2 does not. Continue reading if you'd like to learn more about the MMU and how you can use it yourself – even from within CP/M 2.2

RAM bank switching: deeper dive.

Bank switching for RAM is a single OUT instruction, just like it is for ROM. Port 40h controls which RAM bank is currently selected. Send a 2 to port 40h, and you've just selected RAM bank #2. It's that easy!

Let's write a short assembly language program to access one of the available RAM banks. Consider the following code, bank1.asm:

```
MMU     .EQU 40H

.ORG    100h
    LD  A, 2
    OUT (MMU),A     ; select memory bank #2
                    ; do stuff in bank #2
    LD  A, 0
```

```
    OUT (MMU),A     ; return to bank #0
    RET             ; return to CP/M
.END
```

It doesn't work. CP/M dies. What's going on? Answer: as soon as the first OUT instruction is executed, the lower 48K of memory, *including this program*, is switched out of the address space. The LD A,0 instruction never gets executed, because it is *gone*. It is replaced with whatever data happens to be in RAM bank #2. The last three instructions are never executed.

To avoid getting switched out, our program must reside in the upper 16K common area, beginning at C000. CP/M should also be completely contained in the common area. We cannot use the default 54K system, since its CCP begins at BC00 (see chart above.) The 60, 62, and 64K systems are all OK.

As a second attempt, move to a 62K CP/M system with banked ROM (configuration **0958**, 8958, or similar). Change the program origin to C000. See bank2.asm.

```
.ORG    0C000h      ; reside in nonbanked RAM

LD  A, 2        ; select memory bank #2
OUT (MMU),A     ; make it so.
                ; do stuff in bank #2
LD  A, 0        ; return to bank #0
OUT (MMU),A     ; make it so.
JP  0000        ; restart CP/M.
```

We can't run this code in the normal CP/M way, since it doesn't reside at 0100H. But we can load and run it from the debugger. "DDT bank2.asm", then start it within DDT with the go command: "gC000". Although the program doesn't appear to do anything, it bank-switches RAM without crashing. That's something to build on.

In a previous article I showed how to display a page of memory. Let's fill a bank of memory with some value, then show the same address space for bank 0 and the selected bank:

```
;  Put a page of 4's in bank 4
;
    LD   A,4
    OUT  (MMU),A     ; switch to bank 4
    LD   HL,TPA      ; point to TPA in b4
    LD   A, 44h      ; and fill it with 4's
    CALL FILL        ; fill a page of memory
;
;  Show memory in banks 0 and 4
;
    LD   A,0
    OUT  (MMU),A     ; switch to bank 0
    LD   HL,TPA      ; point to TPA
```

```
CALL DUMP        ; and show it

LD   A,4
OUT  (MMU),A     ; switch to bank 4
LD   HL,TPA      ; point to TPA
CALL DUMP        ; and show it
```

The complete code is in bank3.asm. Like before, load and run it from DDT, remembering that it resides at C000. It will switch between two versions of memory at 0100h: (a) system memory, containing portions of DDT; and (b) an alternate bank, containing a page of 4's.

You can load and run programs within banked RAM. Check out example program bank4.asm, which:

- Fills memory in banks 4 and 5
- Deploys a "Hello, World" program in bank 5
- Copies memory from bank 4 to bank 7, then
- Runs the program in page 5.

Finally, it would be nice to invoke our program from CP/M rather than DDT. To do that we'll need to write a loader program that copies our primary code into the common area and jumps there for further execution.

Copying code into high memory is straightforward, requiring only 4 instructions:

```
; Now move primary code into high memory (C000)
; and then execute it.

    LD   HL,SOURCE      ; source addr
    LD   DE,DEST        ; destination addr
    LD   BC,CODELEN     ; code length
    LDIR                ; move it!
    JP   DEST           ; and start execution there
```

The full loader program is found in bank5.asm. It requires that the primary "payload" program is at a fixed location in memory. We can use DDT to append our loader program with the payload. To do this:

1. Assemble the loader program, "bank5.asm", making sure that the payload address (140h) follows the last byte of the loader (12Eh).
2. Calculate the proper byte offset that will load a hex file for C000 to the payload address of 0140h. Answer: 10140-C000 = 4140h.
3. Invoke DDT with the loader program: DDT BANK5.HEX
4. Append it with the payload program: "iBANK4.HEX", then "r4140"

5. Note the address displayed as the next byte: 02AE, which indicates the total size of the loader + payload = 02AE-0100 = 01AE. It will therefore require 2 disk pages.
6. Leave DDT, save the result as a com file: "SAVE 2 BANK.COM"

The resulting file, bank.com, runs in CP/M 2.2. The front panel lights flicker around address C000 (bits 14 and 15 illuminated) when the program is running.

That's all for now.

I've summarized what I've learned about bank switching on the IMSAI 8080, and how it can be accessed under CP/M 2.2. Please let me know if I've missed something important.

Happy Hacking!

Bruce.

Last Updated: 12 July 2025

Other articles in this series: IMSAI 8080 articles