Floppy Disks
and the IMSAI 8080

by Bruce E. Hall, W8BH

In 1980, I acquired my first computer with a floppy disk drive. Disk drives were a fantastic upgrade over the system ROM and the cassette tape interface. The disk-based operating system was loaded via the all-important "system disk". This system disk had special status. It was marked differently and often kept in a special place. In 1980 this was obvious and common sense. But after 40+ years I had forgotten what my teenage self knew well. Below is my reacquaintance with the mighty system disk as it applies to CPM 2.2 and the IMSAI 8080 replica from The High Nibble. I will also cover disk images and how CP/M 2.2 stores files on floppy disks.

What is a system disk?

A system disk contains the CP/M operating system. A system disk can be booted, which means that CP/M can be loaded and run from the disk. Attempts to boot CP/M from a non-system disk will fail and result in a non-operative system.

For the 8" floppy disks used by the IMSAI8080, the first two tracks are reserved for the operating system. Normally these tracks are unavailable for other purposes; they can only be used to hold a copy of the operating system.

Table of Contents

How to create a system disk
What do SYSGEN and MOVCPM do?
Roll your own CP/M
A look at .SYS files
Adding an auto-executing file
System disk anatomy
The Disk Image (.DSK) file
CP/M file storage

How can you tell if a disk is a system disk or not? The easiest way is to try it! Mount the disk in drive A, press the <ext clr> front panel switch, and see if you get a CP/M prompt. On my current set of disks, "cpm22b01" is a system disk, whereas "Zork" is not.

Acquiring a system disk is much easier than it was in 1980. Download a disk image from GitHub and you're done! But it is instructive to learn how to create one yourself. Below are several ways to create a bootable disk.

How to create a system disk for CP/M 2.2

Here are five methods. The easiest is Method #1, copying a known system disk.

Method 1: copy a system disk on your personal computer

- On the virtual desktop, click on LIB: to see your collection of disk images.
- Drag a known system disk to the download icon at top-right. This will put a copy of the disk image in your downloads directory.
- Make a copy of the disk image on your PC, giving it a new name: system.dsk, etc.
- Drag-drop this renamed disk image back to LIB:

Method 2: using SYSGEN only

- Place a system disk in drive A and a non-system disk in drive B
- Within CP/M, go to drive A and type SYSGEN<return>
- When prompted for source disk, type A<return>
- When prompted for destination disk, type B<return>

Method 3: using MOVCPM and SYSGEN

- Run MOVCPM xx * to create CP/M for a system with xx Kbytes of memory
- Use SYSGEN to save CP/M to another disk (typing <enter> for source drive).

Method 4: using the provided batch file

- Within CP/M, type "SUBMIT sysgen54" to create CP/M for a 54K system. This process will take time as the customized boot and BIOS are assembled and combined with CP/M.
- Use SYSGEN to save CP/M to another disk (typing <enter> for source drive).

Method 5: using an existing .SYS file

- Using the CPMxx.SYS file created by Method 4, copy the cp/m system file to the TPA using DDT. For example, "ddt cpm54.sys", then "g0"
- Use SYSGEN to save CP/M to another disk (typing <enter> for source drive).

You'll notice that several of these methods use the transient CP/M commands, MOVCPM and SYSGEN. Copies of these are usually found on the system disk.

What does SYSGEN do?

SYSGEN is a CP/M command that creates a bootable CP/M system disk. It copies CP/M from a system disk to the TPA at address 0900h and then writes it to another disk. SYSGEN is the only command needed to copy CP/M from one disk to another: see Method #2 above.

What does MOVCPM do?

Address	Component
0900	Bootloader
0980	Start of CP/M (the CCP, then BDOS)
1F80	Start of BIOS
22FF	Last byte of CP/M

Good question! It moves a copy of CP/M into the TPA at address 0900h, shown at right. This memory image is used by a subsequent SYSGEN operation to create a system disk. MOVCPM allows the user to specify the system memory size so that CP/M will install into the last/highest 7K of that memory space, adjusting all internal addresses accordingly. Method #3 uses MOVCPM and SYSGEN to create a system disk.

MOVCPM is distribution-specific. It contains a fixed copy of CP/M. Until recently, the MOVCPM in build B03 of z80pack contained the B02 version of the BIOS! One should take care to keep MOVCPM with the system it belongs to.

Rolling your own CP/M with customized bootloader & BIOS

Method 4, which uses SUBMIT to process a special batch file, creates a new CP/M system by patching in a customized bootloader (cboot) and customized BIOS (cbios). This process of altering CP/M is described in section 6 of the CP/M operating system manual (p 6-1, or page 192 of 317).

It is worthwhile to study this batch file. For example, if one does “SUBMIT SYSGEN60.SUB”, it runs a batch file which does the following:

- Modifies & assembles the bootloader & BIOS with a new memory size (60KB)
- Runs MOVCPM, creating a copy of a CP/M with a new memory size.
- Via DDT, adds the modified bootloader and BIOS to the new CP/M.
- Finally, a SAVE command saves 2200h bytes in memory to file cpm60.sys, which contains all components of the new CP/M system.

When the batch file completes, the new CP/M is still in the TPA, and can be written to another disk via SYSGEN - just hit return when prompted for source drive name.

Let's study the batch file in detail. Here is the batch file for creating a 60K CP/M system. To run it, type "SUBMIT sysgen60". Values that are specific to creating the 60K system are highlighted.

XSUB	XSUB extends SUBMIT functionality by feeding line input to any programs called within the batch file. For example, when the line editor ED is invoked below, subsequent lines in the batch file are fed to ED and treated as Editor commands.
ERA *.SYS ERA *.HEX ERA *.PRN ERA *.SYM	Erase all files generated from a previous CP/M generation
ED BOOT.ASM	Begin editing the bootloader source code
20AFMSIZE 11C2DI**60**^Z E	Editor commands for finding the line containing MSIZE and changing its value to "60" (the default is 54)
ED BIOS.ASM	Begin editing the custom BIOS source code
20AFMSIZE 11C2DI**60**^Z E	Like above, find the line containing MSIZE and change the memory size parameter to "60" (the default is 54)
ERA *.BAK	Erase the ED backup files that were generated above
ASM BOOT MAC BIOS	Assemble the modified bootloader and BIOS, creating hex file outputs for each.
DDT MOVCPM.COM	Load MOVCPM into memory, preparing it for execution
l**60** * g	Execute MOVCPM with parameters "60 *", which creates a 60K version of CP/M in memory starting at 0900h
SAVE 34 CPM**60**.SYS	Temporarily save the new CP/M as file CPM60.SYS
DDT CPM**60**.SYS	Load it back into memory from 0100h to 2300h
iBOOT.HEX r**900**	Add bootloader with a bias of 900h, placing it at 0900h
iBIOS.HEX r**3580**	Add the BIOS with a bias of 3580h, placing it at 1F80h
g0	Leave the DDT debugger
SAVE 34 CPM**60**.SYS	Save the amended CP/M as file CPM60.SYS. 34 pages of 256 bytes each = 2200h = file size of 8.5K, which now contains the bootloader + CCP/BDOS + custom BIOS

One detail needs further explanation. The second invocation of DDT adds the modified bootloader and BIOS. Recall that they were compiled with specific starting addresses (origins). But both must be loaded into TPA memory at locations different from where they

execute. They cannot be loaded directly to their execution addresses. Loading offsets are required.

The bootloader, which loads the operating system from disk, executes from the start of memory. It is therefore compiled with an origin of 0000h. The resulting hex file, boot.hex, normally loads at 0000h. However, to use this file for creating an operating system disk with SYSGEN, it must be loaded at 0900h instead.

DDT accomplishes this task with its input (I) and read (R) commands. The filename is specified by the input command and loaded into memory with the read command – which can take an optional offset, or bias. For example, IBOOT.HEX sets up BOOT.HEX for a subsequent read operation (Internally, it copies the string "BOOT.HEX" to the file control block at 005Ch). Then R900 reads the file, adding an offset of 0900h.

Similarly, the custom BIOS in a 60K system is located at EA00. We need the TPA copy to reside at 1F80. What bias is required to load it there? Mathematically, EA00 + bias = 1F80. No positive bias can make this equation work... until we realize that 11F80 is equivalent to 1F80 in the 16-bit space. Bias is then 11F80-EA00 = 3580h. Try it: adding 3580h to the normal loading address of EA00h results in 1F80h (with an ignored carry). The table below shows another way of calculating it.

CP/M was designed and coded to run in a system with 20K of memory, with CP/M occupying the upper 7K and user programs occupying the lower 13K. In this minimal 20K system, CP/M starts at memory address 3400h and the BIOS is 5½ K higher, at 4A00h. As system memory increased, the location of CP/M changed correspondingly. Its location is often specified as an offset, or bias, from the original 20K system. For example, the location of CP/M in a 54K system is 3400h with a bias of 8800h, or BC00h (3400h + 8000h = BC00h). Below is a table of CP/M component locations for various system sizes.

Item	TPA	20K	54K	60K	62K	64K
Bias from 20K system	D580	0	8800	A000	A800	B000
Start of CP/M (the CCP)	980	3400	BC00	D400	DC00	E400
CCP command buffer	987	3407	BC07	D407	DC07	E407
Start of BDOS	1186	3C06	C406	DC06	E406	EC06
Start of BIOS	1F80	4A00	D200	EA00	F200	FA00
End of BIOS +1	2300	4D80	D580	ED80	F580	FD80

How much offset is needed to copy CP/M into the TPA? It depends on the system size. See the bias values listed in the table above. The offset needed = D580 – bias. For example, the

offset to translate a 54K system to TPA = D580-8800= 4D80h. The BIOS in a 54K system, located at D200, will be moved by MOVCPM to memory address D200 + 4D80 = 1F80h.

For the typical system having 64K memory, it makes sense to put CP/M in the highest address space possible, thereby maximizing the amount of memory available for user programs in the TPA. However, we do not have to. In fact, there are very good reasons not to maximize CP/M memory! We may wish to reserve memory that cannot be touched by CP/M. For example, the CRT video device puts its drivers at F800h. Specifying a smaller CP/M memory size will allow those drivers to operate without fear of being overwritten.

A look at the CPM54.SYS file

The fifth method for creating a system disk uses the CPM54.SYS file. This file can be found on the CPM22 system disk. The "54" in the filename indicates its use to create a 54K CP/M system. This file is an *image* of the CP/M system. It does not "run". Its contents are meant to be transferred to another disk via the SYSGEN command.

We can use DDT to load this system image into the TPA, then study its contents. Just as a reminder, this file is loaded by DDT starting at 0100h. To determine the corresponding file offset location, therefore, you must subtract 0100h.

File Offset	TPA address	Description
0000	0100	Remnants of previous MOVCPM command…
01B6	02B6	For example, "Ready for "SYSGEN" or "SAVE 34…""
0300-07FF	0400-08FF	Almost all zeros; will not be transferred to the system disk
0800	**0900**	**Start of Boot Loader**… JMP 000A
085B	095B	End of Boot Loader = C3 22 00 = JMP 0022.
0880	**0980**	**Start of CCP**… JMP BF5C, JMP BF58
0887	0987	CCP command buffer, starting with length byte
08C0	09C0	"Copyright…DIGITAL RESEARCH" 00 00 00 00
102D	112D	End of CCP = "$$$ SUB" 00 00 00…`
1086	**1186**	**Start of BDOS**… JMP C411
1E25	1F25	End of BDOS… block of zeroes 00 00 00 00 00
1E80	**1F80**	**Start of BIOS**… Jump table JMP <>, JMP <>, JMP <>, etc C3 EE D2 C3 61 D3 C3 DE D3 C3…
21FF	22FF	Last byte of file

Adding an auto-exec batch file to CP/M

Remember the useful AUTOEXEC.BAT file in MS-DOS? You know, the batch file that runs on system boot, handling your choice of startup chores. CP/M 2.2 does not have one, but there is a sneaky way to do it: hard code a command into CCP's command buffer. When CP/M starts, the command executes. If our command is SUBMIT <filename>, then batch file <filename> is processed on startup.

The value stored at address 0986h, 7Fh, sets the buffer length at 127 bytes. The bytes following that are the buffer itself, beginning with the command length byte at memory location 0987h. See the highlighted line in the table above.

If your system already contains the string "SUBMIT PROFILE" in the CCP command buffer, you are all set. Your system should run the batch file PROFILE.SUB on cold boot. If it doesn't, here is how to patch it in:

1. Copy CPM54.SYS into memory with debugger: ddt cpm54.sys
2. Issue at set command starting at 0987: "S987"
3. You will be shown the value at 0987h and prompted to change it. Change it to 0E<enter>, which is the 14-character length of the string you are about the enter.
4. Continue changing the remaining values, pressing <enter> after each value:

   0987: 0E 53 55 42 4D 49 54 20 50 52 4F 46 49 4C 45 00

5. When done, type a period<enter> to stop data entry
6. Check your work: D980
7. Exit the debugger: g0
8. Save your patch (optional): Save 34 CPM54.SYS
9. Run SYSGEN to create a system disk with the patch. When prompted for the source drive, press <enter> to choose memory contents.

Disk Anatomy

The floppy drive for the IMSAI 8080 was an 8" single-sided, single-density drive. The disk format it used is known as IBM-3740, which is also the name of the data entry system that first used it. IBM-3740 is the direct ancestor of the IBM PC floppy disk format.

- Data is transferred in 128-byte units, called records.
- Each data record occupies exactly one sector of the disk.
- There are 26 sectors, numbered 1-25, in a circular track on the disk. Therefore, a track holds 26 * 128 = 3328 bytes of data.

- There are 77 concentric tracks on the disk, numbered 0-76. Therefore, a disk holds 3328 * 77 = 256,256 bytes of data.

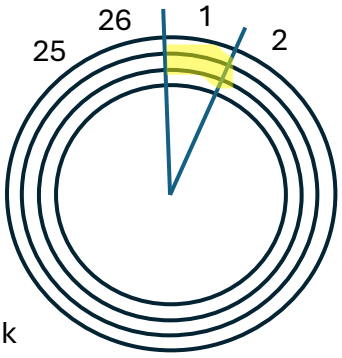
System Disks

CP/M 2.2 reserves Tracks 0 & 1 for system files, *whether or not they are used*. The file directory immediately follows on Track 2, 16 sectors in length. The remaining 241K is available to the user.

Track	Sector(s)	Contents
0	1	Bootloader
0	2-17	CCP
0	18-26	BDOS
1	1-19	BDOS, continued
1	20-26	BIOS
2	1	Start of file directory

See table 6-3 on page 6-14 of CP/M 2.2 Operating System Manual for more information.

That all seems nice and tidy! Unfortunately, reality is a little more complicated. Study a system disk and you will indeed find that the bootloader, CCP, BDOS, and BIO are stored in sequential sectors of the first two tracks. The file directory starts at the beginning of Track 2. However, the directory does not occupy the first 16 contiguous sectors of that track. Its contents are spread throughout the track, out of order. Why? Because early computing hardware could not process data as fast as it could be read. By physically spreading out the data, data processing for one sector could be finished by the time the read head arrived the next sector, improving overall efficiency.



Physical vs Logical Sectors

Sectors are sequentially numbered 1 through 26. These sector numbers correspond to their physical location on the track. CP/M writes and reads sectors in a nonconsecutive order to improve throughput, as discussed above. The physical sectors of a whole track are read/written in the following order:

1, 7, 13, 19, 25, 5, 11, 17, 23, 3, 9, 15, 21, 2, 8, 14, 20, 26, 6, 12, 18, 24, 4, 10, 16, 22

Look closely, and you will see there is a separation of 6 physical sectors from one logical unit to the next: 1 to 7, 7 to 13, 13 to 19, etc. The “skew” of this CP/M system is 6.

Any given sector can be referred to by its physical location or by its logical one. For example, the second logical sector, highlighted above, is physical sector #7; logical sector 3 is physical sector 13, etc. When referring to a particular track and sector, such as “Track 2, Sector 15”, we typically are referring to the physical sector.

So, which numbering system is used in CP/M? The BIOS, which acts the interface between CP/M and the disk hardware, refers to sectors according to their physical location. However, BDOS uses these sectors in their logical (skewed) order.

The Disk Image

We don't use an actual floppy disk with our emulated system. We use a disk image instead, which is a file that represents the disk contents.

The floppy disk image is a 251K file. It contains, byte for byte, the contents of an ibm-3740 formatted 8" Single-sided, Single-density floppy disk, beginning with Track 0, Sector 1 and ending with Track 76, Sector 26. There is no other information in the disk image file. As expected, the sectors are ordered sequentially, mimicking their physical location on the floppy.

The original floppy disks were initialized with the value E5h at every location. To create a blank disk image: open a hex editor, create a file with 256,256 bytes of E5h, and save it with the .DSK file extension. Done! I prefer initializing my blank disks with the value 00h instead, lightly editing them to ensure an empty file directory. These "clean" blank disks are on my GitHub as blank.dsk and blankE5.dsk.

As indicated above, the bootloader, CCP, BDOS, and BIOS are stored on the first two tracks of a system disk. They are stored in **physical sector order**. Remember, the bootloader is responsible for moving these tracks into memory *before* CP/M runs. There is no skew.

Once CP/M is booted, however, the disk is used according to the logical topology. The file directory, beginning with Track 2, and everything else stored on the disk (and disk image file), is in logical order.

With those facts in mind, let's look at the organization of a disk image file. The first two tracks are reserved for storing the system files. The remaining space on the floppy drive consists of 75 tracks * 26 sectors/track = 1950 sectors (243K). The CP/M filesystem divides this space into 243 blocks, each block being 8 sectors or 1K in size. The first two blocks (Block 0 and Block 1) are used by the directory, leaving 241 blocks for files. Blocks are allocated sequentially, starting with Block 2 and ending with Block 242.

The following table shows the contents of a floppy disk image, giving the byte offset for components in the first three tracks. Track 2 details are shown to demonstrate organization of the file directory and the first two data blocks.

Byte Offset	Track	Physical Sector	Logical Sector	Contents
0000	0	1	n/a	Bootloader
0080	0	2	n/a	CCP & BDOS
0D00	1	1	n/a	Start of Track 1
1680	1	20	n/a	BIOS
1A00	2	1	1	**Directory entries 1-4**
1A80	2	2	14	Directory entries 53-56
1B00	2	3	10	Directory entries 37-40
1B80	2	4	23	<Block 2, 7th sector>
1C00	2	5	6	Directory entries 21-24
1C80	2	6	19	<Block 2, 3rd sector>
1D00	2	7	2	Directory entries 5-8
1D80	2	8	15	Directory entries 57-60
1E00	2	9	11	Directory entries 41-44
1E80	2	10	24	<Block 2, 8th sector>
1F00	2	11	7	Directory entries 25-28
1F80	2	12	20	<Block 2, 4th sector>
2000	2	13	3	Directory entries 9-12
2080	2	14	16	Directory entries 61-64
2100	2	15	12	Directory entries 45-48
2180	2	16	25	<**BLOCK 3, 1st sector**>
2200	2	17	8	Directory entries 29-32
2280	2	18	21	<Block 2, Sector 5>
2300	2	19	4	Directory entries 13-16
2380	2	20	17	<**BLOCK 2, 1st sector**>
2400	2	21	13	Directory entries 49-52
2480	2	22	26	<Block 3, 2nd sector>
2500	2	23	9	Directory entries 33-36
2580	2	24	22	<Block 2, 6th sector>
2600	2	25	5	Directory entries 17-20
2680	2	26	18	<Block 2, 2nd sector>
2700	3	1	1	<Block 3, 3rd sector>
…	…	…	…	…
3E8FF				Last byte of Disk Image

The first four lines represent bytes 0 through 19FF of the disk image where CP/M is stored. You may view the file in a hex editor to see each component.

The file directory starts on the yellow highlighted line, at byte location 1A00h. Each directory entry consists of a 32-byte record. Therefore, 4 directory entries fit into a sector. Question: where is the fifth entry located? Remember that the skew of this filesystem is 6,

therefore jump 6 physical sectors to reach the next logical one. Answer: at byte location 1D00h, six lines & six physical sectors down. Can you find it?

The disk directory holds a maximum of 64 entries. It is 16 sectors in length (64 entries / 4 entries per sector = 16). The remaining 10 sectors on Track 2 are the first to be allocated for file storage. Recall that these sectors are allocated in blocks of 8 sectors each. Even a small file of, say 50 bytes, will use up 1K of disk space.

File allocation begins the start of Block 2, on the orange-highlighted line. What does that mean? The first file stored on this disk will be located at byte offset 2380h in the disk image file. If the file is longer than a sector, it will continue in the **next *logical* sector**. Block 2 starts at logical sector 17, so where is logical sector 18? Answer: six lines down, at byte offset 2680h. Did you find it? Blocks cross track boundaries, as seen with Block 3.

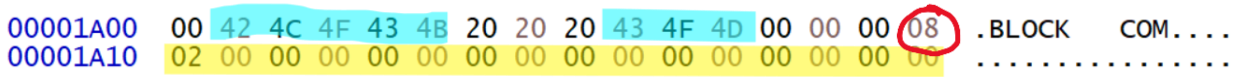
For more information see Block.bas, an XYBASIC program that shows the disk image offsets, tracks and sectors for any given block on a floppy drive.

CP/M file storage

Seeing is believing. And learning. I created a special file that is exactly 1 K (1 block, or 8 sectors) in length. The first sector is filled with hex value 01, the second sector with hex 02, etc. See how CP/M saves it to disk via the following exercise:

1. Create a blank disk (or use a clean one from my GitHub) and mount it as drive C.
2. Download block.hex and copy it to drive A.
3. Type: LOAD BLOCK to convert it to a .COM file. LOAD should report its length as 8 records (0400h bytes).
4. Type: PIP C:=BLOCK.COM, copying the file to drive C.
5. Type: DIR C: to confirm that the C drive contains one and only file, BLOCK.COM

Now let's examine the raw contents of this floppy. Eject the disk from drive C. Then, go to the library and drag the disk to the download icon in the top-right corner of the LIB window. The disk image will appear in your PC's download folder. Open the disk image file with your favorite hex editor (I use HxD). At file offset 1A00h, which is the location of the first directory entry, you should see something like the following:



The first byte, 00, indicates this file is in the default user space. The next 11 characters are the 8-character filename, padded with spaces, and 3-character extension. The byte "08" indicates the length of the file in sectors. The 16 bytes starting at file offset 1A10h, highlighted in yellow, are a list of the blocks allocated to this file. Each directory entry

contains up to 16 block IDs = 16K per associated file. Files larger than 16K therefore use multiple directory entries.

As expected, our special file is one block in length, Block #2, the first available block on this disk. See Obsolescence Guaranteed for a good file directory description.

From the Disk Image table above, can you figure out where Block 2 starts? It should be at byte offset 2380h, which represents Track 2, Sector 20 on a real floppy disk. Your hex editor will display something like this:

```
00002380  C9 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01  É...............
00002390  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01  ................
000023A0  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01  ................
000023B0  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01  ................
```

You should see a 128-byte sector filled with 01h. The first byte was changed to C9 (RETurn instruction) so that attempts to execute the file will gracefully return control to CP/M. Consulting the table again, the second sector of Block 2 should be at file offset 2680h:

```
00002680  02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02  ................
00002690  02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02  ................
000026A0  02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02  ................
000026B0  02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02  ................
```

Continue tracing the file through all 8 sectors, going to 1C80h, 1F80h, etc. You can then similarly create BLOCKA.COM and copy it to the disk, confirming that it occupies the sectors of Block 3.

Conclusion

Armed with the knowledge in this article, you can create your own blank disks and system disks. You can adjust the amount of memory available to CP/M. And by knowing how CP/M stores files on disk, you can dissect the contents of any floppy image file. Happy hacking.

Bruce.

Other articles in this series: Assembly on the IMSAI 8080, Assembly and the Stack.

Last Updated: 12 July 2025

Other articles in this series: IMSAI 8080 articles