CPUville Z80 Computer Disk and Memory Expansion Instruction Manual

for version 2 with gate-array logic IC

By Donn Stewart

Table of Contents

Introduction

The CPUville Disk and Memory Expansion Board is designed to provide the CPUville Z80 computer with 64K RAM memory and an IDE disk interface, allowing the computer to run a disk operating system, such as CP/M[1]. If CP/M is installed, this gives you access to hundreds of programs in the public domain that will run on the Z80 computer. You need the CPUville serial interface board in order to operate the computer with the disk and memory expansion attached.

The disk interface is an IDE interface (also known as parallel ATA, or PATA), that will accommodate an IDE-compatible drive, including compact flash and SD drives with the appropriate adapter. Only the lower 8 bits of each disk data word are transmitted to the Z80 system, so drives that operate in 16-bit mode will have only half the disk space available. This is a trade-off to keep the price of the kit low, because extra hardware would be needed to capture all 16-bits, and convert it to 8-bits for the Z80 data bus. However, since the PATA interface has been replaced by the serial ATA (SATA) interface in commercial computers, IDE drives are now obsolete, and IDE drives with sizes of hundreds of megabytes or even gigabytes are very inexpensive. 8-bit programs are very small, so literally thousands of Z80 programs would fit into a disk space of 100 megabytes. If you have an IDE drive of a few hundred megabytes or more, you can run CP/M and store all the 8-bit data you could reasonably want using the CPUville Z80 computer with the disk and memory expansion.

Testing has shown that not all mechanical IDE drives will work with the current version of the disk and memory expansion board and ROM program. This instruction manual includes a Table of Tested Disk Drives on page 80 that shows which drives work and which don't. I am trying to figure out why some drives don't work. I will update the list of drives as I find out more.

Power for a disk drive that requires only low-current +5V, such as a solid state drive, can be provided by the logic probe connector on the computer board, or by pin 20 on the IDE connector, as described in detail in the section “Building the Disk and Memory Expansion Board”. Power for a drive that requires +12V, or a drive that draws a lot of current from the +5V supply, will need to be provided by an appropriate power supply. Usually, a hobbyist will have an old computer system power supply that can supply both the regulated +5V for the computer, and +12V for the disk drive. These power supplies can also be obtained cheaply. If two separate power supplies are used, they need to share a common ground.

When the disk and memory expansion board is attached to the Z80 computer, the memory and input/output ports on the main computer board need to be disabled by removing jumpers JP1 and JP2. This is because the disk and memory expansion board provides the computer with a new set of input/output ports and memory ICs. Of course, if the disk and memory expansion board is removed, replacing the jumpers restores the ports and memory of the computer board to their original condition. You do not need to physically remove the ROM or RAM from the computer board when they are disabled.

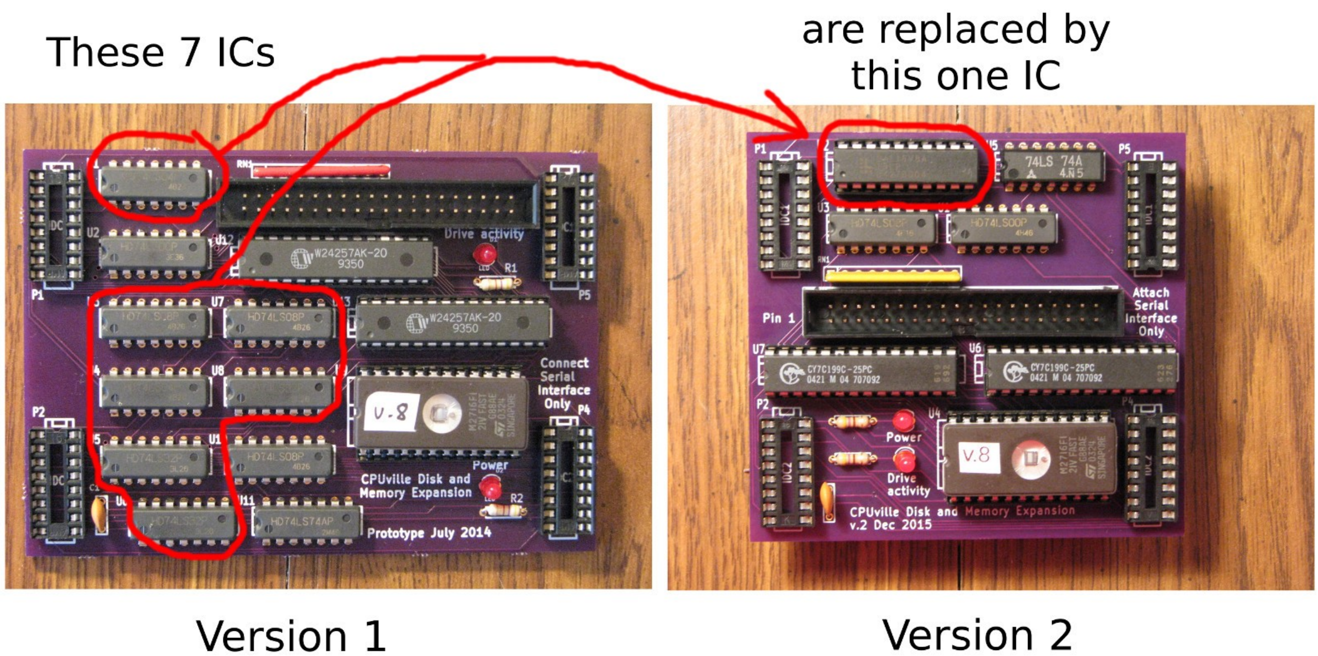
The memory expansion provides two memory configurations. Configuration 0 has 2K of ROM from location 0x0000 to 0x07FF, and 62K of RAM from 0x0800 to 0xFFFF. Configuration 1 is all-RAM, that is, it has 64K of RAM from location 0x0000 to 0xFFFF. The configurations are selected by

1 CP/M is a registered trademark, currently owned by Lineo, Inc.

software OUT instructions to port 0 or port 1, respectively. This system is necessary because the Z80 computer executes code starting at 0x0000 when taken out of reset, so we need ROM there when the computer starts. But, CP/M needs RAM in locations starting at 0x0000. Hence the need for the two configurations.

The expansion board comes with a 2K EPROM with ROM code version 8 and above. It contains the same monitor program and commands as the computer board ROM version 7, with additional commands to read and write disk sectors, and to start CP/M. The version 8 ROM code lacks the simple programs, such as the port reflector, that are present in ROM versions 7 and lower, that use the computer board switches and LEDs for input and output. These computer board input and output ports are disabled when the disk and memory expansion board is in use, as mentioned above, so the simple programs cannot be used anyway. Removing the code that used these ports made space for the extra version 8 commands and subroutines that were added.

This manual is for version 2 of the Disk and Memory Expansion kit. It differs from the original version in that the logic for the two memory configurations is implemented in a programmable gate-array logic (GAL) chip instead of individual discrete logic ICs. This reduces the number of chips needed, and allows the circuit board to be smaller than the version 1 board:



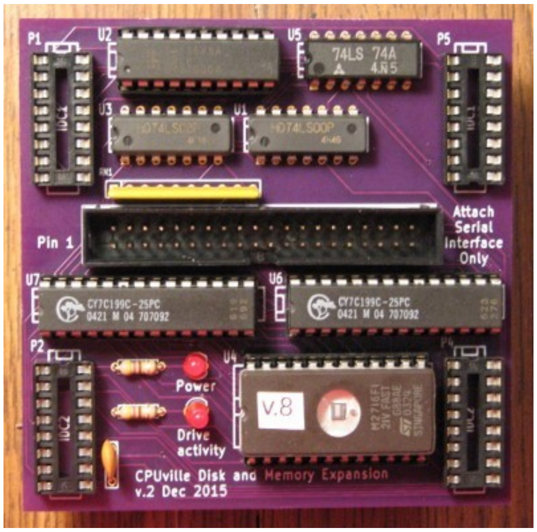
The kit includes a pre-programmed GAL16V8 IC.

Building Tips

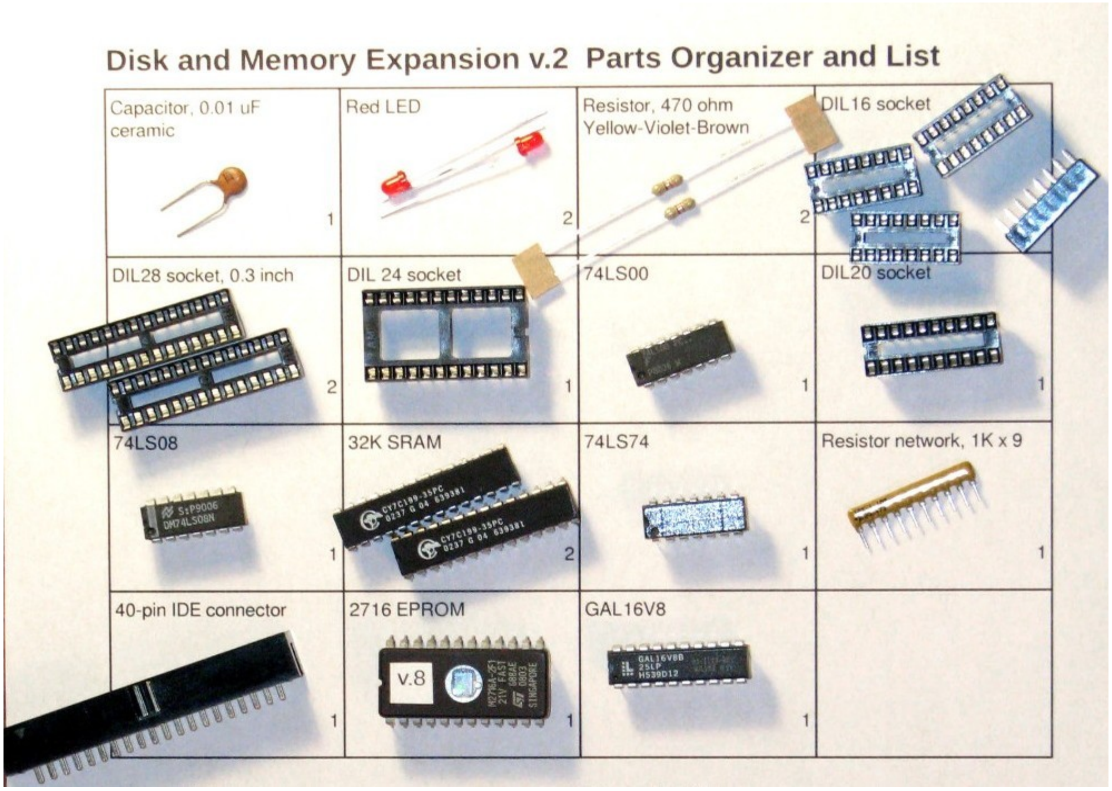
See the “Building Tips” section in the CPUville Z80 Computer Kit Instruction Manual for general help.

The 40-pin IDE drive connector in this kit has pins that are a little more massive than the IC or component pins. This means that more time, or perhaps more wattage, will be required to heat these pins with the soldering iron, to ensure good electrical connections.

Building the Disk and Memory Expansion Board



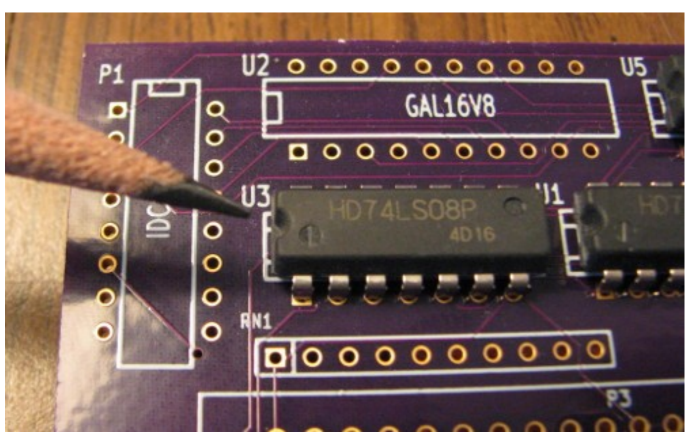
Start by putting the parts on the organizer to make sure you have them all, and to get familiar with them.



Once you have checked the parts you can start to solder them onto the circuit board.

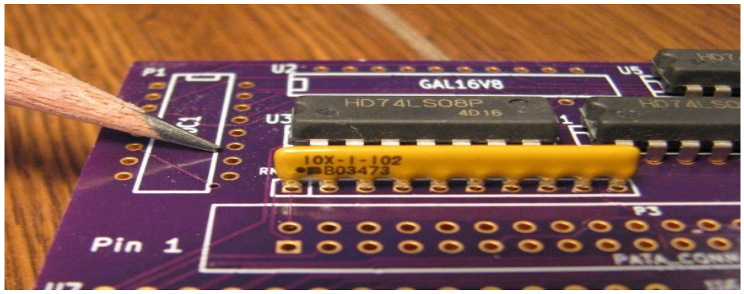
The easiest way to solder the components is to start with the shortest (parts that lie closest to the board) and proceed to the tallest. The order is resistors, ICs, resistor network, sockets, LEDs, capacitor, and 40-pin connector. Some components need to be oriented properly, as described below. The larger ICs,the GAL, EPROM, and RAM, have sockets, so solder the sockets directly to the circuit board, and plug the ICs into the sockets when you are done.

1. The resistors can be soldered first. They do not have to be oriented.
2. The non-socketed (14-pin) ICs are soldered next . The ICs need to be placed with the little cut-out toward the left:

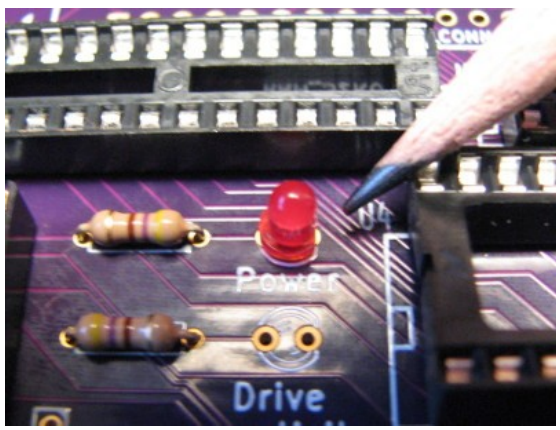


These ICs can be soldered directly to the board without fear of damage if you use a 15-watt or smaller soldering iron.

3. The resistor network can be soldered next. Please note that the marked pin goes to the left, as shown here:

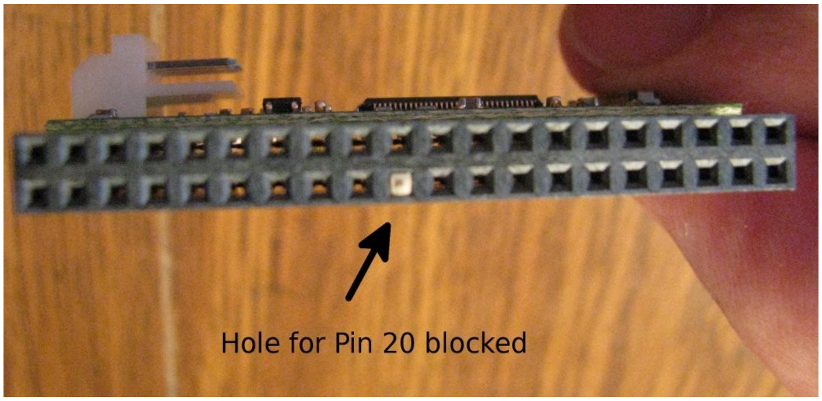


4. The IC sockets are next. They do not need to be oriented.
5. The LEDs are next. The flat side of the plastic base, or the shorter lead, is oriented toward the right:

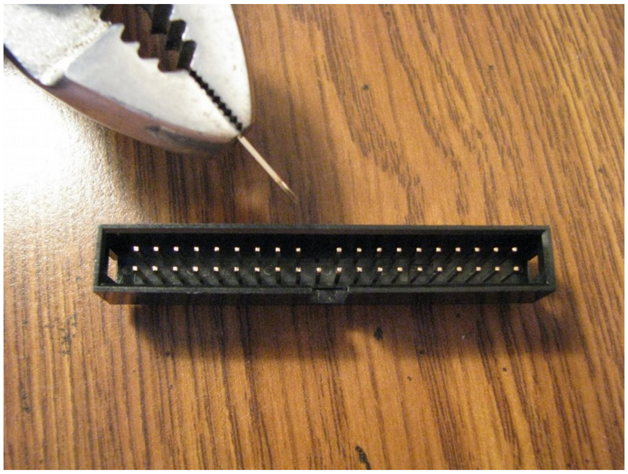


6. The capacitor is next. It does not need to be oriented.

7. Before you solder the 40-pin disk drive connector, take a moment to think about how you will supply power to your disk drive, and if the cable or drive plug you will use is “keyed”. The IDE specification allows for pin 20 to act as a key for orienting the plug in the connector. Here is an example of a keyed plug:



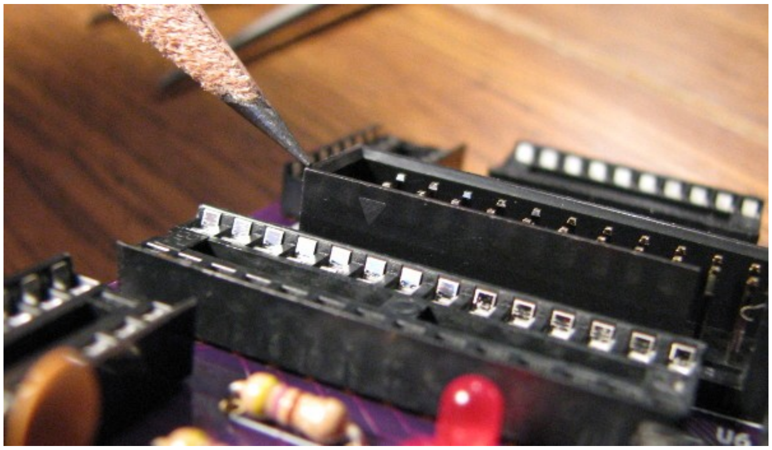
If you are using a keyed plug, and you want to leave the key blocker in its hole, then you should remove pin 20 from the connector before you solder it in. You can push the pin through the plastic from the bottom with a flat hard object, like a screwdriver tip, and then pull it out the rest of the way with pliers:



You can also remove the small key plug from the female socket on your adapter, drive or cable with a needle. Some adapters have the plug hole blocked, but if you remove it, you can provide power to the adapter through pin 20 anyway.

If you have a drive with a non-keyed connector, or one that can use the +5V supplied through pin 20, you should leave it in. If you remove pin 20 without cutting it, you can put the pin back in the connector later if you want to, by pushing it back through the plastic with the pliers and screwdriver, like you did when you removed it.

The 40-pin connector is oriented with pin 1 in the left front corner, as shown by the “Pin 1” label on the circuit board, and the small arrow etched into the plastic shroud:



The cut-out in the shroud should be toward the front of the board.

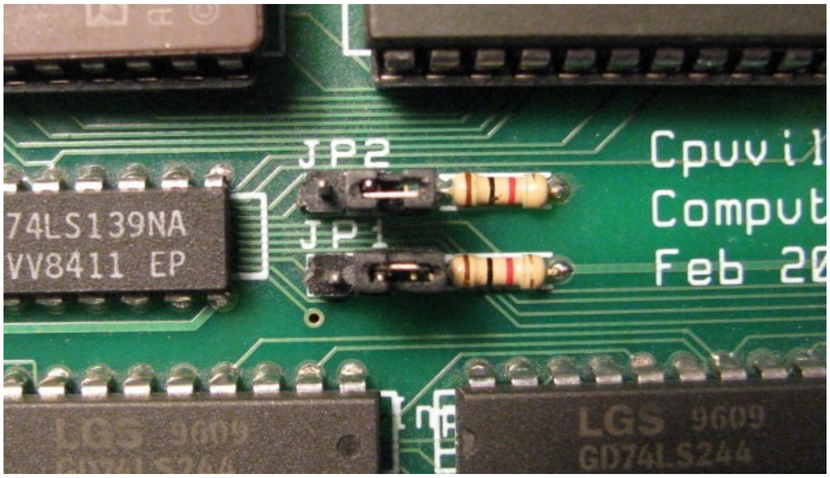
Once you have finished soldering all the pins on the computer, inspect the board to make sure there are no solder bridges or unsoldered pins. Hold the finished board against a bright light. If you can see light coming through a pin hole, you know you forgot to solder it. This does not apply to the vias, the plated holes where a trace goes from one side of the board to the other. These can be left open.

Testing and Using the Disk and Memory Expansion

The following sections assume you are familiar with using the CPUville Z80 computer with the serial interface, connected to a terminal, or to a PC with a terminal emulation program, such as Minicom in Linux, or RealTerm in Windows. If you are not familiar with using the serial interface you should look at the detailed descriptions in the CPUville Z80 Computer Serial Interface Instruction Manual. This is important, because all testing and using the disk drive will be done using the serial interface – the switches and LEDs on the main computer board cannot be used. Also, to get an operating system such as CP/M up and running, you will need to be fairly expert in using the ROM monitor command line, especially the load, dump, bload and run commands.

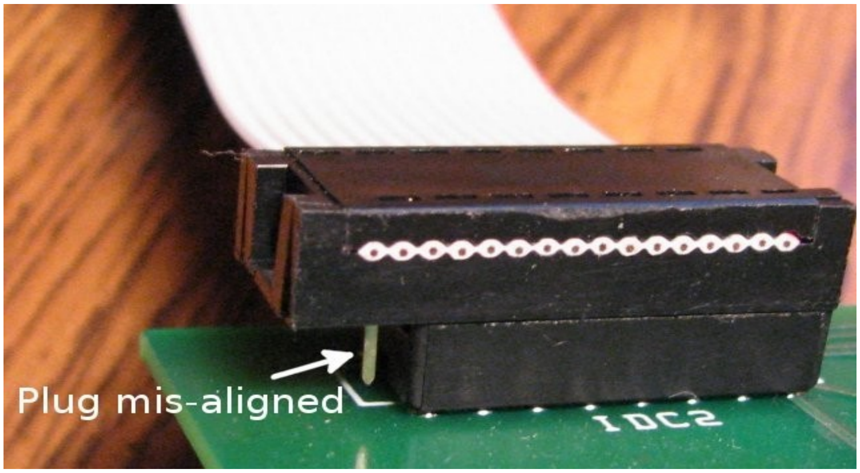
Insert the 16V8 GAL, EPROM (version 8 or higher) and RAM ICs into their sockets on the completed circuit board, being careful not to fold pins underneath. You may have to bend the pins a little to make them go straight down, to better align with the pin holes in the sockets. Take particular care with the RAM IC pins, they tend to be brittle, and if you fold one under, it may break off when you try to straighten it. Note that ROM versions 7 and lower will not work with the disk and memory expansion board.

Before connecting the disk and memory expansion board to the computer, remove the shorting blocks from the JP1 and JP2 jumpers on the main computer board. This picture shows them in the proper,“off” configuration for use with the disk and memory expansion:



This disables the memory and input/output ports on the main computer board, so that the processor can use the memory and ports on the disk and memory expansion board and serial interface board. You do not need to physically remove the EPROM or RAM ICs from the main computer board. Also, there is no need to place a starting address on the main board input port switches; the version 8 ROM code jumps right into the monitor cold start address at startup, and these switches are disabled anyway.

Connect the disk and memory expansion board to the computer using the same ribbon connectors used to connect the bus display board. Make sure the connectors are not misaligned:

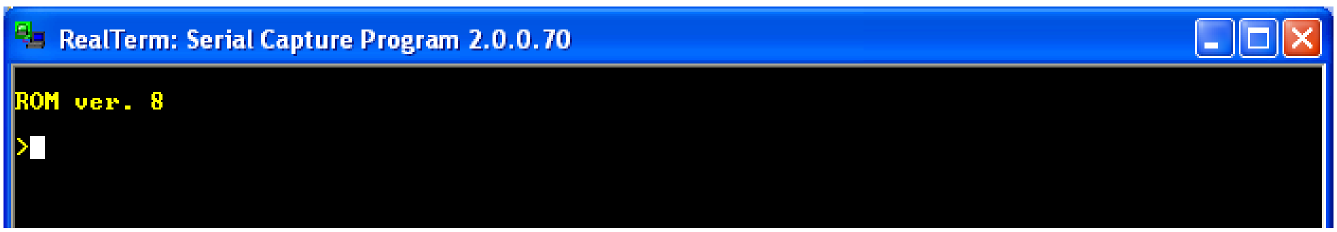


Do the same with the serial interface board, on the right-side connectors. Do not connect a disk to the board at this time.

The main computer should have the fast clock selected, and be in Reset. Now, connect power to the computer board. The Power indicators on the disk and memory expansion and serial interface boards should light up. If they don't, check the ribbon cable connectors again to ensure they are seated properly. Once the Power LEDs are on, check the ICs to make sure none of them are getting hot (can happen if you solder or plug one in backwards by mistake). If everything is OK, connect the serial interface board to a PC serial port using a straight-through serial cable (not a “null modem” crossover cable). The serial interface is configured as 9600 baud, 8 data bits, no parity, one stop bit (8-N-1).

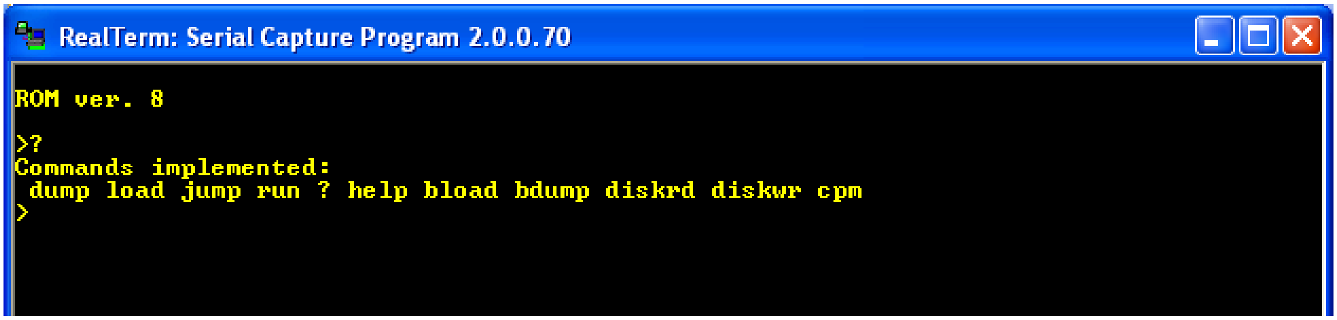
On the PC, start a terminal emulation program. I will use the RealTerm program running under Windows XP on a PC for these examples. For detailed instructions on using RealTerm with the Z80 computer, see the CPUville Z80 Computer Serial Interface Kit Instruction Manual.

Take the computer out of reset. You should see the short ROM v. 8 greeting message:



Note that in ROM version 8 and higher, some messages have been shortened to create more room for program code.

At the monitor prompt enter `?` Or `help` to see a list of available commands:



You see the same set of commands used in the monitor program version 7, with some new additions. They are `diskrd`, `diskwr`, and `cpm`.

The `diskrd` command

This command reads one sector from the disk and writes it into memory at a location you specify. The command takes as input the memory address where the disk data is to be placed as a 4-digit hexadecimal number, and the sector number as a decimal logical block address (LBA) from 0 to 65,535. It reads 256 bytes from the sector, and places this data into memory. Note that the command will read sectors using a 16-bit LBA, but the ROM subroutine underlying the command will take a full 24-bit LBA, and you can write programs using this subroutine to take advantage of this if you want.

The `diskwr` command

This commands takes 256 bytes of data from memory and writes it to one sector of the disk. Like the `diskrd` command, it takes as input the memory address of the data to be written as a 4-digit hexadecimal number, and a 16-bit decimal LBA for the sector to write. Both `diskrd` and `diskwr` need the LBA to be an ordinary decimal number without leading zeros – if you add them, the routines will hang.

The `cpm` command

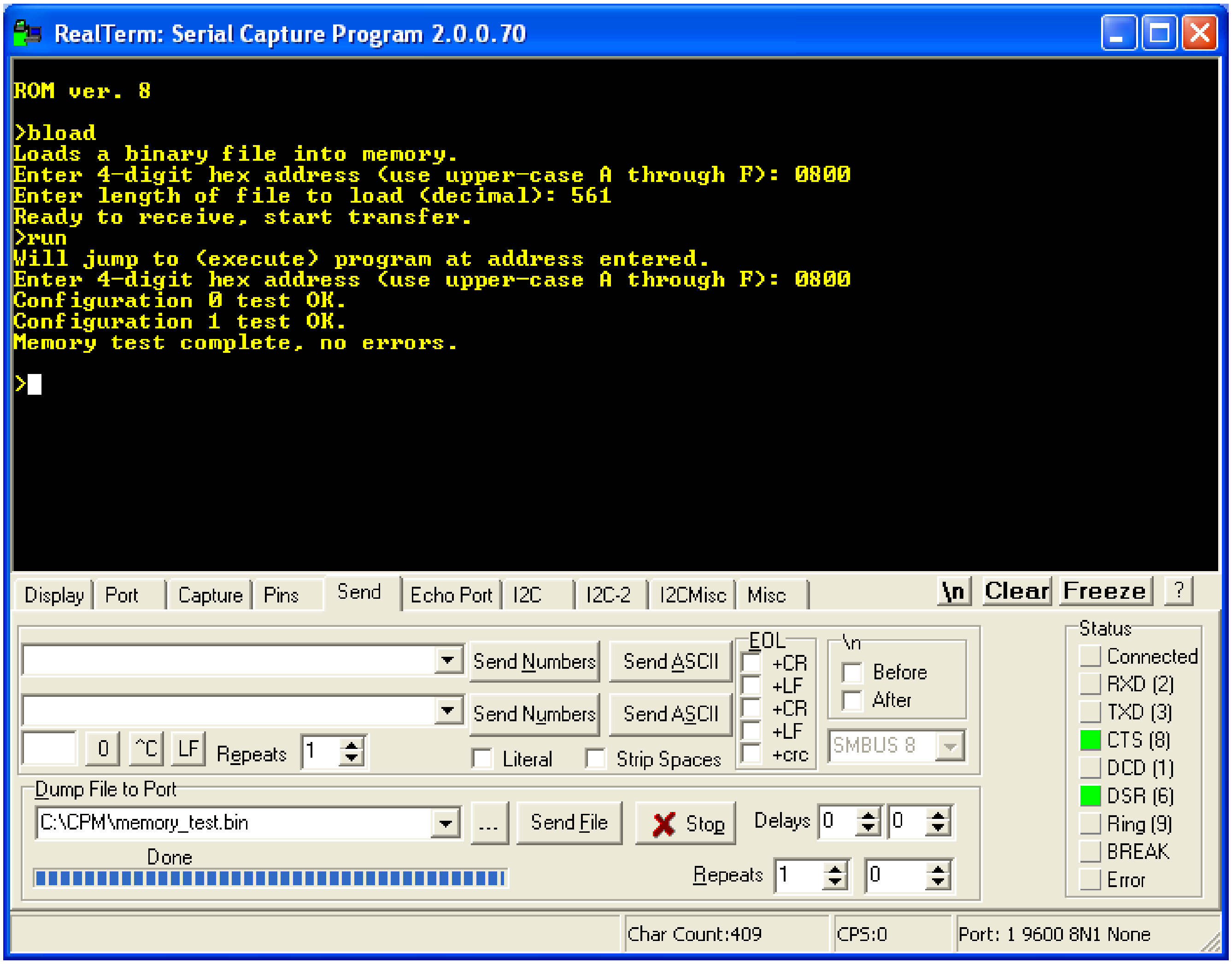
This command loads 256 bytes of data from the first sector of the disk (LBA 0x000000) into memory location 0x0800, then jumps to it. The program in that sector is used to load and start CP/M, but can be used to start any other operating system the user might care to put on the disk.

Do not execute these commands at present. Since there is no disk attached, the system will hang. If the system hangs, just reset the computer to start over – no need to disconnect the power.

Testing the memory expansion

I have written a brief program to verify the presence of 64K RAM, and that the memory configuration flip-flop is working correctly. This program will work without a disk attached. Download the memory_test.bin file from the CPUville website http://cpuville.com/Code/CPM.html.

To do the test, use the `bload` command to load the memory_test.bin file into memory at location 0x0800, then the run command to execute it. It takes about 15 seconds to complete. If successful, it should print output as below:



If the memory test fails, recheck the pins of the RAM ICs to make sure they are seated properly. If you cannot get it to work, please contact me for advice.

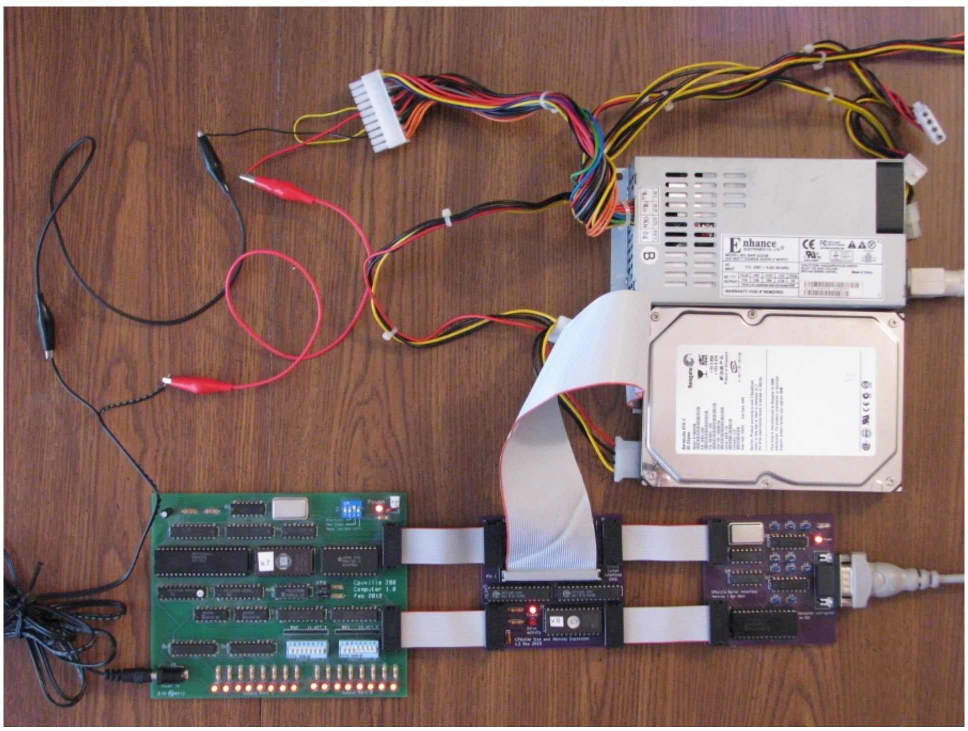
If the memory test works, we can be confident that the board is built correctly. Now, put the computer in reset, and disconnect the power, and connect a disk drive as described in the following section.

Connecting a disk drive

The disk and memory interface will work with most IDE disk drives (see the Table of Tested Disk Drives at the end of this manual). The disk size should be 128 megabytes or higher. This is not to have enough room, because a full-blown CP/M system uses only about 1 megabyte of disk space, but because the CP/M system described here uses simplified code that does not use disk space very efficiently. In particular, it uses simplified arithmetic to map CP/M sectors onto the LBA sectors of the hard disk, which skips a lot of space. Also, the CP/M system I developed uses only 128 bytes of each sector for data. This is the native sector size that CP/M uses, since it came out of the era in the mid-1970s when only floppy disks were used, and those disks used 128-byte sectors. CP/M offers blocking and deblocking code to more efficiently use disk space, by taking 256- or 512-byte sectors and breaking them into 128-byte pieces, but I did not use this code in my system, again out of a desire to make it as simple as possible.

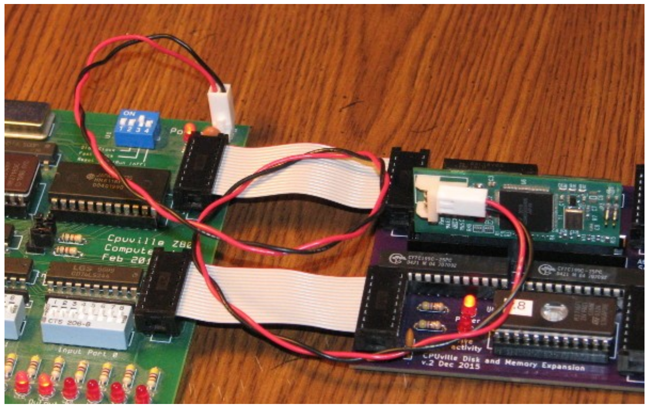
The disk drive plug needs to be oriented correctly. If keyed, as described above, it cannot go into the socket backwards. However, if it is not keyed, you need to take care that pin 1 of the plug goes onto pin 1 of the socket, as indicated by the “Pin 1” label on the circuit board, and by a small arrow engraved on the plastic shroud of the connector.

If using a mechanical disk drive, you can use a computer power supply to provide power to both the drive and the Z80 computer. Take the +5V and ground from the main power connector to the input jack on the Z80 computer board, or to the connector for the logic probe, and connect a power supply disk power connector to the drive. That way, both the computer and disk drive share the same ground, which is important to prevent damage to the computer or the drive electronics. Connect the hard disk to the circuit board IDE socket using a standard 40-conductor IDE cable. Make sure that pin 1 of the circuit board socket is connected to pin 1 of the disk drive socket.

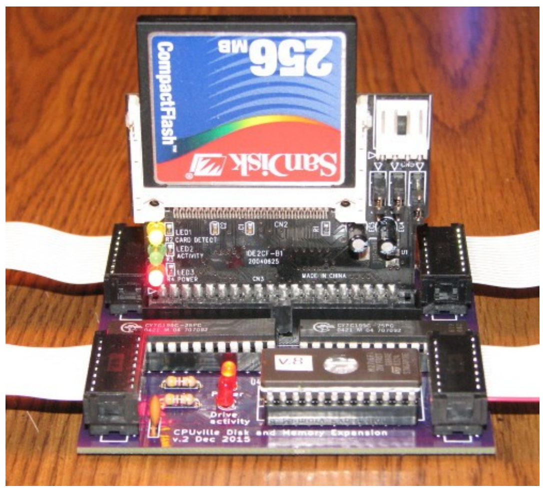


Note in the above picture the AT-type computer power supply, with the hard disk drive receiving power from one of the plugs coming from the power supply. The +5V power and GND for the Z80 computer are coming from the proper pins of the main power supply plug. There is also a jumper wire between the power supply ON input (PS_ON#, pin 14) and ground which is needed for the power supply to turn on.

If you are using a solid-state IDE drive, or a compact flash drive in an adapter with a separate power connector, you can use the logic probe connector to supply low-current +5V power to the drive. You will have to use your own wires to make the connection. Here is a photo of a solid-state IDE drive with attached power supply wires:



Many small solid state flash modules do not require a separate power input; you can get low-current +5V power from pin 20 of the drive connector instead. Here is a photo of a compact flash card in an adapter that can get power from pin 20:

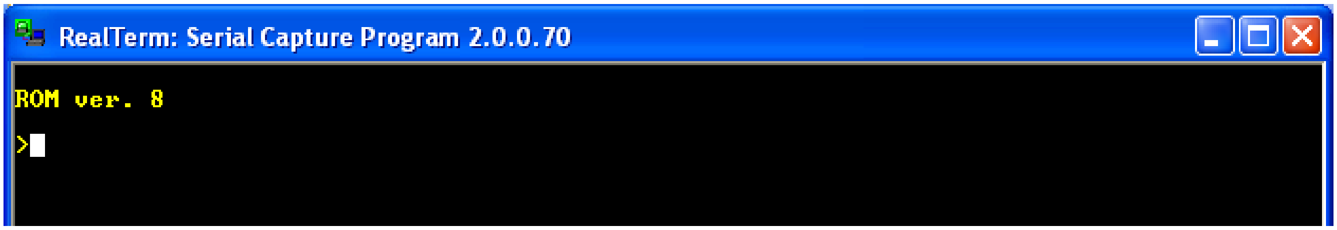


See the section above in Building the Disk and Memory Expansion for more details about pin 20 in the IDE socket.

Testing the Disk Drive

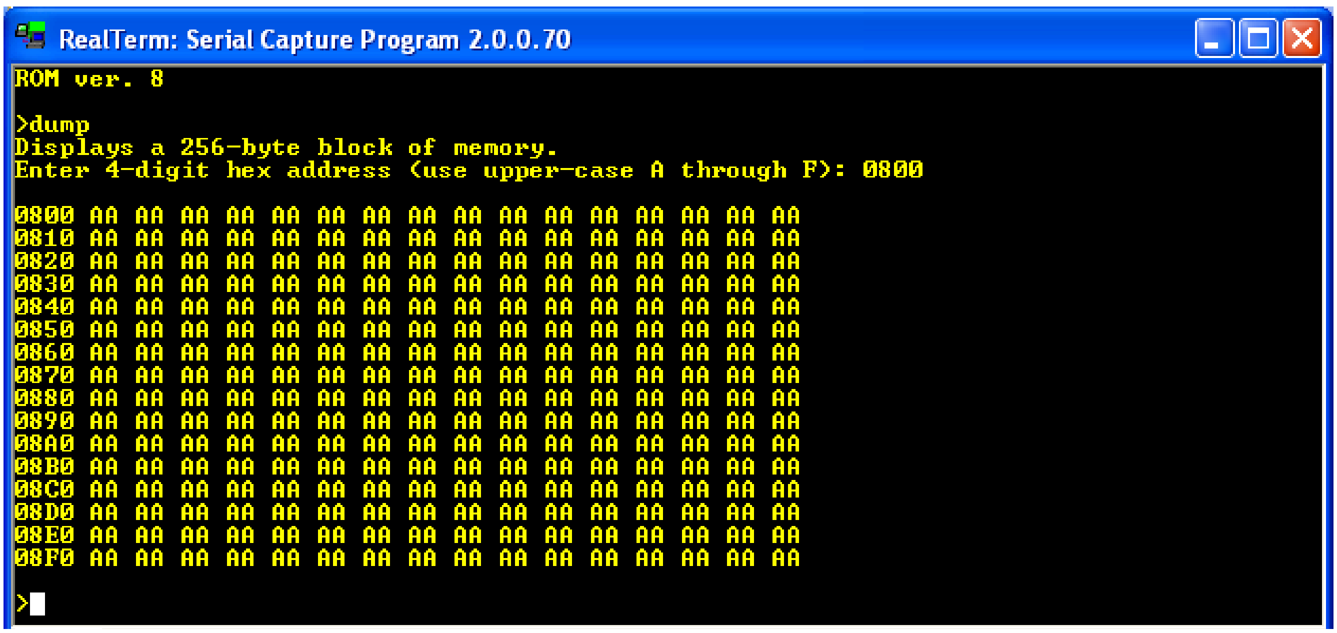
You can test the disk drive using the monitor `load`, `diskwr`, `diskrd`, and `dump` commands. Of course, once you write data to a disk sector, any data on that sector will be overwritten and lost. This is especially true of sector 0, which on most disks will have partition information. The `cpm` command in the ROM monitor reads disk sector 0 into memory, so you will need to place code in this sector if you want to use this command to start the operating system. I suggest you do not try to preserve partitions on your disk, but rather dedicate the disk for use on the Z80 computer for experimentation and to try the CP/M operating system.

With the disk drive connected, apply power to the computer and take it out of reset. You should again see the greeting message and get the monitor prompt.



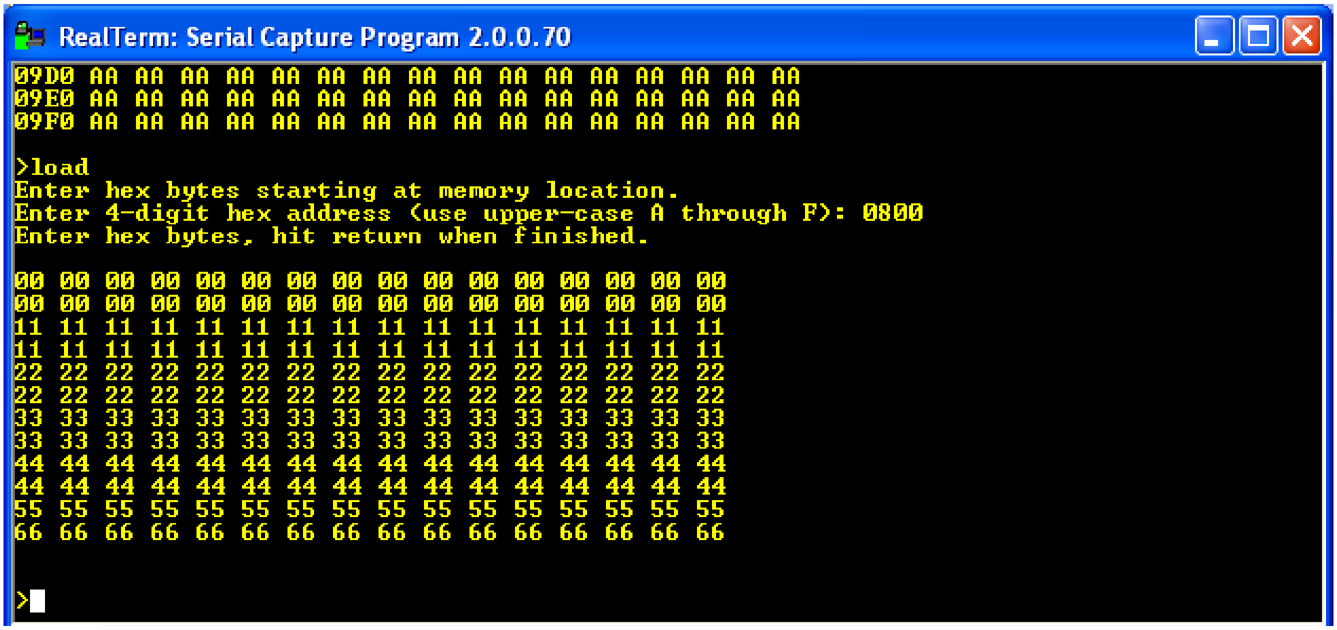
To test the disk, we will use the `load` command to place an easily recognizable data pattern into the computer memory, then write this pattern to a disk sector using the `diskwr` command. Next, we will read it from the disk and place it in a different area of memory using the `diskrd` command. Then, we will examine this second memory area with `dump`, and look for that data pattern. If we see the pattern, we know that the disk write and read commands worked correctly. Here is the detailed test procedure.

First, examine the memory pages (that is, the 256 bytes of memory) at 0x0800 and 0x0900 using the `dump` command:



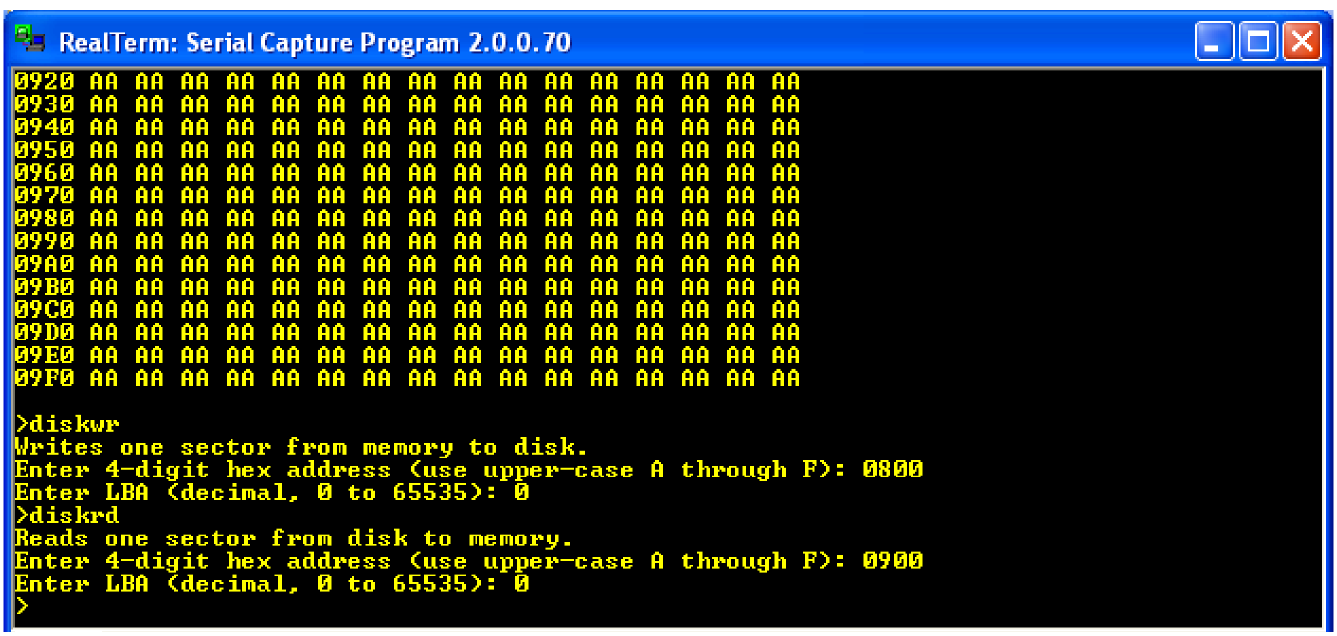
The memory will contain random data at system power-on. Your memory data will probably look different than this.

Now, load page 0x0800 of memory with an easily recognizable pattern of data using the `load` command:

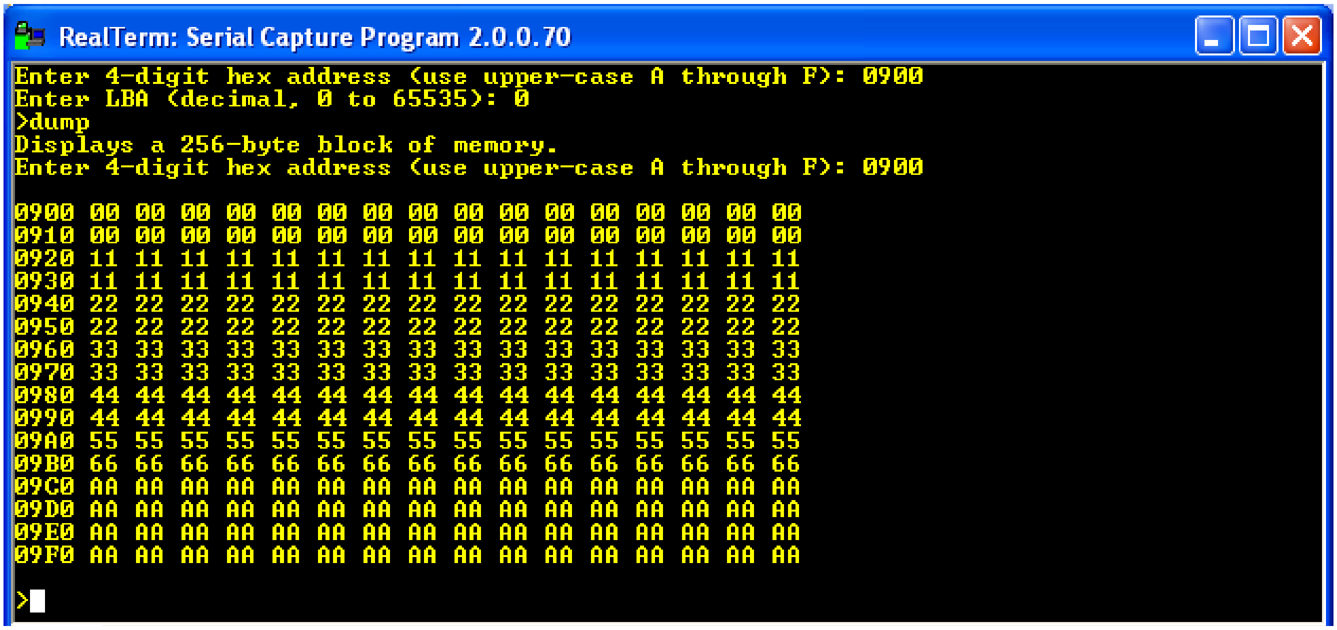


You can use whatever pattern you like, but it should be easily recognizable.

Now, write the memory page at 0x0800 to disk sector 0 using the `diskwr` command. You should see a brief flash on the Drive Activity LED when you do this. Then, read the same sector back into memory at 0x0900 using the `diskrd` command (again, the Drive Activity LED should flash):



Now, display the memory page at 0x0900 using the `dump` command:



If you see your data pattern there, you know your disk is working properly, and you can read and write sectors. You can experiment with other patterns, other memory locations and other sectors.

Once the disk is working properly you can install CP/M onto the disk.

Installing CP/M version 2.2

About CP/M

The CP/M operating system was the first commercially successful disk operating system for microcomputers. As such, it recently received designation by the IEEE as a Milestone in Electrical Engineering and Computing. See the article at http://theinstitute.ieee.org/technology-focus/technology-history/groundbreaking-operating-system-is-named-an-ieee-milestone.

This operating system was designed by Gary Kindall in 1974, to run on microcomputers with an 8080 processor and 8-inch IBM floppy disks. However, it was designed to be portable to many different machine architectures, by having a machine-dependent, customizable basic input-output system (CBIOS) that had the software to operate the disks, console and other peripheral hardware, and a machine-independent basic disk operating system (BDOS) and console command processor (CCP), to process commands and create and use a disk file system. Since the 8080 processor uses a subset of the same machine code as the Z80, CP/M could be used on both 8080 and Z80 machines. CP/M use spread to a wide variety of machines using a wide variety of disk drives and peripherals. Eventually, the introduction of 16-bit microcomputers using MS-DOS made 8-bit microcomputers (and CP/M) obsolete, but it is still used and enjoyed by hobbyists and educators using 8-bit Z80 or 8080 systems.

CP/M Source Code

Even though CP/M is obsolete, it is not yet in the public domain. The operating system was originally owned by Digital Research, Inc. It was passed to a spin-off named Caldera, Inc., and then to Lineo, Inc.

Permission to use CP/M for hobbyist and educational purposes has generally been granted freely, but since I am a commercial enterprise I cannot give you a complete, assembled CP/M system to download.

However, there is a web archive of CP/M software, "The Unofficial CP/M Web Site", that has been granted a license by Lineo, Inc., to make available CP/M source code for download for educational purposes. The site can be found at http://www.cpm.z80.de/

I am allowed to create a CBIOS tailored to the CPUville Z80 computer with the disk and memory expansion, and provide this directly to you. However, you will have to download and assemble your own BDOS and CCP for CP/M 2.2. This should be easy, since there is source code written in Z80 mnemonics that will assemble with only a few modifications.

To obtain the source code for CP/M 2.2 in Z80 assembly language, follow the Digital Research Source Code link on the Unofficial CP/M Web Site page to the source code page (http://www.cpm.z80.de/source.html). On this page, go to the CP/M 2.2 section, and download the .zip file labeled CP/M 2.2 ASM SOURCE (http://www.cpm.z80.de/download/cpm2-asm.zip). Unzip the file. The source code file CPM22.Z80 is the one we will use. It contains source code for the CCP and BDOS in Z80 assembly language.

We need to make some modifications to this source code. First, we need to change the code origin so that it will assemble for a 64K system. Open the file with a text editor. At the start of the file is the MEM constant that tells the assembler how large the system memory is. Change this from 62 to 64, since we will run our CP/M in an all-RAM, 64K system:

```
MEM  EQU  64        ;for a 64k system (TS802 TEST - WORKS OK).
```

The file contains a few errors that are the result of converting the original 8080 code to Z80 code. Here is one example:

```
;
CHECKSUM: LD   C,128          ;length of buffer.
          LD   HL,(DIRBUF)    ;get its location.
          XOR  A         ;clear summation byte.
CHKSUM1:  ADD  A,M       ;and compute sum ignoring carries.
          INC  HL
          DEC  C
          JP   NZ,CHKSUM1
          RET
;
```

In the `ADD A,M` instruction, `M` stands for "memory", and is used in 8080 code. In Z80 code, this is supposed to be `(HL)`. Your assembler will probably find these errors and alert you, and you will have to change them. You can also find them by searching the file for the pattern "`,M`". There are only a few of these errors in the file. The correct Z80 code should be:

```
;
CHECKSUM: LD   C,128          ;length of buffer.
          LD   HL,(DIRBUF)    ;get its location.
          XOR  A         ;clear summation byte.
```

```
CHKSUM1:  ADD  A,(HL)    ;and compute sum ignoring carries.
          INC  HL
          DEC  C
          JP   NZ,CHKSUM1
          RET
;
```

Those are the only code changes that must be made. However, as mentioned in the Serial Interface Instruction Manual, in the section “A Word about Assemblers”, each assembler program has some quirks that may affect the success of your assembly. The TASM assembler, for example, wants all directives to begin with a period (“.”). Thus, you need `.EQU` instead of `EQU`, and `.ORG` instead of `ORG`. The `DEFB` and `DEFW` directives are not recognized by TASM, and need to be changed to `.DB` and `.DW` respectively. The `.DB` directive will not accept strings in single quotes, it wants to see double quotes, but single characters in single quotes are fine – except the semicolon, which it doesn't like for some reason (just substitute the ASCII value 0x3B if this gives you an error). And, in TASM, the `.DB` directive doesn't like long lines of characters, you may need to break some of them up. The z80asm program under Linux wants all labels to end with a colon (“:”), even those for the `EQU` statements. Whichever assembler you use, you will probably need to massage the source code to get it to assemble properly.

At the end of the CPM22.Z80 file you will find the BIOS jump table, with fake destinations. This is present because the BDOS needs to have the addresses of the jump table items in order to assemble properly. The real jump table belongs to the BIOS, and we will overlay this fake BIOS table with the real one when we put the system together in memory. In making your changes to the source code, you might introduce or remove a byte or two from some of the strings if you aren't careful. Then, if you assemble the file, the jump table addresses might be off a little. This has to be fixed before CP/M is installed. You should look at a listing of your assembled CPM22 code, and make sure that the BOOT subroutine address comes out to be 0xFA00, which is the proper start of the BIOS in a 64K system. If not, you should probably go over your changes again, trying not to introduce or remove any characters. If you are off a little, and can't figure out why, you have a few bytes at the end of the file, labeled “Extra space?” that you can remove, or add to, to make the BOOT address exactly 0xFA00:

```
;   Extra space ?
;
     DEFB 0,0,0,0
;
;**************************************************************
;*
;*        B I O S   J U M P   T A B L E
;*
;**************************************************************
```

One more tiny irritant in this code is that the disk drive letter used in the CP/M prompt is lower case. The system will work fine, but if you want it to look like all the other CP/M systems in the world you should change this to upper case:

```
;
```

```
CMMND1:   LD   SP,CCPSTACK    ;set stack straight.
          CALL CRLF      ;start a new line on the screen.
          CALL GETDSK    ;get current drive.
          ADD  A,'a'
          CALL PRINT     ;print current drive.
          LD   A,'>'
          CALL PRINT     ;and add prompt.
          CALL GETINP    ;get line from user.
;
```

Change the character in the `ADD A,'a'` instruction to an upper case A:

```
;
CMMND1:   LD   SP,CCPSTACK    ;set stack straight.
          CALL CRLF      ;start a new line on the screen.
          CALL GETDSK    ;get current drive.
          ADD  A,'A'
          CALL PRINT     ;print current drive.
          LD   A,'>'
          CALL PRINT     ;and add prompt.
          CALL GETINP    ;get line from user.
;
```

Assemble the corrected assembly language program, and name the binary output file cpm22.sys[2]. This file contains the machine code for the CP/M Console Command Processor (CCP) and Basic Disk Operating System (BDOS).

The third part of CP/M, the customized Basic Input Output System (CBIOS) for the CPUville system with the Disk and Memory Expansion, I have written and assembled for you. You can download the CBIOS source and binary files, and other binary helper files mentioned below, from the CPUville web site page at http://cpuville.com/Code/CPM.html. The binary file for the CBIOS is z80_cbios.bin. The other files you will need are format.bin, putsys.bin, cpm_loader.bin, and monitor.bin.

Preparing the disk for CP/M

The CP/M file system directory entries are very simple. The first byte of a directory entry gives the status of the entry. If the entry is inactive (the file has been deleted or not yet created), the status byte has a value of 0xE5. To prepare a disk for the CP/M system, one needs only create a number of directory entries that start with this value.

But it is easier than that, because if a directory entry is inactive, CP/M does not care what else is in the directory. It will create a completely new entry when it needs to. So, all we need to do is write the value 0xE5 to all the sectors of the CP/M disk in order to prepare it.

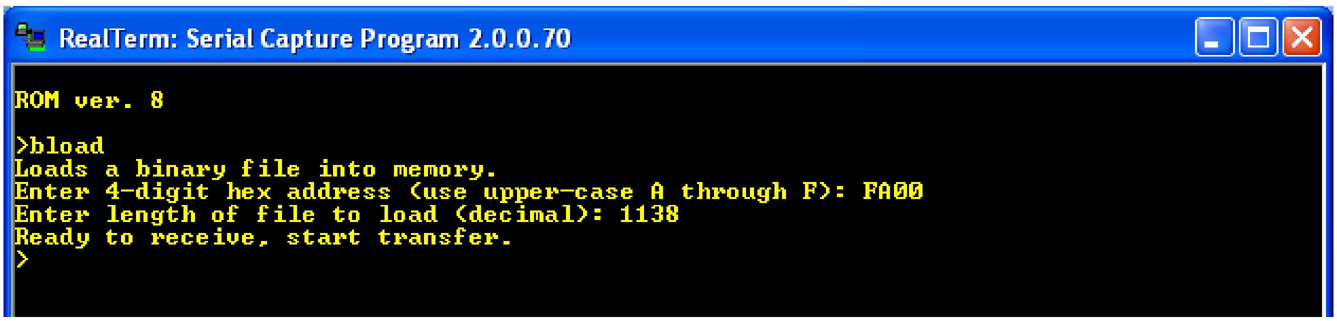
Note that I refer to the “CP/M disk”. This is a logical construct, created by the disk parameter tables in the CBIOS. These tables may or may not accurately represent the physical disk system. In the CBIOS I

2 You can use any name you want for the binary file, like a.out, or a.bin, but these instructions will use the name cpm22.sys.

created, I left the CP/M disk system as it originally was, with four disks, each with 77 tracks, 26 sectors per track. A CP/M call to read or write a particular disk, track, and sector is translated into a unique LBA address for the hard disk by the disk read and write subroutines in the CP/M CBIOS.

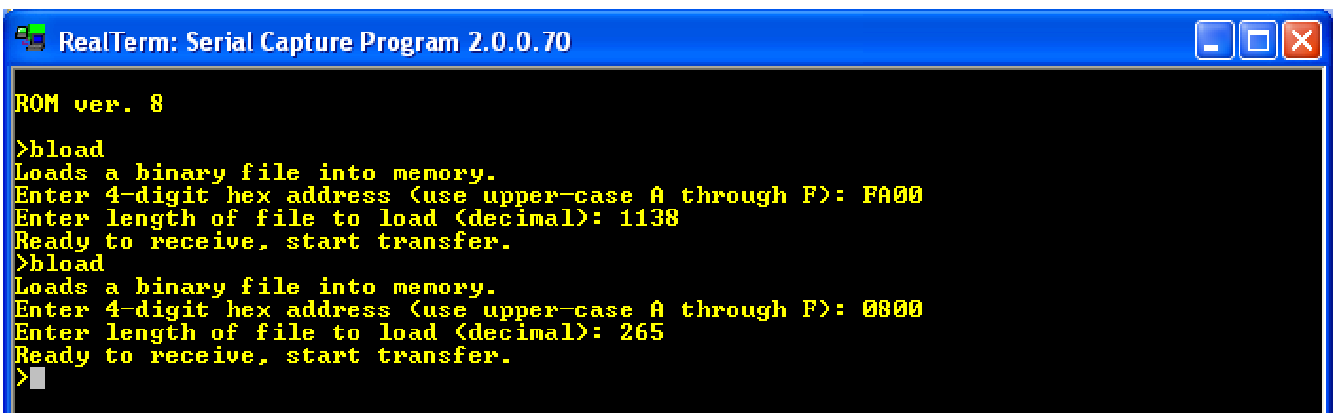
This is important to remember, because in order to prepare the disk, we will need to use the CBIOS calls for writing 0xE5 to the disks. That way, we will write the sectors as CP/M will see them when it creates the file system directory entries.

The format program calls the CBIOS routines to write 0xE5 to all the sectors of the four CP/M disks in our system. In order to work properly, the CBIOS code needs to placed into the system memory at location 0xFA00 before we load and execute the format program. Use the monitor `bload` command, and a binary transfer to load the file z80_cbios.bin into the computer memory at 0xFA00:

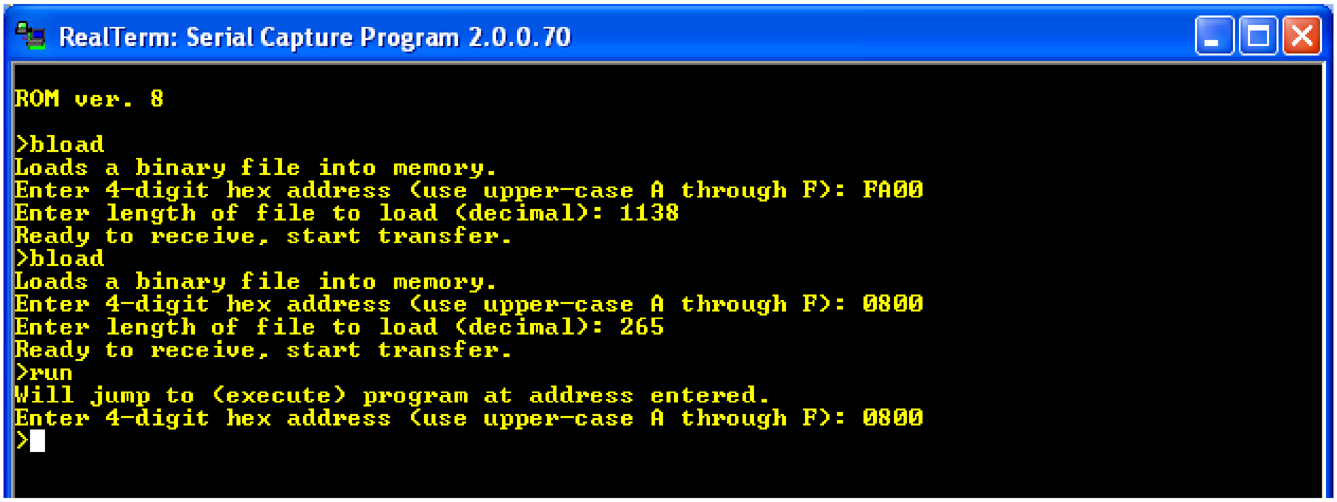


Note the file length in this example may be different from yours if you are using a later or customized version of z80_cbios.bin. Look at the file Properties to get the exact size before you make the transfer.

Next, load the format.bin file into memory at 0x0800:



Now, run the format program using the `run` command:



The Drive Activity LED should light up for about a minute and a half while the format program fills the CP/M disk with 0xE5. When the light goes off, the monitor prompt should re-appear. The disk is now ready for the CP/M files to be placed on it.[3]

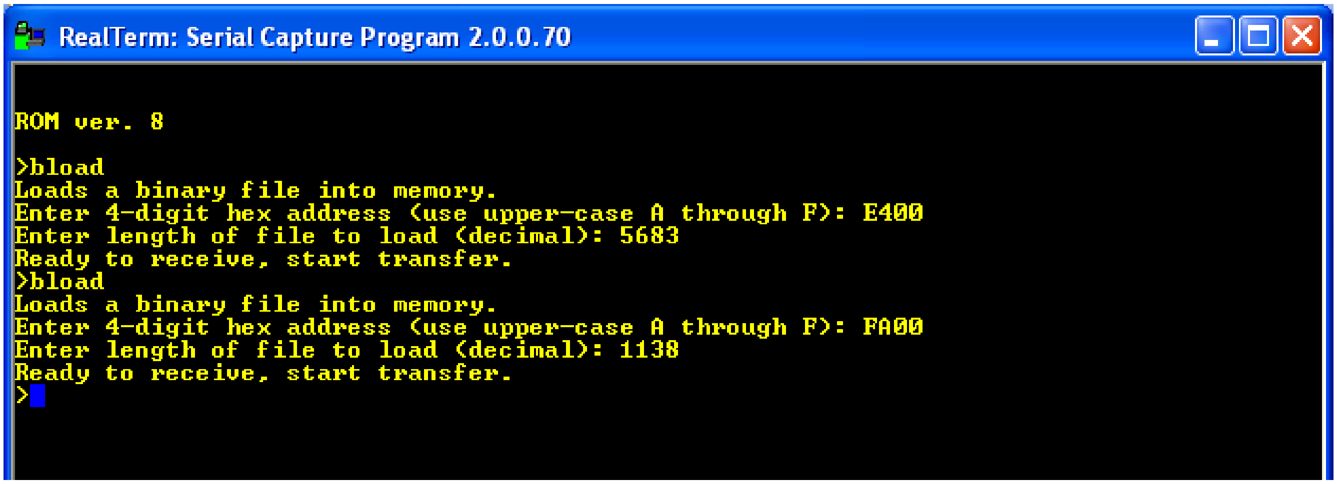
Putting the CP/M System Files onto the disk

The CP/M file system set up in the CBIOS reserves the first 2 tracks of each disk for the system files. This is important, because every time CP/M is started, whether from a cold boot or a warm restart, the system is loaded from the disk into memory. You can see this code in the CBIOS listing, in the WBOOT subroutine. Sector 1 of track 0 is reserved for boot code (not used in this system), and the rest of the sectors in tracks 0 and 1 have a memory image of the operating system.

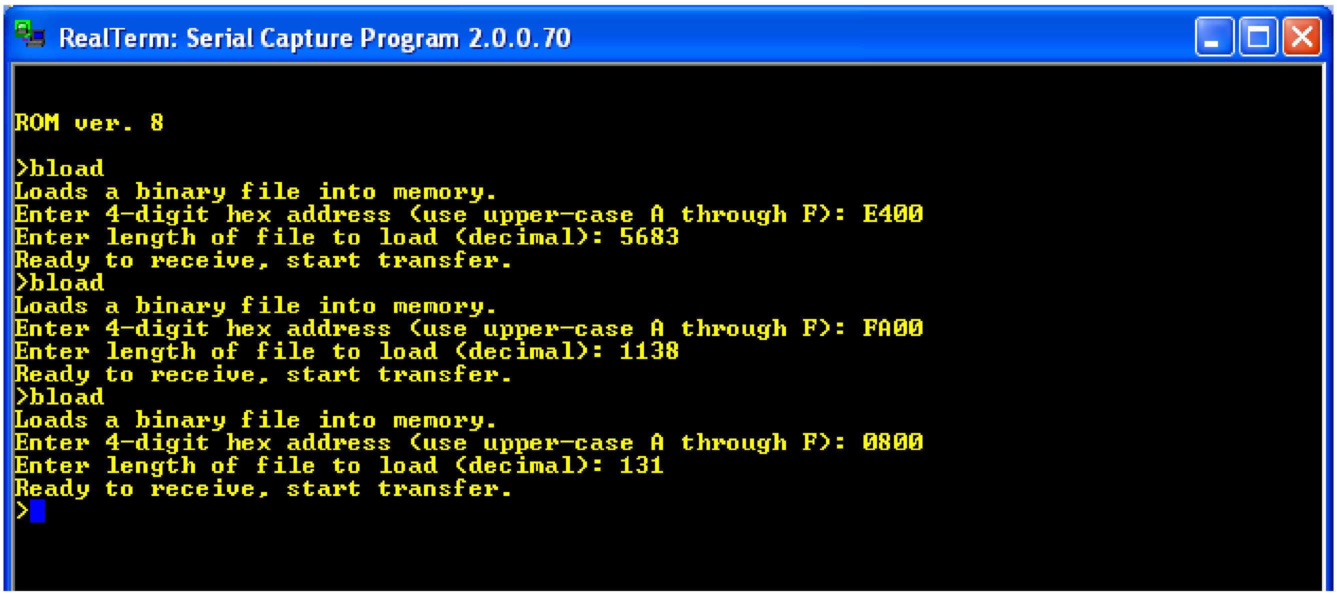
To set this up properly, we need to use the CBIOS routines for disk writing to put the system onto the disk from memory. For this, I have written a putsys program. It is similar to the format program, in that is uses the CBIOS disk write subroutines, but differs in that it copies data from memory, from address 0xE400 to the end of memory, and places it on the disk.

So first, we need to put CP/M into memory. Remember that the cpm22.sys file has the assembled code for the CCP and BDOS, with a dummy BIOS jump table at the end. It is important that we load this file into memory first, then load the z80_cbios.bin file on top of it, so that the true BIOS jump table will be present in memory. We again use the monitor command `bload` to place these files into memory at the proper places. The cpm22.sys file is placed at address 0xE400, and z80_cbios.bin at 0xFA00:

3 With some experimentation I have found that it is not absolutely necessary to format the disk before installing CP/M. If you do not format the disk, when you list the CP/M disk directory, you may get a series of blank entries or jumbled strings displayed. You can fix this by erasing the entire directory with an ERA *.* command.



Then, use `bload` to place the putsys.bin file into memory at location 0x0800:

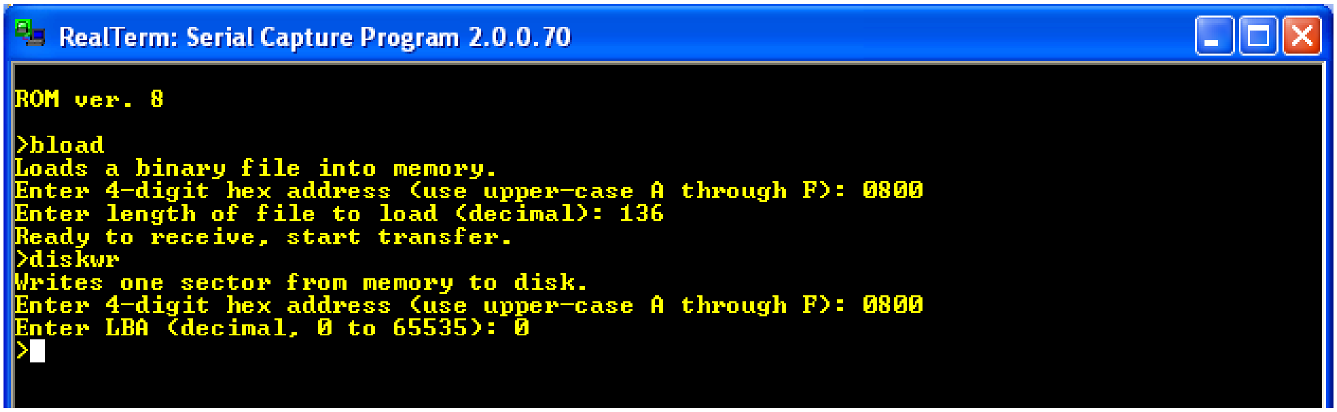


Now, run the putsys program at 0x0800. The drive activity light will light briefly – we are writing many fewer sectors than we wrote with the format program. Now, CP/M will be present on the disk system tracks.

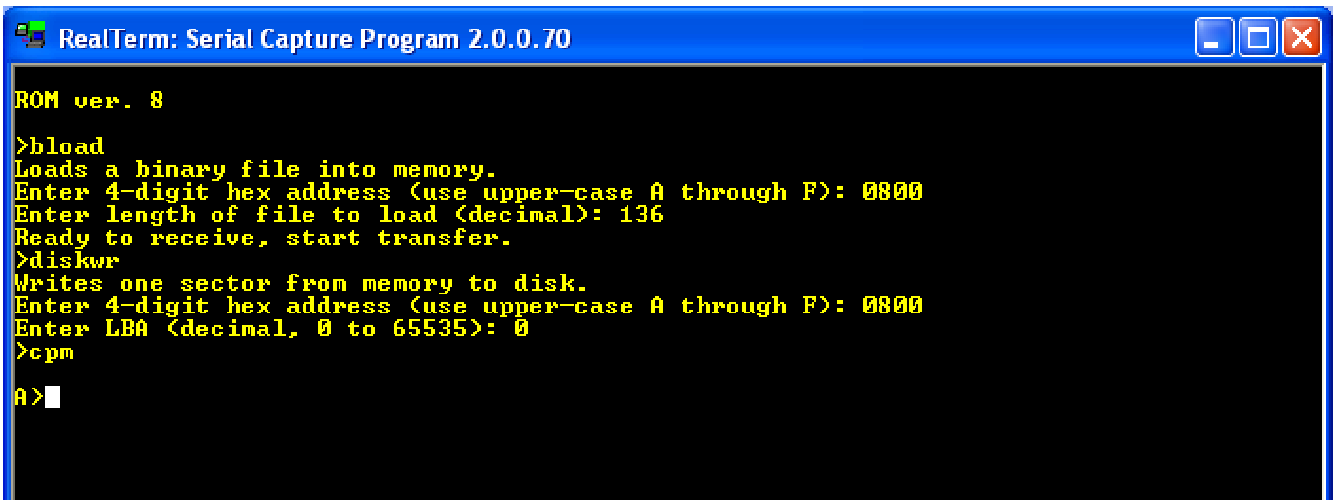
Installing the CP/M loader

The final piece of the puzzle is to place the cpm_loader program into sector 0 of the hard disk. This program is similar to the putsys program, but acts in reverse; that is, it gets the CP/M system from the disk and places it into memory. Since it is designed to run before the CBIOS is in memory, it uses its own versions of the CBIOS disk read routines, combined with ROM monitor subroutines, to get the code from the disk. When it is finished copying CP/M into memory, it switches the memory configuration to all-RAM with an `OUT (1),A` instruction, then jumps to CP/M.

We will use the `bload` command to first place the file cpm_loader.bin into the computer memory, then use the `diskwr` command to put it into sector 0 on the hard disk:



Now that the disk is set up to run CP/M, enter the `cpm` command at the monitor prompt (you might need to reset the computer first to get it to work properly):



You now see the CP/M prompt, A>, which indicates that CP/M is running, and that disk A is active.

To summarize, these are the steps to install CP/M 2.2:

1. Load z80_cbios.bin at 0xFA00
2. Load format.bin at 0x0800
3. Run format.bin
4. Load cpm22.sys at 0xE400
5. Load z80_cbios.bin at 0xFA00
6. Load putsys.bin at 0x0800
7. Run putsys.bin
8. Load cpm_loader.bin at 0x0800
9. Write the memory page 0x0800 to disk sector 0

10. Reset the computer
11. Start CP/M using the monitor `cpm` command.

Running CP/M

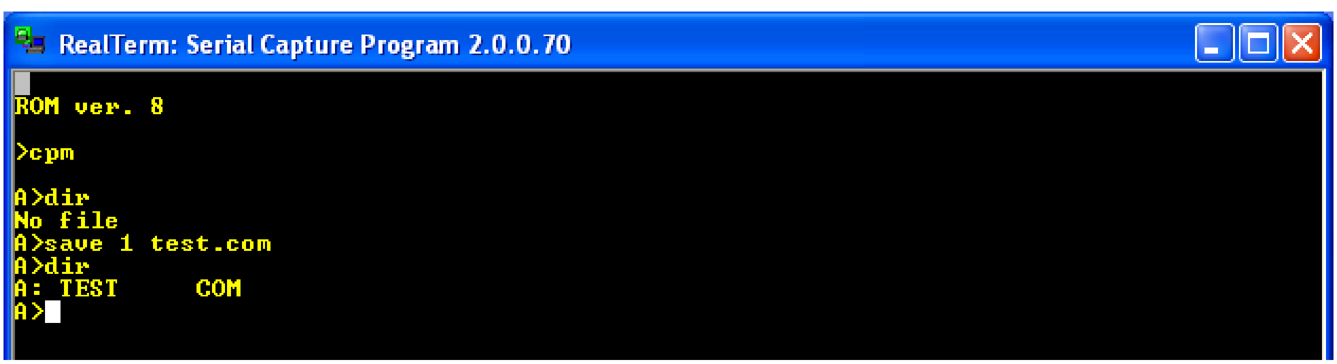
Built-in commands

I will not attempt to reproduce here a guide to running CP/M. The original Digital Research CP/M 2 system manual has been converted into a web page: http://www.gaby.de/cpm/manuals/archive/cpm22htm/. Here you can find all the details about using CP/M, with all the commands listed. However, we need to do a little more work here to create a truly usable CP/M.

CP/M 2.2 has only six built-in commands. These are DIR (list a disk directory), ERA (erase a file), REN (rename a file), SAVE (save memory to a file), TYPE (display a text file on the screen), and USER (change a user number). Note there is no command that will copy or move a file, no command to show how much disk space is available, or what the file sizes are (DIR only displays the file names). These functions can be added later using transient commands (see below).

To get used to the CP/M commands, start with DIR (you can enter commands as upper or lower case):

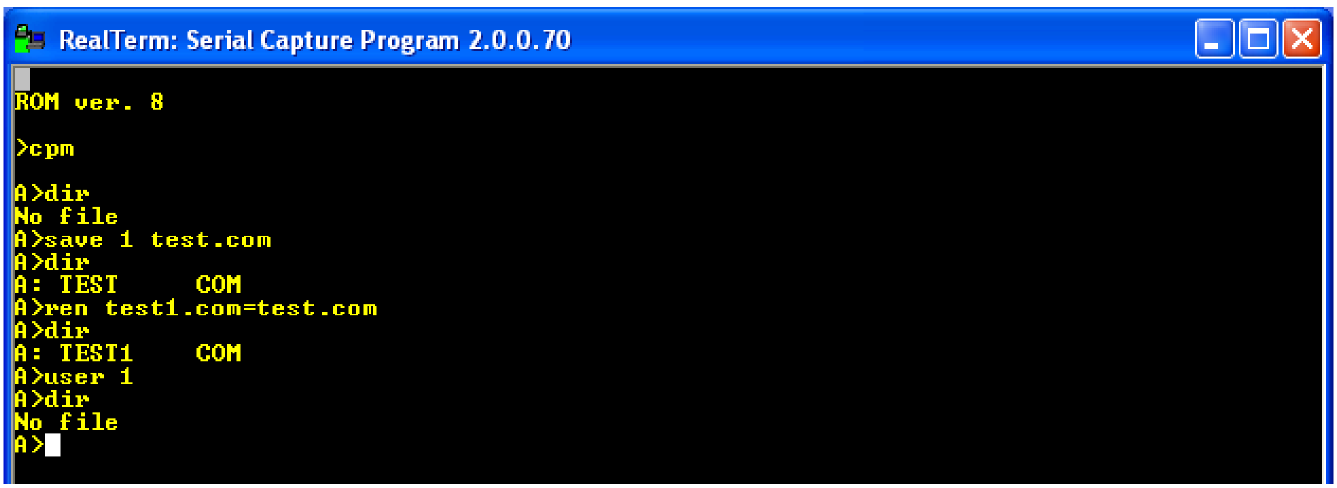
The “No file” output shows that there are no files in the directory of disk A. We can create a file using the SAVE command. This command will take a number of memory pages, starting at 0x0100, and save them to the disk as a CP/M file. For an example, the command “save 1 test.com” save one page (256 bytes) of memory, and give it the name TEST.COM. The file will of course contain garbage, but that is not a concern for now. After entering the SAVE command, enter the DIR command and you will see the directory entry for the file:



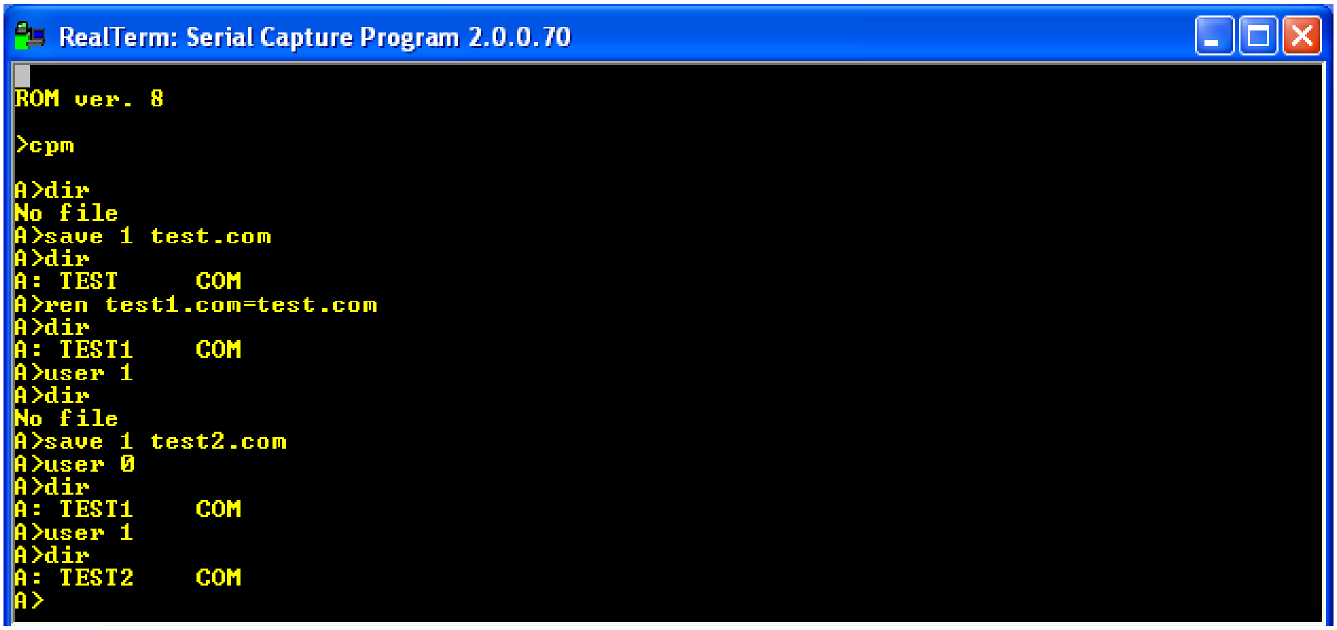
We can rename the file with the REN command:

Note that the target file name comes first in the argument for the REN command.

Each disk maintains a separate directory for each of multiple users, from 0 to 15. This feature is not of much use to us, but for completeness we can demonstrate it. Change to user 1 and enter the DIR command:

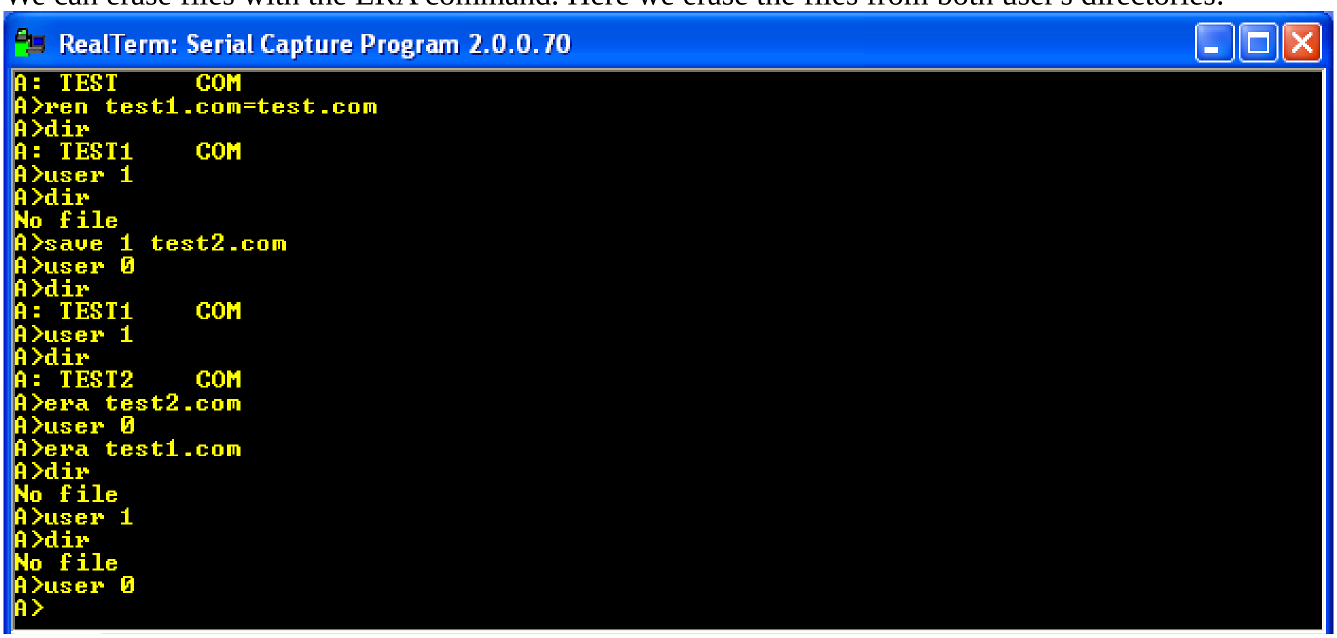


You can see user 1 has no files on disk A. Now create a file, with the name test2.com. Switch back to user 0, and display the directory. You see only test1.com. Switch to user 1, and do DIR, and you see that user's test2.com file.



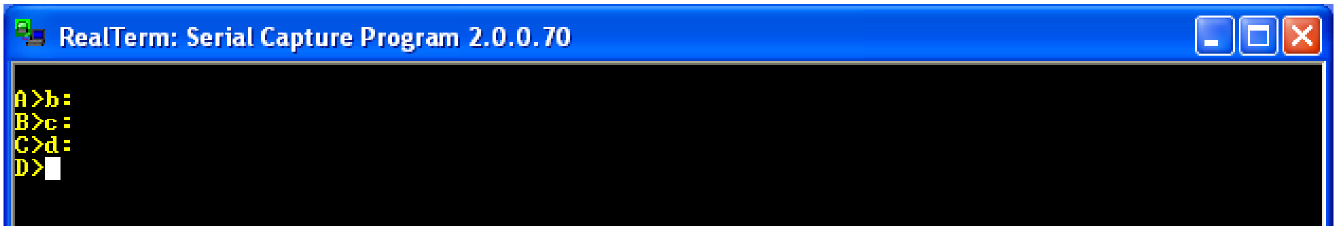
User 1's files are not visible to user 0, and vice-versa.

We can erase files with the ERA command. Here we erase the files from both user's directories:

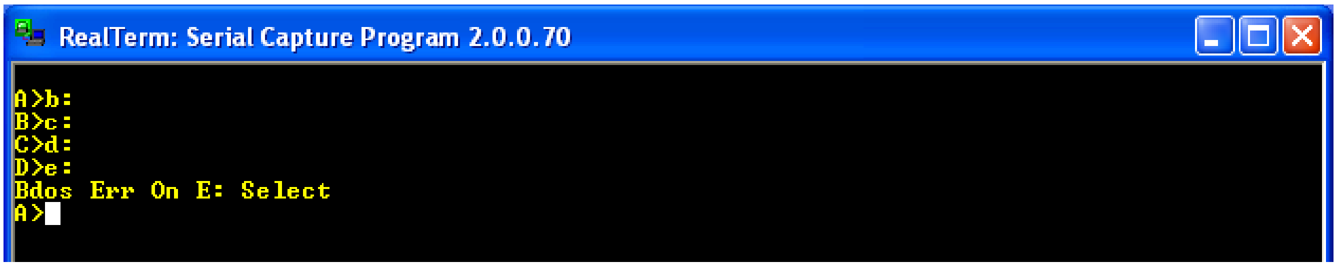


The TYPE command displays a text file to the console, but since we don't have any text file on the disk at present we won't demonstrate it now.

The system configuration set up in the CBIOS has 4 disks. To switch from one disk to another, enter the disk letter followed by a colon:



If you try to access a disk that is not available (here, for example disk E), you will get an error message. Hit return and the system will go back to the A disk:



This is a very limited set of commands. Many more commands are available as transient commands.

Transient commands

Originally, CP/M was created with multiple floppy disks, and the first disk came from the manufacturer with lots of programs (transient commands, or .COM files) that extended the system so that it was easy to create text files (with a text editor, ED.COM), assemble programs (ASM.COM), copy files (PIP.COM), and display disk statistics, such as file size and room remaining (STAT.COM). For example, if STAT.COM was on the A disk, entering STAT at the CP/M prompt would give a display of the room remaining on the disk. Essentially, a .COM file is a command that extends the functions of CP/M. When one enters the command, CP/M searches the directory of the current disk, and if it finds a file with the name of the command and a .COM extension, it loads that file into memory at location 0x0100 and jumps there to execute it. In the original CP/M, getting new programs was as simple as putting a disk in drive B, and copying the files from that disk using the PIP command.

But how can we get CP/M files into the CPUville Z80 system from outside? The CPUville Z80 computer has only one disk interface, and only one serial port. With CP/M running, the serial port is dedicated to the CP/M console, for character input and output, and cannot be used for binary file transfers. If we had two serial ports, we could perhaps use a program like XMODEM running under CP/M to do binary transfers using the second port, but we cannot do that here[4].

The answer is to use a RAM monitor program, that has the same commands as the ROM monitor, but runs in the CP/M environment – that is, with the memory in configuration 1 (all-RAM). Then we can do binary transfers into the Z80 memory through the single serial port using monitor commands.

I created the RAM monitor program by re-assembling the ROM monitor with a target address (code origin) of 0xDC00 instead of 0x0000. I had to put some additional code at the start that copies the rest of the RAM monitor program from location 0x0100, where CP/M would load it, to high memory at 0xDC00, so it would be out of the way of any code that we might want to place into lower memory.

4 It is possible to write an XMODEM program for one port, but the CP/M programs currently available require two.

Another important difference is that the `cpm` command given to the RAM monitor will do a warm boot of CP/M, so any code in memory will not be overwritten by the cpm_loader that is used by the `cpm` command of the ROM monitor.

So, to get a transient command files onto the CP/M disk, we run the RAM monitor, `bload` the command binary file into the Z80 computer's memory at 0x0100 , switch to CP/M, and use the built-in SAVE command to create a .COM file.

So how to get the RAM monitor program itself into memory, and onto the CP/M disk? We need to “bootstrap” it, using the RAM monitor program itself. It is a little complicated, but you only have to do this once. Here is how.

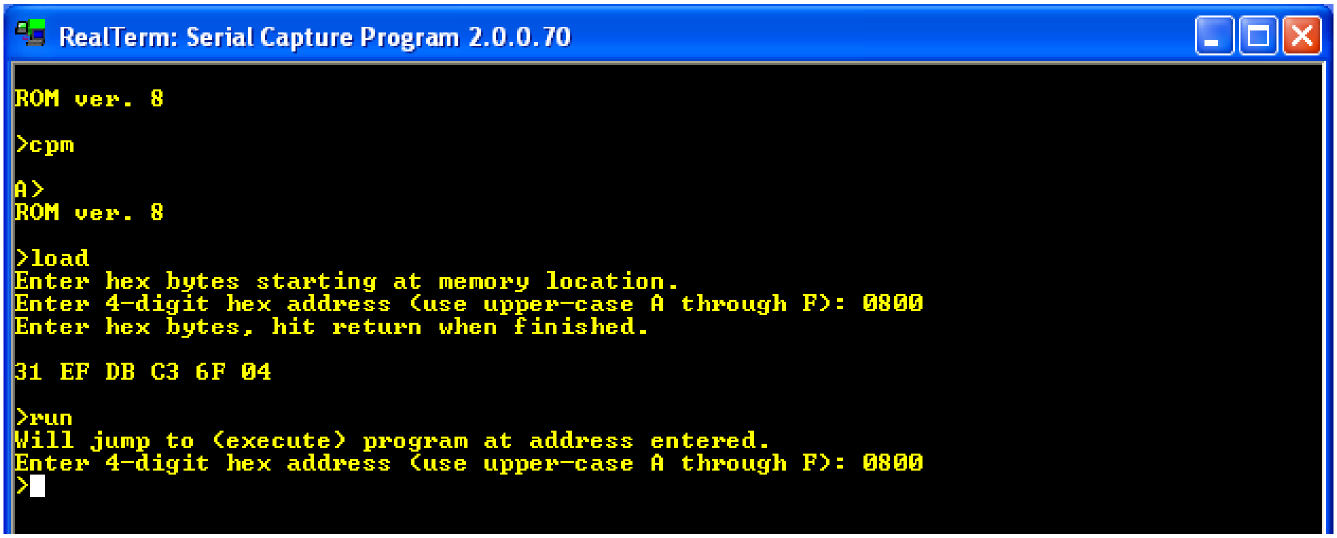
First, we start CP/M with the ROM monitor `cpm` command. This sets the memory configuration to 1, puts the CP/M system into the memory, and sets up memory page 0 (addresses 0x0000 to 0x00FF) with the data CP/M needs to operate. Then, we reset the computer. We see the ROM monitor greeting again. The system reset causes the memory configuration to switch back to configuration 0, so we can use the ROM monitor, but it does not disturb the CP/M memory page 0, or the CP/M code in high memory:

Now, using the ROM monitor, we will place the RAM monitor program (file name monitor.bin) into high memory, but below CP/M.

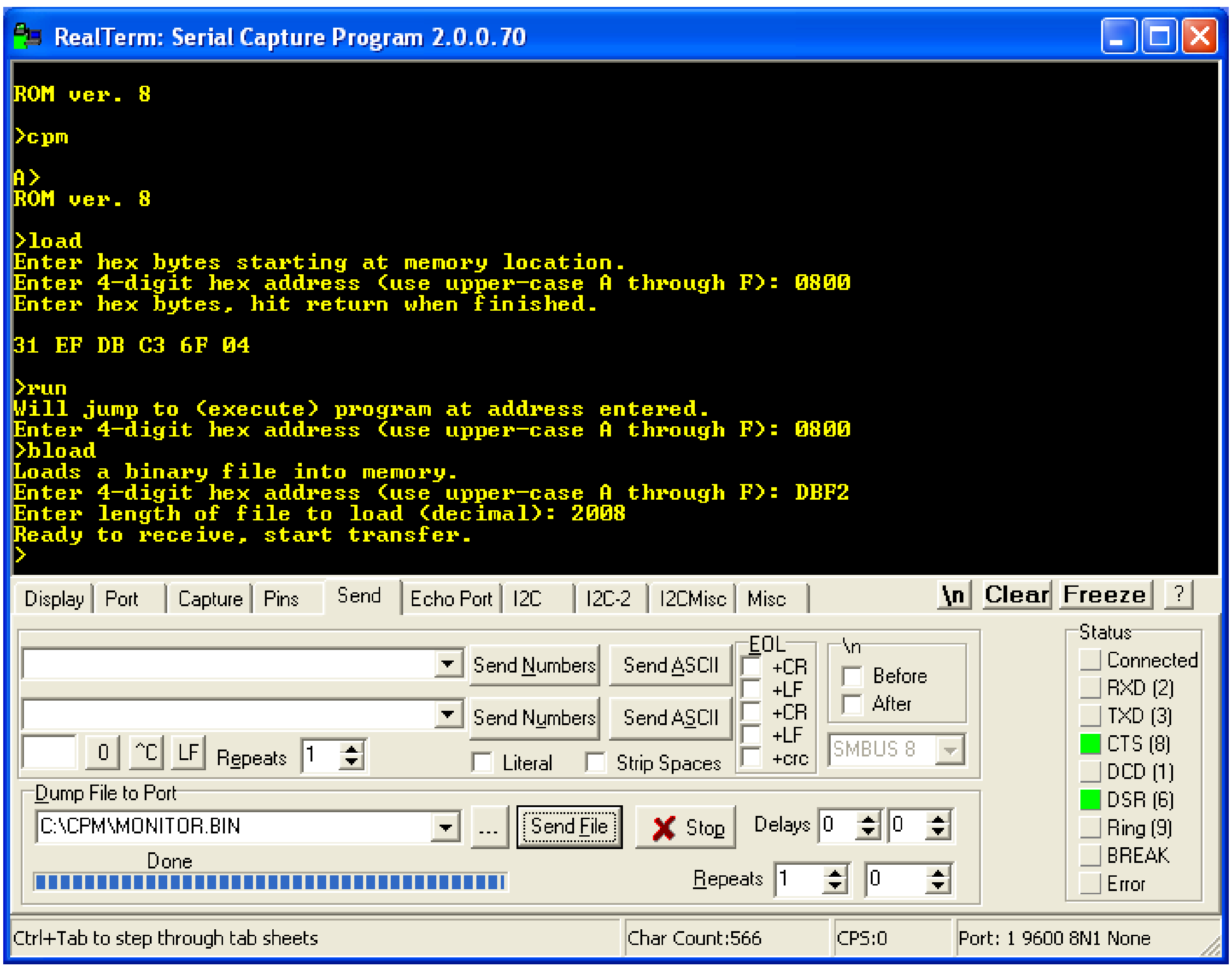
The RAM monitor code, has a short prefix, which will be used to relocate the file when we load it with CP/M. This means that we should load the monitor.bin file at 0xDBF2. Then, the RAM monitor code proper will start at 0xDC00 as designed. But, the ROM monitor uses stack space at 0xDBFF, so if we `bload` the file at 0xDBF2 the stack will be overwritten. To solve this problem, we just move the stack out of the way first with these commands:

```
0800 31 EF DB  ld sp,0DBEFh   ;move stack pointer out of the way
0803 C3 6F 04  jp 046Fh       ;ROM monitor warm start
```

We use the `load` command to put these bytes into memory at 0x0800 and execute them with `run`:

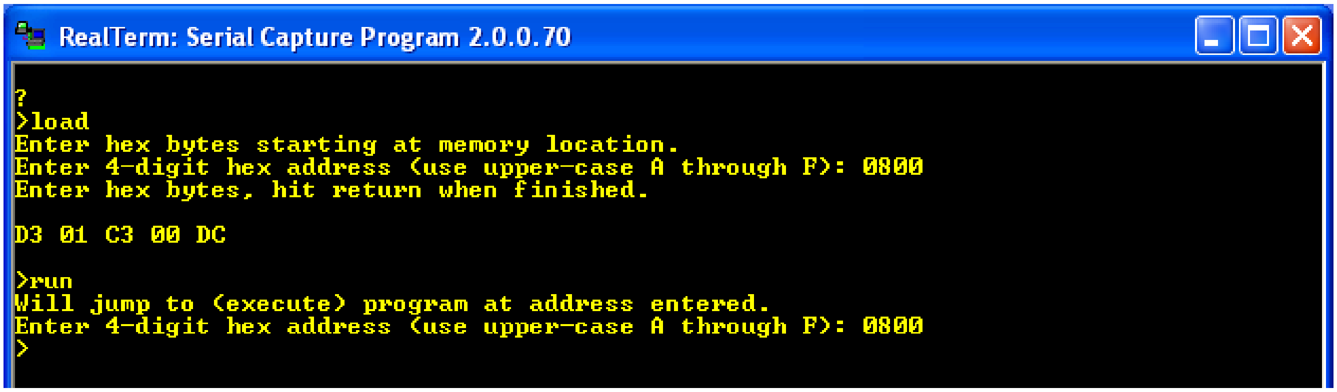


Now we can safely load the RAM monitor.bin file into memory at 0xDBF2:

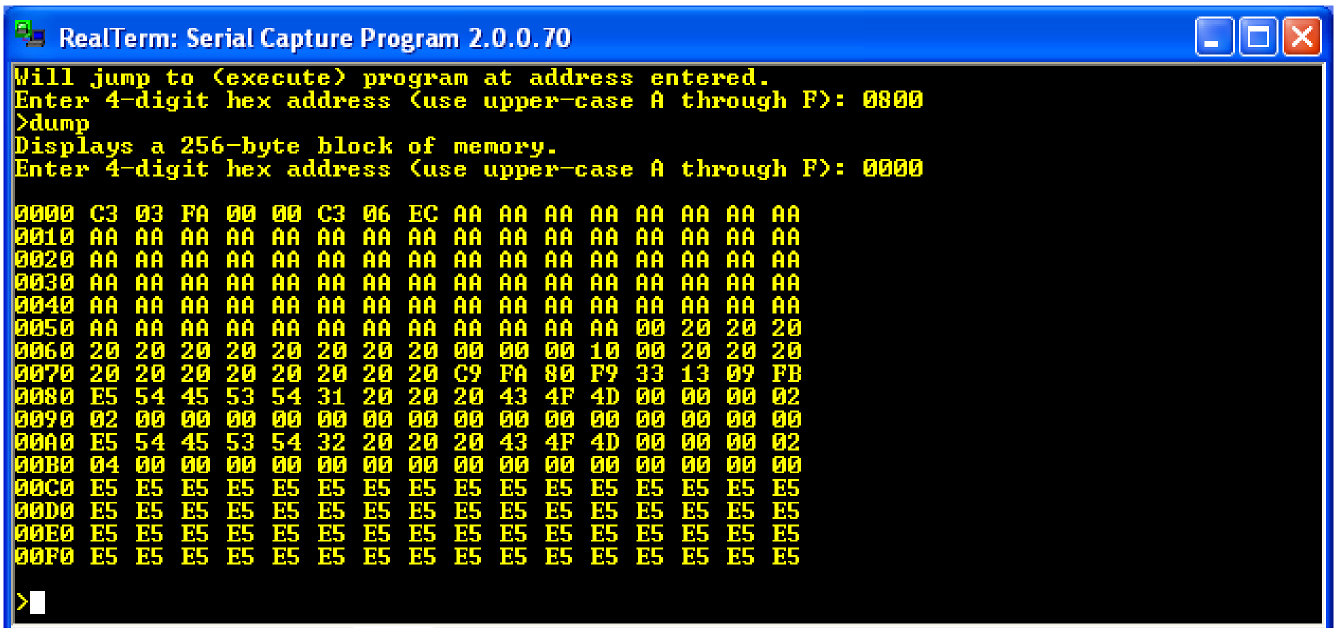


Then, we run some tiny code (again entered with the `load` command) to switch to memory configuration 1 and run the RAM monitor:

```
0800 D3 01     out (1),A ;switch to memory configuration 1 (all-RAM)
0802 C3 00 DC  jp 0DC00h ;jump to start of RAM monitor
```

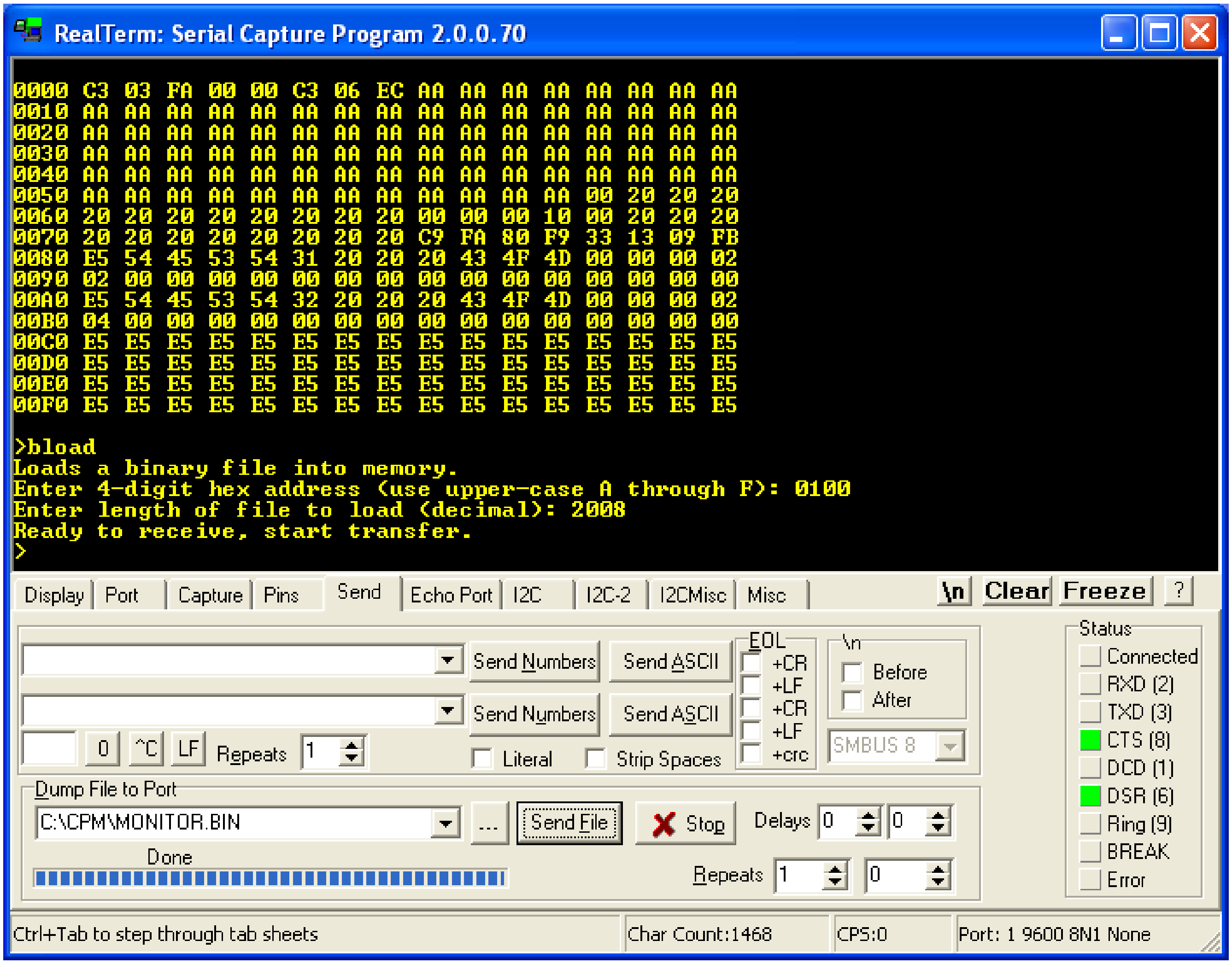


Now you see a monitor prompt (>), but it is now from the RAM monitor, running with the computer memory in configuration 1, and not the ROM monitor. To verify this, look at the first page of memory with the `dump` command:



There you see the CP/M warm start jump command at location 0x0000 with some other data. If we were still in memory configuration 0, this area would be filled with ROM code.

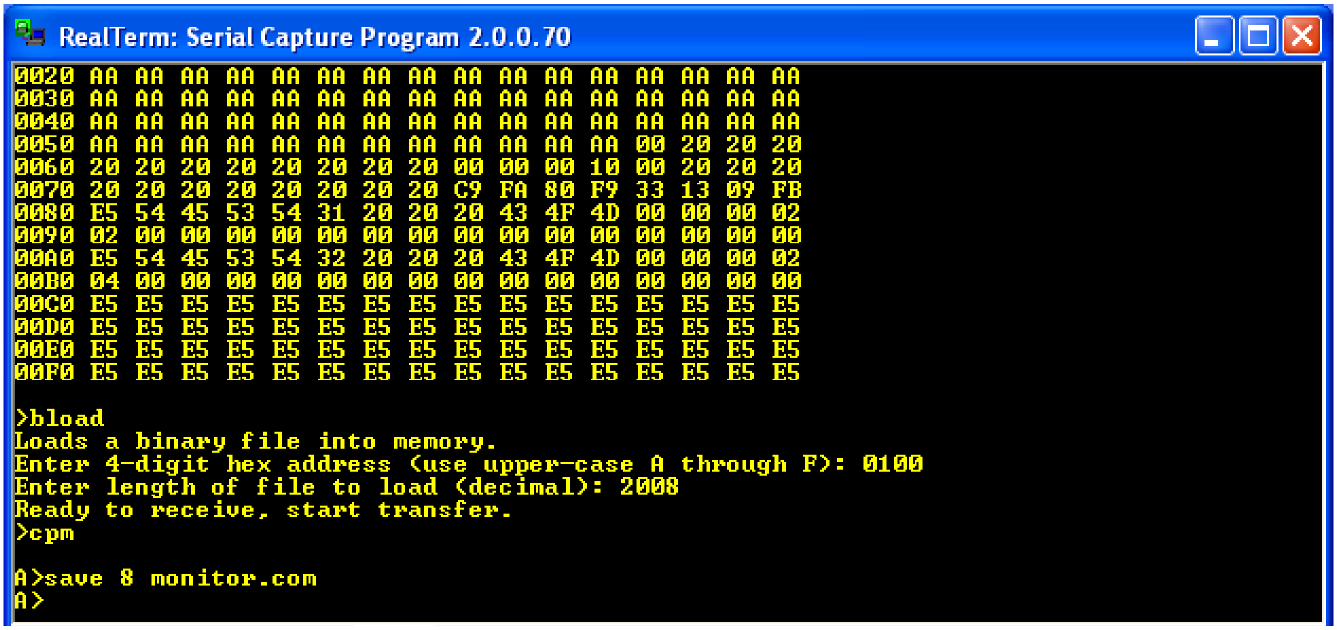
Now, using the RAM monitor, we can load the monitor.bin file again, this time at 0x0100:



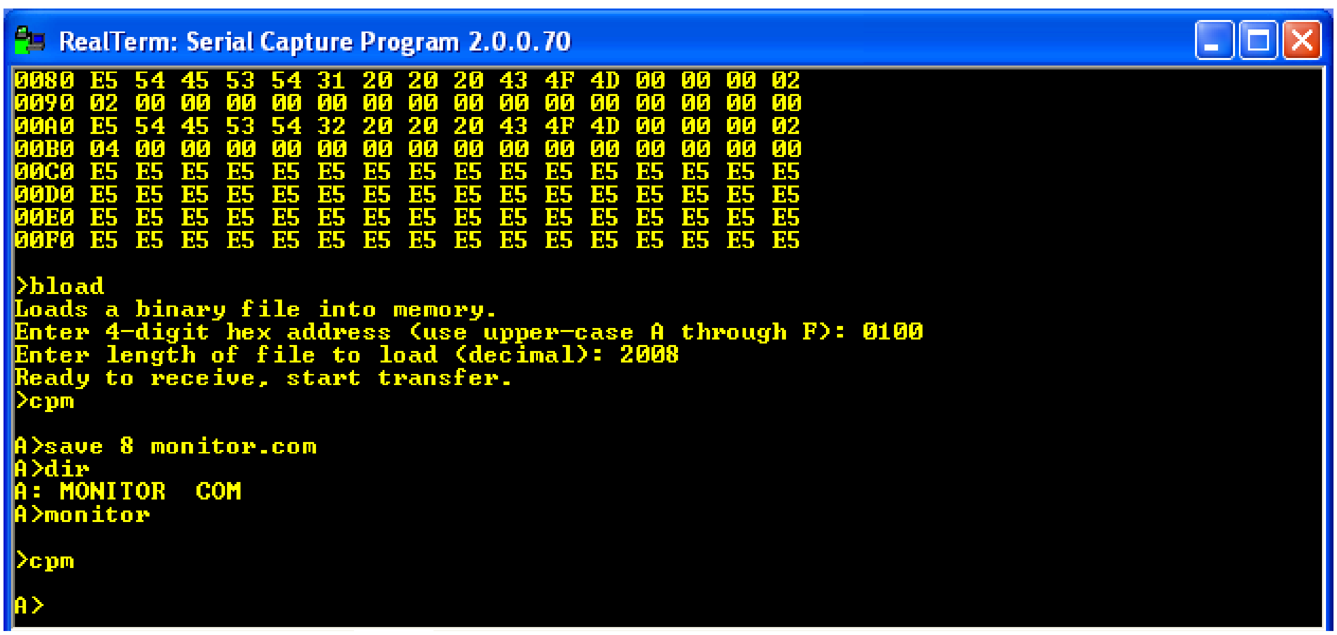
Now switch to CP/M by entering the RAM monitor `cpm` command. Unlike the ROM monitor `cpm` command, the `cpm` command in the RAM monitor does a CP/M warm start. When CP/M does a warm start it uses its own code in the CBIOS (which is in memory from 0xFA00 and above) to copy its BDOS and CCP code from the disk to the memory locations from 0xE400 and higher, but leaves the rest of the memory undisturbed[5]. So, the image of the RAM monitor at 0x0100 stays safe while CP/M reloads and restarts.

Now, we can use the CP/M SAVE command to create the disk file MONITOR.COM. We have to tell CP/M how many memory pages to save (one page = 256 bytes). If we divide the size of the monitor.bin file by 256 we get 2008/256 = 7.84. This means we need to save at least 8 pages of memory with the SAVE command. Give the file the name MONITOR.COM:

5 CP/M behaves this way to allow user programs to use the space from 0xE400 to 0xF9FF for their own code. When user programs return control to CP/M, it will load its code back in this space.



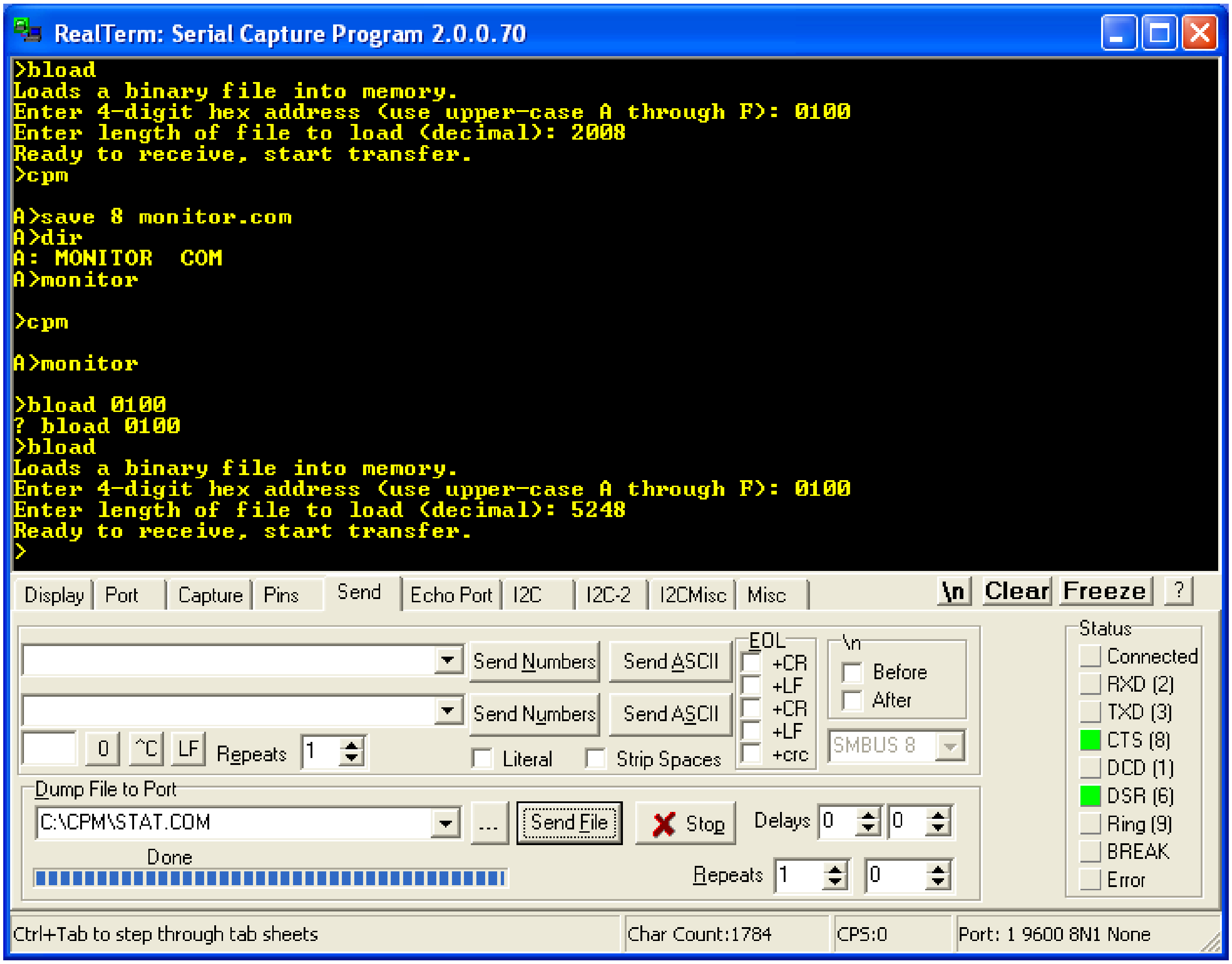
Check the disk directory, and you will see the MONITOR.COM file in place. Once this file is on the disk, all we need to do is enter MONITOR at the CP/M prompt, and we can use the monitor commands to do binary file transfers. When we are done with the monitor, we can enter the `cpm` command to return to CP/M:



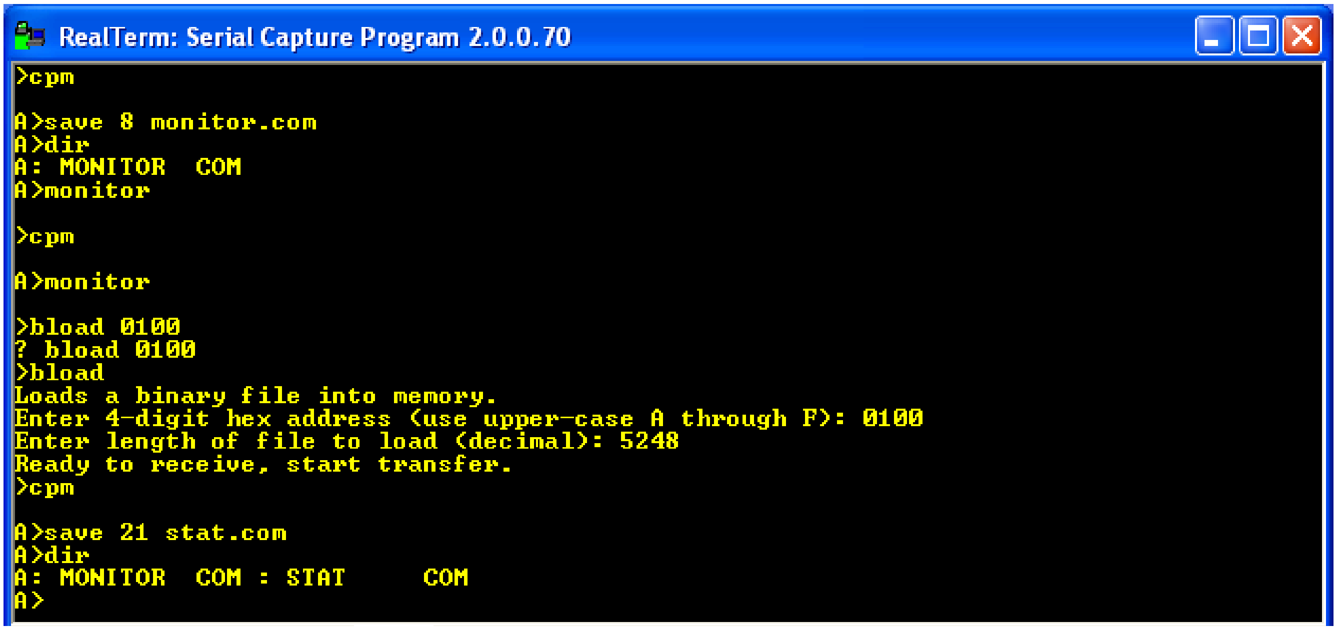
We can use the RAM monitor `bload` command to put any program we want into memory at 0x0100, provided it is not larger than 55,807 bytes (to keep it from running into monitor variables and stack space in page 0xDB00). Once a file is loaded, we can switch back to CP/M, and then SAVE the programs. We can load other types of files as well.

The first files we should load are the standard CP/M transient command files. The binary files for these commands can be obtained from The Unofficial CP/M Web Site. The binaries from a CP/M distribution disk are here: http://www.cpm.z80.de/download/cpm22-b.zip. The important ones are PIP.COM, ED.COM, ASM.COM, LOAD.COM, and STAT.COM. There is also DUMP.COM which displays file contents.

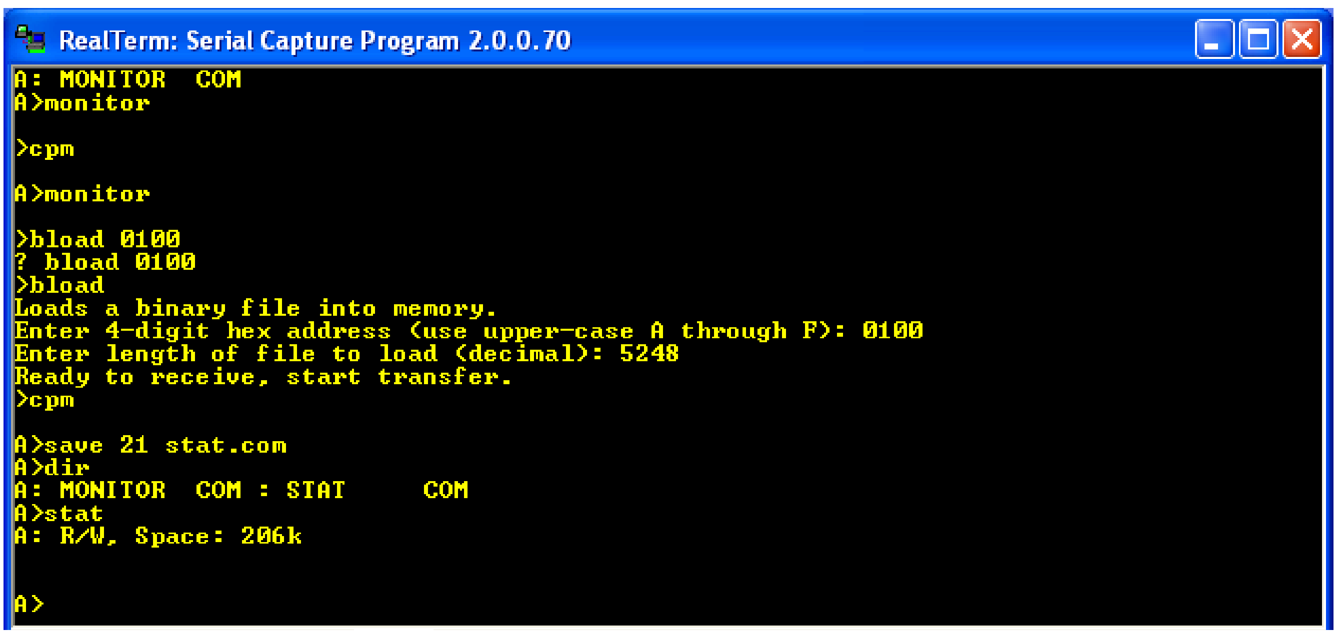
Let's use the MONITOR and SAVE commands to get STAT.COM onto our computer. Download a copy of STAT.COM from the above web site archive, enter the MONITOR command, and use the `bload` command to put the file into the Z80 computer memory at 0x0100:



After the file has been loaded, switch back to CP/M using the monitor `cpm` command. From the File Properties dialog on the PC, you can see that the STAT.COM file is 5,248 bytes long; it takes up 5,248/256 = 20.5 pages. So we need to save 21 pages to get all of the file. After SAVEing the file, you can see the file in the directory:



If you execute the STAT command, you can see how much room is available on the active CP/M disk:



If you give STAT a file name argument, it will tell you how big the file is:

RealTerm: Serial Capture Program 2.0.0.70

```
>bload 0100
? bload 0100
>bload
Loads a binary file into memory.
Enter 4-digit hex address (use upper-case A through F): 0100
Enter length of file to load (decimal): 5248
Ready to receive, start transfer.
>cpm

A>save 21 stat.com
A>dir
A: MONITOR  COM : STAT     COM
A>stat
A: R/W, Space: 206k


A>stat monitor.com

 Recs  Bytes  Ext Acc
   16     2k    1 R/W A:MONITOR.COM
Bytes Remaining On A: 206k

A>
```

Using the PCGET and PCPUT file transfer utilities

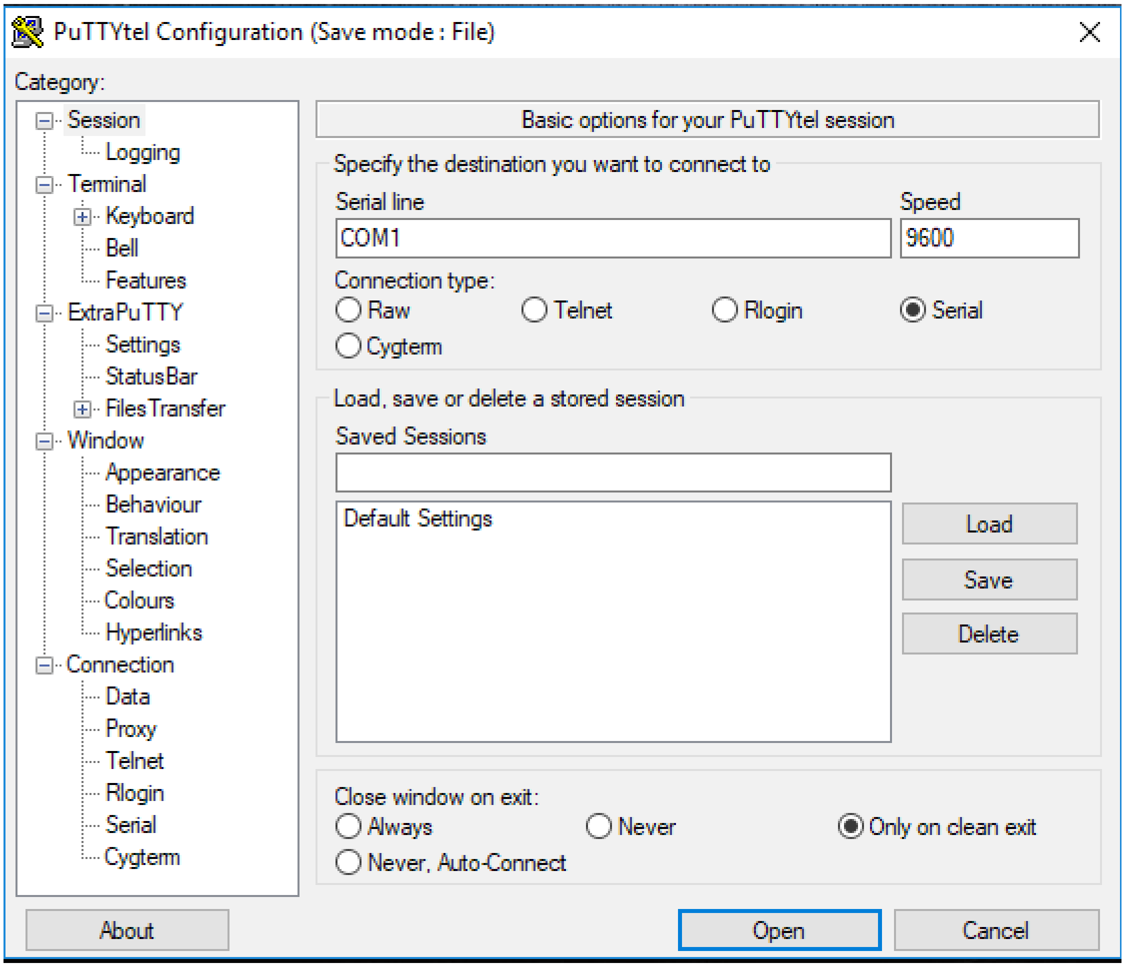
The method of using the MONITOR.COM program to do binary transfers is a little awkward. I had sought to use any of several XMODEM-type CP/M programs to do file transfers, but they all required a system with two serial ports, one for the terminal, and one for a modem to do the file transfer. However, customer Stephen Williams has modified two XMODEM CP/M utilities to perform file transfers from the PC to the CPUville Z80 kit computer over the single serial port. These utilities, PCGET and PCPUT were created by Mike Douglas for his Altair 8800 clone computer. He derived them from the original XMODEM-based file transfer utilities created by Ward Christensen in 1977 for his early bulletin board systems. With the permission of both Mike Douglas and Stephen Williams I have placed the code for these utilities on the CPUville CP/M code page for download.

PCGET will transfer a file from the PC over the serial interface onto the CP/M disk, and PCPUT will transfer a file from the CP/M disk to the PC. To do this, one must be using a terminal emulation program with the ability to do XMODEM-protocol file transfers. In the Linux environment, minicom will do this. In the Mac environment, the serial program will work. In Windows however, the Realterm program used frequently in this instruction manual does not do XMODEM transfers. Instead, you can use the ExtraPuTTY terminal emulaton program.

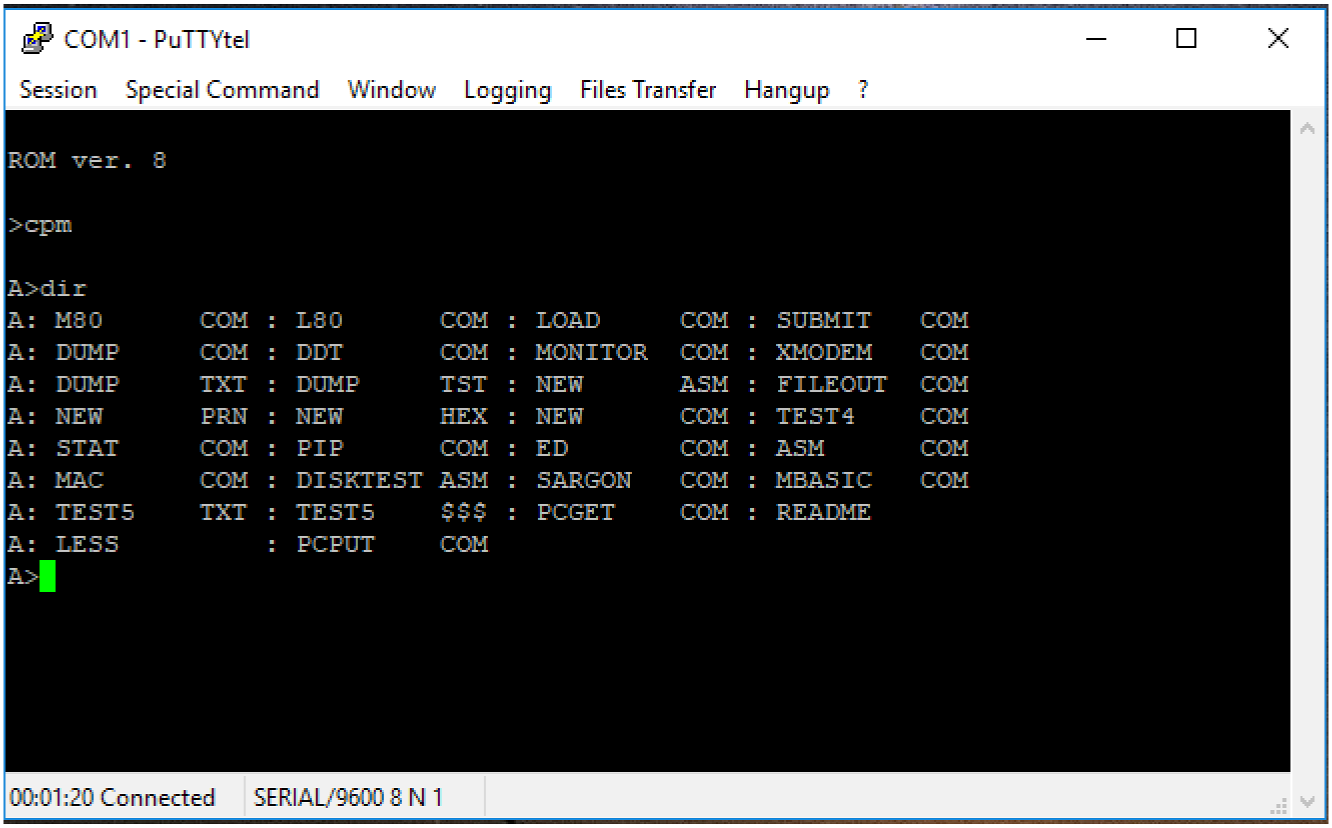
One last thing: to get PCGET.COM onto the CPUville computer you will have to do the MONITOR.COM binary transfer and CP/M SAVE procedure, as explained above. Transfer the PCGET.BIN binary file into memory at 0x0100 using the monitor `bload` command, then switch to `cpm` and use the CP/M SAVE command to create the file PCGET.COM on the CP/M disk. After that, you can use PCGET as a CP/M command to do file transfers for the rest of the CP/M transient commands and other files.

For an example, I will show using PCGET to transfer the file CAPTURE from the PC disk to a CP/M system, using the ExtraPuTTY terminal emulation program in the Windows environment.

Start ExtraPuTTY. On the initial window, select the Serial communication type, the COM port associated with your serial interface (COM1 here), and 9600 baud:

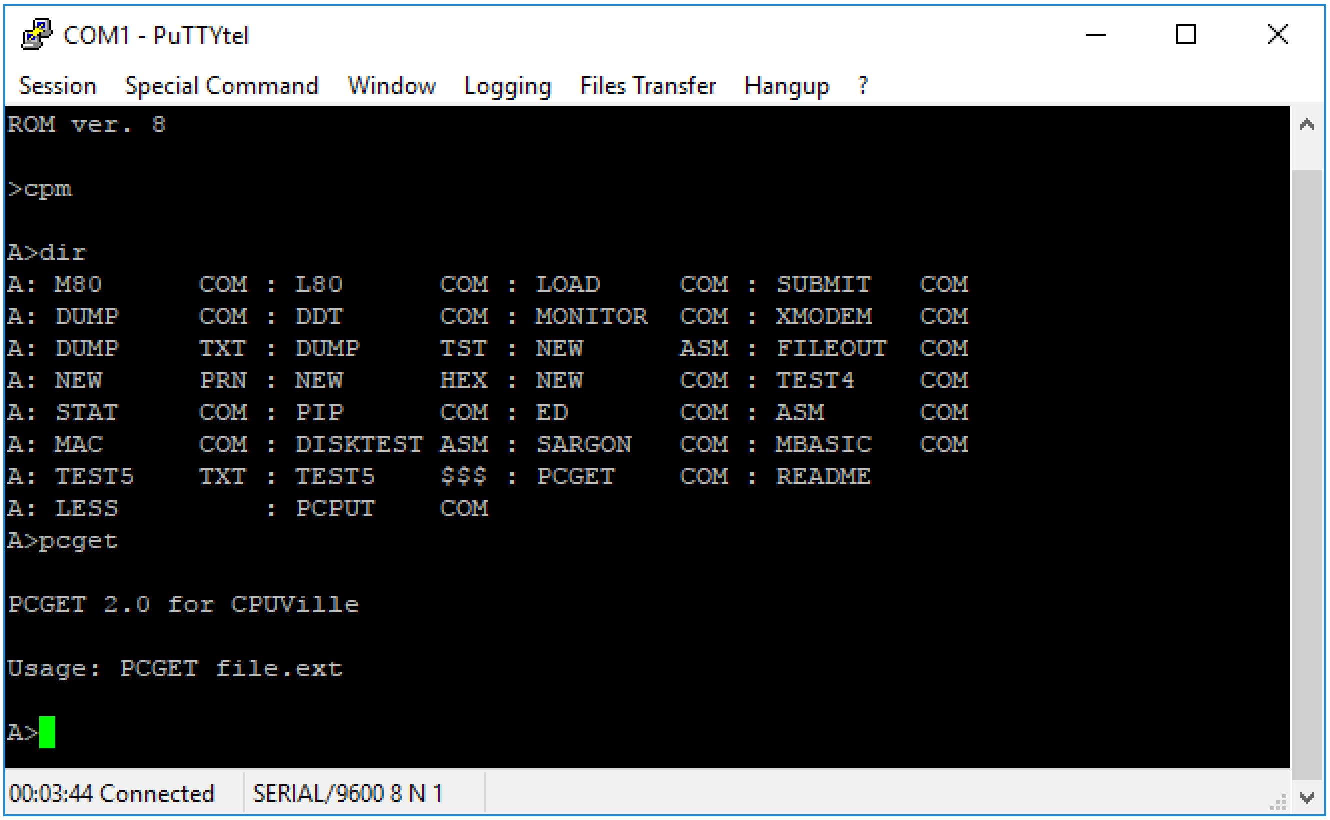


The terminal window opens. Take the Z80 computer out of reset, and you should get the ROM monitor greeting message and prompt. Here, I have started CP/M, and done a CP/M directory display:

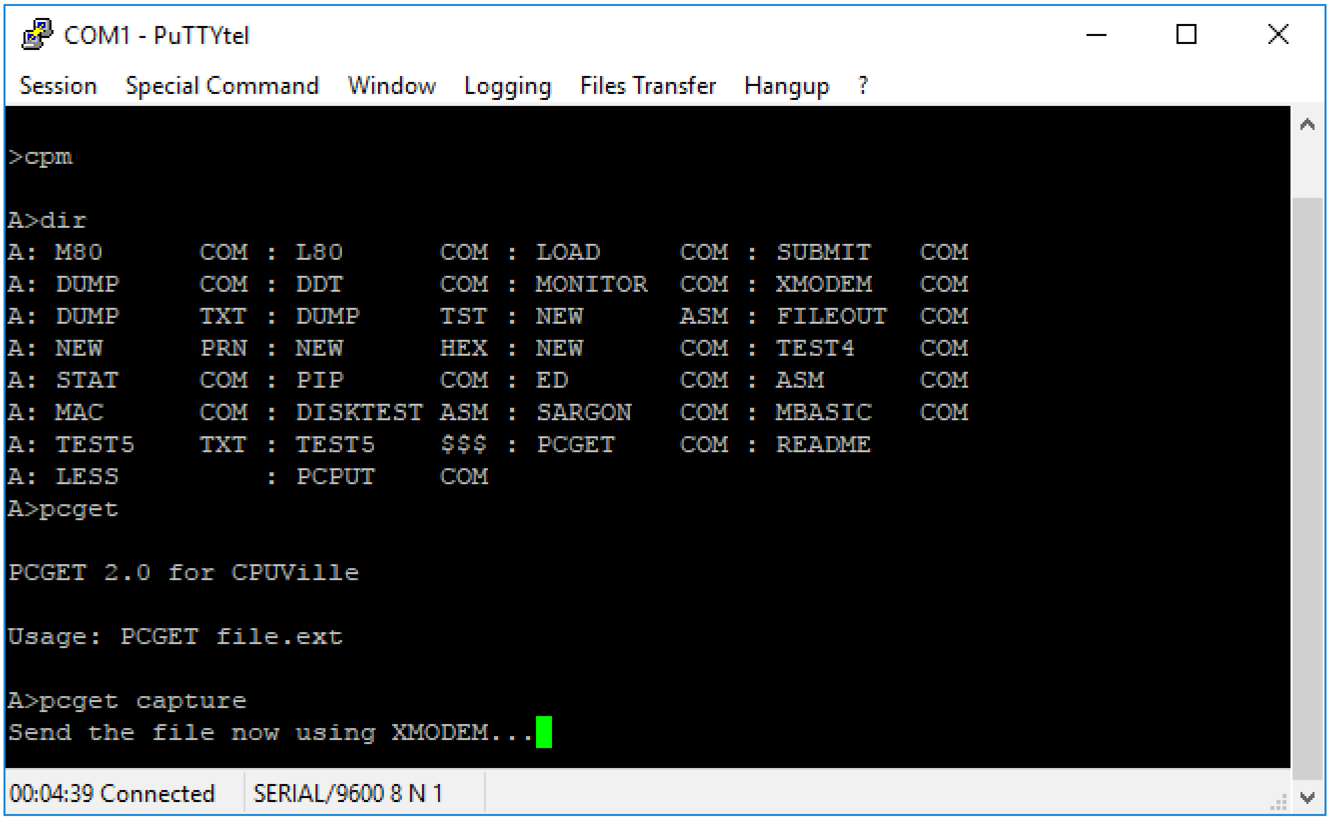


You can see I have already loaded PCGET.COM using the MONITOR.COM method.

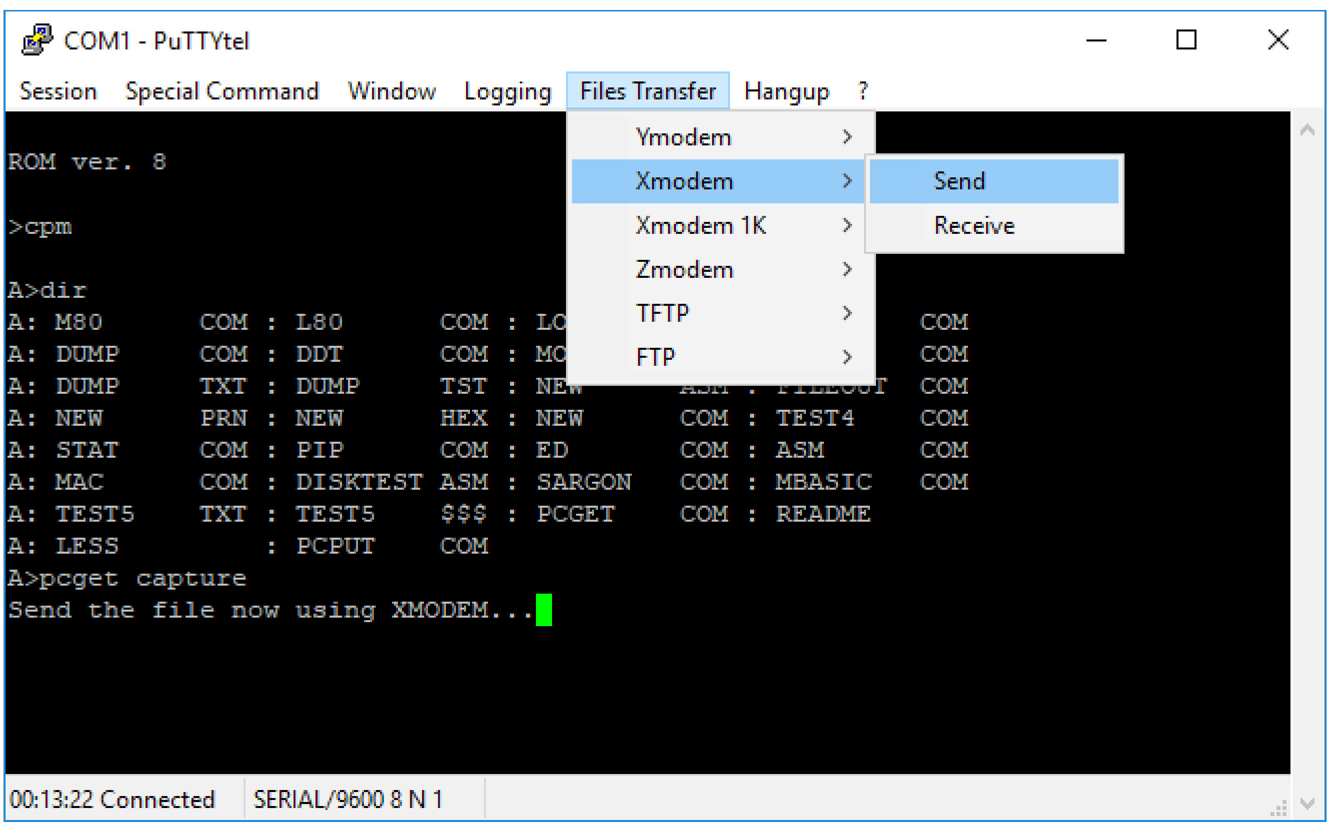
If you execute the PCGET command, a brief display reminds you of the usage:



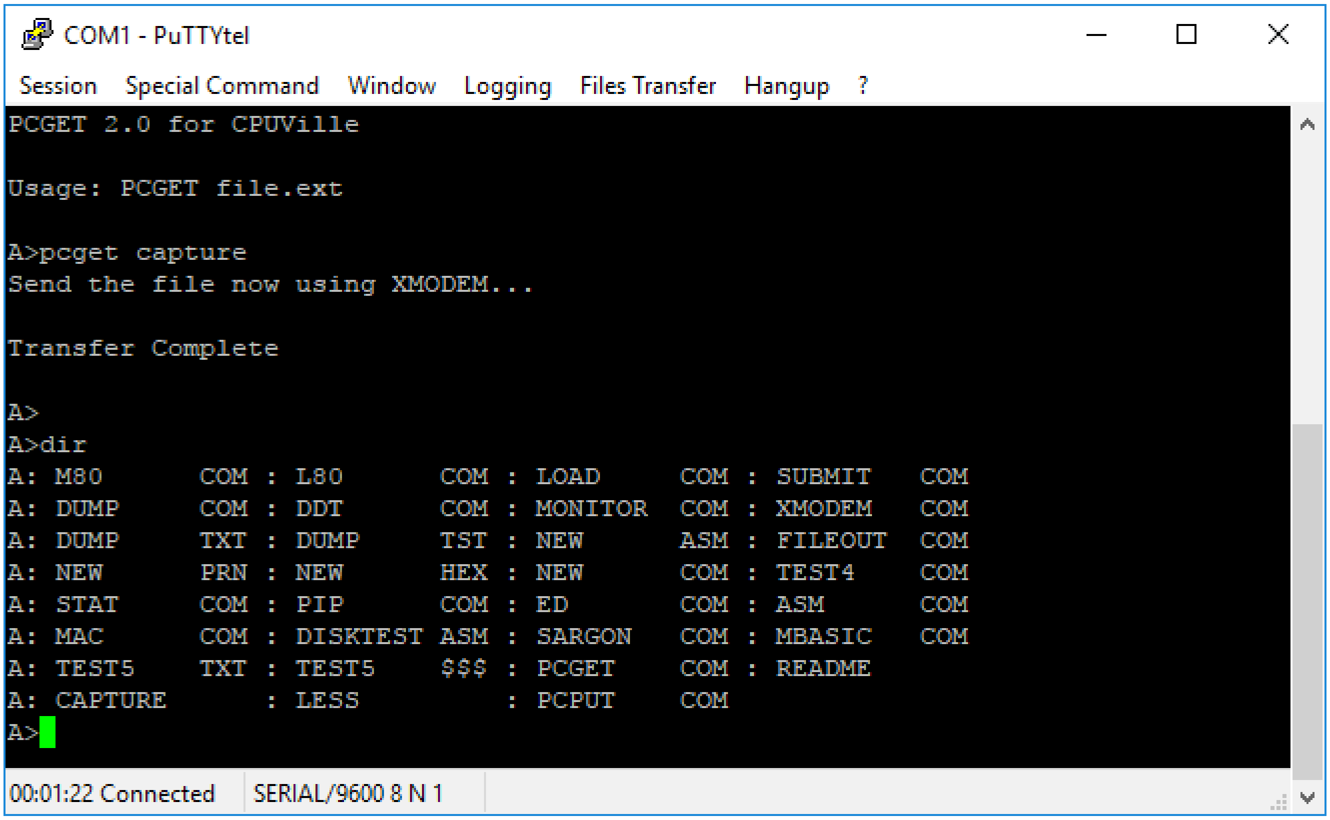
To load a file from the PC to CP/M, execute the PCGET command with the file name. Note that file names are not transferred, so the file name you use as the argument for PCGET is the name CP/M will assign to the file, not the name that the file on PC currently has. In this example, the file name is "capture":



At this prompt, navigate to the Files Transfer menu, and select Xmodem, Send:



A file menu opens that allows you to select the file to send. Click “Open”, and the transfer begins. Once the transfer is finished, PCGET quits with the message “Transfer complete” and sends you back to the CP/M prompt. Another `dir` command should show that the file “capture” is now on the CP/M disk:



PCPUT acts in a similar fashion, except you would select Xmodem, Recieve for the file transfer.

Using minicom in Linux, the procedure is similar. To send the file, do ctrl-A, S to open the Send File menu. You select the XMODEM protocol, then a window to select the file opens. Once a file is selected, the transfer proceeds.

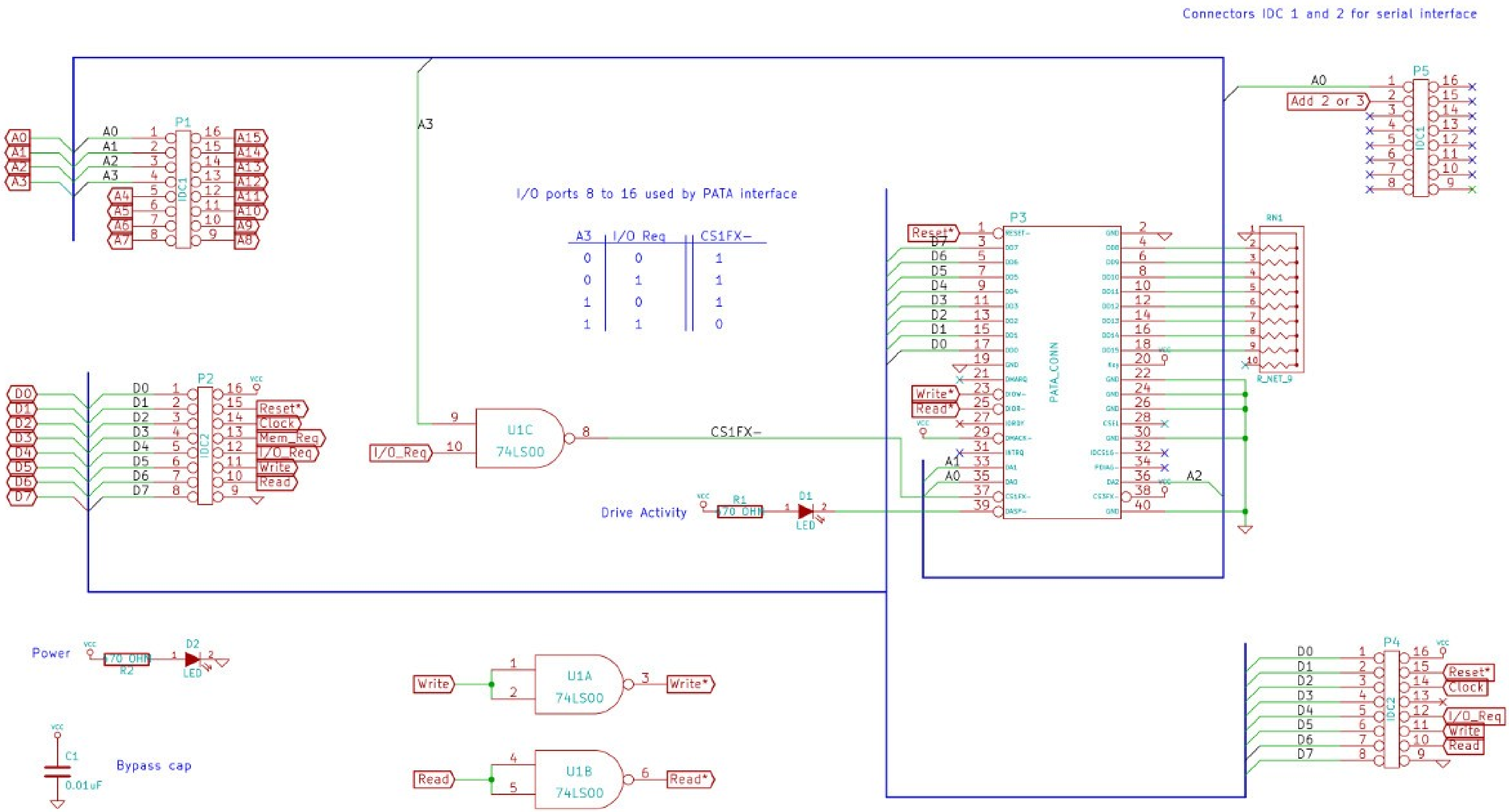
This concludes a description of the basics of using CP/M, including how to get binary files into the CP/M file system through the serial port of the Z80 computer. The Digital Research CP/M 2 System Manual, available on-line as stated above, explains how to use CP/M in full detail.

There are thousands of CP/M programs available, both on the web sites mentioned above, and on other archives. The Humongous CP/M Software Archives at http://www.classiccmp.org/cpmarchives/ is just one example. There is also Retrocomputing Archive at http://www.retroarchive.org/ There I found the Sargon program that plays chess better than I can. This concludes a description of the basics of using CP/M, including how to get binary files into the CP/M file system through the serial port of the Z80 computer. The Digital Research CP/M 2 System Manual, available on-line as stated above, explains how to use CP/M in full detail.

There are thousands of CP/M programs available, both on the web sites mentioned above, and on other archives. The Humongous CP/M Software Archives at http://www.classiccmp.org/cpmarchives/ is just one example. There is also Retrocomputing Archive at http://www.retroarchive.org/ There I found the Sargon program that plays chess better than I can.

Disk and Memory Expansion Schematics and Explanations

IDE Interface

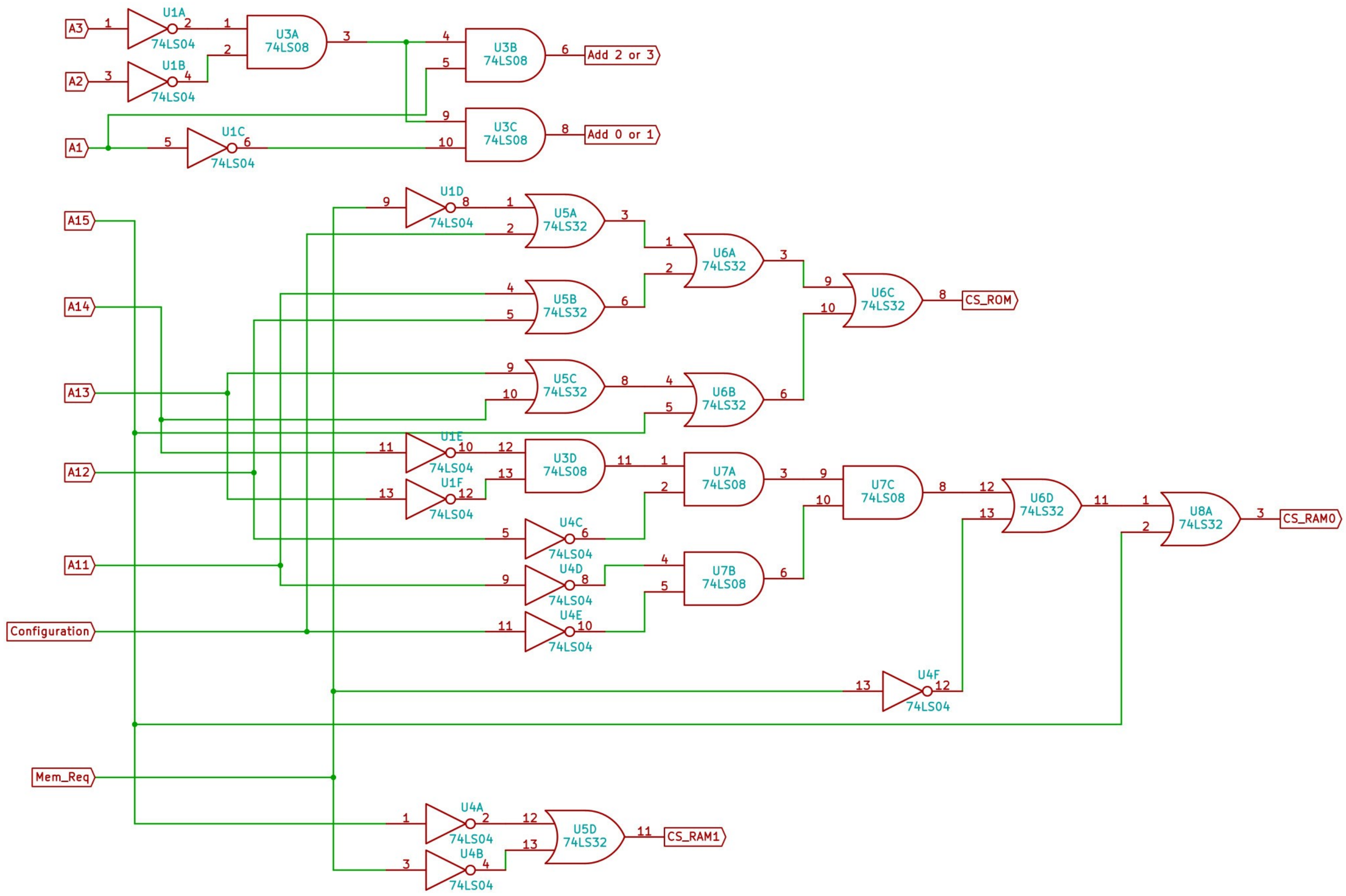


The IDE interface portion of the disk and memory is simpler than you might suppose. The computer system data, address, and control bus signals, with +5V and ground power lines, are brought to the disk and memory board through the P1 and P2 connectors. The data and control lines are passed to the serial interface through the P4 connector. Note the “Add 2 or 3” (meaning address 2 or 3) input to pin 2 of the P5 connector. This signal comes from the logic circuitry described below. It was needed because the serial interface board has minimal chip select logic on it, and will be activated for any input/output request for port addresses with A1 = 1. The Add 2 or 3 signal is sent to the serial interface connector P5 in the place of A1, so that other addresses that have A1 = 1, such as decimal 10, 11, 14, and 15, can be used for the disk interface, while leaving the serial interface undisturbed.

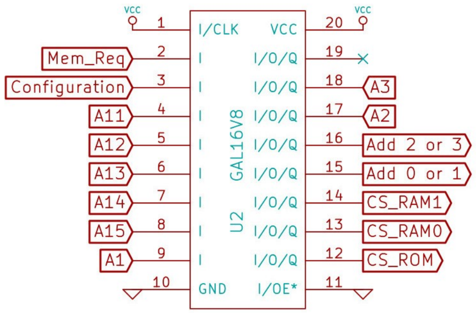
The IDE interface consists of TTL level signals sent to and from a series of input/output ports. The disk interface is selected when A3 = 1, and I/O_Req is asserted. Address lines A0 through A2 determine which IDE interface register is selected for reading/writing. The IDE connector (labeled PATA_CONN in the schematic) can be connected directly to the system data bus, because it has three-state outputs. Here, data bits 0 to 7 are connected, and bits 8 to 15 are grounded through 1K resistors. This means for disks running in 16-bit mode half the data is not retrievable, a trade-off to make the hardware simpler.

Port and Memory Address Logic and Memory Configuration Flip-flop

The original version of the disk and memory expansion board had the following logic circuits, made with 7 discrete logic ICs:



In the current version, these two logic circuits are implemented by a single programmable gate-array logic IC, the GAL16V8:



Notice the 10 input signals for the circuit are fed into the GAL pins 2-9, 17, and 18. The outputs are seen on pins 12 to 16.

The first logic circuit has inputs A1, A2 and A3. One output of this circuit is the signal Add 2 or 3 which is passed to the serial interface connector as described in the section above. The other output is Add 0 or 1 (“address zero or one”) that is an input to the memory configuration port circuit (see below).

This logic circuit performs the following calculations (the Add 0 or 1 and Add 2 or 3 are active-high, that is, are logical 1 or +5V when asserted):

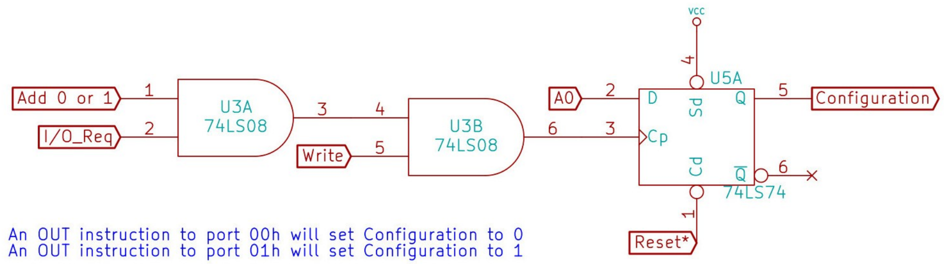
- Assert Add 0 or 1 if A1 and A2 and A3 are all zero.
- Assert Add 2 or 3 if A1 = 1 and, A2 and A3 are zero.

The formal logic equations for these two outputs are:

Add 0 or 1 = ~A3 ~A2 ~A1

Add 2 or 3 = ~A3 ~A2 * A1

Here is the memory configuration port circuit:



There is a small logic circuit made of two AND gates that creates a clock pulse that is fed to the configuration flip-flop when an OUT (0),A or OUT (1),A instruction is executed. The A0 bit is latched and becomes the Configuration bit. Note that the system Reset* signal is fed to the flip-flop. This ensures that the flip-flop is in configuration 0 when the system starts, which is necessary for code execution to start in the ROM.

The configuration bit becomes one of the inputs to the other logic circuit in the GAL, which also has inputs A11 to A15, and Mem_Req. The outputs of this logic circuit are the chip select (CS) signals for the ROM and the two RAM ICs. The logic performs the following calculation (the CS signals are all active-low, that is, are logical 0 or GND when asserted):

- Assert CS_ROM if Configuration is 0, Mem_Req is asserted, and the address is 0x0000 to 0x07FF – that is, if A11 to A15 are all zero.
- Assert CS_RAM0 if Configuration is 0, Mem_Req is asserted, and the address is 0x0800 to 0x7FFF – that is, A15 is zero, and any of A11 to A15 is 1.
- Assert CS_RAM0 if Configuration is 1, Mem_Req is asserted, and the address is 0x0000 to 0x7FFF – that is, if A15 is zero
- Assert CS_RAM1 if Configuration is 0 or 1 (a “don't care”) and the address is 0x8000 to 0xFFFF – that is, A15 is one.

The formal logic equations are here:

CS_ROM = ~Mem_Req + Config + A11 + A12 + A13 + A14 + A15

CS_RAM0 = ~A14 ~A13 ~A12 ~A11 ~Config + ~Mem_Req + A15

CS_RAM1 = ~A15 + ~Mem_Req

I used the Logisim program to help design these logic circuits. I used the Galasm program to create the fuse map file used to program the GAL16V8. Here is the Galasam PLD file:

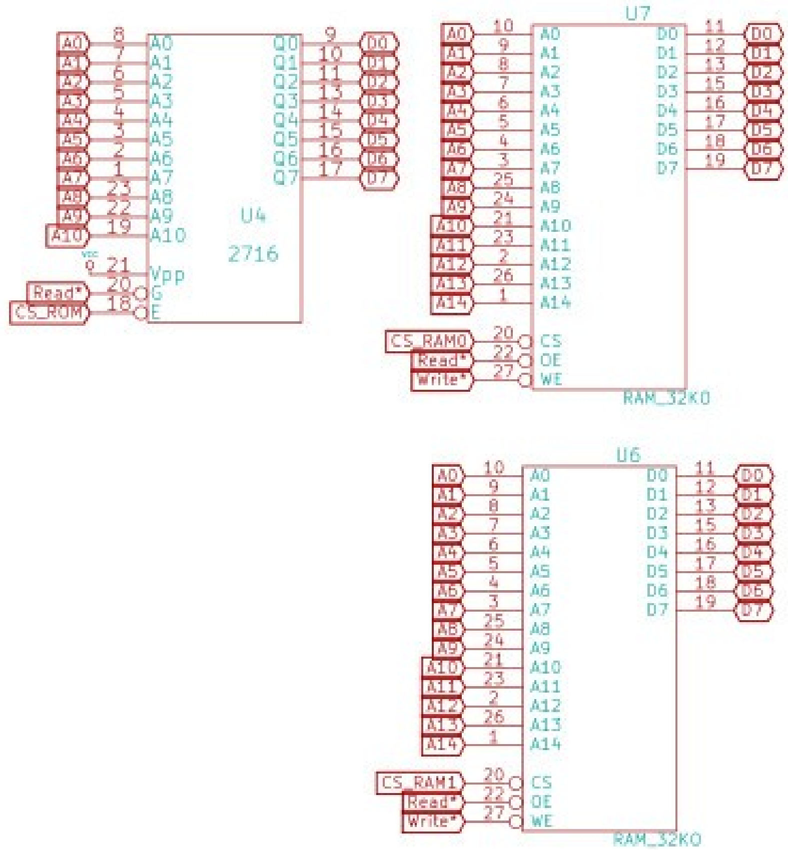
```
GAL16V8          ; this is the GAL type
Memory Logic 1        ; this is the signature

CLK  MemReq   Config   A11   A12  A13  A14  A15  A1  GND       ; this is the pin declaration
/OE  CSROM  CSRAM0  CSRAM1  Addr0or1  Addr2or3  A2  A3  A  VCC

CSROM = /MemReq + Config + A11 + A12 + A13 + A14 + A15       ; here are the pin definitions
CSRAM0 = /A14 * /A13 * /A12 * /A11 * /Config + /MemReq + A15
CSRAM1 = /A15 + /MemReq
Addr0or1 = /A3 * /A2 * /A1
Addr2or3 = /A3 * /A2 * A1

DESCRIPTION:
This is the memory select logic for the CPUville Disk and Memory Expansion board, with the addition of
logic to produce the Add 0 or 1 and Add 2 or 3 outputs.
```

Memory ICs



The ROM and RAM ICs are connected to the address and data buses. The ROM IC is connected to the system Read* signal, and the RAM ICs are connected to both the Read* and Write* signals. The chip select (CS) inputs come from the logic discussed in the section above. The logic is designed so that only one of these three ICs is active at any time.

Disk and Memory Expansion v.2 Parts Organizer and List

Capacitor, 0.01 uF ceramic 1	Red LED 2	Resistor, 470 ohm Yellow-Violet-Brown 2	DIL16 socket 4
DIL28 socket, 0.3 inch 2	DIL 24 socket 1	74LS00 1	DIL20 socket 1
74LS08 1	32K SRAM 2	74LS74 1	Resistor network, 1K x 9 1
40-pin IDE connector 1	2716 EPROM 1	GAL16V8 1	

C1	0.01uF	R2	470 OHM
D1	LED	RN1	R_NET_9
D2	LED	U1	74LS00
P1	IDC1	U2	GAL16V8
P2	IDC2	U3	74LS08
P3	PATA_CONN	U4	2716
P4	IDC2	U5	74LS74
P5	IDC1	U6	RAM_32KO
R1	470 OHM	U7	RAM_32KO

Selected Program Listings

ROM monitor[6]

```
# File 2K_ROM_8.asm
0000                  ;ROM monitor for a system with serial interface and IDE disk and memory expansion board.
0000                  ;Expansion board has 64K RAM -- computer board memory decoder disabled (J2 off).
0000                  ;Expansion board uses ports 2 and 3 for the serial interface, and 8 to 15 for the disk
0000                  ;Therefore the computer board I/O decoder is also disabled (J1 off)
0000                  ;Output to port 0 will cause memory configuration flip-flop to activate 2K ROM 0000-07FF,
0000                  ;with 62K RAM 0800-FFFF
0000                  ;Output to port 1 will cause memory configuration flip-flop to activate all RAM 0000-FFFF
0000                  ;
0000                                    org   00000h
0000 c3 63 04                           jp    monitor_cold_start
0003                  ;
0003                  ;The following code is for a system with a serial port.
0003                  ;Assumes the UART data port address is 02h and control/status address is 03h
0003                  ;
0003                  ;The subroutines for the serial port use these variables in RAM:
0003                  current_location: equ   0xdb00            ;word variable in RAM
0003                  line_count:       equ   0xdb02            ;byte variable in RAM
0003                  byte_count:       equ   0xdb03            ;byte variable in RAM
0003                  value_pointer:    equ   0xdb04            ;word variable in RAM
0003                  current_value:    equ   0xdb06            ;word variable in RAM
0003                  buffer:           equ   0xdb08            ;buffer in RAM -- up to stack area
0003                  ;Need to have stack in upper RAM, but not in area of CP/M or RAM monitor.
0003                  ROM_monitor_stack:      equ   0xdbff            ;upper TPA in RAM, below RAM monitor
0003                  ;
0003                  ;Subroutine to initialize serial port UART
0003                  ;Needs to be called only once after computer comes out of reset.
0003                  ;If called while port is active will cause port to fail.
0003                  ;16x = 9600 baud
0003 3e 4e            initialize_port:  ld    a,04eh            ;1 stop bit, no parity, 8-bit char, 16x baud
```

6 The RAM monitor program monitor.bin is identical to this ROM monitor, except it was assembled to target address 0xDC00, has a small code prefix to move the code to this location after CP/M loads it at 0x0100, and responds to the `cpm` command with a CP/M warm start, not a cold start as does the ROM monitor.

```
0005 d3 03                        out   (3),a             ;write to control port
0007 3e 37                        ld    a,037h            ;enable receive and transmit
0009 d3 03                        out   (3),a             ;write to control port
000b c9                           ret
000c                  ;
000c                  ;Puts a single char (byte value) on serial output
000c                  ;Call with char to send in A register. Uses B register
000c 47               write_char:       ld    b,a               ;store char
000d db 03            write_char_loop:  in    a,(3)             ;check if OK to send
000f e6 01                        and   001h              ;check TxRDY bit
0011 ca 0d 00                     jp    z,write_char_loop ;loop if not set
0014 78                           ld    a,b               ;get char back
0015 d3 02                        out   (2),a             ;send to output
0017 c9                           ret                     ;returns with char in a
0018                  ;
0018                  ;Subroutine to write a zero-terminated string to serial output
0018                  ;Pass address of string in HL register
0018                  ;No error checking
0018 db 03            write_string:     in    a,(3)             ;read status
001a e6 01                        and   001h              ;check TxRDY bit
001c ca 18 00                     jp    z,write_string    ;loop if not set
001f 7e                           ld    a,(hl)            ;get char from string
0020 a7                           and   a                 ;check if 0
0021 c8                           ret   z                 ;yes, finished
0022 d3 02                        out   (2),a             ;no, write char to output
0024 23                           inc   hl                ;next char in string
0025 c3 18 00                     jp    write_string      ;start over
0028                  ;
0028                  ;Binary loader. Receive a binary file, place in memory.
0028                  ;Address of load passed in HL, length of load (= file length) in BC
0028 db 03            bload:            in    a,(3)             ;get status
002a e6 02                        and   002h              ;check RxRDY bit
002c ca 28 00                     jp    z,bload           ;not ready, loop
002f db 02                        in    a,(2)
0031 77                           ld    (hl),a
0032 23                           inc   hl
0033 0b                           dec   bc                ;byte counter
0034 78                           ld    a,b               ;need to test BC this way because
0035 b1                           or    c                 ;dec rp instruction does not change flags
0036 c2 28 00                     jp    nz,bload
```

```
0039 c9                           ret
003a             ;
003a             ;Binary dump to port. Send a stream of binary data from memory to serial output
003a             ;Address of dump passed in HL, length of dump in BC
003a db 03       bdump:           in    a,(3)           ;get status
003c e6 01                        and   001h            ;check TxRDY bit
003e ca 3a 00                     jp    z,bdump         ;not ready, loop
0041 7e                           ld    a,(hl)
0042 d3 02                        out   (2),a
0044 23                           inc   hl
0045 0b                           dec   bc
0046 78                           ld    a,b             ;need to test this way because
0047 b1                           or    c               ;dec rp instruction does not change flags
0048 c2 3a 00                     jp    nz,bdump
004b c9                           ret
004c             ;
004c             ;Subroutine to get a string from serial input, place in buffer.
004c             ;Buffer address passed in HL reg.
004c             ;Uses A,BC,DE,HL registers (including calls to other subroutines).
004c             ;Line entry ends by hitting return key. Return char not included in string (replaced by zero).
004c             ;Backspace editing OK. No error checking.
004c             ;
004c 0e 00       get_line:              ld    c,000h                  ;line position
004e 7c                                 ld    a,h                     ;put original buffer address in de
004f 57                                 ld    d,a                     ;after this don't need to preserve hl
0050 7d                                 ld    a,l                     ;subroutines called don't use de
0051 5f                                 ld    e,a
0052 db 03       get_line_next_char:    in    a,(3)                   ;get status
0054 e6 02                              and   002h                    ;check RxRDY bit
0056 ca 52 00                           jp    z,get_line_next_char    ;not ready, loop
0059 db 02                              in    a,(2)                   ;get char
005b fe 0d                              cp    00dh                    ;check if return
005d c8                                 ret   z                       ;yes, normal exit
005e fe 7f                              cp    07fh                    ;check if backspace (VT102 keys)
0060 ca 74 00                           jp    z,get_line_backspace    ;yes, jump to backspace routine
0063 fe 08                              cp    008h                    ;check if backspace (ANSI keys)
0065 ca 74 00                           jp    z,get_line_backspace    ;yes, jump to backspace
0068 cd 0c 00                           call  write_char              ;put char on screen
006b 12                                 ld    (de),a                  ;store char in buffer
006c 13                                 inc   de                      ;point to next space in buffer
```

```
006d 0c                             inc   c                        ;inc counter
006e 3e 00                          ld    a,000h
0070 12                             ld    (de),a                   ;leaves zero-terminated string in buffer
0071 c3 52 00                       jp    get_line_next_char
0074 79           get_line_backspace:     ld    a,c                ;check current position in line
0075 fe 00                          cp    000h                     ;at beginning of line?
0077 ca 52 00                       jp    z,get_line_next_char     ;yes, ignore backspace, get next char
007a 1b                             dec   de                       ;no, erase char from buffer
007b 0d                             dec   c                        ;back up one
007c 3e 00                          ld    a,000h                   ;put zero in place of last char
007e 12                             ld    (de),a
007f 21 84 03                       ld    hl,erase_char_string     ;ANSI seq. To delete one char from line
0082 cd 18 00                       call  write_string             ;transmits seq. to BS and erase char
0085 c3 52 00                       jp    get_line_next_char
0088              ;
0088              ;Creates a two-char hex string from the byte value passed in register A
0088              ;Location to place string passed in HL
0088              ;String is zero-terminated, stored in 3 locations starting at HL
0088              ;Also uses registers b,d, and e
0088 47           byte_to_hex_string:     ld    b,a          ;store original byte
0089 cb 3f                          srl   a                  ;shift right 4 times, putting
008b cb 3f                          srl   a                  ;high nybble in low-nybble spot
008d cb 3f                          srl   a                  ;and zeros in high-nybble spot
008f cb 3f                          srl   a
0091 16 00                          ld    d,000h             ;prepare for 16-bit addition
0093 5f                             ld    e,a                ;de contains offset
0094 e5                             push  hl                 ;temporarily store string target address
0095 21 ee 00                       ld    hl,hex_char_table ;use char table to get high-nybble character
0098 19                             add   hl,de              ;add offset to start of table
0099 7e                             ld    a,(hl)             ;get char
009a e1                             pop   hl                 ;get string target address
009b 77                             ld    (hl),a             ;store first char of string
009c 23                             inc   hl                 ;point to next string target address
009d 78                             ld    a,b                ;get original byte back from reg b
009e e6 0f                          and   00fh               ;mask off high-nybble
00a0 5f                             ld    e,a                ;d still has 000h, now de has offset
00a1 e5                             push  hl                 ;temp store string target address
00a2 21 ee 00                       ld    hl,hex_char_table ;start of table
00a5 19                             add   hl,de              ;add offset
00a6 7e                             ld    a,(hl)             ;get char
```

```
00a7 e1                               pop   hl                ;get string target address
00a8 77                               ld    (hl),a            ;store second char of string
00a9 23                               inc   hl                ;point to third location
00aa 3e 00                            ld    a,000h            ;zero to terminate string
00ac 77                               ld    (hl),a            ;store the zero
00ad c9                               ret                     ;done
00ae             ;
00ae             ;Converts a single ASCII hex char to a nybble value
00ae             ;Pass char in reg A. Letter numerals must be upper case.
00ae             ;Return nybble value in low-order reg A with zeros in high-order nybble if no error.
00ae             ;Return 0ffh in reg A if error (char not a valid hex numeral).
00ae             ;Also uses b, c, and hl registers.
00ae 21 ee 00    hex_char_to_nybble:  ld    hl,hex_char_table
00b1 06 0f                            ld    b,00fh                  ;no. of valid characters in table - 1.
00b3 0e 00                            ld    c,000h                  ;will be nybble value
00b5 be          hex_to_nybble_loop:  cp    (hl)                    ;character match here?
00b6 ca c2 00                         jp    z,hex_to_nybble_ok      ;match found, exit
00b9 05                               dec   b                       ;no match, check if at end of table
00ba fa c4 00                         jp    m,hex_to_nybble_err     ;table limit exceded, exit with error
00bd 0c                               inc   c                       ;still inside table, continue search
00be 23                               inc   hl
00bf c3 b5 00                         jp    hex_to_nybble_loop
00c2 79          hex_to_nybble_ok:    ld    a,c                     ;put nybble value in a
00c3 c9                               ret
00c4 3e ff       hex_to_nybble_err:   ld    a,0ffh                  ;error value
00c6 c9                               ret
00c7             ;
00c7             ;Converts a hex character pair to a byte value
00c7             ;Called with location of high-order char in HL
00c7             ;If no error carry flag clear, returns with byte value in register A, and
00c7             ;HL pointing to next mem location after char pair.
00c7             ;If error (non-hex char) carry flag set, HL pointing to invalid char
00c7 7e          hex_to_byte:         ld    a,(hl)            ;location of character pair
00c8 e5                               push  hl                ;store hl (hex_char_to_nybble uses it)
00c9 cd ae 00                         call  hex_char_to_nybble
00cc e1                               pop   hl                ;ret. with nybble in A reg, or 0ffh if error
00cd fe ff                            cp    0ffh              ;non-hex character?
00cf ca ec 00                         jp    z,hex_to_byte_err ;yes, exit with error
00d2 cb 27                            sla   a                 ;no, move low order nybble to high side
00d4 cb 27                            sla   a
```

```
00d6 cb 27                          sla   a
00d8 cb 27                          sla   a
00da 57                             ld    d,a                ;store high-nybble
00db 23                             inc   hl                 ;get next character of the pair
00dc 7e                             ld    a,(hl)
00dd e5                             push  hl                 ;store hl
00de cd ae 00                       call  hex_char_to_nybble
00e1 e1                             pop   hl
00e2 fe ff                          cp    0ffh               ;non-hex character?
00e4 ca ec 00                       jp    z,hex_to_byte_err ;yes, exit with error
00e7 b2                             or    d                  ;no, combine with high-nybble
00e8 23                             inc   hl                 ;point to next memory location after char pair
00e9 37                             scf
00ea 3f                             ccf                      ;no-error exit (carry = 0)
00eb c9                             ret
00ec 37           hex_to_byte_err:  scf                      ;error, carry flag set
00ed c9                             ret
00ee ..           hex_char_table:   defm  "0123456789ABCDEF"      ;ASCII hex table
00fe              ;
00fe              ;Subroutine to get a two-byte address from serial input.
00fe              ;Returns with address value in HL
00fe              ;Uses locations in RAM for buffer and variables
00fe 21 08 db     address_entry:    ld    hl,buffer               ;location for entered string
0101 cd 4c 00                       call  get_line                ;returns with address string in buffer
0104 21 08 db                       ld    hl,buffer               ;location of stored address entry string
0107 cd c7 00                       call  hex_to_byte             ;will get high-order byte first
010a da 20 01                       jp    c, address_entry_error  ;if error, jump
010d 32 01 db                       ld    (current_location+1),a  ;store high-order byte, little-endian
0110 21 0a db                       ld    hl,buffer+2             ;point to low-order hex char pair
0113 cd c7 00                       call  hex_to_byte             ;get low-order byte
0116 da 20 01                       jp    c, address_entry_error  ;jump if error
0119 32 00 db                       ld    (current_location),a    ;store low-order byte in lower memory
011c 2a 00 db                       ld    hl,(current_location)   ;put memory address in hl
011f c9                             ret
0120 21 c2 03     address_entry_error:   ld    hl,address_error_msg
0123 cd 18 00                       call  write_string
0126 c3 fe 00                       jp    address_entry
0129              ;
0129              ;Subroutine to get a decimal string, return a word value
0129              ;Calls decimal_string_to_word subroutine
```

```
0129 21 08 db     decimal_entry:          ld    hl,buffer
012c cd 4c 00                             call  get_line         ;returns with DE pointing to terminating zero
012f 21 08 db                             ld    hl,buffer
0132 cd 3f 01                             call  decimal_string_to_word
0135 d0                                   ret   nc                     ;no error, return with word in hl
0136 21 36 04                             ld    hl,decimal_error_msg   ;error, try again
0139 cd 18 00                             call  write_string
013c c3 29 01                             jp    decimal_entry
013f              ;
013f              ;Subroutine to convert a decimal string to a word value
013f              ;Call with address of string in HL, pointer to end of string in DE
013f              ;Carry flag set if error (non-decimal char)
013f              ;Carry flag clear, word value in HL if no error.
013f 42           decimal_string_to_word: ld    b,d
0140 4b                                   ld    c,e                    ;use BC as string pointer
0141 22 00 db                             ld    (current_location),hl  ;save addr. of buffer start in RAM
0144 21 00 00                             ld    hl,000h                ;starting value zero
0147 22 06 db                             ld    (current_value),hl
014a 21 8f 01                             ld    hl,decimal_place_value ;pointer to values
014d 22 04 db                             ld    (value_pointer),hl
0150 0b           decimal_next_char:      dec   bc                     ;next char (moving right to left)
0151 2a 00 db                             ld    hl,(current_location)  ;check if at end of decimal string
0154 37                                   scf                          ;get ready to sub. DE from buffer addr.
0155 3f                                   ccf                          ;set carry to zero (clear)
0156 ed 42                                sbc   hl,bc                  ;cont. if bc > or = hl (buffer address)
0158 da 64 01                             jp    c,decimal_continue     ;borrow means bc > hl
015b ca 64 01                             jp    z,decimal_continue     ;z means bc = hl
015e 2a 06 db                             ld    hl,(current_value)     ;return if de < buffer add. (no borrow)
0161 37                                   scf                          ;get value back from RAM variable
0162 3f                                   ccf
0163 c9                                   ret                          ;return with carry clear, value in hl
0164 0a           decimal_continue:       ld    a,(bc)                 ;next char in string (right to left)
0165 d6 30                                sub   030h                   ;ASCII value of zero char
0167 fa 8a 01                             jp    m,decimal_error        ;error if char value less than 030h
016a fe 0a                                cp    00ah                   ;error if byte value > or = 10 decimal
016c f2 8a 01                             jp    p,decimal_error        ;a reg now has value of decimal numeral
016f 2a 04 db                             ld    hl,(value_pointer)     ;get value to add an put in de
0172 5e                                   ld    e,(hl)                 ;little-endian (low byte in low memory)
0173 23                                   inc   hl
0174 56                                   ld    d,(hl)
```

```
0175 23                                 inc   hl                      ;hl now points to next value
0176 22 04 db                           ld    (value_pointer),hl
0179 2a 06 db                           ld    hl,(current_value)      ;get back current value
017c 3d           decimal_add:          dec   a                       ;add loop to increase total value
017d fa 84 01                           jp    m,decimal_add_done      ;end of multiplication
0180 19                                 add   hl,de
0181 c3 7c 01                           jp    decimal_add
0184 22 06 db     decimal_add_done:     ld    (current_value),hl
0187 c3 50 01                           jp    decimal_next_char
018a 37           decimal_error:        scf
018b c9                                 ret
018c c3 7c 01                           jp    decimal_add
018f 01 00 0a 00 64 00 e8 03 10 27 decimal_place_value:    defw  1,10,100,1000,10000
0199              ;
0199              ;Memory dump
0199              ;Displays a 256-byte block of memory in 16-byte rows.
0199              ;Called with address of start of block in HL
0199 22 00 db     memory_dump:          ld    (current_location),hl   ;store address of block to be displayed
019c 3e 00                              ld    a,000h
019e 32 03 db                           ld    (byte_count),a          ;initialize byte count
01a1 32 02 db                           ld    (line_count),a          ;initialize line count
01a4 c3 d9 01                           jp    dump_new_line
01a7 2a 00 db     dump_next_byte:       ld    hl,(current_location)   ;get byte address from storage,
01aa 7e                                 ld    a,(hl)                  ;get byte to be converted to string
01ab 23                                 inc   hl                      ;increment address and
01ac 22 00 db                           ld    (current_location),hl   ;store back
01af 21 08 db                           ld    hl,buffer               ;location to store string
01b2 cd 88 00                           call  byte_to_hex_string      ;convert
01b5 21 08 db                           ld    hl,buffer               ;display string
01b8 cd 18 00                           call  write_string
01bb 3a 03 db                           ld    a,(byte_count)          ;next byte
01be 3c                                 inc   a
01bf ca 09 02                           jp    z,dump_done             ;stop when 256 bytes displayed
01c2 32 03 db                           ld    (byte_count),a          ;not finished yet, store
01c5 3a 02 db                           ld    a,(line_count)          ;end of line (16 characters)?
01c8 fe 0f                              cp    00fh                    ;yes, start new line
01ca ca d9 01                           jp    z,dump_new_line
01cd 3c                                 inc   a                       ;no, increment line count
01ce 32 02 db                           ld    (line_count),a
01d1 3e 20                              ld    a,020h                  ;print space
```

```
01d3 cd 0c 00                          call  write_char
01d6 c3 a7 01                          jp    dump_next_byte         ;continue
01d9 3e 00        dump_new_line:       ld    a,000h                 ;reset line count to zero
01db 32 02 db                          ld    (line_count),a
01de cd 89 02                          call  write_newline
01e1 2a 00 db                          ld    hl,(current_location)  ;location of start of line
01e4 7c                                ld    a,h                    ;high byte of address
01e5 21 08 db                          ld    hl, buffer
01e8 cd 88 00                          call  byte_to_hex_string     ;convert
01eb 21 08 db                          ld    hl,buffer
01ee cd 18 00                          call  write_string           ;write high byte
01f1 2a 00 db                          ld    hl,(current_location)
01f4 7d                                ld    a,l                    ;low byte of address
01f5 21 08 db                          ld    hl, buffer
01f8 cd 88 00                          call  byte_to_hex_string     ;convert
01fb 21 08 db                          ld    hl,buffer
01fe cd 18 00                          call  write_string           ;write low byte
0201 3e 20                             ld    a,020h                 ;space
0203 cd 0c 00                          call  write_char
0206 c3 a7 01                          jp    dump_next_byte         ;now write 16 bytes
0209 3e 00        dump_done:           ld    a,000h
020b 21 08 db                          ld    hl,buffer
020e 77                                ld    (hl),a                 ;clear buffer of last string
020f cd 89 02                          call  write_newline
0212 c9                                ret
0213              ;
0213              ;Memory load
0213              ;Loads RAM memory with bytes entered as hex characters
0213              ;Called with address to start loading in HL
0213              ;Displays entered data in 16-byte rows.
0213 22 00 db     memory_load:         ld    (current_location),hl
0216 21 ee 03                          ld    hl,data_entry_msg
0219 cd 18 00                          call  write_string
021c c3 66 02                          jp    load_new_line
021f cd 7f 02     load_next_char:      call  get_char
0222 fe 0d                             cp    00dh                   ;return?
0224 ca 7b 02                          jp    z,load_done            ;yes, quit
0227 32 08 db                          ld    (buffer),a
022a cd 7f 02                          call  get_char
022d fe 0d                             cp    00dh                   ;return?
```

```
022f ca 7b 02                       jp    z,load_done             ;yes, quit
0232 32 09 db                       ld    (buffer+1),a
0235 21 08 db                       ld    hl,buffer
0238 cd c7 00                       call  hex_to_byte
023b da 71 02                       jp    c,load_data_entry_error ;non-hex character
023e 2a 00 db                       ld    hl,(current_location)   ;get byte address from storage,
0241 77                             ld    (hl),a                  ;store byte
0242 23                             inc   hl                      ;increment address and
0243 22 00 db                       ld    (current_location),hl   ;store back
0246 3a 08 db                       ld    a,(buffer)
0249 cd 0c 00                       call  write_char
024c 3a 09 db                       ld    a,(buffer+1)
024f cd 0c 00                       call  write_char
0252 3a 02 db                       ld    a,(line_count)          ;end of line (16 characters)?
0255 fe 0f                          cp    00fh                    ;yes, start new line
0257 ca 66 02                       jp    z,load_new_line
025a 3c                             inc   a                       ;no, increment line count
025b 32 02 db                       ld    (line_count),a
025e 3e 20                          ld    a,020h                  ;print space
0260 cd 0c 00                       call  write_char
0263 c3 1f 02                       jp    load_next_char          ;continue
0266 3e 00        load_new_line:    ld    a,000h                  ;reset line count to zero
0268 32 02 db                       ld    (line_count),a
026b cd 89 02                       call  write_newline
026e c3 1f 02                       jp    load_next_char          ;continue
0271 cd 89 02     load_data_entry_error:  call  write_newline
0274 21 1b 04                       ld    hl,data_error_msg
0277 cd 18 00                       call  write_string
027a c9                             ret
027b cd 89 02     load_done:        call  write_newline
027e c9                             ret
027f              ;
027f              ;Get one ASCII character from the serial port.
027f              ;Returns with char in A reg. No error checking.
027f db 03        get_char:         in    a,(3)                   ;get status
0281 e6 02                          and   002h                    ;check RxRDY bit
0283 ca 7f 02                       jp    z,get_char              ;not ready, loop
0286 db 02                          in    a,(2)                   ;get char
0288 c9                             ret
0289              ;
```

```
0289                ;Subroutine to start a new line
0289 3e 0d          write_newline:          ld    a,00dh                  ;ASCII carriage return character
028b cd 0c 00                               call  write_char
028e 3e 0a                                  ld    a,00ah                  ;new line (line feed) character
0290 cd 0c 00                               call  write_char
0293 c9                                     ret
0294                ;
0294                ;Subroutine to read one disk sector (256 bytes)
0294                ;Address to place data passed in HL
0294                ;LBA bits 0 to 7 passed in C, bits 8 to 15 passed in B
0294                ;LBA bits 16 to 23 passed in E
0294                disk_read:
0294 db 0f          rd_status_loop_1:       in    a,(0fh)                 ;check status
0296 e6 80                                  and   80h                     ;check BSY bit
0298 c2 94 02                               jp    nz,rd_status_loop_1     ;loop until not busy
029b db 0f          rd_status_loop_2:       in    a,(0fh)                 ;check      status
029d e6 40                                  and   40h                     ;check DRDY bit
029f ca 9b 02                               jp    z,rd_status_loop_2      ;loop until ready
02a2 3e 01                                  ld    a,01h                   ;number of sectors = 1
02a4 d3 0a                                  out   (0ah),a                 ;sector count register
02a6 79                                     ld    a,c
02a7 d3 0b                                  out   (0bh),a                 ;lba bits 0 - 7
02a9 78                                     ld    a,b
02aa d3 0c                                  out   (0ch),a                 ;lba bits 8 - 15
02ac 7b                                     ld    a,e
02ad d3 0d                                  out   (0dh),a                 ;lba bits 16 - 23
02af 3e e0                                  ld    a,11100000b             ;LBA mode, select drive 0
02b1 d3 0e                                  out   (0eh),a                 ;drive/head register
02b3 3e 20                                  ld    a,20h                   ;Read sector command
02b5 d3 0f                                  out   (0fh),a
02b7 db 0f          rd_wait_for_DRQ_set:    in    a,(0fh)                 ;read status
02b9 e6 08                                  and   08h                     ;DRQ bit
02bb ca b7 02                               jp    z,rd_wait_for_DRQ_set   ;loop until bit set
02be db 0f          rd_wait_for_BSY_clear:  in    a,(0fh)
02c0 e6 80                                  and   80h
02c2 c2 be 02                               jp    nz,rd_wait_for_BSY_clear
02c5 db 0f                                  in    a,(0fh)                 ;clear INTRQ
02c7 db 08          read_loop:              in    a,(08h)                 ;get data
02c9 77                                     ld    (hl),a
02ca 23                                     inc   hl
```

```
02cb db 0f                         in    a,(0fh)                ;check status
02cd e6 08                         and   08h                    ;DRQ bit
02cf c2 c7 02                      jp    nz,read_loop           ;loop until cleared
02d2 c9                            ret
02d3             ;
02d3             ;Subroutine to write one disk sector (256 bytes)
02d3             ;Address of data to write to disk passed in HL
02d3             ;LBA bits 0 to 7 passed in C, bits 8 to 15 passed in B
02d3             ;LBA bits 16 to 23 passed in E
02d3             disk_write:
02d3 db 0f       wr_status_loop_1:       in    a,(0fh)                ;check status
02d5 e6 80                         and   80h                    ;check BSY bit
02d7 c2 d3 02                      jp    nz,wr_status_loop_1    ;loop until not busy
02da db 0f       wr_status_loop_2:       in    a,(0fh)                ;check      status
02dc e6 40                         and   40h                    ;check DRDY bit
02de ca da 02                      jp    z,wr_status_loop_2     ;loop until ready
02e1 3e 01                         ld    a,01h                  ;number of sectors = 1
02e3 d3 0a                         out   (0ah),a                ;sector count register
02e5 79                            ld    a,c
02e6 d3 0b                         out   (0bh),a                ;lba bits 0 - 7
02e8 78                            ld    a,b
02e9 d3 0c                         out   (0ch),a                ;lba bits 8 - 15
02eb 7b                            ld    a,e
02ec d3 0d                         out   (0dh),a                ;lba bits 16 - 23
02ee 3e e0                         ld    a,11100000b            ;LBA mode, select drive 0
02f0 d3 0e                         out   (0eh),a                ;drive/head register
02f2 3e 30                         ld    a,30h                  ;Write sector command
02f4 d3 0f                         out   (0fh),a
02f6 db 0f       wr_wait_for_DRQ_set:    in    a,(0fh)                ;read status
02f8 e6 08                         and   08h                    ;DRQ bit
02fa ca f6 02                      jp    z,wr_wait_for_DRQ_set  ;loop until bit set
02fd 7e          write_loop:             ld    a,(hl)
02fe d3 08                         out   (08h),a                ;write data
0300 23                            inc   hl
0301 db 0f                         in    a,(0fh)                ;read status
0303 e6 08                         and   08h                    ;check DRQ bit
0305 c2 fd 02                      jp    nz,write_loop          ;write until bit cleared
0308 db 0f       wr_wait_for_BSY_clear:  in    a,(0fh)
030a e6 80                         and   80h
030c c2 08 03                      jp    nz,wr_wait_for_BSY_clear
```

```
030f db 0f                         in    a,(0fh)                 ;clear INTRQ
0311 c9                            ret
0312             ;
0312             ;Strings used in subroutines
0312 .. 00       length_entry_string:    defm  "Enter length of file to load (decimal): ",0
033b .. 00       dump_entry_string:      defm  "Enter no. of bytes to dump (decimal): ",0
0362 .. 00       LBA_entry_string:       defm  "Enter LBA (decimal, 0 to 65535): ",0
0384 08 1b .. 00  erase_char_string:      defm  008h,01bh,"[K",000h     ;ANSI seq. for BS, erase to end of line.
0389 .. 00       address_entry_msg:      defm  "Enter 4-digit hex address (use upper-case A through F): ",0
03c2 .. 00       address_error_msg:      defm  "\r\nError: invalid hex character, try again: ",0
03ee .. 00       data_entry_msg:         defm  "Enter hex bytes, hit return when finished.\r\n",0
041b .. 00       data_error_msg:         defm  "Error: invalid hex byte.\r\n",0
0436 .. 00       decimal_error_msg:      defm  "\r\nError: invalid decimal number, try again: ",0
0463             ;
0463             ;Simple monitor program for CPUville Z80 computer with serial interface.
0463 31 ff db    monitor_cold_start:     ld    sp,ROM_monitor_stack
0466 cd 03 00                            call  initialize_port
0469 21 dc 05                            ld    hl,monitor_message
046c cd 18 00                            call  write_string
046f cd 89 02    monitor_warm_start:     call  write_newline     ;re-enter here to avoid port re-init.
0472 3e 3e                               ld    a,03eh            ;cursor symbol
0474 cd 0c 00                            call  write_char
0477 21 08 db                            ld    hl,buffer
047a cd 4c 00                            call  get_line          ;get monitor input string (command)
047d cd 89 02                            call  write_newline
0480 cd 84 04                            call  parse             ;parse command, returns with jump add. in HL
0483 e9                                  jp    (hl)
0484             ;
0484             ;Parses (interprets) an input line in buffer for commands as described in parse table.
0484             ;Returns with address of jump to action for the command in HL
0484 01 ba 07    parse:                  ld    bc,parse_table    ;bc is pointer to parse_table
0487 0a          parse_start:            ld    a,(bc)            ;get pointer to match string from parse table
0488 5f                                  ld    e,a
0489 03                                  inc   bc
048a 0a                                  ld    a,(bc)
048b 57                                  ld    d,a               ;de will is pointer to strings for matching
048c 1a                                  ld    a,(de)            ;get first char from match string
048d f6 00                               or    000h              ;zero?
048f ca aa 04                            jp    z,parser_exit     ;yes, exit no_match
0492 21 08 db                            ld    hl,buffer         ;no, parse input string
```

```
0495 be         match_loop:             cp    (hl)              ;compare buffer char with match string char
0496 c2 a4 04                           jp    nz,no_match       ;no match, go to next match string
0499 f6 00                              or    000h              ;end of strings (zero)?
049b ca aa 04                           jp    z,parser_exit     ;yes, matching string found
049e 13                                 inc   de                ;match so far, point to next char in match
string
049f 1a                                 ld    a,(de)            ;get next character from match string
04a0 23                                 inc   hl                ;and point to next char in input string
04a1 c3 95 04                           jp    match_loop        ;check for match
04a4 03         no_match:               inc   bc                ;skip over jump target to
04a5 03                                 inc   bc
04a6 03                                 inc   bc                ;get address of next matching string
04a7 c3 87 04                           jp    parse_start
04aa 03         parser_exit:            inc   bc                ;skip to address of jump for match
04ab 0a                                 ld    a,(bc)
04ac 6f                                 ld    l,a
04ad 03                                 inc   bc
04ae 0a                                 ld    a,(bc)
04af 67                                 ld    h,a               ;returns with jump address in hl
04b0 c9                                 ret
04b1            ;
04b1            ;Actions to be taken on match
04b1            ;
04b1            ;Memory dump program
04b1            ;Input 4-digit hexadecimal address
04b1            ;Calls memory_dump subroutine
04b1 21 06 06   dump_jump:              ld    hl,dump_message         ;Display greeting
04b4 cd 18 00                           call  write_string
04b7 21 89 03                           ld    hl,address_entry_msg    ;get ready to get address
04ba cd 18 00                           call  write_string
04bd cd fe 00                           call  address_entry           ;returns with address in HL
04c0 cd 89 02                           call  write_newline
04c3 cd 99 01                           call  memory_dump
04c6 c3 6f 04                           jp    monitor_warm_start
04c9            ;
04c9            ;Hex loader, displays formatted input
04c9 21 2d 06   load_jump:              ld    hl,load_message         ;Display greeting
04cc cd 18 00                           call  write_string            ;get address to load
04cf 21 89 03                           ld    hl,address_entry_msg    ;get ready to get address
04d2 cd 18 00                           call  write_string
```

```
04d5 cd fe 00                      call  address_entry
04d8 cd 89 02                      call  write_newline
04db cd 13 02                      call  memory_load
04de c3 6f 04                      jp    monitor_warm_start
04e1                ;
04e1                ;Jump and run do the same thing: get an address and jump to it.
04e1 21 5c 06       run_jump:      ld    hl,run_message         ;Display greeting
04e4 cd 18 00                      call  write_string
04e7 21 89 03                      ld    hl,address_entry_msg   ;get ready to get address
04ea cd 18 00                      call  write_string
04ed cd fe 00                      call  address_entry
04f0 e9                            jp    (hl)
04f1                ;
04f1                ;Help and ? do the same thing, display the available commands
04f1 21 ee 05       help_jump:     ld    hl,help_message
04f4 cd 18 00                      call  write_string
04f7 01 ba 07                      ld    bc,parse_table         ;table with pointers to command strings
04fa 0a             help_loop:     ld    a,(bc)                 ;displays command strings
04fb 6f                            ld    l,a                    ;getting the string addresses from the
04fc 03                            inc   bc                     ;parse table
04fd 0a                            ld    a,(bc)                 ;pass add. of string to HL through A reg
04fe 67                            ld    h,a
04ff 7e                            ld    a,(hl)                 ;hl now points to start of match string
0500 f6 00                         or    000h                   ;exit if no_match string
0502 ca 15 05                      jp    z,help_done
0505 c5                            push  bc                     ;write_char uses B reg, so save first
0506 3e 20                         ld    a,020h                 ;space char
0508 cd 0c 00                      call  write_char
050b c1                            pop   bc
050c cd 18 00                      call  write_string           ;writes match string
050f 03                            inc   bc                     ;pass over jump address in table
0510 03                            inc   bc
0511 03                            inc   bc
0512 c3 fa 04                      jp    help_loop
0515 c3 6f 04       help_done:     jp    monitor_warm_start
0518                ;
0518                ;Binary file load. Need both address to load and length of file
0518 21 91 06       bload_jump:    ld    hl,bload_message
051b cd 18 00                      call  write_string
051e 21 89 03                      ld    hl,address_entry_msg
```

```
0521 cd 18 00                         call  write_string
0524 cd fe 00                         call  address_entry
0527 cd 89 02                         call  write_newline
052a e5                               push  hl
052b 21 12 03                         ld    hl,length_entry_string
052e cd 18 00                         call  write_string
0531 cd 29 01                         call  decimal_entry
0534 44                               ld    b,h
0535 4d                               ld    c,l
0536 21 b4 06                         ld    hl,bload_ready_message
0539 cd 18 00                         call  write_string
053c e1                               pop   hl
053d cd 28 00                         call  bload
0540 c3 6f 04                         jp    monitor_warm_start
0543                  ;
0543                  ;Binary memory dump. Need address of start of dump and no. bytes
0543 21 d8 06         bdump_jump:     ld    hl,bdump_message
0546 cd 18 00                         call  write_string
0549 21 89 03                         ld    hl,address_entry_msg
054c cd 18 00                         call  write_string
054f cd fe 00                         call  address_entry
0552 cd 89 02                         call  write_newline
0555 e5                               push  hl
0556 21 3b 03                         ld    hl,dump_entry_string
0559 cd 18 00                         call  write_string
055c cd 29 01                         call  decimal_entry
055f 44                               ld    b,h
0560 4d                               ld    c,l
0561 21 08 07                         ld    hl,bdump_ready_message
0564 cd 18 00                         call  write_string
0567 cd 7f 02                         call  get_char
056a e1                               pop   hl
056b cd 3a 00                         call  bdump
056e c3 6f 04                         jp    monitor_warm_start
0571                  ;Disk read. Need memory address to place data, LBA of sector to read
0571 21 2f 07         diskrd_jump:    ld    hl,diskrd_message
0574 cd 18 00                         call  write_string
0577 21 89 03                         ld    hl,address_entry_msg
057a cd 18 00                         call  write_string
057d cd fe 00                         call  address_entry
```

```
0580 cd 89 02                         call  write_newline
0583 e5                               push  hl
0584 21 62 03                         ld    hl,LBA_entry_string
0587 cd 18 00                         call  write_string
058a cd 29 01                         call  decimal_entry
058d 44                               ld    b,h
058e 4d                               ld    c,l
058f 1e 00                            ld    e,00h
0591 e1                               pop   hl
0592 cd 94 02                         call  disk_read
0595 c3 6f 04                         jp    monitor_warm_start
0598 21 57 07     diskwr_jump:        ld    hl,diskwr_message
059b cd 18 00                         call  write_string
059e 21 89 03                         ld    hl,address_entry_msg
05a1 cd 18 00                         call  write_string
05a4 cd fe 00                         call  address_entry
05a7 cd 89 02                         call  write_newline
05aa e5                               push  hl
05ab 21 62 03                         ld    hl,LBA_entry_string
05ae cd 18 00                         call  write_string
05b1 cd 29 01                         call  decimal_entry
05b4 44                               ld    b,h
05b5 4d                               ld    c,l
05b6 1e 00                            ld    e,00h
05b8 e1                               pop   hl
05b9 cd d3 02                         call  disk_write
05bc c3 6f 04                         jp    monitor_warm_start
05bf 21 00 08     cpm_jump:           ld    hl,0800h
05c2 01 00 00                         ld    bc,0000h
05c5 1e 00                            ld    e,00h
05c7 cd 94 02                         call  disk_read
05ca c3 00 08                         jp    0800h
05cd              ;Prints message for no match to entered command
05cd 21 eb 05     no_match_jump:      ld    hl,no_match_message
05d0 cd 18 00                         call  write_string
05d3 21 08 db                         ld    hl, buffer
05d6 cd 18 00                         call  write_string
05d9 c3 6f 04                         jp    monitor_warm_start
05dc              ;
05dc              ;Monitor data structures:
```

```
05dc                ;
05dc .. 00          monitor_message:        defm  "\r\nROM ver. 8\r\n",0
05eb .. 00          no_match_message:       defm  "? ",0
05ee .. 00          help_message:           defm  "Commands implemented:\r\n",0
0606 .. 00          dump_message:           defm  "Displays a 256-byte block of memory.\r\n",0
062d .. 00          load_message:           defm  "Enter hex bytes starting at memory location.\r\n",0
065c .. 00          run_message:            defm  "Will jump to (execute) program at address entered.\r\n",0
0691 .. 00          bload_message:          defm  "Loads a binary file into memory.\r\n",0
06b4 .. 00          bload_ready_message:    defm  "\n\rReady to receive, start transfer.",0
06d8 .. 00          bdump_message:          defm  "Dumps binary data from memory to serial port.\r\n",0
0708 .. 00          bdump_ready_message:    defm  "\n\rReady to send, hit any key to start.",0
072f .. 00          diskrd_message:         defm  "Reads one sector from disk to memory.\r\n",0
0757 .. 00          diskwr_message:         defm  "Writes one sector from memory to disk.\r\n",0
0780                ;Strings for matching:
0780 .. 00          dump_string:            defm  "dump",0
0785 .. 00          load_string:            defm  "load",0
078a .. 00          jump_string:            defm  "jump",0
078f .. 00          run_string:             defm  "run",0
0793 .. 00          question_string:        defm  "?",0
0795 .. 00          help_string:            defm  "help",0
079a .. 00          bload_string:           defm  "bload",0
07a0 .. 00          bdump_string:           defm  "bdump",0
07a6 .. 00          diskrd_string:          defm  "diskrd",0
07ad .. 00          diskwr_string:          defm  "diskwr",0
07b4 .. 00          cpm_string:             defm  "cpm",0
07b8 00 00          no_match_string:        defm  0,0
07ba                ;Table for matching strings to jumps
07ba 80 07 b1 04 85 07 c9 04  parse_table:     defw  dump_string,dump_jump,load_string,load_jump
07c2 8a 07 e1 04 8f 07 e1 04                   defw  jump_string,run_jump,run_string,run_jump
07ca 93 07 f1 04 95 07 f1 04                   defw  question_string,help_jump,help_string,help_jump
07d2 9a 07 18 05 a0 07 43 05                   defw  bload_string,bload_jump,bdump_string,bdump_jump
07da a6 07 71 05 ad 07 98 05                   defw  diskrd_string,diskrd_jump,diskwr_string,diskwr_jump
07e2 b4 07 bf 05                               defw  cpm_string,cpm_jump
07e6 b8 07 cd 05                               defw  no_match_string,no_match_jump
07ea
# End of file 2K_ROM_8.asm
07ea
```

Customized BIOS

```
# File z80_cbios.asm
0000            ;     skeletal cbios for first level of CP/M 2.0 alteration
0000            ;     Modified for CPUville Z80 computer with IDE disk interface
0000            ;     Aug, 2014 by Donn Stewart
0000            ;
0000            ccp:        equ   0E400h          ;base of ccp
0000            bdos:       equ   0EC06h          ;bdos entry
0000            bios:       equ   0FA00h          ;base of bios
0000            cdisk:      equ   0004h     ;address of current disk number 0=a,... l5=p
0000            iobyte:     equ   0003h     ;intel i/o byte
0000            disks:      equ   04h       ;number of disks in the system
0000            ;
0000                        org   bios      ;origin of this program
fa00            nsects:     equ   ($-ccp)/128 ;warm start sector count
fa00            ;
fa00            ;     jump vector for individual subroutines
fa00            ;
fa00 c3 9c fa               JP    boot  ;cold start
fa03 c3 a6 fa   wboote:     JP    wboot ;warm start
fa06 c3 18 fb               JP    const ;console status
fa09 c3 25 fb               JP    conin ;console character in
fa0c c3 31 fb               JP    conout      ;console character out
fa0f c3 3c fb               JP    list  ;list character out
fa12 c3 40 fb               JP    punch ;punch character out
fa15 c3 42 fb               JP    reader      ;reader character out
fa18 c3 47 fb               JP    home  ;move head to home position
fa1b c3 4d fb               JP    seldsk      ;select disk
fa1e c3 66 fb               JP    settrk      ;set track number
fa21 c3 6b fb               JP    setsec      ;set sector number
fa24 c3 77 fb               JP    setdma      ;set dma address
fa27 c3 7d fb               JP    read  ;read disk
fa2a c3 d7 fb               JP    write ;write disk
fa2d c3 3e fb               JP    listst      ;return list status
fa30 c3 70 fb               JP    sectran     ;sector translate
fa33            ;
fa33            ;     fixed data tables for four-drive standard
```

```
fa33                ;     ibm-compatible 8" disks
fa33                ;     no translations
fa33                ;
fa33                ;     disk Parameter header for disk 00
fa33 00 00 00 00  dpbase:     defw  0000h, 0000h
fa37 00 00 00 00              defw  0000h, 0000h
fa3b 36 fc 8d fa              defw  dirbf, dpblk
fa3f 32 fd b6 fc              defw  chk00, all00
fa43                ;     disk parameter header for disk 01
fa43 00 00 00 00              defw  0000h, 0000h
fa47 00 00 00 00              defw  0000h, 0000h
fa4b 36 fc 8d fa              defw  dirbf, dpblk
fa4f 42 fd d5 fc              defw  chk01, all01
fa53                ;     disk parameter header for disk 02
fa53 00 00 00 00              defw  0000h, 0000h
fa57 00 00 00 00              defw  0000h, 0000h
fa5b 36 fc 8d fa              defw  dirbf, dpblk
fa5f 52 fd f4 fc              defw  chk02, all02
fa63                ;     disk parameter header for disk 03
fa63 00 00 00 00              defw  0000h, 0000h
fa67 00 00 00 00              defw  0000h, 0000h
fa6b 36 fc 8d fa              defw  dirbf, dpblk
fa6f 62 fd 13 fd              defw  chk03, all03
fa73                ;
fa73                ;     sector translate vector
fa73 01 07 0d 13  trans:      defm   1,  7, 13, 19    ;sectors  1,  2,  3,  4
fa77 19 05 0b 11              defm  25,  5, 11, 17    ;sectors  5,  6,  7,  6
fa7b 17 03 09 0f              defm  23,  3,  9, 15    ;sectors  9, 10, 11, 12
fa7f 15 02 08 0e              defm  21,  2,  8, 14    ;sectors 13, 14, 15, 16
fa83 14 1a 06 0c              defm  20, 26,  6, 12    ;sectors 17, 18, 19, 20
fa87 12 18 04 0a              defm  18, 24,  4, 10    ;sectors 21, 22, 23, 24
fa8b 10 16                    defm  16, 22            ;sectors 25, 26
fa8d                ;
fa8d              dpblk:      ;disk parameter block for all disks.
fa8d 1a 00                    defw  26                ;sectors per track
fa8f 03                       defm  3                 ;block shift factor
fa90 07                       defm  7                 ;block mask
fa91 00                       defm  0                 ;null mask
fa92 f2 00                    defw  242               ;disk size-1
fa94 3f 00                    defw  63                ;directory max
```

```
fa96 c0                     defm  192              ;alloc 0
fa97 00                     defm  0                ;alloc 1
fa98 00 00                  defw  0                ;check size
fa9a 02 00                  defw  2                ;track offset
fa9c              ;
fa9c              ;     end of fixed tables
fa9c              ;
fa9c              ;     individual subroutines to perform each function
fa9c              boot: ;simplest case is to just perform parameter initialization
fa9c af                     XOR   a                ;zero in the accum
fa9d 32 03 00               LD    (iobyte),A       ;clear the iobyte
faa0 32 04 00               LD    (cdisk),A        ;select disk zero
faa3 c3 ef fa               JP    gocpm            ;initialize and go to cp/m
faa6              ;
faa6              wboot:      ;simplest case is to read the disk until all sectors loaded
faa6 31 80 00               LD    sp, 80h          ;use space below buffer for stack
faa9 0e 00                  LD    c, 0             ;select disk 0
faab cd 4d fb               call  seldsk
faae cd 47 fb               call  home             ;go to track 00
fab1              ;
fab1 06 2c                  LD    b, nsects        ;b counts * of sectors to load
fab3 0e 00                  LD    c, 0             ;c has the current track number
fab5 16 02                  LD    d, 2             ;d has the next sector to read
fab7              ;     note that we begin by reading track 0, sector 2 since sector 1
fab7              ;     contains the cold start loader, which is skipped in a warm start
fab7 21 00 e4               LD    HL, ccp          ;base of cp/m (initial load point)
faba              load1:      ;load one more sector
faba c5                     PUSH  BC               ;save sector count, current track
fabb d5                     PUSH  DE               ;save next sector to read
fabc e5                     PUSH  HL               ;save dma address
fabd 4a                     LD    c, d             ;get sector address to register C
fabe cd 6b fb               call  setsec           ;set sector address from register C
fac1 c1                     pop   BC               ;recall dma address to b, C
fac2 c5                     PUSH  BC               ;replace on stack for later recall
fac3 cd 77 fb               call  setdma           ;set dma address from b, C
fac6              ;
fac6              ;     drive set to 0, track set, sector set, dma address set
fac6 cd 7d fb               call  read
fac9 fe 00                  CP    00h              ;any errors?
facb c2 a6 fa               JP    NZ,wboot         ;retry the entire boot if an error occurs
```

```
face             ;
face             ;     no error, move to next sector
face e1                      pop   HL                 ;recall dma address
facf 11 80 00                LD    DE, 128            ;dma=dma+128
fad2 19                      ADD   HL,DE              ;new dma address is in h, l
fad3 d1                      pop   DE                 ;recall sector address
fad4 c1                      pop   BC                 ;recall number of sectors remaining, and current trk
fad5 05                      DEC   b                  ;sectors=sectors-1
fad6 ca ef fa                JP    Z,gocpm            ;transfer to cp/m if all have been loaded
fad9             ;
fad9             ;     more  sectors remain to load, check for track change
fad9 14                      INC   d
fada 7a                      LD    a,d                ;sector=27?, if so, change tracks
fadb fe 1b                   CP    27
fadd da ba fa                JP    C,load1            ;carry generated if sector<27
fae0             ;
fae0             ;     end of      current track,     go to next track
fae0 16 01                   LD    d, 1               ;begin with first sector of next track
fae2 0c                      INC   c                  ;track=track+1
fae3             ;
fae3             ;     save  register state, and change tracks
fae3 c5                      PUSH  BC
fae4 d5                      PUSH  DE
fae5 e5                      PUSH  HL
fae6 cd 66 fb                call  settrk             ;track address set from register c
fae9 e1                      pop   HL
faea d1                      pop   DE
faeb c1                      pop   BC
faec c3 ba fa                JP    load1              ;for another sector
faef             ;
faef             ;     end of      load operation, set parameters and go to cp/m
faef             gocpm:
faef 3e c3                   LD    a, 0c3h            ;c3 is a jmp instruction
faf1 32 00 00                LD    (0),A              ;for jmp to wboot
faf4 21 03 fa                LD    HL, wboote         ;wboot entry point
faf7 22 01 00                LD    (1),HL             ;set address field for jmp at 0
fafa             ;
fafa 32 05 00                LD    (5),A              ;for jmp to bdos
fafd 21 06 ec                LD    HL, bdos           ;bdos entry point
fb00 22 06 00                LD    (6),HL             ;address field of Jump at 5 to bdos
```

```
fb03              ;
fb03 01 80 00               LD    BC, 80h           ;default dma address is 80h
fb06 cd 77 fb               call  setdma
fb09              ;
fb09 fb                     ei                      ;enable the interrupt system
fb0a 3a 04 00               LD    A,(cdisk)         ;get current disk number
fb0d fe 04                  cp    disks             ;see if valid disk number
fb0f da 14 fb               jp    c,diskok          ;disk valid, go to ccp
fb12 3e 00                  ld    a,0               ;invalid disk, change to disk 0
fb14 4f           diskok:   LD    c, a              ;send to the ccp
fb15 c3 00 e4               JP    ccp               ;go to cp/m for further processing
fb18              ;
fb18              ;
fb18              ;     simple i/o handlers (must be filled in by user)
fb18              ;     in each case, the entry point is provided, with space reserved
fb18              ;     to insert your own code
fb18              ;
fb18              const:      ;console status, return 0ffh if character ready, 00h if not
fb18 db 03                  in    a,(3)             ;get status
fb1a e6 02                  and   002h              ;check RxRDY bit
fb1c ca 22 fb               jp    z,no_char
fb1f 3e ff                  ld    a,0ffh            ;char ready
fb21 c9                     ret
fb22 3e 00        no_char:  ld    a,00h             ;no char
fb24 c9                     ret
fb25              ;
fb25              conin:      ;console character into register a
fb25 db 03                  in    a,(3)             ;get status
fb27 e6 02                  and   002h              ;check RxRDY bit
fb29 ca 25 fb               jp    z,conin           ;loop until char ready
fb2c db 02                  in    a,(2)             ;get char
fb2e e6 7f                  AND   7fh               ;strip parity bit
fb30 c9                     ret
fb31              ;
fb31              conout:     ;console character output from register c
fb31 db 03                  in    a,(3)
fb33 e6 01                  and   001h              ;check TxRDY bit
fb35 ca 31 fb               jp    z,conout          ;loop until port ready
fb38 79                     ld    a,c               ;get the char
fb39 d3 02                  out   (2),a             ;out to port
```

```
fb3b c9                 ret
fb3c            ;
fb3c            list: ;list character from register c
fb3c 79                 LD    a, c             ;character to register a
fb3d c9                 ret                    ;null subroutine
fb3e            ;
fb3e            listst:     ;return list status (0 if not ready, 1 if ready)
fb3e af                 XOR   a                ;0 is always ok to return
fb3f c9                 ret
fb40            ;
fb40            punch:      ;punch      character from    register C
fb40 79                 LD    a, c             ;character to register a
fb41 c9                 ret                    ;null subroutine
fb42            ;
fb42            ;
fb42            reader:     ;reader character into register a from reader device
fb42 3e 1a              LD     a, 1ah          ;enter end of file for now (replace later)
fb44 e6 7f              AND    7fh             ;remember to strip parity bit
fb46 c9                 ret
fb47            ;
fb47            ;
fb47            ;     i/o drivers for the disk follow
fb47            ;     for now, we will simply store the parameters away for use
fb47            ;     in the read and write   subroutines
fb47            ;
fb47            home: ;move to the track 00   position of current drive
fb47            ;     translate this call into a settrk call with Parameter 00
fb47 0e 00              LD     c, 0            ;select track 0
fb49 cd 66 fb           call   settrk
fb4c c9                 ret                    ;we will move to 00 on first read/write
fb4d            ;
fb4d            seldsk:     ;select disk given by register c
fb4d 21 00 00           LD    HL, 0000h        ;error return code
fb50 79                 LD    a, c
fb51 32 35 fc           LD    (diskno),A
fb54 fe 04              CP    disks            ;must be between 0 and 3
fb56 d0                 RET   NC               ;no carry if 4, 5,...
fb57            ;     disk number is in the proper range
fb57            ;     defs  10                 ;space for disk select
fb57            ;     compute proper disk Parameter header address
```

```
fb57 3a 35 fc                   LD    A,(diskno)
fb5a 6f                         LD    l, a              ;l=disk number 0, 1, 2, 3
fb5b 26 00                      LD    h, 0              ;high order zero
fb5d 29                         ADD   HL,HL             ;*2
fb5e 29                         ADD   HL,HL             ;*4
fb5f 29                         ADD   HL,HL             ;*8
fb60 29                         ADD   HL,HL             ;*16 (size of each header)
fb61 11 33 fa                   LD    DE, dpbase
fb64 19                         ADD   HL,DE             ;hl=,dpbase (diskno*16) Note typo here in original source.
fb65 c9                         ret
fb66                  ;
fb66                  settrk:     ;set track given by register c
fb66 79                         LD    a, c
fb67 32 2f fc                   LD    (track),A
fb6a c9                         ret
fb6b                  ;
fb6b                  setsec:     ;set sector given by register c
fb6b 79                         LD    a, c
fb6c 32 31 fc                   LD    (sector),A
fb6f c9                         ret
fb70                  ;
fb70                  ;
fb70                  sectran:
fb70                        ;translate the sector given by bc using the
fb70                        ;translate table given by de
fb70 eb                         EX    DE,HL             ;hl=.trans
fb71 09                         ADD   HL,BC             ;hl=.trans (sector)
fb72 c9                         ret                     ;debug no translation
fb73 6e                         LD    l, (hl)           ;l=trans (sector)
fb74 26 00                      LD    h, 0              ;hl=trans (sector)
fb76 c9                         ret                     ;with value in hl
fb77                  ;
fb77                  setdma:     ;set  dma address given by registers b and c
fb77 69                         LD    l, c              ;low order address
fb78 60                         LD    h, b              ;high order address
fb79 22 33 fc                   LD    (dmaad),HL        ;save the address
fb7c c9                         ret
fb7d                  ;
fb7d                  read:
fb7d                  ;Read one CP/M sector from disk.
```

```
fb7d             ;Return a 00h in register a if the operation completes properly, and 0lh if an error occurs
during the read.
fb7d             ;Disk number in 'diskno'
fb7d             ;Track number in 'track'
fb7d             ;Sector number in 'sector'
fb7d             ;Dma address in 'dmaad' (0-65535)
fb7d             ;
fb7d 21 72 fd                            ld    hl,hstbuf               ;buffer to place disk sector (256 bytes)
fb80 db 0f       rd_status_loop_1:       in    a,(0fh)                 ;check status
fb82 e6 80                               and   80h                     ;check BSY bit
fb84 c2 80 fb                            jp    nz,rd_status_loop_1     ;loop until not busy
fb87 db 0f       rd_status_loop_2:       in    a,(0fh)                 ;check      status
fb89 e6 40                               and   40h                     ;check DRDY bit
fb8b ca 87 fb                            jp    z,rd_status_loop_2      ;loop until ready
fb8e 3e 01                               ld    a,01h                   ;number of sectors = 1
fb90 d3 0a                               out   (0ah),a                 ;sector count register
fb92 3a 31 fc                            ld    a,(sector)              ;sector
fb95 d3 0b                               out   (0bh),a                 ;lba bits 0 - 7
fb97 3a 2f fc                            ld    a,(track)               ;track
fb9a d3 0c                               out   (0ch),a                 ;lba bits 8 - 15
fb9c 3a 35 fc                            ld    a,(diskno)              ;disk (only bits
fb9f d3 0d                               out   (0dh),a                 ;lba bits 16 - 23
fba1 3e e0                               ld    a,11100000b             ;LBA mode, select host drive 0
fba3 d3 0e                               out   (0eh),a                 ;drive/head register
fba5 3e 20                               ld    a,20h                   ;Read sector command
fba7 d3 0f                               out   (0fh),a
fba9 db 0f       rd_wait_for_DRQ_set:    in    a,(0fh)                 ;read status
fbab e6 08                               and   08h                     ;DRQ bit
fbad ca a9 fb                            jp    z,rd_wait_for_DRQ_set   ;loop until bit set
fbb0 db 0f       rd_wait_for_BSY_clear:  in    a,(0fh)
fbb2 e6 80                               and   80h
fbb4 c2 b0 fb                            jp    nz,rd_wait_for_BSY_clear
fbb7 db 0f                               in    a,(0fh)                 ;clear INTRQ
fbb9 db 08       read_loop:              in    a,(08h)                 ;get data
fbbb 77                                  ld    (hl),a
fbbc 23                                  inc   hl
fbbd db 0f                               in    a,(0fh)                 ;check status
fbbf e6 08                               and   08h                     ;DRQ bit
fbc1 c2 b9 fb                            jp    nz,read_loop            ;loop until clear
fbc4 2a 33 fc                            ld    hl,(dmaad)              ;memory location to place data read from
```

```
disk
fbc7 11 72 fd                          ld    de,hstbuf             ;host buffer
fbca 06 80                             ld    b,128                 ;size of CP/M sector
fbcc 1a          rd_sector_loop:       ld    a,(de)                ;get byte from host buffer
fbcd 77                                ld    (hl),a                ;put in memory
fbce 23                                inc   hl
fbcf 13                                inc   de
fbd0 10 fa                             djnz  rd_sector_loop        ;put 128 bytes into memory
fbd2 db 0f                             in    a,(0fh)               ;get status
fbd4 e6 01                             and   01h                   ;error bit
fbd6 c9                                ret
fbd7
fbd7             write:
fbd7             ;Write one CP/M sector to disk.
fbd7             ;Return a 00h in register a if the operation completes properly, and 0lh if an error occurs
during the read or write
fbd7             ;Disk number in 'diskno'
fbd7             ;Track number in 'track'
fbd7             ;Sector number in 'sector'
fbd7             ;Dma address in 'dmaad' (0-65535)
fbd7 2a 33 fc                          ld    hl,(dmaad)            ;memory location of data to write
fbda 11 72 fd                          ld    de,hstbuf             ;host buffer
fbdd 06 80                             ld    b,128                 ;size of CP/M sector
fbdf 7e          wr_sector_loop:       ld    a,(hl)                ;get byte from memory
fbe0 12                                ld    (de),a                ;put in host buffer
fbe1 23                                inc   hl
fbe2 13                                inc   de
fbe3 10 fa                             djnz  wr_sector_loop        ;put 128 bytes in host buffer
fbe5 21 72 fd                          ld    hl,hstbuf             ;location of data to write to disk
fbe8 db 0f       wr_status_loop_1:     in    a,(0fh)               ;check status
fbea e6 80                             and   80h                   ;check BSY bit
fbec c2 e8 fb                          jp    nz,wr_status_loop_1   ;loop until not busy
fbef db 0f       wr_status_loop_2:     in    a,(0fh)               ;check      status
fbf1 e6 40                             and   40h                   ;check DRDY bit
fbf3 ca ef fb                          jp    z,wr_status_loop_2    ;loop until ready
fbf6 3e 01                             ld    a,01h                 ;number of sectors = 1
fbf8 d3 0a                             out   (0ah),a               ;sector count register
fbfa 3a 31 fc                          ld    a,(sector)
fbfd d3 0b                             out   (0bh),a               ;lba bits 0 - 7 = "sector"
fbff 3a 2f fc                          ld    a,(track)
```

```
fc02 d3 0c                            out   (0ch),a                 ;lba bits 8 - 15 = "track"
fc04 3a 35 fc                         ld    a,(diskno)
fc07 d3 0d                            out   (0dh),a                 ;lba bits 16 to 20 used for "disk"
fc09 3e e0                            ld    a,11100000b             ;LBA mode, select drive 0
fc0b d3 0e                            out   (0eh),a                 ;drive/head register
fc0d 3e 30                            ld    a,30h                   ;Write sector command
fc0f d3 0f                            out   (0fh),a
fc11 db 0f       wr_wait_for_DRQ_set:    in    a,(0fh)                 ;read status
fc13 e6 08                            and   08h                     ;DRQ bit
fc15 ca 11 fc                         jp    z,wr_wait_for_DRQ_set   ;loop until bit set
fc18 7e          write_loop:             ld    a,(hl)
fc19 d3 08                            out   (08h),a                 ;write data
fc1b 23                               inc   hl
fc1c db 0f                            in    a,(0fh)                 ;read status
fc1e e6 08                            and   08h                     ;check DRQ bit
fc20 c2 18 fc                         jp    nz,write_loop           ;write until bit cleared
fc23 db 0f       wr_wait_for_BSY_clear:  in    a,(0fh)
fc25 e6 80                            and   80h
fc27 c2 23 fc                         jp    nz,wr_wait_for_BSY_clear
fc2a db 0f                            in    a,(0fh)                 ;clear INTRQ
fc2c e6 01                            and   01h                     ;check for error
fc2e c9                               ret
fc2f             ;
fc2f             ;     the remainder of the cbios is reserved uninitialized
fc2f             ;     data area, and does not need to be a Part of the
fc2f             ;     system      memory image (the space must be available,
fc2f             ;     however, between"begdat" and"enddat").
fc2f             ;
fc2f 00...       track:           defs  2           ;two bytes for expansion
fc31 00...       sector:          defs  2           ;two bytes for expansion
fc33 00...       dmaad:           defs  2           ;direct memory address
fc35 00...       diskno:          defs  1           ;disk number 0-15
fc36             ;
fc36             ;     scratch ram area for bdos use
fc36             begdat:          equ   $           ;beginning of data area
fc36 00...       dirbf:           defs  128         ;scratch directory area
fcb6 00...       all00:           defs  31          ;allocation vector 0
fcd5 00...       all01:           defs  31          ;allocation vector 1
fcf4 00...       all02:           defs  31          ;allocation vector 2
fd13 00...       all03:           defs  31          ;allocation vector 3
```

```
fd32 00...        chk00:      defs  16          ;check vector 0
fd42 00...        chk01:      defs  16          ;check vector 1
fd52 00...        chk02:      defs  16          ;check vector 2
fd62 00...        chk03:      defs  16          ;check vector 3
fd72              ;
fd72              enddat:     equ   $           ;end of data area
fd72              datsiz:     equ   $-begdat;   ;size of data area
fd72 00...        hstbuf: ds  256         ;buffer for host disk sector
fe72                    end
# End of file z80_cbios.asm
fe72
```

Format

```
# File format.asm
0000              ;Formats four classical CP/M disks
0000              ;Writes E5h to 26 sectors on tracks 2 to 77 of each disk.
0000              ;Uses calls to cbios, in memory at FA00h
0000              seldsk:                 equ   0fa1bh            ;pass disk no. in c
0000              setdma:                 equ   0fa24h            ;pass address in bc
0000              settrk:                 equ   0fa1eh            ;pass track in reg C
0000              setsec:                 equ   0fa21h            ;pass sector in reg c
0000              write:                  equ   0fa2ah            ;write one CP/M sector to disk
0000              monitor_warm_start:     equ   046fh
0000                                      org   0800h
0800 31 09 09                             ld    sp,format_stack
0803 3e 00                                ld    a,00h             ;starting disk
0805 32 64 08                             ld    (disk),a
0808 4f           disk_loop:              ld    c,a               ;CP/M disk a
0809 cd 1b fa                             call  seldsk
080c 3e 02                                ld    a,2               ;starting track (offset = 2)
080e 32 66 08                             ld    (track),a
0811 3e 00        track_loop:             ld    a,0               ;starting sector
0813 32 65 08                             ld    (sector),a
0816 21 69 08                             ld    hl,directory_sector     ;address of data to write
0819 22 67 08                             ld    (address),hl
081c 3a 66 08                             ld    a,(track)
081f 4f                                   ld    c,a               ;CP/M track
0820 cd 1e fa                             call  settrk
```

```
0823 3a 65 08     sector_loop:           ld    a,(sector)
0826 4f                                  ld    c,a                ;CP/M sector
0827 cd 21 fa                            call  setsec
082a ed 4b 67 08                         ld    bc,(address)       ;memory location
082e cd 24 fa                            call  setdma
0831 cd 2a fa                            call  write
0834 3a 65 08                            ld    a,(sector)
0837 fe 1a                               cp    26
0839 ca 43 08                            jp    z,next_track
083c 3c                                  inc   a
083d 32 65 08                            ld    (sector),a
0840 c3 23 08                            jp    sector_loop
0843 3a 66 08     next_track:            ld    a,(track)
0846 fe 4d                               cp    77
0848 ca 52 08                            jp    z,next_disk
084b 3c                                  inc   a
084c 32 66 08                            ld    (track),a
084f c3 11 08                            jp    track_loop
0852 3a 64 08     next_disk:             ld    a,(disk)
0855 3c                                  inc   a
0856 fe 04                               cp    4
0858 ca 61 08                            jp    z,done
085b 32 64 08                            ld    (disk),a
085e c3 08 08                            jp    disk_loop
0861 c3 6f 04     done:                  jp    monitor_warm_start
0864 00           disk:                  db    00h
0865 00           sector:                db    00h
0866 00           track:                 db    00h
0867 00 00        address:               dw    0000h
0869              directory_sector:
0869 0xe5...                       ds    128,0e5h          ;byte for empty directory
08e9 00...                         ds    32                ;stack space
0909              format_stack:
0909                          end
# End of file format.asm
0909
```

Putsys

```
# File putsys.asm
0000            ;Copies the memory image of CP/M loaded at E400h onto tracks 0 and 1 of the first CP/M disk
0000            ;Load and run from ROM monitor
0000            ;Uses calls to cbios, in memory at FA00h
0000            ;Writes track 0, sectors 2 to 26, then track 1, sectors 1 to 25
0000            seldsk:         equ   0fa1bh          ;pass disk no. in c
0000            setdma:         equ   0fa24h          ;pass address in bc
0000            settrk:         equ   0fa1eh          ;pass track in reg C
0000            setsec:         equ   0fa21h          ;pass sector in reg c
0000            write:          equ   0fa2ah          ;write one CP/M sector to disk
0000            monitor_warm_start:     equ   046Fh ;Return to ROM monitor
0000                            org   0800h
0800 0e 00                      ld    c,00h           ;CP/M disk a
0802 cd 1b fa                   call  seldsk
0805            ;Write track 0, sectors 2 to 26
0805 3e 02                      ld    a,2             ;starting sector
0807 32 80 08                   ld    (sector),a
080a 21 00 e4                   ld    hl,0E400h       ;memory address to start
080d 22 81 08                   ld    (address),hl
0810 0e 00                      ld    c,0             ;CP/M track
0812 cd 1e fa                   call  settrk
0815 3a 80 08   wr_trk_0_loop:  ld    a,(sector)
0818 4f                         ld    c,a             ;CP/M sector
0819 cd 21 fa                   call  setsec
081c ed 4b 81 08                ld    bc,(address)    ;memory location
0820 cd 24 fa                   call  setdma
0823 cd 2a fa                   call  write
0826 3a 80 08                   ld    a,(sector)
0829 fe 1a                      cp    26
082b ca 3f 08                   jp    z,wr_trk_1
082e 3c                         inc   a
082f 32 80 08                   ld    (sector),a
0832 2a 81 08                   ld    hl,(address)
0835 11 80 00                   ld    de,128
0838 19                         add   hl,de
0839 22 81 08                   ld    (address),hl
083c c3 15 08                   jp    wr_trk_0_loop
```

```
083f              ;Write track 1, sectors 1 to 25
083f 0e 01        wr_trk_1:        ld    c,1
0841 cd 1e fa                      call  settrk
0844 2a 81 08                      ld    hl,(address)
0847 11 80 00                      ld    de,128
084a 19                            add   hl,de
084b 22 81 08                      ld    (address),hl
084e 3e 01                         ld    a,1
0850 32 80 08                      ld    (sector),a
0853 3a 80 08     wr_trk_1_loop:   ld    a,(sector)
0856 4f                            ld    c,a                 ;CP/M sector
0857 cd 21 fa                      call  setsec
085a ed 4b 81 08                   ld    bc,(address)        ;memory location
085e cd 24 fa                      call  setdma
0861 cd 2a fa                      call  write
0864 3a 80 08                      ld    a,(sector)
0867 fe 19                         cp    25
0869 ca 7d 08                      jp    z,done
086c 3c                            inc   a
086d 32 80 08                      ld    (sector),a
0870 2a 81 08                      ld    hl,(address)
0873 11 80 00                      ld    de,128
0876 19                            add   hl,de
0877 22 81 08                      ld    (address),hl
087a c3 53 08                      jp    wr_trk_1_loop
087d c3 6f 04     done:            jp    monitor_warm_start
0880 00           sector:          db    00h
0881 00 00        address:         dw    0000h
0883                               end

# End of file putsys.asm
0883
```

CP/M loader

```
# File cpm_loader.asm
0000              ;Retrieves CP/M from disk and loads it in memory starting at E400h
0000              ;Uses calls to ROM routine for disk read.
0000              ;Reads track 0, sectors 2 to 26, then track 1, sectors 1 to 25
```

```
0000                ;This program is loaded into LBA sector 0 of disk, read to loc. 0800h by ROM and executed.
0000                hstbuf:          equ   0900h       ;will put 256-byte raw sector here
0000                disk_read:       equ   0294h       ;in 2K ROM
0000                cpm:             equ   0FA00h            ;CP/M cold start entry
0000                                 org   0800h
0800                ;Read track 0, sectors 2 to 26
0800 3e 02                           ld    a,2               ;starting sector
0802 32 84 08                        ld    (sector),a
0805 21 00 e4                        ld    hl,0E400h         ;memory address to start
0808 22 86 08                        ld    (dmaad),hl
080b 3e 00                           ld    a,0               ;CP/M track
080d 32 85 08                        ld    (track),a
0810 cd 61 08       rd_trk_0_loop:   call  read
0813 3a 84 08                        ld    a,(sector)
0816 fe 1a                           cp    26
0818 ca 2c 08                        jp    z,rd_trk_1
081b 3c                              inc   a
081c 32 84 08                        ld    (sector),a
081f 2a 86 08                        ld    hl,(dmaad)
0822 11 80 00                        ld    de,128
0825 19                              add   hl,de
0826 22 86 08                        ld    (dmaad),hl
0829 c3 10 08                        jp    rd_trk_0_loop
082c                ;Read track 1, sectors 1 to 25
082c 3e 01          rd_trk_1:        ld    a,1
082e 32 85 08                        ld    (track),a
0831 2a 86 08                        ld    hl,(dmaad)
0834 11 80 00                        ld    de,128
0837 19                              add   hl,de
0838 22 86 08                        ld    (dmaad),hl
083b 3e 01                           ld    a,1               ;starting sector
083d 32 84 08                        ld    (sector),a
0840 cd 61 08       rd_trk_1_loop:   call  read
0843 3a 84 08                        ld    a,(sector)
0846 fe 19                           cp    25
0848 ca 5c 08                        jp    z,done
084b 3c                              inc   a
084c 32 84 08                        ld    (sector),a
084f 2a 86 08                        ld    hl,(dmaad)
0852 11 80 00                        ld    de,128
```

```
0855 19                         add   hl,de
0856 22 86 08                   ld    (dmaad),hl
0859 c3 40 08                   jp    rd_trk_1_loop
085c d3 01       done:          out   (1),a            ;switch memory config to all-RAM
085e c3 00 fa                   jp    cpm
0861
0861             read:
0861             ;Read one CP/M sector from disk 0
0861             ;Track number in 'track'
0861             ;Sector number in 'sector'
0861             ;Dma address (location in memory to place the CP/M sector) in 'dmaad' (0-65535)
0861             ;
0861 21 00 09                   ld    hl,hstbuf        ;buffer to place raw disk sector (256 bytes)
0864 3a 84 08                   ld    a,(sector)
0867 4f                         ld    c,a              ;LBA bits 0 to 7
0868 3a 85 08                   ld    a,(track)
086b 47                         ld    b,a              ;LBA bits 8 to 15
086c 1e 00                      ld    e,00h            ;LBA bits 16 to 23
086e cd 94 02                   call  disk_read        ;subroutine in ROM
0871             ;Transfer top 128-bytes out of buffer to memory
0871 2a 86 08                   ld    hl,(dmaad)       ;memory location to place data read from disk
0874 11 00 09                   ld    de,hstbuf        ;host buffer
0877 06 80                      ld    b,128            ;size of CP/M sector
0879 1a          rd_sector_loop: ld   a,(de)           ;get byte from host buffer
087a 77                         ld    (hl),a           ;put in memory
087b 23                         inc   hl
087c 13                         inc   de
087d 10 fa                      djnz  rd_sector_loop   ;put 128 bytes into memory
087f db 0f                      in    a,(0fh)          ;get status
0881 e6 01                      and   01h              ;error bit
0883 c9                         ret
0884 00          sector:        db    00h
0885 00          track:         db    00h
0886 00 00       dmaad:         dw    0000h
0888                        end
```

```
# End of file cpm_loader.asm
0888
```

Table of Tested Disk Drives

Drive	Year of manufacture	Size	Passed diskrd/diskwr test	CP/M installed successfully
Mechanical Hard Disk Drives				
Seagate ST3290A		261.3 Mb	Yes	Yes, but gave bad sector errors
Western Digital Caviar 32500	1996	2559.8 Mb	No	Not attempted
Seagate Medalist 4321	1999	4.3 Gb	No	Not attempted
Seagate Medalist 4310	1999	4.3 Gb	No	Not attempted
Western Digital WD200	2001	20.0 Gb	No	Not attempted
Western Digital WD400	2003	40.0 Gb	No	Not attempted
Western Digital Caviar 31600	1995	1624.6 Mb	No	Not attempted
Western Digital Caviar 153BA	2000	15.3 Gb	No	Not attempted
Maxtor 71626AP	1996	1630 Mb	Yes	Yes
Maxtor 90845D4	2000	8.5 Gb	Yes	Yes
Seagate Medalist 10232	1999	10 Gb	Yes	Yes
Seagate Barracuda ATA II	2000	15.3 Gb	Yes	Yes
Maxtor DiamondMax Plus 9	2003	120 Gb	Yes	Yes
Seagate U4 ST36421A	2000	6.4 Gb	Yes	Yes
Seagate U6 ST380020A	2002	80 Gb	Yes	Not attempted (I wanted to

				preserve disk contents)
Fujitsu MPE3102AT	1999	10.2 Gb	Yes	Yes
Seagate Barracuda ATA V Model ST380023A	2003	80 Gb	Yes	Yes
Maxtor DiamondMax Plus 8	2003	40 Gb	Yes	Yes
Seagate Barracuda 7200.7 Model ST380011A	2004	80 Gb	Yes	Yes
SATA drive with SATA to IDE adapter[7]				
Fujitsu MHV2080BH PL HD SATA		80 Gb	Yes	
Solid State (Flash) IDE drives				
Silicon Drive SSD-M01G-3100		1 Gb	Yes	Yes
SimpleTech 94000-00964 solid state IDE drive			Yes	Yes
Transcend 40-pin IDE flash module[8]		1 Gb	Yes	Yes
Compact Flash drives in IDE Adapter[9]				
Sandisk CF SDCFB	2003	256 Mb	Yes	Yes
Canon FC-32MH	2002	32 Mb	Yes	Not attempted – drive too small
Iomega Microdrive DMDM-10340[10]		340 Mb	Yes	Yes

7 Generic IDE to SATA or SATA to IDE Adapter, purchased on Amazon $2.99
8 Some Transcend modules may not work. The tested module had identification number 145194R 0502 SS63 1G 0632.
9 SYBA SD-CF-IDE-DI IDE to Compact Flash Adapter (Direct Insertion Mode), purchased from Newegg $8.49
10 This is a mechanical drive in a CF enclosure and needs +12V to operate.

SD Card in IDE Adapter[11]				
Canon MultiMediaCard MMC-16M		16 Mb	Yes	Yes – only drive A (card too small for B, C, and D)

11 SLOA063 40-Pin Female IDE To SD Card Adapter made by Soarland, purchased on Amazon $18.98