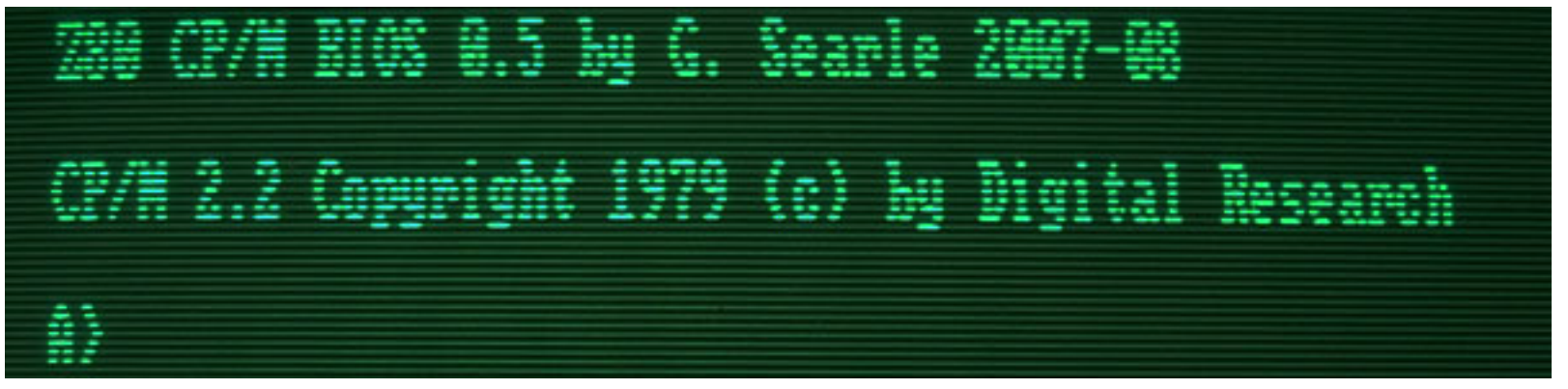
CP/M on breadboard



(Only 9 chips needed for a serial-only version with full 64K RAM and power-up ROM)
(Or only 8 chips if using a USB to TTL serial cable)

Details of the add-on display and PC keyboard controller are shown at the bottom of the page

VERIFIED WORKING
RE-BUILT FROM INFO PROVIDED HERE

by Grant Searle

For news and updates, follow me on Twitter:

Last update: 8th September 2013

[FOR A SLIGHTLY SIMPLER Z80 COMPUTER (RUNNING BASIC), CLICK HERE]

Major changes log:
** March 2007 - First working version of my CP/M BIOS*
** June 2008 - BIOS Version 0.5*
** 11th August 2012 - Update to the schematic to correct the SIO/2 pinout*
** 12th August 2012 - The SIO and Compact Flash resets now connected to the /COLD RESET line*
** 13th August 2012 - SIO initialisation now writes register 4 before the others, as in line with some datasheets. No change to version number as this was a very minor change and does not affect functionality.*
** 21st November 2012 - Video and keyboard interface added (updated 22nd Nov 2012)*
** 1st December 2012 - Keyboard now supports LED setting. Num lock implemented,*
** 16th December 2012 - Corrected an error on the schematic - two of the serial port pins were incorrectly connected.*
** 23rd December 2012 - Changed CF connection schematic to show pin names.*
** 30th December 2012 - Added important construction notes for the main circuit and the compact flash here . Updated FilePackager (v1.1) application here*
** 17th January 2013 - BIOS version 1.0 released here - now handles the IOBYTE settings - thanks to Andrew Quinn for his help and support.*
** 19th January 2013 - LBA disk addressing details added here.*
** 25th April 2013 - Small update to the schematic to include a decoupling capacitor on the MAX232*
** 17th July 2013 - Updated software for the keyboard and display interfaces. Also details of control codes and characters here*
** 22nd July 2013 - Keyboard/display interface redesigned to use an ATmega328 instead of the bigger ATmega32.*
** 23rd July 2013 - DCDA and DCDB pins should be connected to ground (as on my pictures). Schematic corrected - thanks to Michael Berger for spotting this.*
** 6th August 2013 - Details on how to create your own transfer file using the file format that I use here.*
** 8th September 2013 - ROM BASIC updated slightly. Details of BASIC in ROM added to this page. See below for details.*

Contents

INTRODUCTION

This project shows how it is possible to construct a Z80 computer running full CP/M 2.2 with disk storage using a minimal chip count. Source code and schematics are included to allow modifications if needed. Several commercial CP/M packages have been tried on this and they work perfectly. A large library of applications are available elsewhere on the web, and these can be loaded onto this machine using my own custom-made transfer software.
Full usage instructions are included on this page.
Also included is a ROM version of BASIC to allow the machine to be programmed without CP/M.

A LOT of work has gone into this project, as I have not only designed the hardware, but written the Boot ROM, BIOS, Formatting, System Transfer and File transfer programs. Also, many hours spent in making the ROM BASIC as clean as possible, with extended commands written by myself within it.

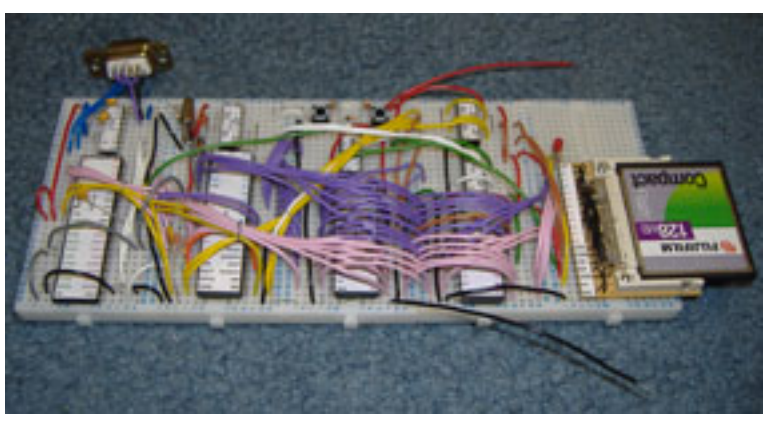
Features
CPU: Z80 running at 7.3728 MHz (* see below)
Interface: Two high-speed serial ports at 115200 Baud. One with a fully compatible RS232 driver.
Fully buffered serial input on both serial ports. Supported by interrupt-driven drivers and circular buffers. Buffer size is 60 bytes. Full signal indicated when 20 chars received to accommodate run-on from the sender. Empty signal sent when 5 chars remain in the buffer.
Disk: 64MB or 128MB Compact Flash card support, containing 8 or 16 logical drives, respectively.
RAM: 64K Byte
ROM: 16K Byte, switched off when CP/M active. ROM contains the bootloader and memory load utilities. Also contains Microsoft BASIC, as used on the NASCOM computer, modified to remove code that is not relevant to a serial-interfaced board (eg. screen handling and keyboard matrix scanning).
Resets: Both cold (full reset) and warm reset (used to return to CP/M prompt) circuitry
Power consumption: Less than 250mA
Chip count: 9 for a fully functional CP/M implementation with an RS232 compliant serial port
CP/M support: 2.2 with included software.

The Z80 is overclocked to 7.3728 MHz. All Z80 chips that I have tested (ST, Zilog) will **easily** overclock to about 10MHz, some to 12 MHz, provided the clock edges are FAST, so the clock rate that I use is well within my tested maximum for the chips. I have been running the CPU overclocked for a long time and it has never caused any problems.

* - You can use lower clock speeds provided you adjust the terminal's serial port speed to match (ie. a 3.6864MHz crystal will give a 57600 baud interface, and a 2.4576MHz crystal will give a 38400 baud interface). However, the display interface shown later will only work at 115200 unless you alter the source code or alter the crystal on the ATmega88 as needed. All descriptions on this page assumes the 7.3728 MHz clock is being used.

I use a Z80 SIO/2 chip to give me 2 serial ports. One serial port is available to connect to a PC running at 115200.

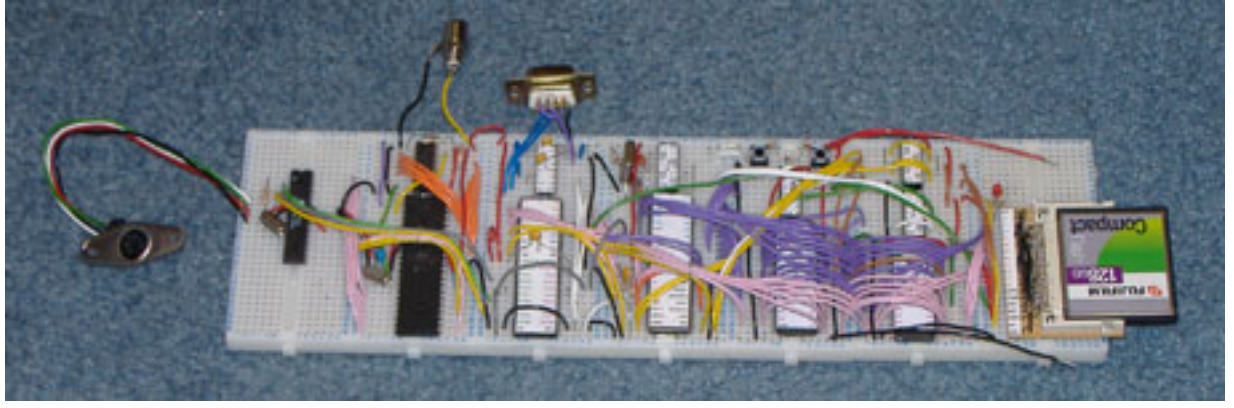
9-chip serial version (8 if using a USB to TTL serial cable)



Expanded version
As above, but the second port is connected internally to an ATMEGA88 processor which buffers the data between the CP/M computer and a PS/2 keyboard and an 80x25 character display.

The keyboard interface is internal to the ATmega88 processor, along with the serial buffers.

The 80x25 text video controller was created using an ATmega328 processor. This was based on a design and code produced by Daryl Rictor (he produced a 40x25 display using an ATmega8). I adapted the code to use the larger memory of the ATmega328 (originally on an ATmega32) and modified it significantly to make an 80 char display, along with an ANSI-compatible command interpreter. 2-wire handshaking between the display processor and the IO processor allows correct synchronisation between the processors when transferring serial data to the display processor.
Using a separate processor with it's own memory allows the Z80 processor to have as much memory available as possible. I have written a minimal-ANSI interpreter within the display processor to handle escape sequences so that CP/M software can be set to use an ANSI terminal setting, but can run on a standard PAL compatible monitor/television.
So, in effect, using the ATmega processors, I have created a small minimum-ANSI compatible terminal!



Connections visible in this picture are:
Left-hand side : PS/2 compatible keyboard
Top-left : Monitor connection
Top-mid-left : 115200 serial connection

The display processor runs at 16 MIPS and is surprisingly fast at updating and scrolling, making my CP/M machine perform much faster than the commercial machines that I own/used.

General (for both boards)

The two pushbuttons on the top mid-right of the boards are for cold reset and warm reset.
Warm reset is the standard CP/M reset to return to the command prompt. The cold reset will page the ROM back in to the bottom 16K of memory and start the monitor.
The complete machine, with a PC keyboard connected consumes approx 240mA.

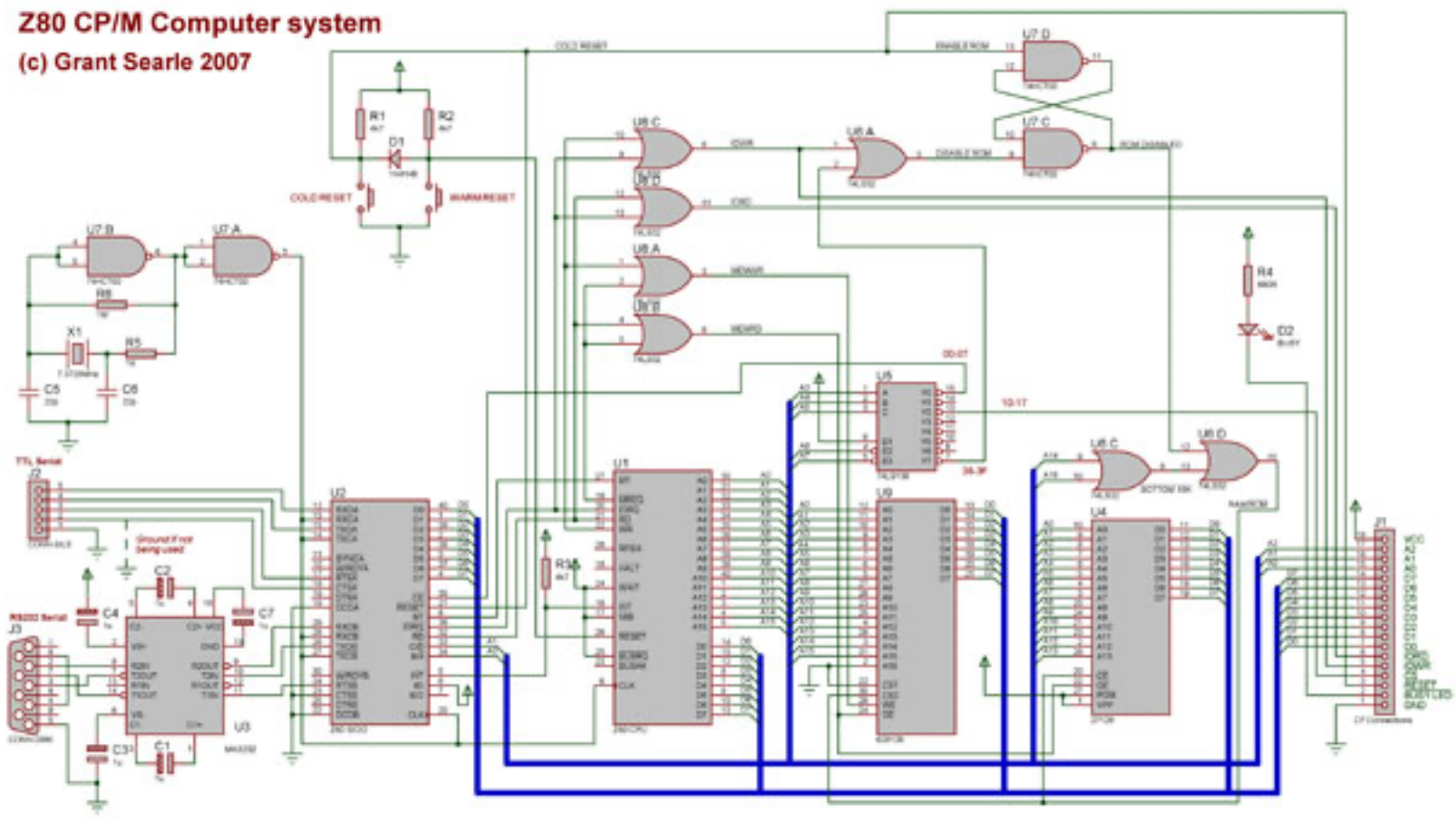
The ROM is turned off by writing to port 38H.

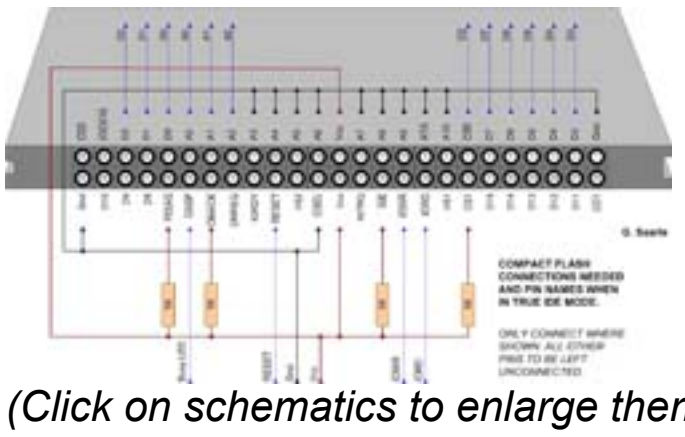
The main CP/M machine, operating with a single RS232 serial port (and a second TTL-level port) has only 9 ICS !
(I could have eliminated the 74LS138 to reduce the chip count to 8, but this would have given a less well-defined I/O memory map. I wanted this design to remain expandable so I have explicit I/O decoding of the first 64 locations. The spare outputs on the '138 are decoded to 8-location blocks to allow easy expansion of extra I/O devices.)

The 80x25 display and the PS/2 keyboard interface is a total of 3 ICs, so a complete stand-alone machine has been built with 12 ICs.

SCHEMATIC

Here is the circuit diagram of the machine (serial only, without display), and also the connections that were needed for the compact flash card:





(Click on schematics to enlarge them)

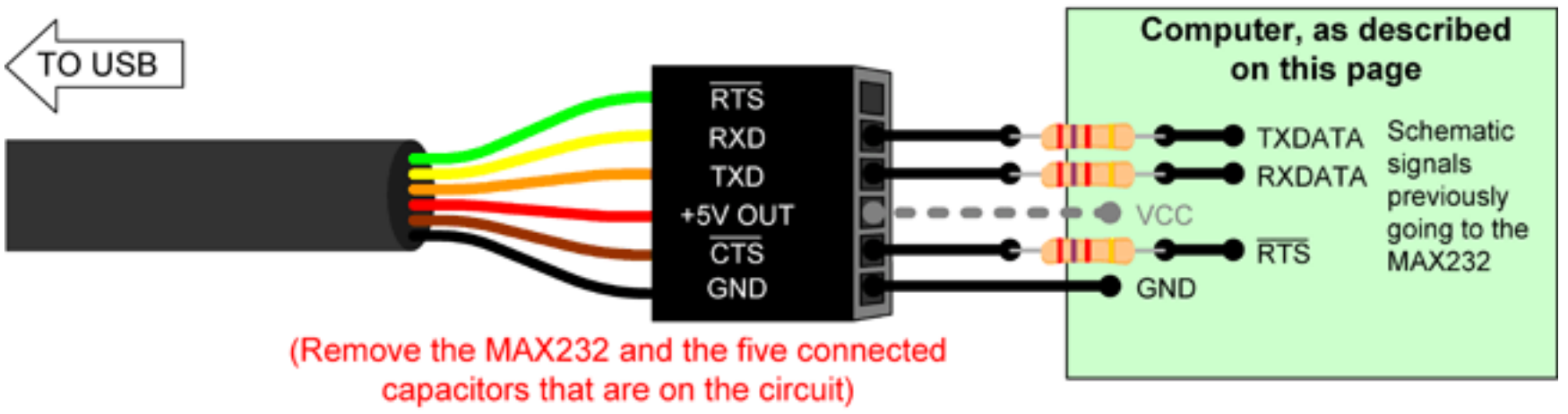
UPDATE 1: 11th August 2012 - Correction to schematic to show correct pinout for the Z80 SIO/2 (breadboard pics below are correct, however). 12th August 20120 - The /RESET on the SIO and Compact Flash are now taken to the COLD reset so that they are reset properly if needed. Thanks to

Andrew Quinn for his feedback on this.
UPDATE 2: 16th December 2012 - David Asta has spotted an error (thanks!) on the RS232 serial connector on my schematic (when comparing it to the picture that I have on the page).
I had pin 4 connected to pin 9. It's supposed to be pin 4 (DTR) connected to pin 6 (DSR). Also, I had Pin 8 connecting to T1 out. It should have been pin 7 (RTS) connected to T1 out.
UPDATE 3: 25th April 2013 - Include the decoupling capacitor on the MAX232 and change the capacitor values from 10uF to 1uF.
UPDATE 4: 23rd July 2013 - Thanks to Michael Berger for spotting an error with the schematic - the DCDA and DCDB pins should be connected to ground. They are on my pictures, but I missed them off the schematic. This has now been corrected.

Alternative serial (and power) connection

The circuit assumes a connection to a standard RS232 port. However, this could also be connected to a USB to TTL (5V) serial cable instead. If doing this, the MAX232, capacitors C1 to C5 and the serial connector are not needed. This therefore reduces the chip count by 1.

Using a USB to 5V (TTL) serial cable instead of an RS232 interface



(Remove the MAX232 and the five connected capacitors that are on the circuit)

Resistors are present to safely limit current along the cable if the USB or computer is not powered.
These are not needed if using +5V OUT to VCC to power the computer - connect directly instead.

By using a USB to serial cable the board can also be powered from the USB port down the same cable (a powered hub could be used if higher current needed), eliminating the need for a separate power supply for this board. Just connect the +5V out on the cable to the Vcc supply on the board, as shown on the dotted line.
IMPORTANT - only connect Vcc to the USB cable if there is no other power supplied to the board.
The 2k7 resistors are present in the diagram because the board may be powered but not plugged in to the USB cable, or the USB cable could be plugged in and active without power to the board. These limit the current to avoid power being drawn through the interface pins. If always powering the board via the cable then no resistors are necessary.

IMPORTANT CONSTRUCTION NOTES

MAIN CIRCUIT
1. The Z80 SIO can be an SIO/0, SIO/1 or SIO/2 (as I have used). However, the pin-out for some of it is different due to some of the pins having different functions. It is therefore important that you check the datasheet for the type that you are using to ensure the correct pins are used - they are not directly interchangeable even though all will work.
2. The clock generator needs strong pull-ups to ensure correct CPU operation (fast clock edges). U7 must be a CMOS (and TTL compatible) type ie. 74HCT00. HC is not suitable as the inputs are not TTL compatible (could possible add pull-ups where needed, but this is untested). The LS version is not suitable. All other logic can be LS types.
3. No decoupling capacitors are shown. These can be added across the clock IC and the CPU etc as needed.
4. Power pins not shown in the schematics - this is normal practice for logic circuits and these are assumed, and all ICs must be connected to the 0V and +5V power rails.
5. Optional: You may wish to use a CMOS Z80 for U1 and 74HCT CMOS devices for U5 and U8 (see CF notes below). All other LS chips can be replaced with 74HCT devices if needed.
6. A MAX202 could be used instead of the MAX232, in which case C1-C5 should be 0.1uF.

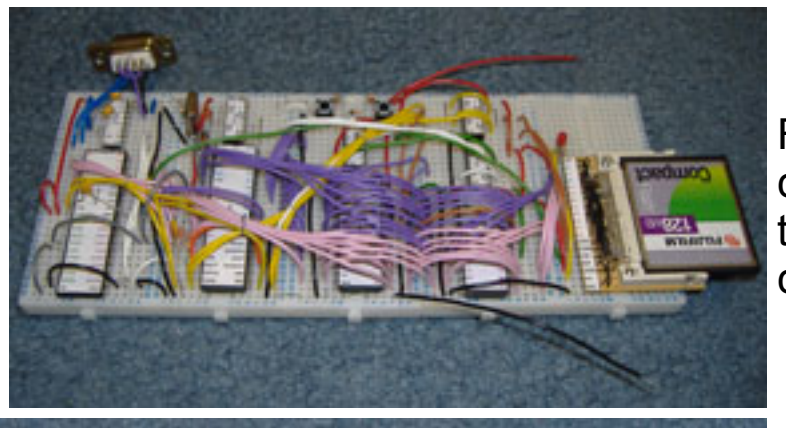
COMPACT FLASH
1. For reliable operation, the leads to the CF card must be short.
2. Some cards are more sensitive to noise on the lines. FujiFilm cards work very well, SanDisk cards work but need short wires. To get one card working I needed to connect a very small capacitor (15-22pF) between the address lines and Gnd. Ferrite beads on the lines also help.
3. If the results are intermittent, ensure you follow the above guidelines.
4. The CF specification says that the logic levels should be "CMOS" values. However, from ALL cards that I have tried, although this is specified, the TTL levels work perfectly. If in doubt, you could replace the Z80 with a CMOS device and replace the 74LS138 with a 74HCT138 and replace the 74LS32 with 74HCT32. This way, all lines to the CF will be CMOS levels. However, this should not be necessary.
5. Cards larger than 128MB can be used (use the 128MB BIOS and formatter) but only the first 128MB will be recognised by CP/M - this should be more than ample, though. Avoid new high speed cards as these are very sensitive to noise (due to the high speed) and will require short leads.
6. Route leads as directly to the Z80 and to U5 and U8 as possible. On my picture, the address lines were taken from the end of a chain of connections which works perfectly for my FujiFilm cards. However, the SanDisk card was not reliable with this, but worked perfectly when I took the address lines straight to the Z80 CPU.

IMPLEMENTATION ON BREADBOARD

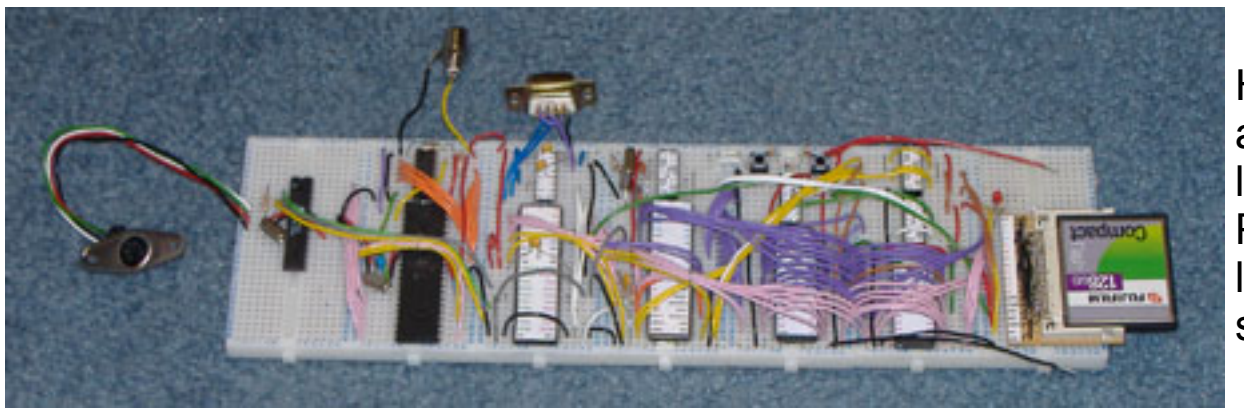
Here are some photos of my machine.

Labels attached to the chips to make construction easier can be obtained HERE.

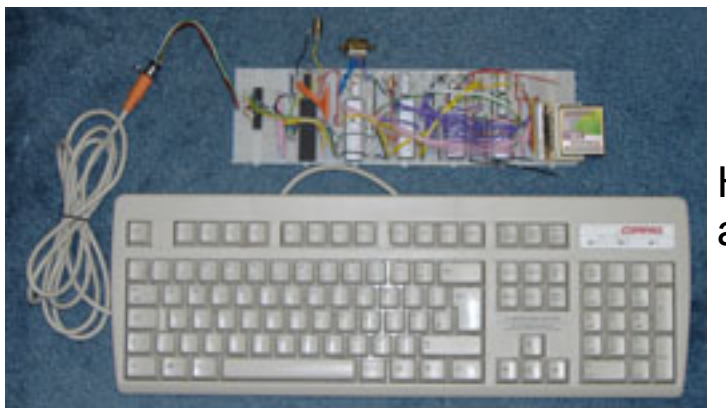
(Click on any of the pictures to enlarge)



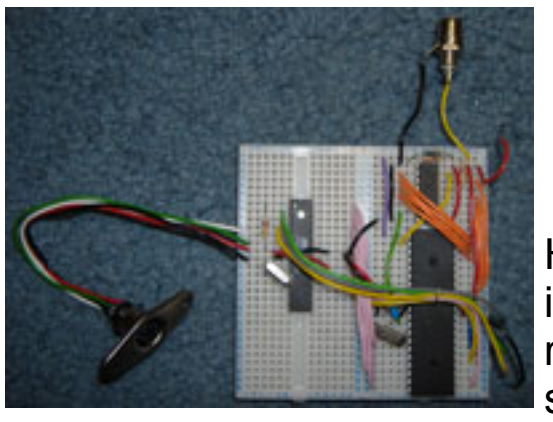
Full 64K memory CP/M implementation. This is the "9-chip" serial-only version. The connector on the top-left is the serial port, and the buttons in the middle at the top are cold reset and warm reset.



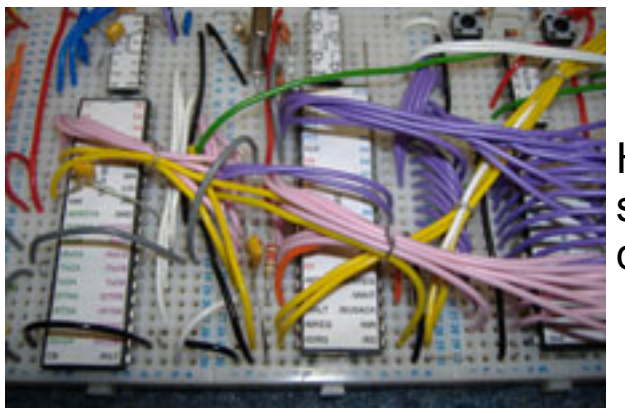
Here is the same machine with the PC (PS/2) keyboard and 80x25 display processors added (the 3 chips on the left-hand side). The connector on the top to the left is the PAL-compatible monitor output, and the connector on the left is the PS/2 connector for a standard PC keyboard. The serial port is still fully functional.



Here it is with a "standard" PC keyboard plugged in, to allow you to see the actual size.



Here is the PS/2 (PC) keyboard and monochrome monitor interface (my "mini-terminal"). This attaches to a serial port running at 115200 baud. I attach this to the TTL-compatible spare serial port on my CP/M machine to allow the existing RS232 serial port to remain active.



Here is a close-up of some of the chip labels that I made to stick on the chips. This allows me to make the connections quickly and accurately.

The disk drive and the fine soldering needed to make a CF adapter to plug into the breadboard. Bend every other pin upwards (to create a zig-zag). This then allows sufficient space for soldering when using a fine bit and wire.

ROM FILES AND CP/M SYSTEM FILES

DOWNLOAD FILES HERE

UPDATE: 13th August 2012 - Very minor change to the SIO initialisation in the CBIOS and MONITOR files so that the SIO register 4 is written before some of the others. This then matches the notes in some datasheets. Previously 0,1,2,3,4,5 now 0,4,1,2,3,5. I haven't experienced any problems previously, though, and this change is just to ensure 100% compatibility in case there are some device versions that need it. As it is such a small change I have not changed the version number on the BIOS - still at 0.5. Thanks to David Asta for pointing this out to me.
UPDATE: 17th January 2013 - BIOS versions now set to 1.0 - changes include the implementation of IOBYTE and also a small fix to ensure changing to

an invalid drive letter will reset the working drive back to A:

The ZIP file presented here contains the following files. Please read the rest of the page to see what files are needed and when to use them.

ROM contents
ROM.HEX (all that is needed to program the ROM. Contains the BASIC.HEX and MONITOR.HEX)

Windows file packager
FilePackage.exe
COMDLG32.OCX <== *Copy to the same folder as FilePackage.exe if you get an error saying it is missing when running the packager program.*

CP/M "standard" transient programs
CPM211FilesPkg.txt
Optional, but available pre-packaged ready to install if needed. See further down as to how to use the package files (basically, open it and paste all into a terminal window showing the CP/M prompt after the download program has been installed on the CP/M A: drive). Contains LOAD, PIP, STAT, DDT, DISPLAY, DUMP, ED and ASM, as supplied with CP/M 2.2.

CP/M Installation files
BASIC.HEX
DOWNLOAD2.HEX
MONITOR.HEX
CBIOS64.HEX
CBIOS128.HEX
CPM22.HEX
FORM64.HEX
FORM128.HEX
PUTSYS.HEX

Source code for the installation files
basic.asm
download.asm
monitor.asm
cbios64.asm
cbios128.asm
cpm22.asm
form64.asm
form128.asm
putsys.asm

DOS/Windows assembler
_ASSEMBLE.BAT
TASM.EXE
TASM80.TAB
TASM_RELNOTES.TXT
TASM.TXT

Acknowlegements
The BIOS, Format, Putsys and the file transfer programs are my own work.
CPM22 is a modified version found on the net. Originator copyright within the file.
Monitor is partially my own work, but using routines written by Joel Owens.
BASIC is Microsoft BASIC 4.7 for the NASCOM, heavily modified by myself to remove references to different monitors, screen handlers and keyboard matrix reading.
TASM assembler is a partial distribution of the package from Speech Technology Incorporated.

MEMORY MAPS

INITIAL BOOT

0000-3FFF ROM
  Within the ROM...
    0000 Boot monitor
    2000 Microsoft BASIC interpreter
4000-FFFF RAM

Once CP/M is loaded

0000-FFFF RAM

0100 Transient program area (applications)
D000 CP/M System
E600 BIOS

DISK DETAILS

Using a 64MB Compact Flash, eight drives are available.
The compact flash is not a full 64MB, so the drive sizes are:

A: to G: - 8MByte
H: - 5MByte

Using a 128MB Compact Flash, the full sixteen CP/M drives are available.
The compact flash is not a full 128MB, so the drive sizes are:
A: to O: - 8MByte
P: - 2MByte

The block size was chosen as a compromise between smallest possible, but without using too much RAM (because CP/M stores a complete allocation bitmap of all blocks, so smaller blocks would require more RAM to be allocated to each drive).
So, the chosen block size is 4096 bytes (*).
512 Directory entries for each drive was chosen as being suitable (*).
These can be changed within the source code, but require changes to the BIOS and also to the formatter.
The sizes that I have used have been fine for me, though, and I recommend they remain as-is, unless you are very familiar with the parameter tables.

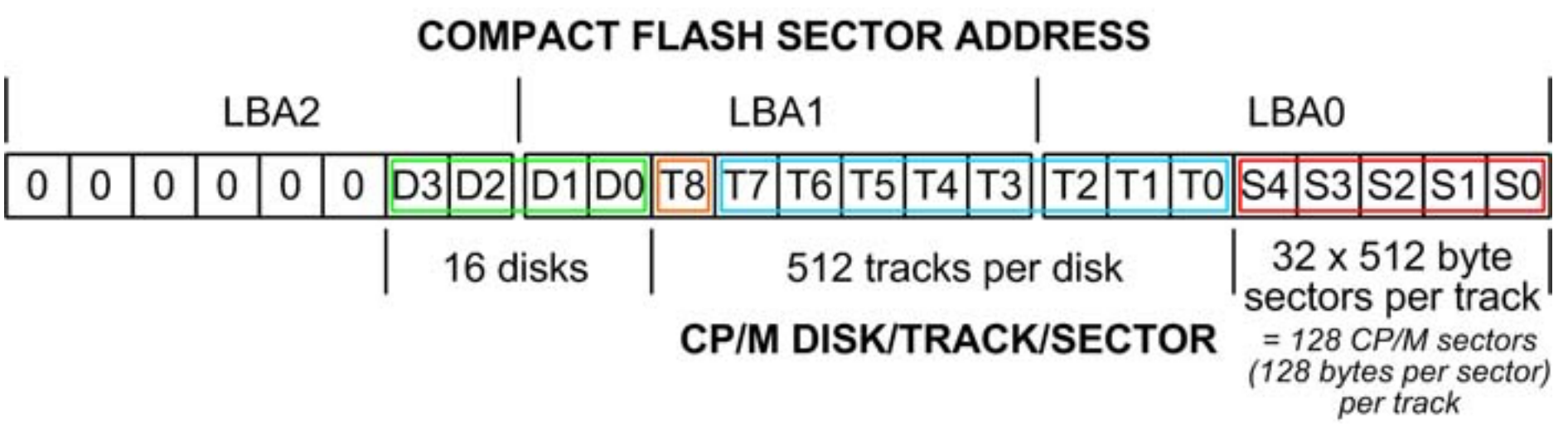
The standard CP/M sector size is 128 bytes, and I have used the published blocking/deblocking algorithms (by Digital Research) to allow the 512 byte sectors to be used (ie. 4 CP/M sectors are contained in each Compact Flash sector).
The CF is set up to use 8 bit transfers, so all bytes in a drive are available when using the 8-bit data bus. Additionally, the CF is set up as "IDE" mode so it only requires 3 address lines and one chip select. This method of operation also suits the disk operations performed by CP/M.

** - Each directory entry points to up to 8 blocks. For the values that I have chosen each block is 4096 bytes. So, if all files were 4K, a maximum of 2MB on a drive will be used (512 x 4 k files). However, if an average of 16K is assumed (each entry therefore pointing to 4 blocks) then the full 8MB will be used. This is largely academic, however, as you are very unlikely to fill either the disk capacity or directory entries.*
More directory entries would reduce the amount of free RAM available to the user. A smaller block size would also have the same affect.

Sector addressing

LBA addressing is used because not all cards have the same drive geometry (sectors per track). All CF cards support LBA. Using LBA, a contiguous sector address is used to address the complete disk.

The CP/M drive, track and sectors values are bit-shifted and transferred into the LBA addresses (higher LBA address bits set to zero) as shown below...



The shifting, splitting and combining of each of the values can be seen in the CBIOS code ("setLBAaddr").
This is the physical sector address. Each sector is 512 bytes and contains 4 x CP/M sectors (128 bytes each).
The CP/M track size is arbitrary, so 128 sectors per track seems a suitable value to use - this makes 32 "blocked" sectors per track.
To get 8MB disks, this therefore requires 512 tracks, so 9 bits are used for the track number.

IOBYTE details

Many thanks to Andrew Quinn for his help and support on this - most appreciated.

(If you don't know what the CP/M IOBYTE is then you can safely ignore this section!)

For full details on how to map logical devices to physical devices, please refer to the CP/M 2.2 manual available on the internet.

The CP/M IOBYTE (location 0003 in the memory map) is implemented to allow logical devices to be re-directed to the two physical ports.
The physical ports on this machine have been assigned to "CRT:" (the TTL serial port A) and "TTY:" (the RS232 serial port B).
For flexibility with all logical devices...
the CRT: port (TTL serial) is also assigned to physical devices UP1:, UP2:, UR1: and UR2:
the TTY: port (RS232 serial) is also assigned to physical devices LPT:, UL1:, RDR:, and PUN:

During startup you are asked to press the Spacebar on the window that is being used. This allows the machine to identify which serial port is connected to the user terminal. Depending on which of the serial ports is identified (CRT or TTY) the

IOBYTE is set appropriately.

So...
If you activate the terminal on the CRT port, the IOBYTE is set to 00000001b ie. List is TTY:, Punch is TTY:, Reader is TTY:, Console is CRT:
If you activate the terminal on the TTY (RS232) port, the IOBYTE is set to 00000000b ie. List is TTY:, Punch is TTY:, Reader is TTY:, Console is TTY:

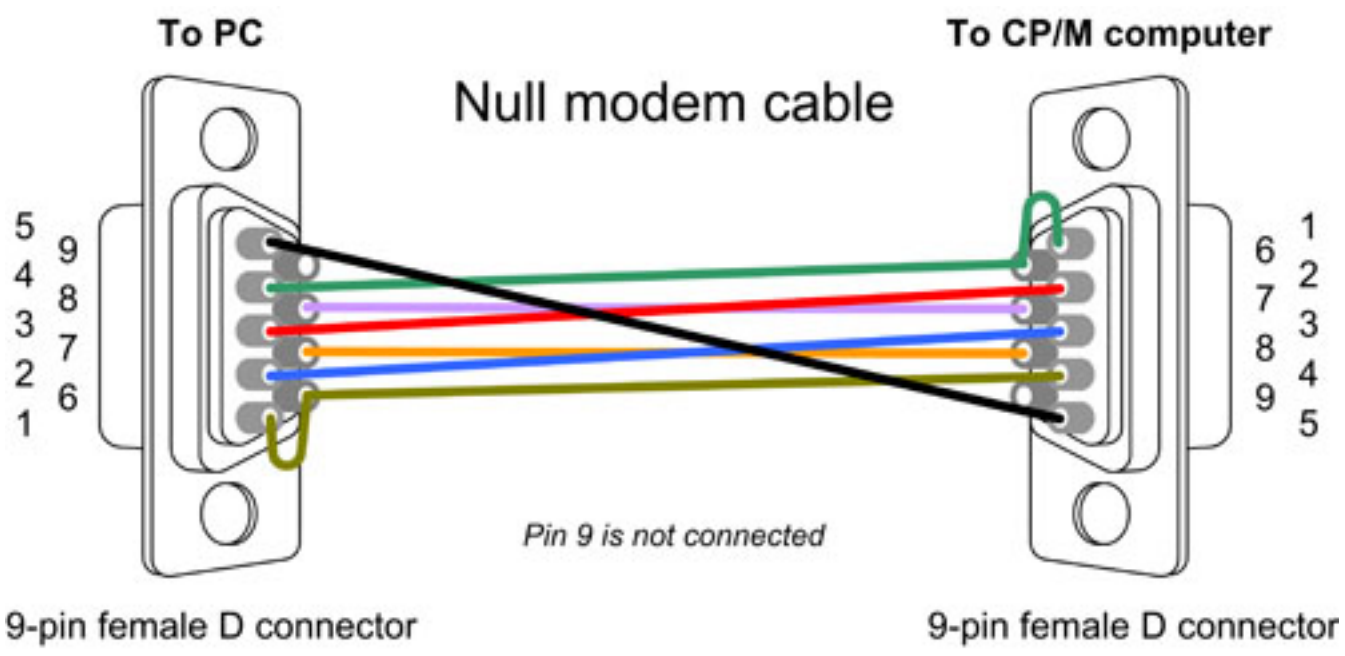
These can be changed dynamically as required.
eg. from the CP/M prompt, if STAT is installed...
STAT CON:=CRT: - to set the console to the TTL port
STAT CON:=TTY: - to set the console to the RS232 port
...see the CPM 2.2 manual for more details.
Alternatively, the I/O can be changed within, say, Microsoft CP/M BASIC by "poke"ing the IOBYTE (location 3) with a suitable value. This allows programs to utilise BOTH serial ports for input and output within the same program.

PIP and STAT use these physical and logical mappings. Other communications programs (eg. Kermit) will also work with them.

USING THE MACHINE

Connect the board to a 5V regulated supply capable of supplying at least 250mA (or 500mA if you have a PC keyboard connected).

A suitable cable needs to be made to connect the CP/M machine to a PC to allow the system to be transferred from the files supplied below. Here is a fully configured null modem cable that I use successfully:



.
The board is set up for 115200 Baud, hardware handshake, 1 stop bit and no parity.
Connect it to a PC or similar running a terminal program set to these parameters, then press the cold reset button.

```
Press [SPACE] to activate console
```

...is displayed on both serial ports. Only one is active at any time, so a spacebar press on the terminal will allow the board to determine which port is to be used for the console I/O.

Once spacebar is pressed, the following is displayed...

```
Z80 SBC Boot ROM 1.1 by G. Searle
Type ? for options

>
Typing ? will show
R - Reset
BC or BW - ROM BASIC Cold or Warm
X - Boot CP/M (load $D000-$FFFF from disk)
:nnnnnn... - Load Intel-Hex file record
>
```

BC
This starts the Microsoft BASIC ROM interpreter in "Cold" mode (ie. clears memory)

You will see

```
Memory top?
(press ENTER for full RAM usage)

Z80 ROM BASIC Ver 4.7b
Copyright (C) 1978 by Microsoft
48691 Bytes free
```

```
Ok
```

BASIC is then ready for programming

BW
This starts the Microsoft BASIC ROM interpreter in "Warm" mode (ie. doesn't clear memory, so a program already in memory will remain)

X
Boots CP/M
You will see

```
Boot CP/M?
```

displayed. Press "Y" to allow CP/M to be loaded from the disk. This will also turn off the boot ROM, allowing full 64K RAM access to CP/M.
If a properly initialised disk is present, the following is then displayed...

```
Loading CP/M...
screen clears here, so you may not see the above line
Z80 CP/M BIOS 1.0 by G. Searle 2007-13

CP/M 2.2 Copyright 1979 (c) by Digital Research

A>
```

CP/M is then loaded and ready to use.

INSTALLING CP/M

Installing CP/M onto a new card is a straightforward process, and involved firstly formatting the card then transferring the CP/M system to the first sectors on the card.
The board will need to be connected to a PC (or whatever) which has a terminal program and contains the ".HEX" files needed for the installation.
Reset the machine, and press spacebar, as described above. The machine is then ready to load Intel-Hex files if they are transferred from the terminal.
The following steps require Intel-Hex files to be loaded. To do this, open the ".HEX" file in something like notepad, select the whole contents (CTRL-A) then "Copy" to clipboard. In the terminal, paste the contents into the window - the board will automatically load them to the appropriate locations specified in the HEX file. Dots will be shown on the terminal for each byte loaded. Once the end of the hex file is reached, "File loaded" will be displayed.
If for any reason something fails, simply cold-reset the machine and repeat the steps.

PART 1. FORMATTING THE DRIVE

Load the Intel-HEX dump of FORMAT into memory by copying the contents of FORM64.HEX (for 64MB Flash drives) or FORM128.HEX (for 128MB or larger drives) into the terminal window.
FORMAT resides at $5000 when loaded.
Executing address $5000 will start the FORMAT program which will then initialise the directory structures (ie. a quick format, as that is all that is needed on a compact flash card) for ALL of the logical drives on the flash card.

Once loaded, type **G5000** then press ENTER.

You will see...

```
CP/M Formatter by G. Searle 2012
Then the following will appear in sequence (only takes a few seconds)
ABCDEFGHIJKLMNOP
Finally
Formatting complete

>
```

Each drive is 8MB (except for the last drive on each card), so a 64MB card will have drives A: to H:, and a 128MB card will have drives A: to P: on one card.

PART 2. "PUTSYS"

CP/M is "installed" on the first track of a disk. When the disk is then booted, the first track is read into the correct area of memory then executed.
To write the CP/M system to the disk, it is firstly loaded into memory, then a program is executed to write the memory to the first sectors on the flash disk.

To install, the following steps are needed in this order:
1. Load the Intel-HEX dump of CP/M into memory by copying the contents of CPM22.HEX into the terminal window. This will take about 10 seconds to load.
2. Load the Intel-HEX dump of CBIOS into memory by copying the contents of CBIOS64.HEX (for 64MB Flash drives) or CBIOS128.HEX (for 128MB or larger drives) into the terminal window. This takes about 3 seconds to load.
3. Load the Intel-HEX dump of PUTSYS into memory by copying the contents of PUTSYS.HEX into the terminal window.

PUTSYS resides at $5000 when loaded.
Executing address $5000 will start the PUTSYS program which will copy the CP/M and CBIOS from memory to the flash drive.
Type **G5000** then press ENTER.

You will see...

```
CP/M System Transfer by G. Searle 2012

System transfer complete

>
```

Once these are complete, CP/M is then installed on the disk and can be booted. Applications will then need to be installed.

INSTALLING APPLICATIONS

To install files, a small client application needs to be installed on the A: drive on the board. Once this is done, the windows program to package the files is used to create a text file which is then pasted into the terminal window.

PART 1 - The CP/M client application installation

The installation of the client application only needs to be done once (provided it is not deleted) on each flash card.

1. Install CP/M as show above then boot from CP/M (press X and then Y at the monitor prompt, as described above).
2. Press the cold reset button to get back to monitor
3. Paste DOWNLOAD2.HEX (a relocated DOWNLOAD.HEX) to terminal window. This will load the download client into high memory. This normally resides at 0100, but as the ROM is active, this area is not available.
4. Type **GFFE8** and press ENTER (this will relocate RAM 4100-->0100 and restart CP/M)
6. When in CP/M, **SAVE 2 DOWNLOAD.COM** and press ENTER

A full terminal session of this is shown here...

```
<<<PRESS COLD RESET>>>
Press [SPACE] to activate console
<<<PRESS SPACE>>>
Z80 SBC Boot ROM 1.1 by G. Searle
Type ? for options

>x <=== Type char
Boot CP/M?
<<<PRESS Y>>>
Loading CP/M...
Z80 CP/M BIOS 1.0 by G. Searle 2007-13

CP/M 2.2 Copyright 1979 (c) by Digital Research

A>
<<<PRESS COLD RESET>>>
Press [SPACE] to activate console
<<<PRESS SPACE>>>
Z80 Monitor
Type ? for options
<<<PASTE THE DOWNLOAD2.HEX FILE INTO THE TERMINAL WINDOW>>>
>................................
>................................
>................................
>................................
>................................
>................................
>................................
>................................
>................................
>................................
>................................
>................................
>Load complete.

>GFFE8    <=== Type chars and press ENTER
Screen clears here
Z80 CP/M BIOS 1.0 by G. Searle 2007-13
```

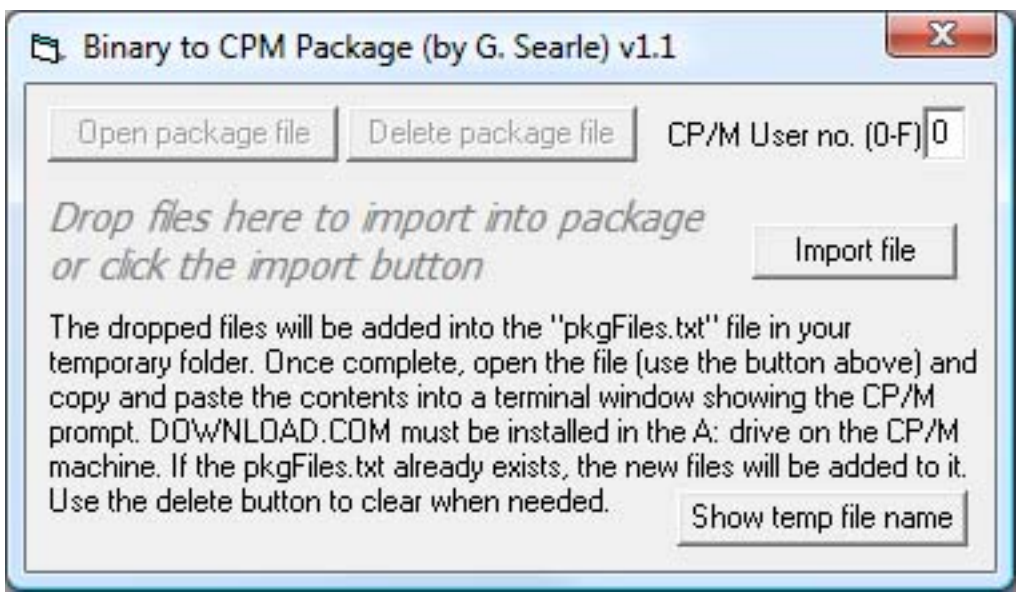
```
CP/M 2.2 Copyright 1979 (c) by Digital Research

A>SAVE 2 DOWNLOAD.COM    <=== Type chars and press ENTER
A>
```

DOWNLOAD.COM is now present on the A: drive, and the machine is now ready to be loaded with application files. There is no need to repeat the above steps unless the DOWNLOAD.COM file is deleted from the A: drive.

PART 2 - Using the Windows packager program

This can be used as many times as required and can transfer single or multiple files into the current drive that is active in the terminal window.

This is a small Microsoft Windows application that will allow BINARY CP/M files to be dropped onto it, which will then add them to a package text file. If you prefer, you can use the "import" button to select a file instead of dropping files onto the window.



NOTE: The "pkgFiles.txt" file will be created in the temporary folder of your machine. Only limited error handling has been added to this program, so if it fails please re-run.
Once all files that are to be transferred into a particular drive have been dropped, the package file can then be opened.
Copy the COMPLETE contents of this file and paste it into the terminal window of the machine.
The terminal must be showing the CP/M prompt. This can be on any drive. The packaged files will be created in the current drive on the CP/M machine.
The "DOWNLOAD.COM" file on A: drive on the CP/M board will then start and read the HEX byte stream, storing it in a file.
Multiple files are permitted in a package file, allowing you to transfer a complete application containing many files in a single operation.
Partial contents of this file are, as an example...

```
A:DOWNLOAD BBCBASIC.COM  <==== Command to create a file BBCBASIC.COM in the current drive
U0  <==== User selection
:C33401C37B01C3A501C3CA01C37A01C37301C38501C37201DDE5F  <==== Data stream, starts with a colon
CB5FD21003CCA120FC5E5CDD209E1CDD50ACC9209C179B7CA990  <==== Data stream
36F7E0>0085  <==== Data stream, ends with a greater-than symbol, followed by a checksum
A:DOWNLOAD STAT.COM  <==== Command to create second file STAT.COM in the current drive
U0  <==== User selection
:C33304202020436F7079726967687420245763634 56E2342EEA2  <==== Data stream for second file, starts with a colon
B0E0ECD4C14C9212A15702B712A2915EB0E015CD3805C6414F  <==== Data stream
A1A1A1A1A1A1A1A>80DD  <==== Data stream, ends with a greater-than symbol, followed by a checksum
```

An example terminal session showing these files being loaded is shown here...

```
Z80 CP/M BIOS 1.0 by G. Searle 2007-13

CP/M 2.2 Copyright 1979 (c) by Digital Research

A>C:  <=== Select drive where files are to be loaded and press ENTER
<<<PASTE CONTENTS OF PKGFILES.TXT>>>
C>A:DOWNLOAD BBCBASIC.COM <== Start of pasted text
.........................................  <=== Import progress of first file
.........................................
OK   <=== Successful import of a file completed
C>
C>A:DOWNLOAD STAT.COM
.....................................  <=== Import progress of second file
OK   <=== Successful import of second file completed
C>
C>DIR    <=== A DIR command will show the created files
C: BBCBASIC COM : STAT COM
C>BBCBASIC   <=== Running one of the imported files
BBC BASIC (Z80) Version 3.00
(C) Copyright R.T.Russell 1987
>
```

You can, of course, install your own transfer program using the above then use that.

The format of my transfer files is simple so you can also create your own program to build the transfer text files if you don't want to use my Windows app. You need to create a text file with the following contents (also refer to my example above)

First line:
A:DOWNLOAD <filename> *where <filename> is the CPM file that is required. Must be a space between DOWNLOAD and the filename.*
Second line:
U0 *This is the "user" number that can see the destination file. If you are not familiar with CP/M users, then always use U0*
Third line:
A continual stream of HEX values *(must be 2 chars each) for each byte in the file, followed by a* ***>*** *character, followed by the* ***checksum***

Each line is terminated with a carriage return / linefeed ($0D, $0A) as standard for text files

The checksum is two pairs of HEX values:
The first one is the low-byte of the length of the file being uploaded. Because CP/M files are normally saved in blocks of 128, this is normally "80" or "00", but doesn't need to be.
The second one is the low-byte of the SUM of each byte being uploaded.
The checksum is very simple, but more than adequate for confirming a simple data transfer, as it can detect missing, extra or changed characters.

If multiple files are to be transferred, then all can be included in the same text file immediately after one another, with each following the three line format shown above.

ROM BASIC (Microsoft BASIC 4.7) - details of what has been included/excluded

Update 8th September 2013 (Boot ROM 1.1) -
1. The HEX and BINARY identifiers have changed to &Hnnnn and &Bnnnn to match Microsoft implementations for other processors.
2. Width setting changed to 255 default, so no auto CR/LF are added when printing long strings.
3. HEX$(nn) and BIN$(nn) now have leading zeroes suppressed, as for other numeric output.
4. CR/LF processing changed slightly internally so that an LF is no longer auto-generated when sending a CR to the terminal.

INCLUDED TOKENS

SGN,INT,ABS,USR,FRE,INP,POS,SQR,RND,LOG,EXP,COS,SIN,TAN,ATN,PEEK,DEEK,LEN,STR$,VAL,ASC,CHR$,LEFT$,RIGHT$

END,FOR,NEXT,DATA,INPUT,DIM,READ,LET,GOTO,RUN,IF,RESTORE,GOSUB,RETURN,REM,STOP,OUT,ON,NULL,WAIT,

DEF,POKE,DOKE,LINES,CLS,WIDTH,MONITOR,PRINT,CONT,LIST,CLEAR,NEW

TAB,TO,FN,SPC,THEN,NOT,STEP

+,-,*,/,^,AND,OR,>,<,=

Note: there is also SET,RESET,POINT that call user-defined entry points, as in the ORIGINAL Nascom ROM (ie. not changed). Don't use unless you have defined the calling points for them (see the assembly listing for details).

EXCLUDED TOKENS (don't do anything if called)

SCREEN,CLOAD,CSAVE

NEW (my additional implementations)

HEX$(nn) - convert a SIGNED integer (-32768 to +32767) to a string containing the hex value
BIN$(nn) - convert a SIGNED integer (-32768 to +32767) to a string containing the binary value
&Hnn - interpret the value after the &H as a HEX value (signed 16 bit)
&Bnn - interpret the value after the &B as a BINARY value (signed 16 bit)

IMPORTANT NOTES

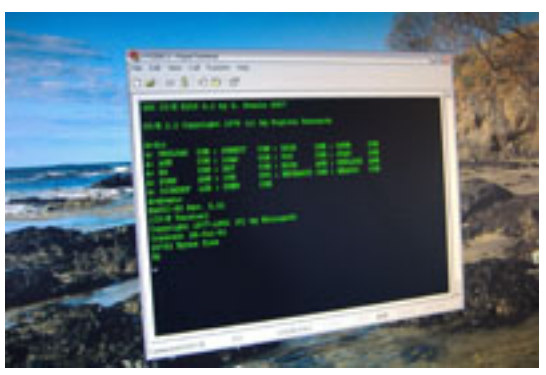
Integers in this version of BASIC are SIGNED - ie. -32768 to +32767. This includes memory locations when peek, poke, deek,doke commands are issued (etc.)
So, to refer to location "n" above 32767, you must provide the 2's compliment number instead (ie."n-65536") otherwise you will get an "?FC Error" message.

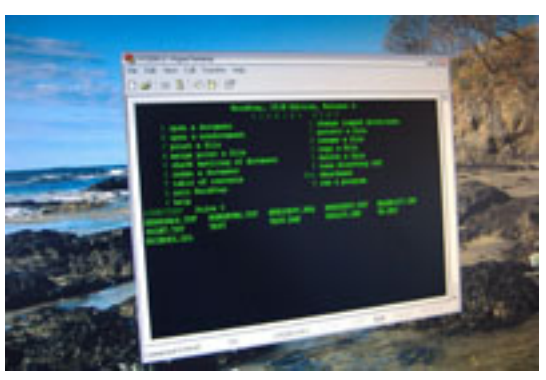
Functions that return integer values (such as the memory FRE(0) are also **signed** (!)) so anything bigger than 32767 will appear as a negative value.

TERMINAL SCREENSHOTS

(Click on the images to see enlargements)

Here is this machine running on a monochrome (green) monitor using the monitor interface described later on this page:


Here is the command prompt of my computer running using a Windows terminal.


And here is the infamous Wordstar (version 4) running on my machine in a Windows terminal.

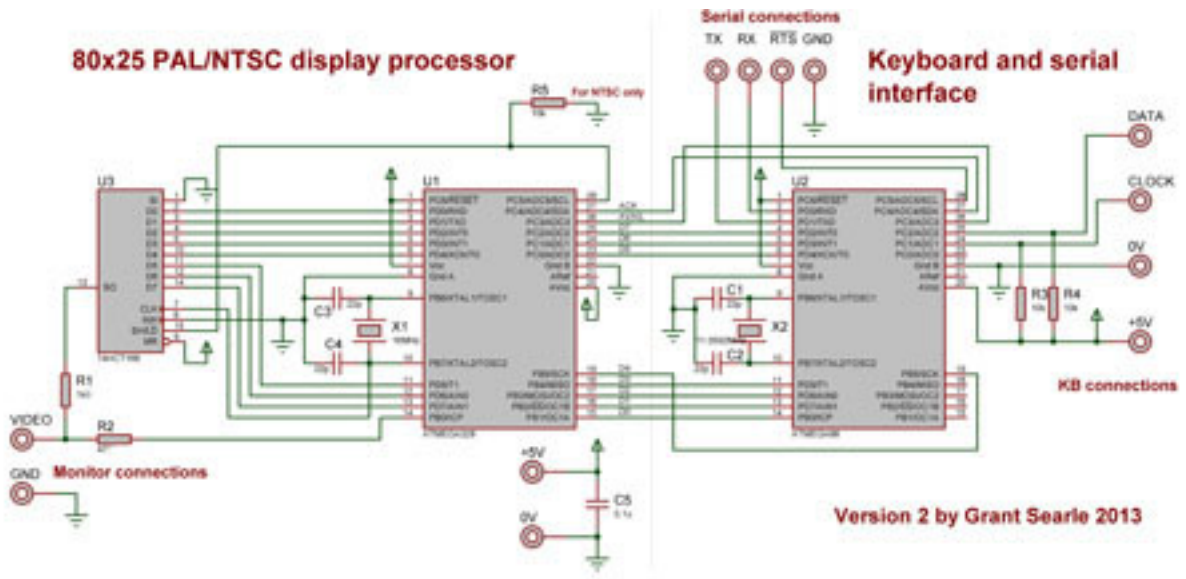
MONITOR AND KEYBOARD INTERFACE

This is a generic 80 column x 25 line ANSI terminal using a TTL compatible serial interface running at 115200 baud (can be changed in software) so can be used for any other computer project that has a serial I/O and needs a keyboard and display.

VERSION 2 - Now uses an ATmega328 processor instead of the older ATmega32. The ATmega328 is a cheaper, newer and smaller processor (28 pin instead of 40 pin).

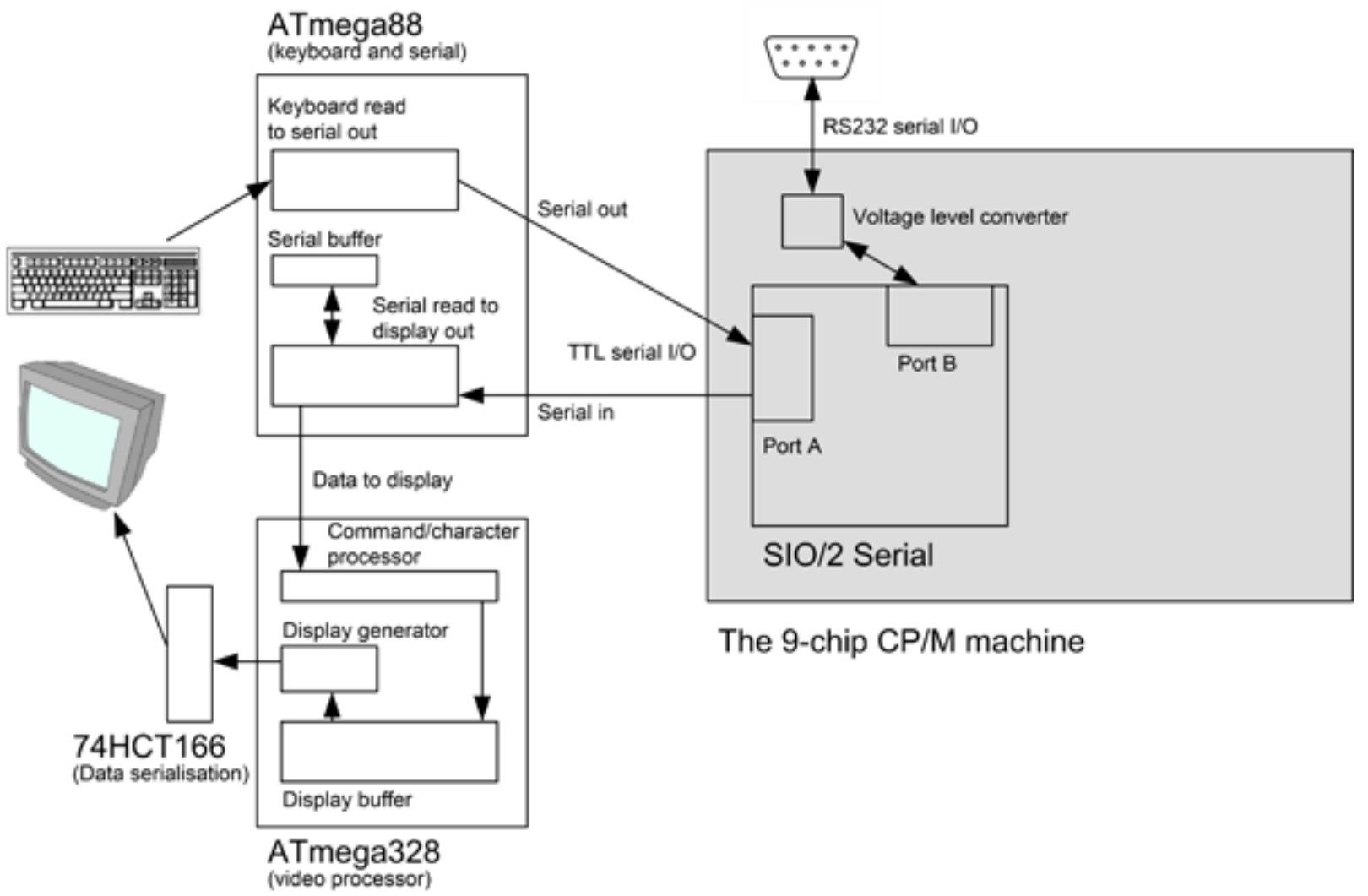
Details of the interface has been moved to its own page HERE, as it is suitable for other systems in addition to the one show above,

This uses an ATmega88 for the keyboard and serial buffer and an ATmega328 for the display processor.
Since the display is independent of the Z80 processor, the Z80 has no processing overhead so this CP/M system runs much faster than many of the old systems.



Note: The standard input handler in CP/M is 7 bit, so any codes that are passed to the CP/M machine that are higher than 127 have their 7th bit reset to 0.

Here is a block diagram of the display and keyboard interface when connected to the CP/M machine:



To connect to the main CP/M machine you will need to make the following connections...
1. Vcc to +5V of the main machine
2. Gnd to Gnd of the main machine
3. /RTS to /CTSA on the SIO/2 (Pin 18)
4. TXD to RxDA on the SIO/2 (Pin 12)
5. RXD to TxDA on the SIO/2 (Pin 15)

Connect a keyboard with a PS/2 connector and connect the Video out to a suitable video input on a monitor or TV ("AV Input")

Power up and press reset and you will see the "Press [SPACE] to activate console" appear on the top of the display.

CLICK HERE FOR DETAILS

LINKS

Some of my pages

INDEX PAGE - Go here for access to all of my pages along with my eMail details
Build your own ZX80 - my page showing you how to build this old micro
  |__ ZX80 to ZX81 conversion - build the NMI generator needed to convert the ZX80 circuit into a ZX81
  |__ ZX80 software - Type in a Space Invaders game into the ZX80
Build your own Jupiter Ace - my page showing you how to build this old micro
Build your own UK101 - my page showing you how to build a greatly simplified version of this old micro
A simple 6809 machine - can't get simpler than this for a machine that can run Microsoft Basic
CP/M with 9 chips - build a fully functional CP/M machine with a minimal chip count and CF card storage
Pong - Pictures of my build of the Atari classic arcade game
Practical Wireless Tele-Tennis - A rebuild of the 1974 articles for this home Pong game
My Machines - My collection of classic 80's micros
My TV games - my collection of "pong" style games from the 70's
My electronics kits - my collection of electronics kits from 60's, 70's and 80's

---

Grant.

Back to main menu