

Grant's 7-chip Z80 computer

(only 6 chips if using a USB to TTL serial cable)

- a fully operational Z80 computer running BASIC can't get simpler than this!



by Grant Searle

[For news and updates, follow me on Twitter:](#)

[Follow](#)

Last update: 12th October 2017

[\[FOR A 6809 VERSION OF THE MINIMAL COMPUTER, CLICK HERE\]](#)

[\[FOR A 6502 VERSION OF THE MINIMAL COMPUTER, CLICK HERE\]](#)

[\[FOR A VERSION ON A LOW-COST FPGA BOARD CLICK HERE\]](#)

[\[FOR A SLIGHTLY MORE COMPLEX Z80 VERSION RUNNING CP/M, CLICK HERE\]](#)

Please note that you are NOT allowed to reproduce any of this page elsewhere on the Web without my permission.

Index

[Specification](#)

[Memory and I/O map](#)

[Circuit diagram](#)

[Circuit description](#)

[Prototype](#)

[ROM assembly and hex files](#)

[ROM BASIC - what's included/excluded and new](#)

[Powering up](#)

[Loading and saving](#)

[Interfacing](#)

Specification

8K ROM

56K RAM (A version with 32K RAM is [HERE](#))

Z80 Processor (overclocked - all processors 4MHz+ (Z80B) that I have tried overclock to this with no issues) with a 7.3728MHz clock. You can halve this, ie. to stop the overclocking, by changing the crystal to 3.6864MHz but the serial I/O speed is then also halved to 57600 baud.

115200 Baud serial interface, RS232 specification voltage levels. Full interrupt driven input with buffer and hardware handshaking so no incoming data loss.

Power consumption - approx 200mA

Microsoft BASIC, as used in the Nascom 2 computer modified for the SBC with all I/O via serial. Commands not applicable for the SBC have been removed.

Minimal possible component count - 7 ICs and a small number of discrete components.

Easily interfaced to a large number of connections by adding input ports or output ports and a very small amount of logic.

Memory Map

0000-1FFF 8K ROM

2000-FFFF RAM (56K) (32K RAM modifications [HERE](#))

I/O Map

00-7F Free (128 input and 128 output ports)

80-81 SERIAL INTERFACE (minimally decoded, actually covers locations 80 to BF)

Circuit diagram

(Note: Circuit and other details for the 32K RAM version is [HERE](#))

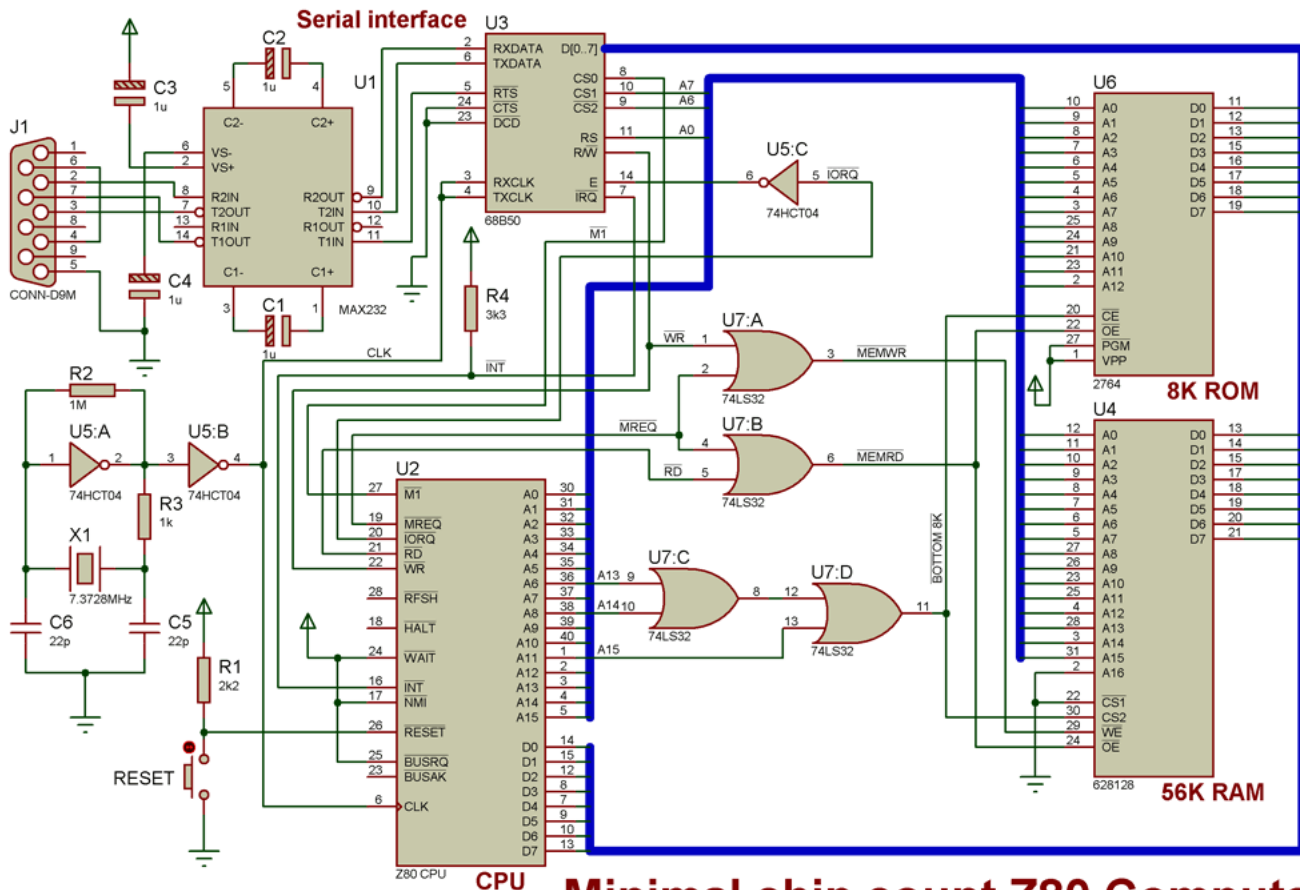
The purpose of this computer is to create the simplest possible machine with a high speed interface, good amount of RAM and also a good implementation of BASIC.

The design that I produced is shown here, and is probably the simplest Z80 circuit that can be done to fulfil what I need. This can be used as the basis of more complex machines.

The ACIA is very minimally decoded but still leaves an I/O address space of 192 free input ports and 192 free output ports (ie. 1536 input bits plus 1536 output bits) available so would be more than adequate. If a more refined decoding is needed then it is straightforward to add additional gates or a 74LS138 decoder. However, even with this simple arrangement, there are many I/O addresses free for expansion.

Note: Chip selection - All of the standard Z80 and 6850 chips that I have worked perfectly at the speeds required for this circuit even though the circuit requirement is faster than their specification. Therefore, you are unlikely to have any issues. However, I would recommend you buy the "B" speed grade ACIA ie. 68B50 and get the 6MHz or 8MHz version of the Z80 processor.

Power supply pins and any (optional) decoupling capacitors are not shown and need to be connected to the appropriate power rails.



Minimal chip count Z80 Computer by Grant Searle 2007

Note: /CTS is not used and is grounded on the 6850 as I didn't want this board to stop running (sending serial data) if the PC is disconnected. PCs are also plenty fast enough to keep up with the serial output from this board so handshaking isn't really required. However, if required, this can be connected to the serial interface via the MAX232 chip (R1OUT to /CTS on the 6850 and R1IN to the serial port pin 8). A MAX202 could be used instead of the MAX232, in which case C1-C5 should be 0.1uF.

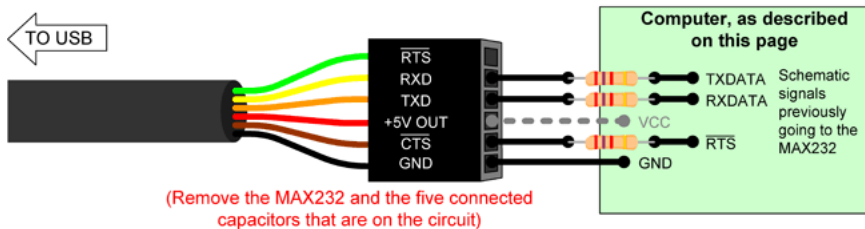
Serial port pinout (standard 9-pin):

- (IN) - Data Carrier Detect (DCD) - not used
- (IN) - Receive Data (RX) - data from PC to this board
- (OUT) - Transmit Data (TX) - data from this board to the PC
- (OUT) - Data Terminal Ready (DTR) - connected to pin 6 to show the board is ready whenever PC is ready (common practice)
- Signal Ground
- (IN) - Data Set Ready (DSR) - connected to pin 4 - see pin 4
- (OUT) - Request To Send (RTS) - used to control the data from the PC to this board
- (IN) - Clear To Send (CTS) - not used, but can be if required (see above)
- (IN) - Ring Indicator (RI) - not used

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



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.

Circuit description

The crystal oscillator is quite a standard design with the second inverter in it used as a buffer to ensure the crystal operation is not affected by the connected circuitry.

A Z80 memory read is determined whenever the /MREQ pin goes low when the /RD pin or /WR pin is also low. Two OR gates are used to combine the memory access (/MREQ) and the read/write signals (RD/WR) to produce "memory read" and "memory write" to simplify the interfacing to the ROM and RAM. These are then connected to the appropriate pins on the ROM and RAM devices.

Address decoding using A13, A14 and A15 will identify the lower 8K (low when A13, A14 and A15 are all low). This is connected to the active-low chip select of the ROM and also to the active-high chip select of the RAM. This way, either the ROM will be active (lower 8K) otherwise the RAM will be active (the remaining 56K). The entire 64K memory space is therefore used.

The serial interface is accessed using the Z80 I/O addressing mechanism. When accessing I/O devices, the /IORQ signal is taken low along with /RD or /WR. The 6850 does not follow this convention, however, as it uses a single Enable (E) pin, along with a single R/W pin to identify whether a read or write is to be performed. The R/W pin is taken directly to the /WR pin of the Z80 and the Enable pin is taken to an inverted signal from the /IORQ.

A problem exists, however, as the /IORQ is also taken low during an interrupt acknowledge. If this was allowed, then the data/control values in the serial interface would be corrupted when the interrupt is acknowledged by the Z80. Interrupt acknowledge is identified by /IORQ and /M1 signals going low, so the 6850 is to be disabled if /M1 is low (the Z80 /M1 signal connected to the active-high CS0 of the 6850). This ensures that only real I/O triggers the serial interface. The /M1 signal is much wider than the /IORQ and totally masks the /IORQ signal. This ensures that there is no possibility of the ACK signal enabling a read or write. The remaining chip selects on the serial interface are connected to A7 and A6, so any I/O on address 10xxxxxx will select the serial chip. Finally, the register select on the serial interface is connected to A0, so the control register is addressed on port 80H and the data register is addressed on port 81H.

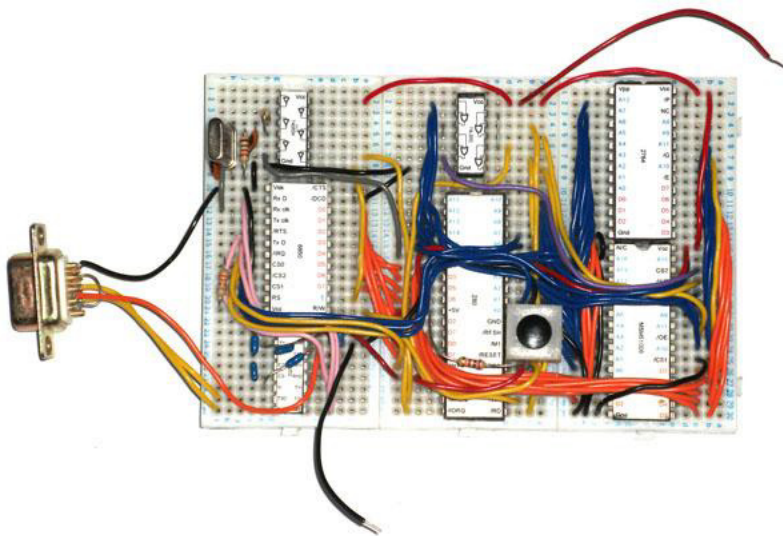
The remaining chip is a MAX232 (or MAX202) which is a level converter to convert the TTL signals to RS232 values.

/RTS is passed from the 6850 to the serial interface to signal to the PC when to stop transmitting.

/CTS is not used and is grounded on the 6850 (see circuit note, above).

Prototype

The circuit shown on this page is built here. As you can see, I have attached labels (download [HERE](#)) to the chips to allow the wiring to be very straightforward and error-free. Blue wiring is the address lines, orange is the data lines, red and black are power lines and various colours for the other connections.



The serial port is on the left, and the reset button is shown on the right hand side of the Z80 CPU.

ROM files and assembly listings

Update 8th September 2013 - ROM BASIC updates. See the below for details.

Update 10th September 2013 - Cold/warm start no longer offered as an option for the first-time reset after power-up because cold is the only valid option. Thanks to Dave Curran for this suggestion.

This is the same BASIC that I use in the ROM on the CP/M computer. Originally from a Nascom computer, modified to remove all hardware-specific code.

I then removed all code and commands that were specific to the screen, graphics, keyboard and sound.

I wrote a serial handler to control the text I/O, along with suitable Control-C break handling.

Unused variable locations were removed.

The command/function tables were updated as needed for the updated command set, and pointer values adjusted accordingly.

[Here](#) is the zip containing the following:

Source files

intmini.asm - the interrupt driven mini startup program needed to boot into BASIC

basic.asm - BASIC 4.7b - a conversion of Microsoft BASIC 4.7, as used on the Nascom computers (see below for details)

Output files

INTMINI.HEX

BASIC.HEX

ROM.HEX - the complete 8K ROM ready for burning to an EPROM - the unused contents are filled with FF values.

Within the ROM, the serial handler is first (starting at address 0000H), followed by the BASIC interpreter (starting at 0100H).

Assembler files (for Windows/DOS)

_ASSEMBLE.BAT - double-click in Windows to run the assembly if needed

TASM.EXE

TASM80.TAB

Acknowledgements:

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.

(Note: files for the 32K RAM modifications are [HERE](#))

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

Update 8th September 2013 -

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\$,MID\$

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

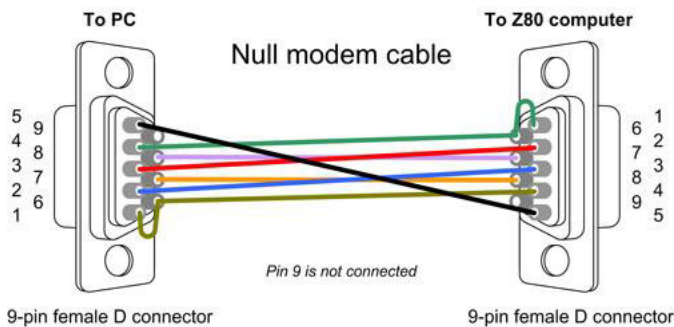
Integers in this version of BASIC are SIGNED - ie. -32768 to +32767. This includes memory locations when peek, poke, deek,doke commands are issued (etc.)

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.

Powering-up

You will need a suitable connection to a PC or other terminal display. You will probably need a null-modem cable to connect this to a PC serial port. Here is my recommended wiring.



Connect to a PC or similar terminal, with the following details (when using a 7.3728MHz crystal):
115200 baud, 8 bits, hardware (RTS/CTS) handshake, no parity, 1 stop bit.

Power on the board and press the RESET button.

You will see

Z80 SBC By Grant Searle

If BASIC was already started (ie. reset has been pressed after initial startup) then you will now see the following...

Cold or warm start (C or W) ?

Press C for normal cold start. (Press W if you want to resume a program previously entered in memory).

If this is the first-time reset after power up then the cold/warm option will not be presented, as only cold start is the valid choice.

You will now see the following...

Memory top?

Press ENTER to use full memory, otherwise enter the value that you require.

After a few seconds (memory test) the following will appear on the terminal:

```
Z80 BASIC Ver 4.7b
Copyright (C) 1978 by Microsoft
56958 Bytes free
Ok
```

The machine is now ready to use.

Note: typing

PRINT FRE(0)

will give (current ROM version, may be slightly different with future releases)...

-8578

Ok

This shows that the 56K RAM has been recognised correctly.

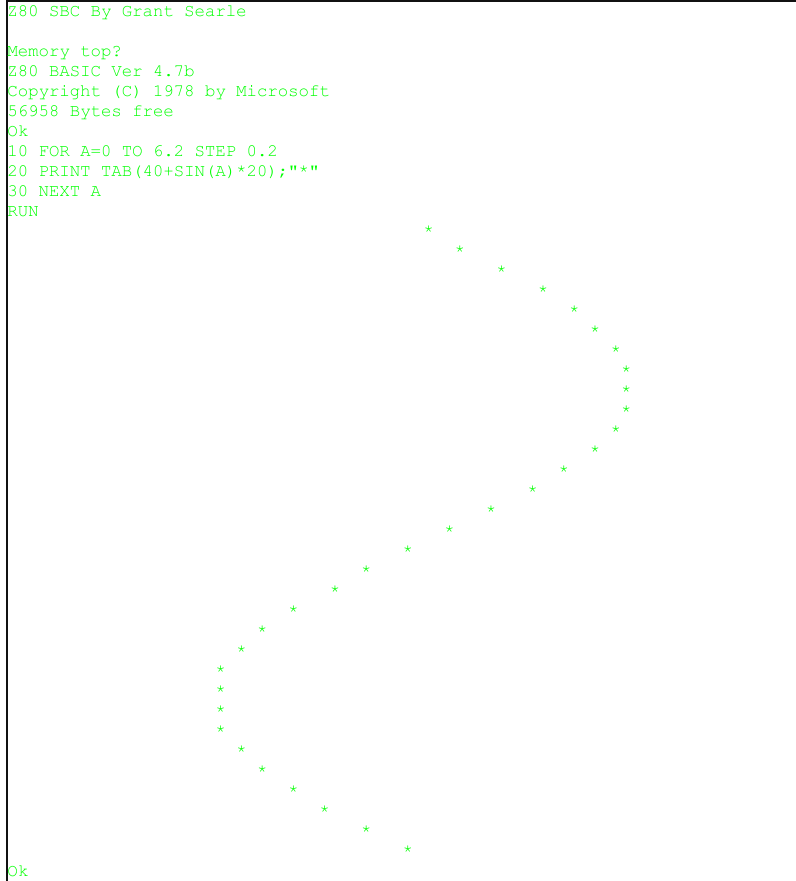
Note: The integer values are SIGNED, so this is actually 65536-8578 = 56958

...the rest is up to you.

A simple program entered and run on the terminal is shown here...

```
Z80 SBC By Grant Searle

Memory top?
Z80 BASIC Ver 4.7b
Copyright (C) 1978 by Microsoft
56958 Bytes free
Ok
10 FOR A=0 TO 6.2 STEP 0.2
20 PRINT TAB(40+SIN(A)*20); "*"
30 NEXT A
RUN
```



```
Ok
```

Loading and saving

Save

Type LIST but don't press enter.

In your terminal window on the PC, or whatever you are using to operate this board, enable the capture to a text file.

Press ENTER.

The listing will then be spooled to the text file on your PC.

Once the listing is complete, end the capture. The contents of the program will then be on the PC for later use.

Load

Most terminal programs have a method to pass a text file to the destination.

Ensure the board is ready to receive characters and then transfer the text file from the PC to the board using the terminal.

You can transfer the text at full speed - the interrupt-driven code will capture the data with no data loss. The handler will also control the hardware handshaking (RTS) to ensure the sender waits until the BASIC interpreter is ready for it.

Interfacing

The Z80 family includes several ICs suitable for interfacing to the Z80 bus to allow parallel or serial data I/O. These don't have to be used, however. For outputs, a 74HCT374 latch is suitable (provides 8 output bits) and a 74HCT245 can be used for an 8-bit input port. The circuit below shows how these can be used.

It is important to use 74HCT devices because:

1. They have extremely small input current requirements so many can be attached to the Z80 buses without any noticeable loading - no buffer ICs are necessary. There should be no problem in attaching a large number of devices providing the bus lengths are kept as short as possible. Quite a reasonable length can be achieved with no instability, though (see my picture below)
2. The outputs are virtually rail-to-rail (ie. 0V or 5V) unlike the 74LS devices.
3. The outputs have strong high and low currents so can drive (for example) LEDs connected to either ground or Vcc easily (must use current limiting resistors, as is normal for LEDs).

An example

Here is an example of a very simple way of having a very large number of input and output connections (32 inputs and 32 outputs) added to the circuit described above. This should be plenty for virtually ALL interfacing requirements. If not so many bits are required then only use the number of 74HCT245 and 74HCT374 chips that are required.

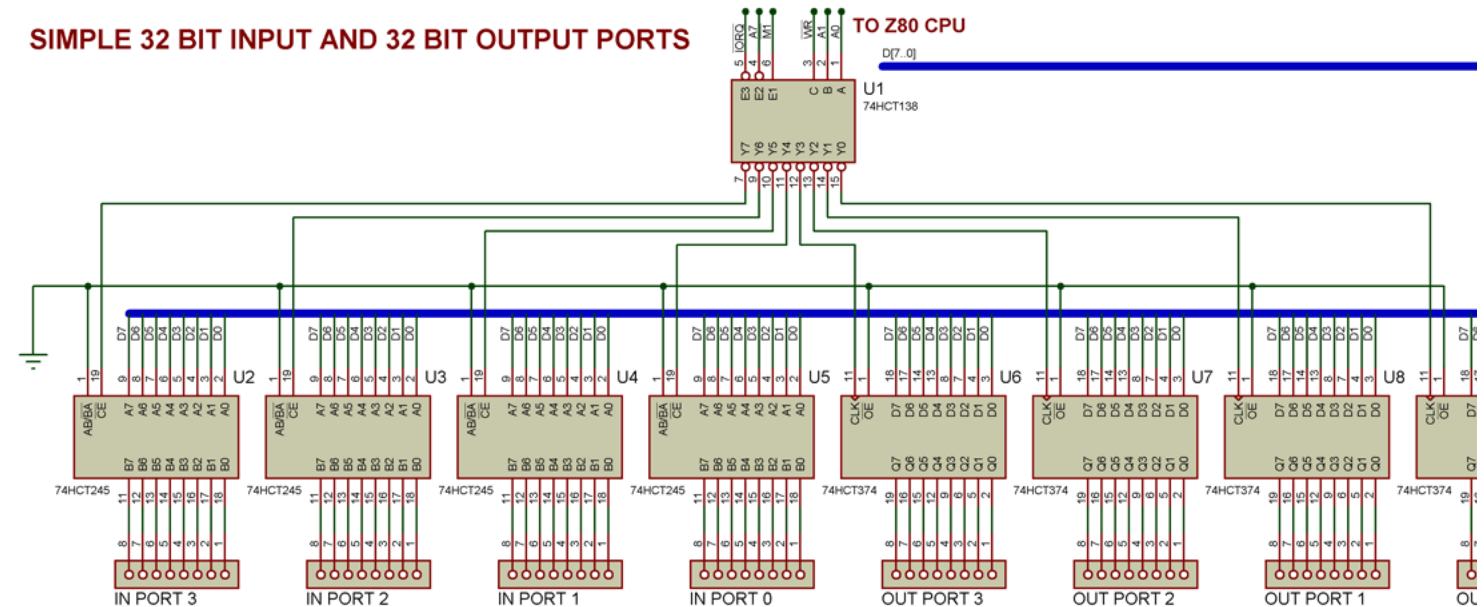
A 74HCT138 is used to provide the logic needed to activate 4 input ports and 4 output ports. No other decoding is necessary. The /WR line is connected to the high address selection on the '138 so that when /WR is low, Y0 to Y3 are selected (the output port select pins) and when /WR is high (ie. /RD would be low) Y4 to Y7 would be selected (the input port select pins). A0 and A1 is used to select which of the input or output ports are to be selected. /M1 is connected to an active high pin to prevent "ACK" signals triggering I/O, as described for the 6850 earlier in this page. A7 is connected to an active low input so any I/O with address 0xxxxxxx will address this circuitry. /IORQ is connected to an active low input to allow Z80 I/O to activate the ports.

Therefore...

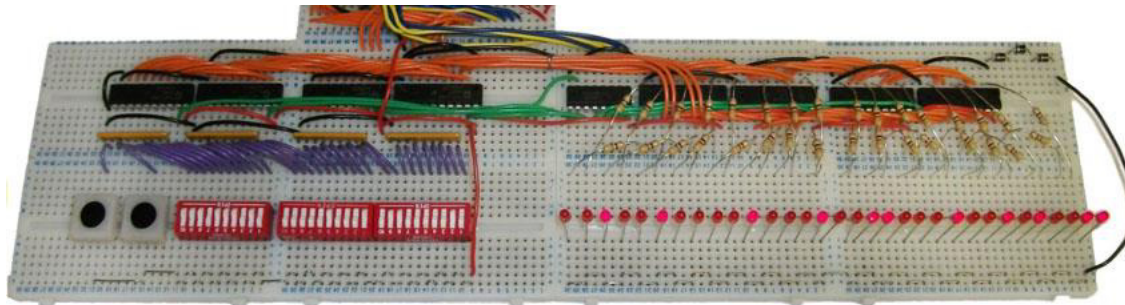
I/O address 0xxxxx00 = input/output port 0
 I/O address 0xxxxx01 = input/output port 1
 I/O address 0xxxxx10 = input/output port 2
 I/O address 0xxxxx11 = input/output port 3

This example still uses **minimal** decoding, so takes up a large number of spaces in the I/O address range. If needed, further address decoding can be added. However, as this provides a large number of input and outputs, it may not be necessary to free up any more I/O space as this may be sufficient for all the interfacing that is required.

Here is the circuit:



Here is my example implementation. I have 32 input switches (two pushbuttons and 30 DIP switches) and 32 LEDs.



Orange wires are the data bus, green wires are the chip select connections.

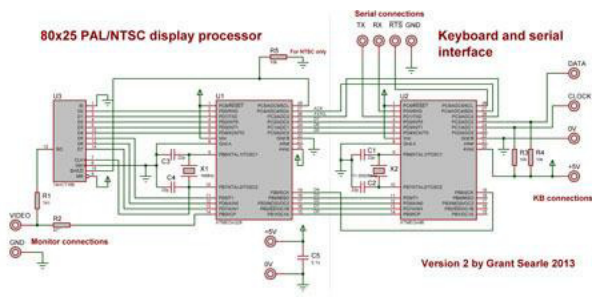
The three diodes top-right are the brightness adjustment - the more in series with the cathodes, the lower voltage drop across each LED resistor and therefore lower current/brightness. The resistors in series with the LEDs are 470R, so max LED current is a bit under 7mA with no diodes in series with the ground connection (this is the limit for the '374 chips if all LEDs are lit), or about 3mA with the three diodes in series to ground. 7mA is more than enough current for the LEDs, and 3mA is fine for most lighting conditions.

The input ports have resistor packs taking each input to ground (0) when not connected. The switches then allow each input to connect to Vcc (1).

These ports are accessed in BASIC using **x=INP(n)** to read the input port, or **OUT n,x** to write to the output port.

Using a keyboard and monitor instead of the PC interface

This board, as for my other boards, can use my monitor and keyboard interface [here](#)



My other pages

This computer with 32K RAM modifications are [HERE](#)

[CLICK HERE TO GO TO MY MAIN PAGE FOR MY OTHER PROJECTS](#)

I hope this page has been useful.

Grant.

To see my other pages or to see my contact address, please click [here](#). Please note that this address may change to avoid spam.

Note: All information shown here is supplied "as is" with no warranty whatsoever, however, please let me know if there are any errors. All copyrights recognised.