Orton SC/MP computer manual

Index

Disclaimer

The information in this document is provided purely as a record of the construction of one of my computer projects. I provide no assurance or guarantee whatsoever as to the accuracy, safety or originality of the contents of this document. I provide no warranty whatsoever as to the suitability for any purpose of the contents of this document. I accept no responsibility whatsoever for any deaths, injuries or losses resulting from the use of the information contained in this document. I have no claims to most of the technology used in my projects or described in this document, indeed, it must be assumed that I have used much copyrighted and/or patented material. But since I do what I do for purely recreational, educational or personal instructional purposes, I do not believe that I am breaking the law. It is up to the individual using the information in this document to determine whether, and to ensure that, their use of the information I provide is both legal and safe. This is supposed to be fun.

Chapter 1 PIC-based terminal

Introduction

My interests in early microcomputers are skewed towards the very earliest 8 bit era when use of an ASR 33 teletype was not uncommon. The microcomputers of that time (e.g. the Altair 8800) had no backup storage whatsoever and storage to cassette tape was still in its infancy. When using said teletypes, it was usual to fall back on these machines' paper tape punch and reader as a means of program storage and retrieval. In that respect, I suspect it is no coincidence that when you LIST a BASIC program, the format in which it is printed is compatible with typed entry. This means that you can capture a LISTing to paper tape, and then replay it to the computer at a later date using the paper tape reader, at which time the computer's 'perception' is that you are typing the program very quickly.

There is a downside to this trick however - BASIC interpreters need time to load completed lines into memory following each received carriage return (CR) character, and while we don't notice a pause when typing normally, the speed of paper tape input could potentially result in characters being lost while the interpreter is processing each line. But since ASR 33 teletypes work at only ten characters per second such loss was rare.

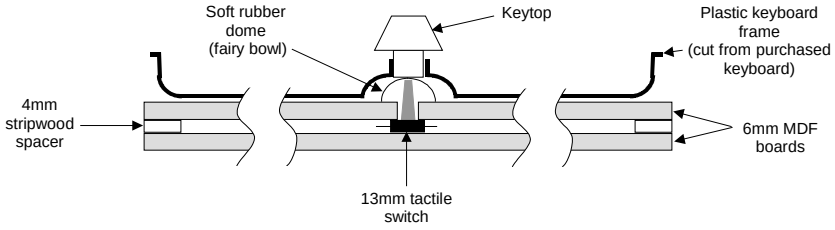
In honour of those pioneering times I have made a PIC-based terminal that simultaneously scans an ASCII keyboard and generates a text display. The PIC is directly interfaced to a host microprocessor unit (MPU) using the PIC's Parallel Slave Port (PSP) thereby replacing the usual UART. In line with teletype practice, the terminal has local echo. In addition to basic terminal functions, the PIC firmware includes a 100 character per second cassette interface that uses FSK modulation for high reliability.

The cassette port serves the function of the paper tape punch and reader: All characters delivered to the terminal from the host MPU are delivered to the cassette port. Similarly, all characters demodulated on the cassette input are delivered to the host. This arrangement provides the functional equivalent of a teletype's paper tape facilities: pressing the cassette recorder's 'Record' key corresponds to activating the teletype's hole punch, while pressing the 'Play' key corresponds to activating the paper tape reader.

Keyboard

I decided to construct my own keyboard rather than use a standard USB item since these require complex drivers. PS/2 keyboards, while much easier to drive, are rather rare now. My keyboard was constructed using the plastic frame and keys/plungers of a low cost purchased USB keyboard. The frame was cut down to leave only the core keys, dispensing with the function keys, numeric pad, arrow keys, etc. The frame was then mounted onto a piece of 6mm thick MDF board.

The 'fairy bowls' that provide hysteresis to the key travel were retained but the membrane contact layer was replaced with 13mm tactile switches. I did this mainly because I didn't want to be tied to the matrix provided on the membrane (in any case it proved impossible to make reliable connection to the cut down membrane). Use of tactile switches also allows easy repair of the keyboard, should any switch become faulty. The result was a somewhat stiff but 'clicky' keyboard with a retro feel. The membrane sheets, though ultimately discarded, provided a precise drilling guide for the holes that admit the tactile switch plungers.



Display resolution

The terminal generates a monochrome 625 line, 50Hz non-interlaced video signal bearing a character display of 16 lines of 25 characters. Each character is formed from a grid of 8 wide by 14 high pixels. The terminal uses the standard ASCII character set* and the font is a 'classic PC' font, which is very readable. Lower case is supported with descenders. An extra 31 graphic characters are also supported.

* The exceptions are the ASCII DEL character (character 127), which has been replaced with the British currency symbol '£'. Also, shift-tab is programmed to generate character 15_{10}.

Terminal operation

There are essentially three sources of characters within the terminal. They are handled as follows:

Source	Sent to host	Sent to screen	Sent to Cassette
Keyboard	Yes	Yes	Yes
Host	No	Yes	Yes
Cassette	Yes	Yes	No

Keyboard interface

The terminal scans an 8 by 7 matrix comprising of 52 keys, plus shift and control keys. Two character roll-over is included, i.e. a second key stroke can be accepted that begins before the previous key has been released.

Cassette interface

The cassette interface uses FSK encoding and records at 100 characters per second. A '0' is recorded as a single cycle of a 900Hz tone while a '1' is recorded as two cycles of an 1800Hz tone. Characters are recorded as nine bits: a zero start bit, seven bits of ASCII character information (LSB first) followed by a single stop bit of one. No parity is recorded.

The terminal has no buffering and so the cassette interface determines the rate of character transfer between the terminal and the host. Due to day to day speed variations or differences in playback speed between individual cassette recorders, playback may sometimes exceed 100 characters per second. This may lead to occasional playback characters being lost to the video display. However, the cassette decoder itself has a wide speed tolerance and so it is assured that the host will be in receipt of all characters.

The cassette interface is polarity sensitive. Since it cannot be predicted whether a cassette recorder will return a signal with the same polarity as was delivered to it, links are included to allow the replay polarity to be matched to that of the recorder.

Bell sounder

The terminal is able to respond to ASCII character 7 and trigger an audio bleep circuit.

Escape sequences

Escape sequences allow access to some useful graphic characters and are encoded as an Escape character (number 27) followed by one of the following characters:

@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	┘	└	─	┴	│	┐	┤	┌	├	┬	┼	-1	¼	½	¾

P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
°	±	π	µ	²	³	¢	€	Ω	×	÷	√	♥	♣	♠	♦

Host communications protocol

Host communications use the PIC16F877 Parallel Slave Port (PSP). This port may be interfaced to a microprocessor as a single byte location, with chip select (/CS), read (/RD) and write (/WR) strobes being accepted by the PSP. Flow control is implemented as follows.

Seven bit ASCII characters are sent by the terminal to the host as single bytes, the most significant bit (MSB) of which is used as a sequence bit, i.e. it alternates as

each new character is presented. The host should therefore watch for a change in the MSB of the read data (or indeed any change) to detect new characters. It is recommended that each character is re-read on each such detection, just in case the host noticed the change at the moment when the data lines of the read port were in transition.

Seven bit ASCII characters are sent by the host to the terminal by simply writing them to the PSP. When the terminal has accepted and processed each written character, it will issue a low-going pulse on the /ACK line. The pulse duration is in the range 60..130 microseconds. The MSB of written characters should always be zero (there is one exception to this rule - see later).

It is recommended that the host employ a time-out of around 20 milliseconds on acknowledgment, just in case an acknowledgement pulse goes unnoticed by the host. Monitoring of the /ACK line is best done using a set/rest latch, a counter, or an interrupt, as otherwise the host would have to waste time polling this signal.

The terminal program requires a short period to initialise following a power-on. During this time the /ACK line is in a high impedance state. Once the terminal has completed its initialisation this line will be asserted HIGH, indicating that the terminal is ready to receive characters from the host. It is advised that the /ACK line is pulled to the LOW state by a 10k resistor so that the host microprocessor can detect this readiness.

A word of caution: A problem arises when the host changes direction from sending characters to the terminal, to receiving characters from the terminal. For reasons unknown, the last written character can be duplicated if a read is attempted before the previous write has been acknowledged. For this reason, it is necessary to check for acknowledgement before a character read is attempted.

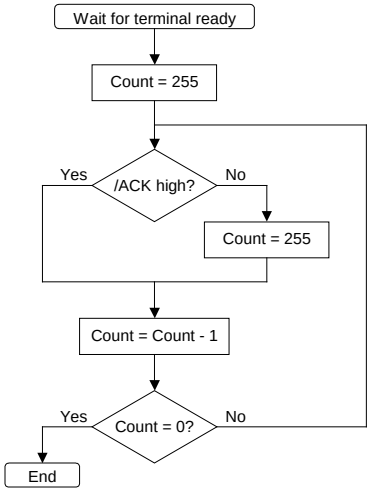
Carriage return and line feed

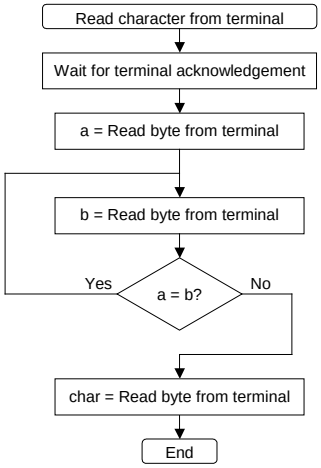
On receipt of a Carriage Return character (number 13) the terminal starts a new line. Line Feed characters (number 10) are ignored. The only other control characters that the terminal responds to are Bell (number 7), Backspace (number 8) and Escape (number 27).

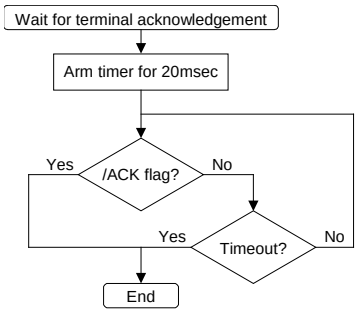
Video output

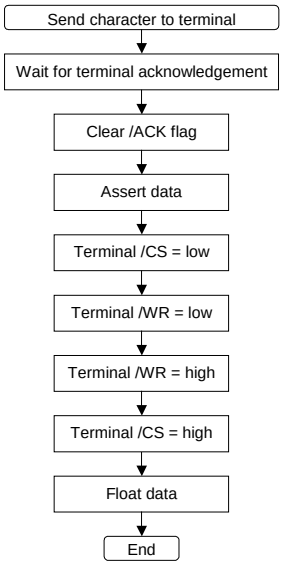
The video output requires a 75R load in order to develop a standard $1V_{p-p}$ signal. Most video monitors present this load but if there is evidence of excess signal, or if the monitor struggles to lock to the signal, then an explicit load resistor may be required. Note that some modern TVs cannot lock to non-interlaced analogue video.

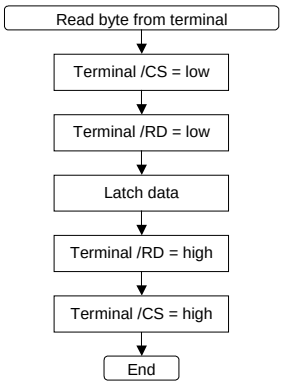
Terminal communication flowcharts











Cassette throttling

During replay from cassette, the host computer may need time to process each line of text as it is received. To ensure that this time is available on playback, it may be necessary to insert pauses during recording. This can be achieved by writing the character code 0xFF to the PSP. This code will have no effect on the terminal display but will ensure that a 10 millisecond pause is inserted into the data recorded

to cassette. Note that writing any other character to the PSP with the MSB set to '1' will have unpredictable results.

Chapter 2 PIC SC/MP Computer with National's Industrial BASIC Language (NIBL BASIC)

Presented here is a host microprocessor system that makes use of the PIC-based terminal. It uses a Microchip PIC emulation of the National SC/MP microprocessor, running a fully expanded NIBL tiny BASIC (that is, the computer has 32kBytes of memory).

Terminal case logic

NIBL expects upper case keywords and program statements. To avoid the need to constantly hold down the shift key, a special version of the PIC terminal was built for this computer where the shift key behaviour is reversed. The keyboard alpha keys therefore default to upper case, while lower case is selected by holding down the shift key. Although NIBL was designed to work in systems that only support upper case, NIBL will allow lower case in text strings.

Serial port

The computer includes a serial port which is fixed for 1200 Baud, 8 data bits, no parity and 1 stop bit. The port includes RTS/CTS bi-directional flow control (i.e. 'five wire connection'). This port has several possible uses. Firstly, it can be used to connect a printer. Secondly, it can be used to transfer programs to and from a modern computer. Alternatively, the serial port can be used to control external peripherals.

The SC/MP Status Register

The SC/MP microprocessor has an eight bit Status Register, five bits of which provide access to user flags and sense inputs - physical digital I/O pins on the microprocessor chip. NIBL provides access to the Status Register via the 'STAT' pseudo-variable. In this implementation, the three user flags (F_0, F_1, and F_2) along with the interrupt enable bit (IE) and the two sense inputs (S_A and S_B) are used to control the serial port. The Status Register is initialised to zero thus ensuring that the serial port is disabled.

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
SC/MP	CY/L	OV	S_B	S_A	IE	F_2	F_1	F_0
This implem-entation			SIOVF	SIMON	NPAD	SIEN	CTSEN	SOEN

Bit 3 of a real SC/MP status register is the SC/MP interrupt enable bit. NIBL does not support interrupts and so this bit has been re-purposed.

Bit	Description	Comment
0	SOEN	When this bit is set, NIBL output is sent to the serial port as well as to the terminal. This bit can be toggled by typing Control-P.
1	CTSEN	When this bit is set, flow control using CTS is enabled for serial port output.
2	SIEN	When this bit is set, NIBL input is sourced from the serial port instead of the terminal.
3	NPAD	A delay is automatically added to the end of each text line that is sent to the terminal. This delay is essential when saving programs to cassette (it ensures that NIBL is given sufficient time to process each text line when a program is played back from cassette). When this bit is set, this feature is TURNED OFF thereby allowing faster output to the terminal.
4	SIMON	This bit is only relevant when the SIEN is set. At other times this bit has no effect. When set, characters received from the serial port are sent to the terminal for monitoring purposes. Note that characters can be received on the serial port somewhat faster than the terminal can process them and so some characters may not be displayed.
5	SIOVF	This bit is set when an 'RTS overrun' occurs. On receipt of a Carriage Return on the serial port, the computer negates RTS to instruct the attached device to cease sending. Should the device continue to send characters, and exceed the buffer space for RTS overrun, then characters will be lost and this bit will be set. Consequently, a '1' in this bit position indicates that the attached device is taking no notice of RTS. This bit does not clear itself automatically and so an assignment to the NIBL 'STAT' pseudo-variable is required to set this bit back to '0'.

The ability to redirect NIBL input and output to the serial port has several possible uses:

Redirected to serial port	Printer	Modern computer	Device
NIBL input	-	Terminal emulator BASIC program download	Use of INPUT to receive readings
NIBL output	Hardcopy Log of session	Terminal emulator BASIC program upload	Use of PRINT to issue instructions and to send parameters

There are a couple of hazards to be aware of when enabling the serial port:

Firstly, enabling CTS flow control will lead to the system hanging up if the CTS line is not driven properly or left floating. In this event, the Break key should be pushed. This will clear SOEN thereby turning off serial port output. The computer will then resume but sending output to the terminal only.

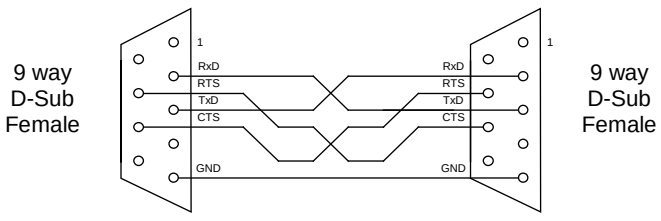
The other hazard is a BASIC program stopping for some reason and leaving the source of NIBL input pointing at the serial port. In this event, pressing the Break key will clear SIEN thereby switching NIBL input back to the terminal. The computer will then resume but taking input from the terminal. Using the Break key to interrupt program execution will also ensure that SIEN and NPAD are cleared.

Note that RTS flow control signals are always generated by the serial port, and may be sensed by a connected device so that characters are not sent to the computer at times when it is not attentive. NIBL is sufficiently fast that it can process the characters within a text line without resort to flow control. It is only on receipt of a Carriage Return that NIBL requires significant processing time.

RTS is negated following receipt of a Carriage Return character. The serial port will then continue to accept characters (up to a limit of 64) until the connected device ceases sending. This ensures that characters already in the sending device's hardware buffer are not lost. Once NIBL has processed a text line (and any RTS overrun characters) RTS is re-asserted so that the connected device can send more characters.

Connecting the serial port to a modern computer

Both the SC/MP NIBL computer and a PC are 'DTE' devices and so must be connected using a 'null modem' cable. This must use a five wire connection as follows:



A terminal emulation program permits control of the SC/MP NIBL computer as if it were the computer's own keyboard and screen*. Such a program must be set up for 1200 baud, 8 data bits, no parity and 1 stop bit. RTS/CTS flow control must be enabled and the emulation should be set for local echo. When NIBL is taking input from the serial port, a break can be instructed by placing a break condition on the serial line (by typing Control-Break in Hyperterminal). A break can always be instructed using the 'Break' button on the SC/MP computer itself.

* There are exceptions - see later.

Key timing parameters for program loading

Like some other early BASIC languages, NIBL relied on the paper tape punch and reader of a teletype as the means of program storage (NIBL has no 'LOAD' or 'SAVE' commands). This simplified the NIBL interpreter but provided no means to detect or correct any errors that might occur during data transcription to or from paper tape. Transferring a program to paper tape required the following steps:

1. Type 'LIST' but do not type a Carriage Return
2. Turn on the paper tape punch
3. Type a Carriage Return
4. NIBL will list the entire program which the punch will duly record
5. When the listing is complete, turn off the tape punch

Loading a program from paper tape required the following steps:

1. Type 'NEW' followed by a Carriage Return
2. Turn on the paper tape reader
3. The paper tape reader will play back the program to NIBL
4. When program load is complete, turn off the paper tape reader
5. Type a Backspace (Control-H)

The last step of the loading procedure is simply to delete the prompt '>' that was recorded to paper tape at the end of the save process. The procedure for saving to cassette is very similar:

1. Type 'LIST' but do not type a Carriage Return
2. Press the 'Record' button on the cassette recorder
3. Allow a few seconds to record a lead-in on the tape
4. Type a Carriage Return
5. NIBL will list the entire program which will be recorded to cassette
6. When the listing is complete, turn off the cassette recorder

Loading from cassette requires the following procedure:

1. Cue the tape to the lead-in (clear tone)
2. Type 'NEW' followed by a Carriage Return
3. Press the play button on the cassette recorder

4. Delete any spurious characters using Backspace (Control-H)
5. The cassette recorder will play back the program to NIBL
6. When program load is complete, turn off the cassette recorder
7. Type a Backspace (Control-H)

There is an inherent hazard in the loading process, namely that the paper tape reader or the cassette recorder might send characters too quickly for NIBL to process. The saving grace in the teletype era was the low speed of teletypes - ten characters per second - and this allowed sufficient time between characters for NIBL to cope. But at the higher data rate of the cassette interface or serial port, some means of 'throttling' must be used.

In the context of the serial port, throttling can be achieved by means of the built-in RTS/CTS hardware flow control. For program load from cassette however, delays must be inserted during recording to ensure that NIBL is not overwhelmed on playback. To get some metrics on this issue the NIBL emulator was timed to see how long it required to process key strokes. It was determined that this emulated version of NIBL requires 320 microseconds to process ordinary key strokes (i.e. not control keys). Carriage Return however is another matter, and the key measurement here is the time required for NIBL to append a new line onto a program. To get a handle on this figure, a NIBL BASIC program was written which creates a 'worst case program':

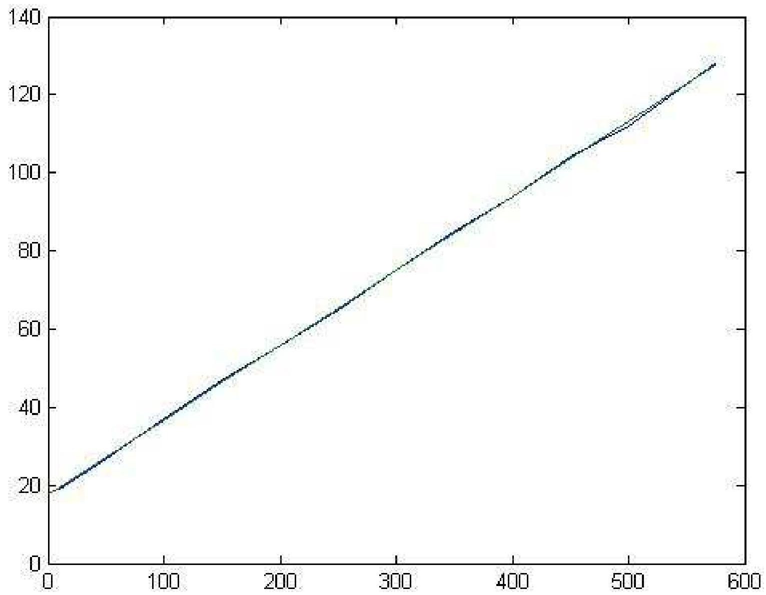
```
10 REM WORST CASE PROGRAM GENERATOR
20 @#2001=13
30 FOR N=1 TO 574
40 A=#2000+(N*7)-5
50 @A=N/256
60 @(A+1)=N
70 @(A+2)=7
80 @(A+3)=82
90 @(A+4)=69
100 @(A+5)=77
110 @(A+6)=13
120 NEXT N
130 @(A+7)=255
140 @(A+8)=255
```

This program was typed into NIBL page 1 and run. On termination, a 574 line program will be found in NIBL page 2, where each line contains only the letters 'REM'. This is very nearly the maximum size of a NIBL program in terms of line count. Line count matters because when loading a program from cassette, a significant portion of the time required to process each line is spent finding the program edit point. Provided a 'NEW' command is issued prior to starting cassette input, loading will be strictly sequential and will not involve any relocation of existing program text.

The ultimate test was to time NIBL's response to entry of a new line 575 for this huge program. This line consists of 70 letter 'A's which is very nearly the longest possible NIBL line:

```
PAGE=2
575AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAA
```

The NIBL implementation required 130 milliseconds to add this line to the end of the synthetic program in page 2. Were one to add a period of silence of this duration to every line recorded to cassette, then a more typical 100 line BASIC program would require 33 seconds to save and load, 13 seconds of which would be end-of-line pauses. A more thorough investigation was undertaken whereby the same experiment was performed for a number of program lengths. The results are shown in the following plot.



The horizontal axis is program line count while the vertical axis is milliseconds. The blue trace is actual measurements while the green trace is a linear regression. The linear relationship is to be expected - the more lines in the program, the longer NIBL requires to skip over them to find the edit point. The regression identified a formula as follows:

$$T_{LOAD} = 17.6 \text{ milliseconds} + 191 \text{ microseconds per line}$$

On this basis, and for programs of up to 400 lines in length, a 100 milliseconds pause at the end of each line that is written to cassette would guarantee that NIBL would not be overwhelmed when the program is replayed for reloading. Using this fixed end-of-line pause, a 400 line program would typically save and load in about 80 seconds.

The linear relationship hints at a possible efficiency measure: During recording to cassette, the appended pause could start at 20 milliseconds and then increase by 200 microseconds for each line that is output. For programs of 400 lines in length, this would cut the save and load time by around 16 seconds. While significant, it is felt that this saving does not justify the extra complexity of this scheme.

Program checksum facility

As mentioned earlier, NIBL has no means to validate received characters. At the time that NIBL was conceived, the expectation was that a NIBL computer would operate in the same room as the teletype to which it was connected, thereby eliminating errors resulting from long cables and interference on the public phone network. This principle still stands and so no additional measures have been included in this implementation.

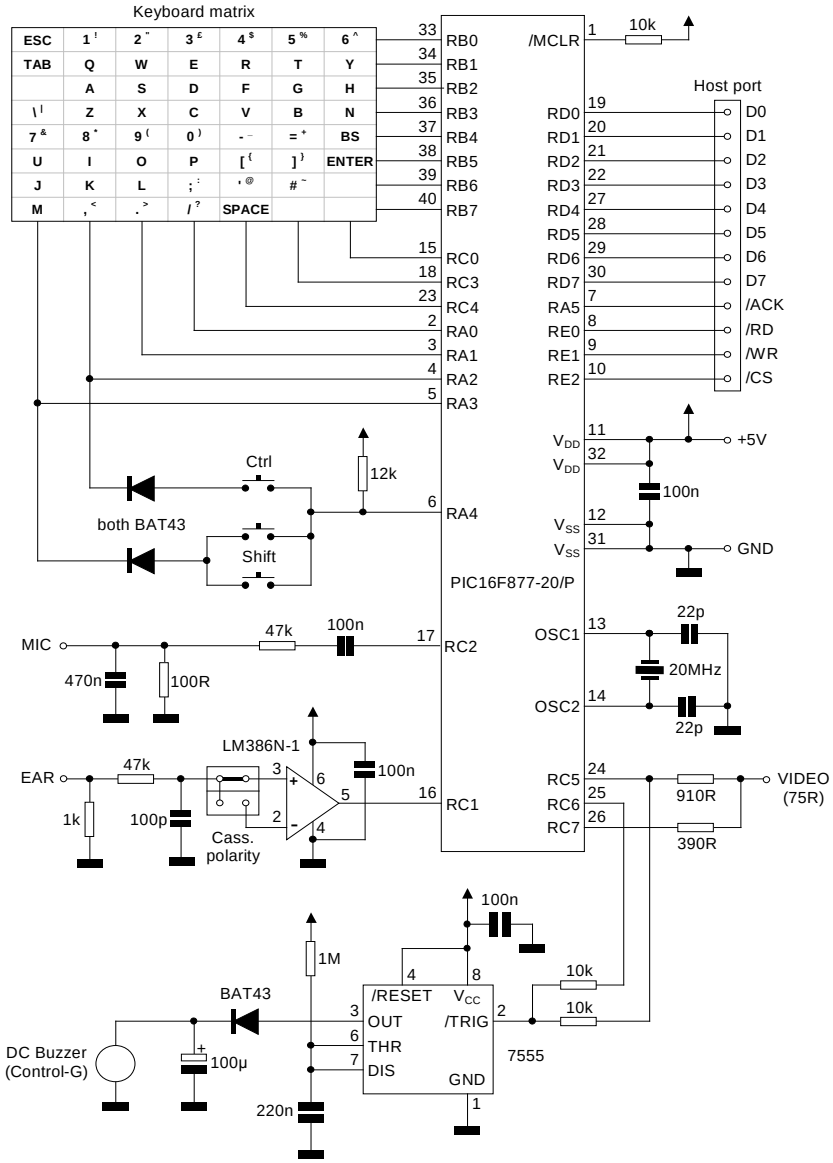
The one exception is cassette storage and retrieval, whose bit error rate is less than exceptional. Occasional cassette read errors might lead to a corrupt program being loaded and so a facility has been included to compute a checksum for an in-memory program. The checksum consists of a four digit hexadecimal number. To obtain the checksum for the program in the current page (as revealed by the NIBL 'PAGE' pseudo-variable) the user simply types Control-S. The user can then use the returned value to either verify a loaded program, or to append to a program in preparation for saving. The latter is easily done by dedicating the last line as a checksum line:

```
>{Control-S}
FE76
>LIST 32000

32000 FE76

>_
```

Provided program execution cannot reach this line then no errors will result from this line's presence. The checksum calculation excludes lines numbered 32000 or above, thereby avoiding a 'chasing one's tail' situation where the act of recording the checksum alters the checksum. Note that the program checksum facility is available only when NIBL is taking input from the terminal.

Chapter 3 Hardware

Terminal



SC/MP NIBL computer

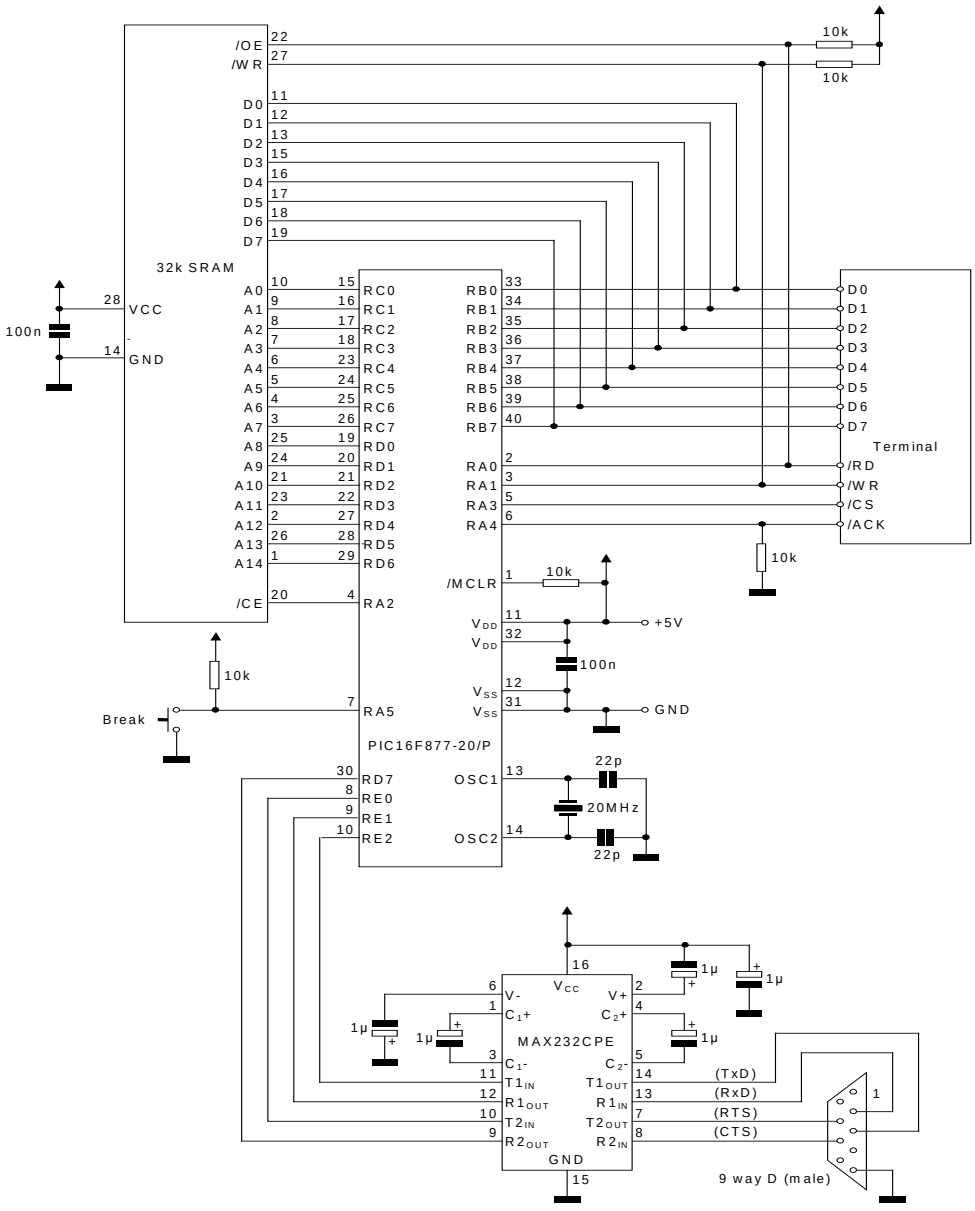


Photo of SC/MP NIBL computer

Chapter 4 Software

There are a number of supporting files associated with the Orton SC/MP computer:

File	Purpose
SCMP computer.ht	Hyperterminal file for computer
PIC NIBL.asm	PIC source for the SC/MP NIBL computer
trmnl.asm	PIC source for the terminal