ZX-TERM*80

A FULL-FEATURE COMMUNICATIONS PROGRAM

FOR THE ZX81, TS1000 and TS1500

Written by Fred Nachbaur

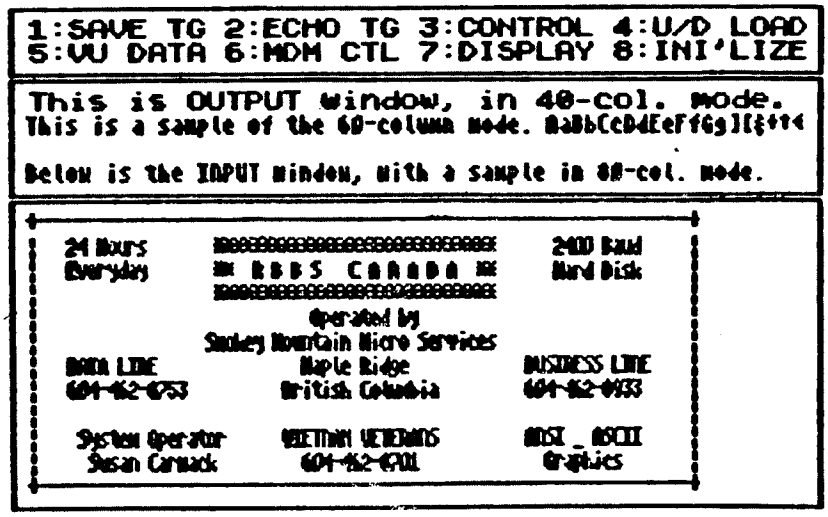
XMODEM portion by Harvey Taylor

WRX16 High-res core by Wilf Rigter

Produced by:

Silicon Mountain Computers
C-12, Mtn. Stn. Group Box
Nelson, BC V1L 5P1

CANADA



WHAT IS ZX-TERM*80?

ZX-TERM*80 is a complete communications package for the ZX81, TS1000 or TS1500 computer, with Westridge (TS2050) or Byte-Back (MD-2) modem. It is a true 8-bit ASCII terminal program, allowing you to communicate with Bulletin Board Services (BBS) as well as any of the other communications services (Compuserve, The Source, etc.) You can, of course, also communicate directly with other terminals, including but not limited to other Timex/Sinclair users with the appropriate software.

ZX-TERM*80 is much more, however. It can also be used to upload (send out) or download (receive) ASCII files, Sinclair programs and/or their associated variables data, word-processor files, or virtually anything else. It uses the popular XMODEM protocol to insure accurate transmission and reception of data files.

ZX-TERM*80 is a quantum leap over other communications packages that have appeared for the ZX81 family of computers. The biggest thing that sets it aside from all the rest is its display. You can easily select between 40, 60 and even 80-column displays! Many BBS's use 75 or 80 columns, so you won't be forever trying to cope with split lines on the traditional 32-column Sinclair display. Lower case is displayed on-screen in all character widths, as are all the ASCII (as well as all the Sinclair) characters.

ZX-TERM*80 completely redefines the Sinclair keyboard, turning it into a true terminal keyboard. Unshifted keys are lower-case, shifted keys are capitals just like on a typewriter. All keys auto-repeat. The program is menu-driven, using the shifted number keys for the main menu options. SHIFT SPACE is used universally as an ESCAPE key, to return to the previous menu level. Prompts and menus make the program extremely easy for even beginners to use, while maintaining full flexibility for more advanced users.

ZX-TERM*80 allows true windowing. Two window modes are selectable; in 1-window mode, all input and output are displayed on the same screen, much like the popular MINI-XMODEM program on which it is based. In 3-window mode, however, three separate windows are employed. The top window is reserved for menus and prompts, the middle window is for your typed output, and the bottom window is for the input you receive from the other terminal. The relative sizes of the send/receive windows can be easily varied using a menu option. All windows auto-scroll independently.

ZX-TERM*80 comes with its own relocator, as well as several other user-modifiable options. Even allows continuous printing to full-size printers! "Textwriter" mode, program/variables scan, duplex toggle... the list goes on and on.

ZX-TERM*80 is written entirely in hand-assembled and optimized machine code. In spite of its many features, it takes up only 4K at the top of your memory (plus the 8K static board).

WHAT WILL I NEED?

You will, of course, need a Sinclair ZX81, Timex/Sinclair 1000 or 1500 computer. You need at least 16K of "user RAM." You also need either a Byte-Back or Westridge 300-baud modem.

Another vital element is a suitable 8K static RAM, mapped in the 8-16K region. The popular "Hunter" board will work, provided that a minor modification has been made. The little NVM circuit published in SyncWare News #4:1 will also work. Silicon Mountain Computers sells a "SCRAM" board which will also work with no changes. If you have successfully run any of our other high-resolution programs, you are already set to go.

64K USERS:

If you have a ZX81/TS1000 and 64K of memory, you can relocate ZX-TERM*80 into the 32-48K region of memory, allowing more space for your data files, and allowing you to up/down load larger Sinclair programs or variables. However, your "ULA" chip must have its M1 NOT line decoded to allow machine-code to run in the 32-48K block. See SyncWare News #2:5, or contact Silicon Mountain Computers about a plug-in adaptor that will do the job with no soldering or trace-cutting. Note: even with 64K RAMs, you will still need a CMOS (static) RAM in the 8-16K region. If possible, disable the 8-16K region of your 64K RAM. (In most cases, however, this is not essential; so if it's impossible or inconvenient to disable this portion of your 64K RAM, try it anyway. On all the packs we've tested, this and other WRX16-based programs still run properly, provided that a suitable static RAM is also mapped in this region.)

TS1500

The TS1500 computer is a little different. It is actually the superior machine for this program, but has slightly different requirements. As with the ZX81/TS1000, you will need a suitable static RAM. However, you ALSO need an external 16K RAM pack (TS1016). Without this, the display routines will not work.

On the TS1500, the M1 NOT line is already decoded, so there is no need for an "M1 NOT adaptor" or modification. Simply plug in your static RAM and 16K RAM pack, and you're ready to go. This arrangement makes it possible to relocate ZX-TERM*80 into the 32-48K region, freeing up all of the 16-32K region for your files and program up/downloads.

64K RAM packs that work on the TS1500 in normal display mode should work fine in high-res mode. Except for the point about M1 NOT decoding, all other points mentioned for 64K ZX81's applies to the 1500 also.

WHY READ THIS?

This manual fully details the care and feeding of ZX-TERM*80. If you have used "MINI-XMODEM," you will find the upgrade to ZX-TERM*80 completely natural. This manual will point out the differences.

The manual is also for the beginner, who has never used a terminal program. Read carefully, it WILL show you how to use ZX-TERM*80 to the fullest. However, it will NOT teach you about modem communications in general, it will NOT teach you BBS etiquette, and it will NOT give you any "good numbers" to call. Consult other publications, get advice from "old-timers," and join a user group. The manual is also provided for hackers who like to dive right in, as a last resort if all else fails.

WHAT ABOUT MY PRINTER?

Yes, ZX-TERM*80 will work with just about any printer imaginable.

TS2040

If you have a TS2040, include it in your line-up. You can copy the contents of any window at almost any time, and can also dump your gathered data to the printer after acquiring it.

"BIG" PRINTERS

If you have a full-size printer and a suitable interface, hook it up also. Patches are supplied in the BASIC driver, for all the popular printer interfaces I know of. Note that even the relatively inexpensive Tasman and Aerco "2068" interfaces are included.

Supporting big printers was, in a way, my biggest headache in writing this program. There's nothing particularly rough about these per se, but the vast diversity of printers and interfaces makes the task somewhat complex. I had to strike a balance between versatility and memory economy; for this reason, some of you might be disappointed in the lack of comprehensive big-printer options. However, you CAN print you data as it being sent/received. You simply enable your printer, and subsequent text gets printed; take the printer off-line, and the program continues just as if the printer wasn't even there. With a little ingenuity you can work up ways of sending your acquired files to the "big printer" after the fact, as well. Further discussion of this and related topics is reserved for later on in this manual.

READ ME MY RIGHTS...

Like the classic "Hot Z" program, ZX-TERM*80 "reveals all." It can even be used to upload itself. All of its machine-code is in a line 0 REM statement in the BASIC loader program. Its data and character sets are in a dimensioned variable A$. This is the only such program I'm aware of, which can be used to transmit/receive copies of itself. In a truly flexible and usable program, there is no room for "protection schemes," which generally don't work anyway.

But BE ADVISED: just because you CAN upload ZX-TERM*80 to your favorite BBS does not mean that you have the right to do so. We had better not EVER, EVER see this program on any BBS or program exchange. You might think, "how will they know?" We have our sources. In a small computing community such as ours, there are much worse fates than lawsuits. Don't shoot us, your fellow computerists, and yourself in the feet.

However, this chapter is not all fire and brimstone. ZX-TERM*80 is designed to allow it to be customized to mimic other graphics sets; e.g. ASCII, ANSI, or IBM. All such data tables are in the afore-mentioned A$ variable. Such customizations are entirely up to you, and a utility is even provided to let you transfer your modified character tables into the variables area. If you wish to share them with other users, you are hereby granted permission to do so by uploading the VARIABLES part of ZX-TERM*80 ONLY. Other ZX-TERM*80 users can then download your customized variables set into their own copy of the ZX-TERM*80 program itself.

If you get tempted to upload the program, or make tapes for your friends, ask yourself this question: how would you feel? Would you be inclined to spend months developing new programs after discovering your hard work in the freebie bin?

AN INTRODUCTORY TOUR

LOAD "ZXT80" into your computer. You can make a back-up copy after loading by pressing "S" at the cover screen. Pressing any other key on completion of loading, you will be taken through various prompts of a "set-up" nature. These are covered in further detail in the "meaty" portions of this manual; for now, simply answer as follows:

MODEM TYPE: Answer (W)estridge or (B)yte-Back.

PRINTER INTERFACE: Answer appropriately. Press "X" if you are not using a big printer.

CHANGE MODE WORD: Answer (N)o. This will give the standard XMODEM default parameters, i.e. 8 bits, no parity, 1 stop bit.

RELOCATE? Y/N: Answer (N)o.

MEMORY 16/32 or 64?: If you are using a 64K RAMpack, press "2". Else (as with 16 or 32K, or with TS1500+16K, answer "1".)

SAVE CONFIGURATION: Answer (N)o this first time through.

The ZX-TERM*80 machine-code is then transferred to its run-time location at the top of your 16K RAM, and you automatically enter the main program. The screen will come up in the 3-window mode, and 40-column character width. The small top window is where you'll get all your prompts and menus. On entering the program, this will say, "DIAL NOW...", prompting you to pick up the phone and contact the board of your choice. For the first part of the tour, you won't need to do this; let's take a look some of the general features of the program first.

You will notice two larger windows below the prompt window. The middle window, presently containing the ID legends, is where your typed OUTPUT will appear. Henceforth, this window will be referred to as "window 2." The bottom window, which I call "window 3," is where the INPUT from the other terminal will appear.

At the very bottom is an auxiliary prompt/menu line. This presently says, "PRESS ANY KEY TO CONTINUE." Doing so will initialize your modem, and jump to the main loop. The main menu options will appear, as always, in window 1 (the top window). Note that all of these options are accessed using the SHIFTED number keys.

The bottom line now shows the following, from left to right:

MODE CURSOR: Just where you'd expect the usual input cursor, is a "mode" cursor. This will normally be inverse "L", for "letters." However, this changes to an inverse "S" for "symbols" (following SHIFT ENTER) or to an inverse "C" for "control" characters (following SHIFT 3). The symbol and control modes are good for one keypress, just like the FUNCTION mode in the normal Sinclair system.

S/SP=ESC - This is a constant reminder that you can go back to the previous level using SHIFT SPACE. Try it; you will be taken to the "end" of the program, with the prompt "SAVE FILES? Y/N" and the usual "SHIFT SPACE TO ESCAPE" legend. Pressing "N" at this time turns off the modem (if using a Westridge) and returns you to BASIC; get back into ZX-TERM*80 with RAND USR 32270. Pressing "Y" at this time will do the same, since your "data buffer" was not initialized. Pressing SHIFT SPACE (henceforth referred to as "Escape") gets you back to the main loop.

S/NL=SYMBOLS - This reminds you that the shifted ENTER (NL=newline) key ("function") is used to type symbols; we'll get into that in a minute.

SAVE - This shows the status of the "save" flag, which controls whether or not the incoming text/data is being stored in memory.

For our introductory tour, the first thing you'll want to do if you have a Westridge is to turn the modem back off. Else you'll get nasty prerecorded messages from the phone company, telling you to "dial or get off the line." If using a Byte-Back modem, you may want to change from ORIGINATE to ANSWER mode.

SHIFT 6: MODEM CONTROL

If you are using a Westridge, pressing shifted 6, will get you the prompt "Hangup/Carrier." Press H (unshifted) to hang up. When you're ready to actually do some modemming, use shift 6 to restart the modem with the (C)arrier option. As always, you can use Escape to go back to the main loop if you got here by mistake.

SHIFT 6 with the Byte-Back is used to select ANSWER or ORIGINATE mode. Press "A" for answer, any other key for originate. If you are communicating with BBS's, Compuserve, etc. then you will use ORIGINATE mode. However, if you're communicating with another Byte-Back modem, one of you will have to be in ANSWER mode.

OK, we've kept the phone company happy while we play around to get the feel of this new software. Now what?

KEYBOARD RESPONSE

In the main loop, start typing something... anything. You'll see that whatever you type will appear in window 2. Pressing ENTER gives a carriage return (newline). Pressing shifted keys gives capitals, unshifted=lower case just like a typewriter. When you reach the bottom of the window, it will automatically scroll with any newline or wraparound.

If you hold down any key, it will auto-repeat. Try it! The repeat rate can be changed, as detailed in the section "SOME USEFUL POKES."

You'll have to be careful when typing stuff in all-capitals. Let go of the shift when typing spaces between words, or you will of course be dumped out of the main loop (remember, shift space = escape). If this happens, all is not lost... simply press shift space once more to get back on the road.

As mentioned earlier, the shifted number keys 1-9 are used for the menu options. (Actually, shift 9 "MEMORY REPORT" is not printed in the menu, but is available nonetheless.) Shifted 0 is, as usual, DELETE. However, you can't delete in the normal sense, since you can't "unsend" a character that was already sent! (Software that can travel backwards in time has yet to be written.) So, ZX-TERM+80 prints the "delete character" as a left-pointing arrow. However, most systems recognize the DELETE character (ASCII 127) to mean "Ignore the last character sent." You can therefore send one or more DELETEs if you make an error in typing a message, and the BBS will fix it up before returning it to you for verification or editing. Similarly, we don't go to the trouble of implementing deletes while receiving; if you get left-pointing arrows, realize that the sender intends the last character(s) sent to be zoofed.

SYMBOL MODE: SHIFT ENTER

If you press SHIFT ENTER, the mode cursor changes to a "S" (for "symbol"). Pressing one of the keys with a single symbol on it (bottom row, J, K, L, U, I, O or P) gives the appropriate symbol. The other 22 keys are unused in this mode; however, if you're so inclined, they can be hooked up to send other characters by suitably modifying the keyboard data table. (See Appendices.)

If you select symbol mode and SHIFT the symbol keys, you'll get all the other standard ASCII characters that aren't present in the normal Sinclair character set. If you've used Memotext, you'll have no trouble remembering these; they follow the Memotext convention almost exactly. If not, practise until you get the hang of where everything is.

The period/comma key is a little different; "Letter" mode gives a period if unshifted, comma if shifted. In symbol mode, unshifted gives the percent symbol, shifted gives apostrophe.

SYMBOL MODE SUMMARY:

Press SHIFT ENTER (similar to "FUNCTION" in normal BASIC), then any of the following keys:

UNSHIFTED	SHIFTED
Z :	`
X ;	^
C ?	!
V /	\
B *	DEL (another way to send DELETE)
N (	_
M)	~
J -	]
K +	@
L =	[
U $	&
I (	{
O)	}
P "	|
sp. ▮	ESC (see Appendix V)

CONTROL MODE: SHIFT 3

Shift 3 is used to send control characters A-Z. These show as inverse letters (to differentiate from data), and are useful to control what happens at the other end. For instance, some boards use CONTROL X or CONTROL K to abort the current operation. Similarly, CONTROL Q and S can be used to pause or restart the other modem. Control A sends ASCII CHR$ 1, control B sends an ASCII 2, etc. So control J would be ASCII 10 (line feed), M is ASCII 13 (carriage return - use this if you want a "hard way" to send a newline), and so on.

Unlike data characters, controls will echo in window 3 (except for control J = linefeed and control M = carrige return). This is because the other terminal will not automatically echo these when you send them. Though it might look messy, it's better than not having any assurance that a control was sent if you happen to have your ECHO TOGGLE (shift 2, covered later) turned off. Note, however, that such controls are NOT saved in your save buffer if the save flag is on, nor will they be sent to your printer if you have the printer flag set; the echo is to the screen (window 3) only.

Now let's take a look at the display options. (Demonstrate these to those poor unfortunates who are still under the delusion that you're trying to compute with a doorstopper!)

DISPLAY: SHIFT 7

Pressing SHIFT 7 gets you the display menu. You have a total of 8 options; the first letter (or number) of each choice in inverted, to emphasize the key that should be pressed for each action. Remember, you can ALWAYS get back to the previous level using ESCAPE.

(C)LEAR WINDOW

Pressing C gets the prompt "Window (2) or (3)" Press "2" to clear your output window of all the garbage you've typed in so far. You are then returned to the main loop. If you managed to get a bunch of garbage in window 3 from not hanging up in time, call this option again and select "3."

In 1-window mode, of course, there is no need for a choice; in this case, the single window is cleared immediately.

(A)DJUST WINDOWS

This is rather fun. Use the 6 and 7 keys (unshifted) to vary the relative sizes of windows 2 and 3. If you'll be doing a lot of typing (e.g. leaving a long message for SYSOP or other users), adjust it way down so that you have a big window for your output. On the other hand, if you're perusing a database, you won't need a big window for your typed commands, and will want to adjust it up. When the windows are the size you want, use ESCAPE to get back to the main loop. The print positions will be reset to the tops of the respective windows.

NOTE: ADJUST clears both windows; don't press A after SHIFT 7 unless you don't mind losing the contents of the windows.

(P)RINT WINDOW

This copies either window to the TS2040 (only). As mentioned earlier, big printers are an entirely different bag of worms; their added versatility makes it necessary to take a different printing approach (covered later).

(M)ODE TOGGLE

This toggles between 3-window and 1-window modes. I'm sure that you will like the 3-window mode most of the time, but there are certain advantages to the 1-window mode also. For example, it's sometimes nice to have your communications laid out in the order they occurred. It's also quite handy for reading long messages or other files, since you have a total of 3 more lines available for display.

In 1-window mode, you won't get the "window 1" menus and prompts... so don't use it until you know the options by heart. Actually, about all you really need to remember is that SHIFT 7 gets you to the DISPLAY OPTIONS menu, from where you can return to 3-window mode.

The (C)lear option is redundant in 1-window mode, as the screen is cleared anyway for the prompts. Similarly, the (P)rint option is unusable in this mode.

Up/download temporarily goes to 3-window mode even if you're in 1-window mode when you call it. In this case, the program will return to 1-window mode after the operation is completed or aborted. Conversely, the "view data" option temporarily goes into 1-window mode, returning you to 3-window mode when done, if that's where you were originally.

(R)EVERSE DISPLAY

The program starts out with white ink on black paper, which I personally find more pleasing (especially for the incredibly scrunched 80-column characters). However, you can reverse the display using this option. It will remain however you set it until you change it again.

40/60/80 COLUMN SELECT

Pressing 4, 6 or 8 sets up the desired character width. On start-up, the program comes up with the nice big 40-column characters. Also, regardless of which width you select, prompts and menus will always be in the easy-to-read 40-column width.

Unless you're in 1-window mode, the window print positions are not changed after a character-width reselection. So it's possible to change character width even in the middle of the line. However, this can cause the last character on the line to be "clipped," depending on how many characters were already printed in the previous width. Therefore, it's best to change modes after a carriage return, or send a carriage return immediately after returning to the main loop. However, even if a character gets "clipped," it will still be saved in your data buffer (assuming that the save flag is on).

The 80-column characters are actually the same as the 60-column set, except that there is no space between characters. Unfortunately, this makes this mode a little hard to read at times. This is especially a problem with random files, long number sequences, etc. Surprisingly, normal text is entirely readable; the human brain is amazingly adept at making sense of patterns such as words, even if the individual letters are hopelessly scrunched together. You'll find that, with a little experience, 80-column text can be read almost as easily as 60-column.

All of the standard 7-bit ASCII codes (32-127) are printed, as well as all of the Sinclair graphics. Some boards allow graphics (ASCII 128-255) to be sent in addition to the 7-bit ASCII character set. ZX-TERM*80 does not directly support these, because of the many different schemes available (ANSI, ASCII, IBM, etc. etc.). However, it DOES use a full 8-bit character set for all width modes, so if you're interested in configuring the program for one of these graphics sets, consult the Appendices.

ECHO TOGGLE (SHIFT 2)

This function from the main loop turns off your typed output. Why on earth would you want to do that? Well, many boards automatically "echo" back whatever you send. This means that whatever you type will also appear in window 3. Turning ECHO TOGGLE OFF by pressing SHIFT 2 will print the message, "DISPLAY ECHO: OFF" to be printed at the next output line, and stops subsequent output from being printed as it is sent. Pressing SHIFT 2 again will turn the ECHO TOGGLE back ON, with the appropriate message.

It should be apparent that shutting off the local echo turns window 2 into that much "dead air." For this reason, it is recommended that you use 1-window mode when communicating with boards that echo everything you type, else adjust the windows as high as they'll go.

Some boards allow you to select whether you want an echo or not. This can be handy if you want your typed butput to appear in the DATA REM (covered in detail later) along with the other terminal's input, without resorting to any POKEs. It's also useful if you have doubts about the reliability of your phone connection.

Note that there is no provision to automatically echo back whatever your receive. There is a remote possibility that you'll run into the occasional user with an inferior system that does not allow a "local echo," relying on other peoples' modems to "do all the housekeeping." The best thing you can do in such a case is to get him on voice, and suggest that he buy a Timex and ZX-TERM*80.

Some boards "intelligently" echo back only certain things, like your name and password, menu choices, etc. Messages, longer replies, etc. are not echoed unless you request verification. In these cases, simply leave the display echo on; the little memory you save in the buffer isn't worth the constant SHIFT 2 toggling that would be required.

SUMMARY

SHIFT 2: ECHO TGL
Toggles local echo on/off

SHIFT 6: MDM CTRL.
Westridge - "pick up" phone and start Carrier, or hang up.

Byte-Back: Select Answer or Originate mode

SHIFT 7: DISPLAY
Adjust, clear, or print windows; select window mode; reverse screen; select character width mode

THE CHARACTER SETS

Before we check out the next menu option, a few words about character code sets are in order. Most of the computing world uses the 7-bit ASCII character set. Each number from 0 through 127 corresponds to a specific character; the codes from 0 through 31 are the "control codes," and the codes from 32 through 127 are the various alphanumeric and symbol characters. The codes from 128 through 255 are "uncommitted," and are assigned to more-or-less arbitrarily by different computer and peripheral manufacturers. Several 8-bit systems exist, which use these high-order codes for graphics symbols, etc. In 7-bit systems, the eighth bit is sometimes used for parity checking.

As you probably know, the ZX machines use an entirely different character coding system. These "Clive codes" (for lack of a better term) run from 0 through 63 (normal characters) and 128 through 191 (inverse characters). Codes 64-127 and 192-255 (bit 6 high) are not printable characters, but are instead unused, assigned to the keyword and function "tokens", or various system codes.

Having a computer that uses a different character set from the rest of the world is a mixed blessing. On the one hand, it means that Sinclair data files (as text) will appear as complete gibberish to anyone not "in the know." On the other hand, it means that we have to translate between the two different code sets in order to communicate with the "real world." ZX-TERM*80 does the necessary translation with skill and ease.

Internally, ZX-TERM*80 uses a modified version of "Clive-code," but this is automatically translated to ASCII when receiving data from other terminals. In this modified version, all "bit 6 low" characters (0-63 and 128-191) are the same as before. Some of the rest are the new lower-case characters and special symbols, and five of them are system control characters. The rest are unused.

Translation from ASCII to Clive-code and vice versa is done via a pair of data tables, which can be modified any way your heart desires. This is described in detail later in the Appendices. Meanwhile, let's get a "feel" for the different code sets by playing around with the next option.

VIEW DATA: SHIFT 5

Presuming that you've followed the introductory tour "to the letter," the BASIC program and variables that installed ZX-TERM*80 will still be present in your computer's memory. So let's take a look at it using ZX-TERM*80, and learn a little about its display options in the process. I suggest that you first select either 40 or 60 column mode, since 80-column mode will be a little cumbersome for this part of the tour.

Press SHIFT 5, and you'll be prompted "Program or Variables?" Select "P" for "Program." You are now asked "Sinclair or ASCII mode?" Select "S" for "Sinclair." You will now see the first "page" of the Sinclair program area, starting at address 16520. This happens to be the start of the "0 REM" statement that contains the ZX-TERM*80 machine code. At the bottom of the screen you'll see the possible options:

COPY - Pressing "C" copies the current screen to the TS2040 (only).

SCROLL - Pressing "S" scrolls the display up by one line. Auto-repeat works, so you can simply hold "S" for a continuous scroll. At the right edge of the screen you will see a "scroll bar." This is handy for selective printing, since it will let you know how much of that page had been scrolled, and how much was still part of the last page.

NEXT PAGE - Pressing "N" displays the subsequent page.

SH/SP=ESC - As always, Escape returns you to the main loop.

Since we're looking at machine-code, the display you get won't make much sense. However, there are a few things I should point out. Firstly, the codes from 0-63 and 128-191 are the same as "standard" Clive code; 0-63 are normal video, 128-191 are inverse. You'll see that there aren't any "tokens" (BASIC words like PRINT, LET, and so on). This is because the characters with bit 6 high (64-127 and 192-255) are used for the lower-case and special symbol characters. Again, bit 7 low (64-127) is normal video, bit 7 high (192-255) is inverse.

Use NEXT PAGE or SCROLL to scan through the rest of the BASIC area. Starting at the eighth screen (page), you'll start to see things that make sense; these are the various messages that are printed in the menu and prompt lines. After these is the cover-screen data, after that you'll see the rest of the BASIC driver. Much of it won't make sense because tokens are replaced with lower-case characters, but it will at least look vaguely familiar. Note that KEYWORD tokens (like PRINT, LET, etc.) are inverse lower case letters corresponding to the key that is assigned to that keyword. So PRINT is replaced by inverse "p", RAND by inverse "t", etc. FUNCTION keywords, on the other hand, won't make much sense at all. Graphics are as before.

After ESCAPEing to the main loop, repeat the experiment, but this time select "V" for "Variables" instead of "Program." This will display the contents of the variables area, starting with string variable A$ which holds the ASCII-Sinclair translation tables, character bit patterns and other data used by the program. Toward the end are the rest of the messages that wouldn't fit in the main block. All this is stored in your static RAM, between the high-res display file and the start of the BASIC program area.

Repeat the experiment using "ASCII" mode instead of "Sinclair." See how different the data looks now? When you've seen enough garbage, escape to the main loop.

Normally, you'll use SHIFT 5 to view data that you received from the other terminal. This will be in the ASCII format, so you'd use the ASCII mode option for this. However, you can also use it to preview or review a Sinclair file that you have just downloaded (or are planning on uploading). The ability to view either the program area or the variables area gives you tremendous flexibility.

At this point, it would be wise to note that ZX-TERM*80 differentiates between a normal Sinclair program, and a DATA REM; even though the DATA REM is in the normal BASIC program area. Any time you select PROGRAM, ZX-TERM*80 checks to see if the DATA REM has been initialized or not, and acts accordingly. As a result, if you try to view an "empty" DATA REM, you'll only see the very first character; on the other hand, normal Sinclair programs will be displayed all the way to the end.

Finally, we're in a position where we know enough about the program to get on-line with a BBS and use ZX-TERM*80 for its intended purpose.

INITIALIZE DATA REM: SHIFT 8

Before you get on-line using SHIFT 6 then (C)arrier, you should use SHIFT 8 to set up a "data REM" which will be used to store your received data. This option creates a big line 1 REM line in the program area. The length of this REM (and therefore the amount of data you can store) will be 2BC0h (11200 decimal) bytes with ZX-TERM*80 at the default location. If you locate it higher, it will automatically lengthen this REM by the same amount. NOTE: this automatically

wipes out any program or data you presently have in the "user RAM space." So, if you are planning on uploading a program or other data, beware of using this option! As always, ESCAPE can be used to bail out if you get here by mistake.

Note also, that the SAVE TOGGLE (Shift 1), MEMORY REPORT (Shift 9) and DOWNLOAD (Shift 4) options cannot be used unless you have set up the data REM using Shift 8. If you try to use these functions without a proper DATA REM present, you will be informed "Data Buffer not Initialized" before being returned to the main loop.

SAVE TOGGLE (SHIFT 1):

Pressing SHIFT 1 will toggle the "SAVE" flag. The status of the save flag is shown in the bottom report line; it defaults at OFF when you first start up. When this is ON, anything that comes in from the other terminal (i.e. everything that appears in window 3) is automatically saved into the DATA REM.

Note that what YOU type (i.e. your OUTPUT) is NOT saved in the Data REM. You can, however, change this if you so desire. See the "Some Useful POKEs" section.

MEMORY REPORT (SHIFT 9):

Pressing this after your DATA REM was initialized gives you a report of how many bytes have been used, and how many are left in the DATA REM. The numbers are in hexadecimal. If you're not familiar with hex, the following table might help out:

0001 = 1 byte
000A = 10 bytes
000C = 12 bytes
000F = 15 bytes
0010 = 16 bytes
0040 = 64 bytes
0080 = 128 bytes
00C0 = 192 bytes
0100 = 256 bytes (1/4K)
0400 = 1024 bytes (1K)
0800 = 2048 bytes (2K)
0C00 = 3K
1000 = 4K
2000 = 8K
4000 = 16K

TERMINAL OPERATION

After initializing your DATA REM (SHIFT 8), turning the SAVE TOGGLE ON (SHIFT 1), and adjusting your display as desired, press MODEM CONTROL (SHIFT 6). Pick up the phone and dial a local board. When the board answers, press "C" to turn on the modem's carrier (Westridge only). Physically hang up the phone; the modem will keep it "off the hook," but you don't want acoustic noise to interfere with your data. Hitting ENTER a few times will usually start the other modem, and you'll see its sign-on message in window 3. Use SHIFT 1 as desired to avoid saving unwanted menus and messages.

THE INPUT BUFFER

ZX-TERM*80 uses a novel way of insuring that you never lose an input character. This was necessary because of the relatively long time it takes to scroll those high-res windows. Here's how it works. The "Non-maskable Interrupt" routine was completely rewritten and revectored. Whenever the program is in high-res mode, the new interrupt routine is active. During the time that the normal ZX system scans the keyboard, ZX-TERM*80 checks the modem for an incoming character. If a character comes in, it is stored in a circular "buffer" or "queue," 64 bytes long. This happens 60 times a second, so in principle ZX-TERM*80 might be usable with modems modified for 600 baud (there are, however, no guarantees!).

During the main loop, the program checks the input buffer pointer to see if any new data has come in during the interrupt. If so, it prints the new input until the "print queue pointer" catches up with the "input queue pointer."

This novel approach opens up other possibilities. For instance, you can change character width WHILE CONTINUING TO RECEIVE DATA by quickly pressing shift 7, then 4 or 6 or 8. On getting back to the main loop, the print routine will madly print from the input queue until it catches up, without missing a single character! Note, however, that you must be quick; at 300 baud, 64 bytes are received in about 2 seconds, so you only have 2 seconds to "do your business" before you have to be back in the main loop. If you're too late, you might lose one or more blocks of 64 characters; however, these are only lost on the screen, as they will have been saved to the buffer if the SAVE FLAG was on. See "Useful POKEs", #13, for enlarging this buffer.

Similarly, you can in principle intialize your DATA REM, clear a window, toggle the SAVE flag, reverse the screen, etc. "on the fly." Other operations, like adjusting windows, copying a window to TS2040, etc. are best done while the other modem is in a "wait" state, as when waiting for a reply or command, or while in a "pause" mode after sending an XOFF control character.

BIG PRINTERS AND ZX-TERM*80

The input queue scheme allows continuous printing to big printers. If location 28679 is POKEd to zero or any even number, this feature is deactivated. If it is POKEd to 1 or any odd number, each character will be sent to the printer as well as to screen window 3. This is automatically turned on if you selected an interface during set-up. To change this during operation, quit ZX-TERM*80 with ESCAPE from the main loop followed by "N". POKE 28679, (0 or 1), then get back in with RAND USR 32270.

Note that the printer driver does NOT wait for the printer BUSY line to signal that the printer is ready to receive. There are two reasons for this:

1: This makes it easy to stop automatic printing by simply taking the printer off-line; the program will not get "stuck" as a result.

2: In the event of a printer malfuntion or slow-down, you will not lose any incoming text as a result.

Needless to say, this assumes that you have a printer that is fast enough to handle 300 baud character rates. Normally this won't be a problem with today's dot-matrix (or even most daisy-wheel) printers. However, if you're using an old 10 CPS teletype or something, you may not get the results you expect.

INTERFACE NOTES:

Even those interfaces that are I/O-mapped (like Aerco, Tasman, Byte-Back CP, etc.), you should get proper result although the program is running in SLOW mode. Special interrupt-handling insures that the OUT commands don't happen at an embarrassing moment.

If your interface requires initialization before use (EPROM Services, Memotech RS232, ENER-Z), then you must initialize the interface BEFORE loading and running ZX-TERM*80. Consult the manufacturer's instructions and any supplied software.

There is an "oddity" when using the Memotech RS232, which is actually rather interesting. This device is memory-mapped; as it happens, the memory location used falls smack in the middle of the high-res display file. You'll therefore get a little 8-pixel line that continually changes as new characters come in. This is interesting to check out even if you don't own this device.

Interfaces with on-card EPROMs (Memotech parallel and serial, Aerco CP-ZX, possibly EPROM Services) might have trouble because of mapping conflicts. Normally, RAM tends to override (swamp) EPROMs, but if you experience strange problems the only recourse may be to disable the offending EPROM. Note that the EPROM code is NOT needed for use with ZX-TERM*80; its printer drivers page the input/output ports directly.

ENDING OFF

After you're done with the board, you will of course want to hang up, and perhaps review the data/information you just picked up. DO NOT simply hang up on the board! This is extremely bad etiquette; if you persist on doing this, you might one day find yourself black-balled from your favourite boards. Use the board's main menu "Goodbye" or "Quit" option to exit; wait until the board hangs up before shutting down your modem (using SHIFT 6 with the Westridge, or manually with the Byte-Back).

When you've broken the connection, turn off the SAVE TOGGLE and use main loop option SHIFT 5 to view the data you just gleaned. You are not restricted to the character-width mode you used to gather the data; you might find it handy (as I do) to use the easier-to-read 40-column mode for data gathering, and the (usually) easier-to-format 60 or 80-column modes for review. When you're done, press ESCAPE to exit the main loop.

SAVING YOUR DATA

Exit the main loop with ESCAPE. You are then asked, "SAVE DATA Y/N?" If you answer "N", you are simply returned to BASIC. When using a Westridge, the modem is turned off if you haven't done so already.

If you answer "Y", the following sequence of events takes place:

1: You are prompted for a NAME under which to save the data.

2: The DATA REM is shortened to the length of the actual data.

3: You are prompted to start the tape; pressing ENTER saves the data to tape, in normal Sinclair format.

A "YES" answer to the "SAVE?" option is handy anytime you want to shorten the DATA REM to its actual length. For instance, let's say you just downloaded a file, and wish to upload it for verification or send it on to someone else. If you don't "clip" the REM, you will send back the entire length of the original REM, almost 11K (or more, if you relocated ZX-TERM*80 higher in memory). If you want to shorten the REM to its actual data length, use the SAVE YES option even if you don't want to actually save to tape. Simply follow it through until it starts saving, then press BREAK. RAND USR 32270 to get back in.

WARNING: Once the DATA REM has been clipped short, it cannot be extended further. Attempting to turn the save toggle on under this condition will give a "DATA BUFFER FULL" report.

Again, there's a way around this. Let's say that you want to save what you have so far, without clipping the DATA REM short. For example, you might have gathered some data from one board, and wish to add to that data with info gleaned from the same or another board at a later time. Simply answer "N" on the "SAVE?" prompt. This will return you to BASIC, and you can save to tape manually (immediate mode). The entire, unabridged DATA REM will be saved to tape, along with the "current position" pointer. On loading it back later, you can jump back into ZX-TERM*80 with RAND USR 32270, turn your save toggle on, and add to the data already present.

SUMMARY

SHIFT 8: INITIALIZE DATA
Sets up a data buffer in the program area, for storing incoming data.

SHIFT 9: MEMORY REPORT
Displays memory used and remaining in buffer.

SHIFT 1: SAVE TGL
Turns SAVE flag on/off, as indicated in lower right screen corner.

SHIFT 5: VU DATA

Use to view contents of DATA REM, Sinclair program area, or Variables area. May be viewed in either Sinclair or ASCII mode. COPY any screen to TS2040.

UP/DOWNLOADING

The up/download portion of ZX-TERM*80 uses the Xmodem protocol to transfer entire files from one system to another. When you send a file to another terminal, this is called UPLOADING. If you transfer a file from the other system into your terminal, you are DOWNLOADING.

This portion of ZX-TERM*80 is similar to, and based on MINI-XMODEM by Harvey Taylor. The main changes made were the removal of some minor bugs, compaction of code, and addition of the "Variables" option. For those of you who are not already legal users of MINI-XMODEM, here is an excerpt of the manual, by the original author.

WHAT IS XMODEM?

by Harvey Taylor

Sometimes called the "Christensen protocol," Xmbdem is a method of transferring files over unreliable telephone lines. A reliability of 99%+ is claimed. Basically, the technique transfers 80h (128d) bytes, and tells the other computer; this is the nth block of data, and this is the checksum; do you agree? If the sender and receiver compute different checksums, the block of data is re-transmitted a maximum of 10 times, after which the program aborts. There is a built-in timeout each time a block repeats, to ensure synchronization of the two systems.

When the sender and receiver agree that the last block was correctly transferred, the next block is sent. More detailed info can be found in many CP/M systems in a file called "MODEMPRT.001".

Note that ZX-TERM*80 will upload a Sinclair program up to the display file, or its variables that lie on the other side of the display file. This is different from MINI-XMODEM, which could only upload the program area. Note also that Xmodem can use "Cyclic Reduncancy Check" (CRC), which is another way of catching errors in reception. This is not implemented in MINI-XMODEM or ZX-TERM*80; for this reason, when sending a file to Xmodem you need to specify "RC", for "Receive with Checksum."

USING XMODEM

When you are signed onto a bulletin board which uses CP/M, you must go to the files section (usually by entering "F" at the command prompt). The system will then do some housekeeping, and present you with the CP/M prompt (A1>). To invoke Xmodem you need only type the word "XMODEM". This tells CP/M that the Xmodem file needs to be loaded. Then you have to tell Xmodem what you want it to do; your options are "S" for Send (from the remote system) or "RC" for Receive (with checksum protocol) from your system.

The only other information that Xmodem needs are the drive where the file is located and the filename and filetype, such as "A2:MODEMPRT.001". (See "Using CP/M".)

Once you have typed in this information, the remote system will do some housekeeping (tell you which version of Xmodem it is, how long it will take to transfer a given file, or on which drive it will be received). Then, invoke SHIFT 4 from the ZX-TERM*80 main loop. Press (U)pload or (D)ownload, and whether it is to/from the program or the variables area. (More about that later.) The transfer is automatic from there on. You will see the data being transferred on-screen, in the form of hex numbers. The program will tell you when the transfer is completed or aborted; pressing any key returns you to the main loop.

TYPICAL EXCHANGES

Once you are in the CP/M files section, downloading a file might proceed thus:

```
A1>XMODEM S XMODEM.HLP
```

```
A1: [CP/M prompt]
XMODEM [invoke Xmodem]
S [tell it to send]
XMODEM.HLP [the filename and filetype]
```

If the file was not on A1:, the system would respond:

```
***REQUESTED FILE NOT FOUND***
```

You would then have to specify which drive the file was on; for example,

```
A1>XMODEM S B5:XMODEM.HLP
```

```
S B5 [Send from B5]
```

At this point, the system would typically respond:

```
Xmodem V 7.3
Send time 3 min. 44 sec.
File open Ready to Send
```

At this point, you go to SHIFT 4 in ZX-TERM*80, press "D" to download the file, then press "P" to specify the program area.

UPLOADING

To upload a file you would first load ZX-TERM*80 and relocate it if desired or necessary. Then return to BASIC using ESCAPE followed by "N", and load or otherwise generate the file you wish to upload. This can be any Sinclair program, with or without variables. If you wish to upload a program AND its variables to a BBS or other database, you have to do it in two steps; I recommend first sending the PROGRAM, then the VARIABLES. This is because when someone subsequently downloads the files, he will have to download the PROGRAM area first, then the VARIABLES.

When the desired file is in memory, re-enter ZX-TERM*80 with RAND USR 32270 (or the relocated entry address) and sign on the Bulletin Board. Get into the CP/M files section, and enter a command such as the following:

```
A1> XMODEM RC CMOSUTIL.SIN
```

```
A1> [CP/M prompt]
XMODEM [Invoke Xmodem]
RC [Tell it to receive with checksum]
CMOSUTIL.SIN [Program name and filetype]
```

At this point, the system would typically respond:

```
Xmodem V 9.0
Checksum enabled
File will be received on B0:
Cntrl-S to Pause, Cntrl-X to cancel
File open Ready to Receive
```

At this point you go to SHIFT 4 in ZX-TERM*80, press "U", and select PROGRAM or VARIABLES. The program will tell you when the transfer is done, or of any problems.

NOTE: Many versions of Xmodem have HELP files built-in, e.g.:

A1>XMODEM [newline]

Simply entering XMODEM without any further data will induce the program to type out its help file.

USING CP/M

CP/M is one of the more popular operating systems around. Part of this success is due to the CBBS programs which allow the remote operation of the CP/M system with all the options that entails. A complete description of CP/M is beyond the scope of this document (see Suggested Reading list, below). However, there are some very basic things you need to know.

When CP/M expects a command from you, it will prompt you with A1>. This also tells you that you are in Drive A, User Area 1. You may change to another Drive and User Area by typing in, for example, USER B5 . Note that spaces are significant in using these command words. CP/M uses spaces to know where the ends of commands are. If you tell CP/M to do something that doesn't make sense to it, the system will echo your command with a question mark.

There are several built-in commands which you can always execute in CP/M, the others have to be loaded in from the disk drive (if they are implemented). Among the built-in commands are "DIR ", which stands for DIRectory, and "TYPE ", which will type out a file for you. You can also ask to see a specific Drive and User Area by entering, for example, "DIR E4: ". To have the entire files area displayed, enter "DIR A:*.* $AD ". To exit the CP/M system always use "BYE ".

Files are stored in CP/M with a Name and File Type. Typical examples are:

XMODEM.HLP	(Help)
READLISA.BAS	(Basic)
MEXREF.DOC	(Document)
BOOTASM.SIN	(Sinclair)

The Filename may be up to 8 characters long, and the Filetype three characters. There are a group of non-alphabetic characters which may not be used in Filenames. The Name and Type must be separated by a period. There is a set of standardized Types in use which you can find in the reading list.

SUGGESTED READING ON CP/M

Osborne CP/M User Guide - T.Hogan
The CP/M Handbook - Rodney Zaks
Mastering CP/M - A.Miller
CP/M Primer - Murtha & Waite
CP/M Revealed - J.Dennon
Using CP/M - Fernandez & Ashley
The Programmer's CP/M Handbook - A.Johnston Laird

UP/DOWNLOAD (SHIFT 4)

This command will display the options, "(U)pload (D)ownload (T)ransfer". As always, and in subsequent queries, ESCAPE returns to the previous level. Similarly, ESCAPE will abort an up/download, should it get stuck or otherwise bomb.

After selecting U or D, you are asked whether you wish to transfer from (or to the PROGRAM or VARIABLES areas. Note that if you are downloading into the PROGRAM area (which would be used for ASCII documents, etc. as well as actual Sinclair programs), you must have previously initialized the DATA REM using SHIFT 8. Memory is tested, so if the file you're trying to download is longer than your available space, you will be informed of this and the program will abort.

When downloading to the variables area, any variables that may have been present will be overwritten. Again, you cannot load a longer file than there is space for.

You may be wondering what the (T)ransfer option does. Good question. This is used after downloading a Sinclair program to "install" the program in it normal run-time location. Although it is already in the "program area" after downloading (i.e. the DATA REM), it is only stored as data, not as a runnable program. The Transfer command shifts the program downwards by 11 bytes, then figures out where the display file should go, and installs it there before returning to the main loop. This must be done before attempting to run the program, or loading any variables (if applicable).

DO NOT try to transfer non-Sinclair files. Attempting this will cause a crash (at worst) or give a "Transfer Error" report (at best).

The Transfer option is NOT needed for variables; the variables are already stored in their final resting place just after the display file. After downloading, ZX-TERM*80 automatically takes care of the housekeeping to insure that the E_LINE, STK_BOT and STK_END system variables point to the right places. Again, however, if the received file does not "make sense" as Sinclair variables, the variables area is cleared (to prevent trouble) and you will get the report "Transfer Error."

FURTHER NOTES ON UP/DOWNLOADING

Why did I go to the trouble (and it WAS trouble) to implement a variables up/download? The reason is that many programs are useless without their associated variables. For example, variables may have been defined manually, to save program space. Similarly, most word-processors store their generated text in the variables area. ZX-TERM*80 is unique in allowing you to transfer the program area and the variables area as separate files.

When uploading a Sinclair program and variables to a BBS, it may be done in either order. However, when downloading, you MUST download the program first, then download the variables. If you try to use the reverse order, the variables just downloaded will be overwritten by the generation of the DATA REM using SHIFT 8.

It might be desirable to download variables specific to a certain program (e.g. Wordprocessors like Memotext or WordSinc, alternate character data sets for ZX-TERM*80, etc.). In such cases, load ZX-TERM*80 first, then quit to BASIC and load the object program. Then re-enter ZX-TERM*80 and download the variables (the text files and any other "housekeeping" variables or buffers that the program may require).

Remember, you must use the (T)ransfer option to install a downloaded program before attempting to load its variables. This was made a user function rather than being automatic, because ZX-TERM*80 has no way of knowing whether a downloaded file is a Sinclair program or just data. It is also wise to use the VU DATA command (SHIFT 5) to look over the program before transferring, to insure that it is indeed what you assume it to be.

There is a timeout of 10 seconds built into Xmodem (and the Xmodem portion of ZX-TERM*80), so every once in a while (when first downloading and on repeating blocks), the program will appear to be doing nothing. However if nothing happens for longer periods of time, there is always the ESCAPE command key if you want to bail out. This will not always work because sometimes the program is busy testing ports and suchlike, but if something has gone fatally wrong, then holding the ESCAPE key (shift space) will usually return control eventually. When you invoke this, data is shortened at best, corrupted at worst, and you must try again from scratch.

SUMMARY

SHIFT 4: UPLOAD, DOWNLOAD, TRANSFER

UPLOAD: Send a data file or Sinclair program to the other terminal.

DOWNLOAD: Receive a data file (which may be a Sinclair program) or Sinclair variables set from the other terminal.

TRANSFER: If the downloaded file is a Sinclair BASIC program, install it in its run-time location in RAM. Program will then be runnable after quitting ZX-TERM*80.

PROGRAM: Use for ASCII text files or Sinclair programs.

VARIABLES: Use for Sinclair variables sets ONLY.

USER OPTIONS

After loading the original ZX-TERM*80 from tape, you'll be taken through various prompts to set up the system to your needs and tastes.

MODEM TYPE:

You may use either the Westridge (TS2050) or the Byte-Back MD-2 modems. If you have some other type of modem (possibly home-brew), it might be possible to modify ZX-TERM*80 to suit. Study what the BASIC driver does, and modify the program lines appropriately. [These modify the MODEM CONTROL message, and change the actual MODCON routine as required. Most importantly, the various I/O subroutines are matched to the modem. These are four bytes each (long enough for either a memory-mapped or I/O-mapped command, plus RET), and are in the sequence CONTROL IN, CONTROL OUT, DATA IN, DATA OUT (12 bytes total)].

RELOCATE?:

If you answer "Y", you will be asked for the new run-time base address (in decimal). As supplied, the main program runs in the 4K at 7000-7FFFh (28672-32767d). Your RAMTOP will be set to the relocation address, and the ZX-TERM*80 program boots just beyond. A couple useful relocations are:

32768 (8000h-8FFFh) - puts ZX-TERM*80 just beyond the 32K mark. This will let you load (and subsequently upload) a full 16K program, while leaving 9000-BFFFh free for other programs or data.

45056 (B000-BFFFh) - puts ZX-TERM*80 at the highest possible address in 32-64K of RAM. (As with TS1500's with 16K external RAM; the minimum requirement for the TS1500. Or with TS1000's with 32K of user RAM, as a Sinclair 16K plus a Memotech 16K, or 64K.)

Both of these relocations assume that you have a decoded M1 NOT line to the ULA, or that you are using a TS1500. Another interesting relocation is described in the next section.

In principle, it would have been possible to allow locating ZX-TERM*80 at lower locations than the default; however, this would give you even less available memory. If you study the memory map, you'll see that the entire 8-16K region is already in use by the program, so it would not be possible to move it down there.

This manual assumes the default location for the program; all POKE addresses, etc. will relate to this. You will have to change these as required for your new relocation. Also, the RAND USR address will be offset from the default value of 32270 by the amount of relocation. This address will always by 3598 higher than your new base address.

PRINTER INTERFACE:

Select whichever printer interface you are using with your "big" printer.

As mentioned earlier, if you have the Memotech interface, or any other interface that maps an EPROM in the 8-16K region, you MIGHT run into trouble because of conflicts. In my experience, static RAM chips seem to be able to "override" most EPROMs; on my systems, enabling both the Memotech interface and the static RAM causes the RAM to "swamp" the contents of the EPROM. Result - the system works,

even though theoretically it shouldn't. So don't simply assume that it won't work because of mapping conflicts. Note that the EPROM code is NOT needed for proper operation; ZX-TERM*80 pages the interface's ports directly.

Incidentally, the Aerco 2068 interface will work just as well (perhaps even better, since it lacks the EPROM card) as the 1000 version, at a lower cost.

Also as mentioned earlier, ZX-TERM*80's driver does NOT check for printer BUSY status. It is therefore up to you to provide a printer that is fast enough to keep up with the data rates used in ZX-TERM*80 (not normally a problem, but you should be aware of it).

CHANGE MODE WORD:

Both the Westridge and the Byte-Back modems use the Intel 8251 USART chip. Its operation is controlled by a "mode word" which can be modified by you after loading the program. You can also change this in the course of your modemming by returning to BASIC, POKEing address 28707 with the desired value, and re-entering ZX-TERM*80.

Specific information about this mode word can be found in the data sheets for the 8251. However, here are a few of the more common settings:

DATA	STOP	PARITY	MODE WORD
8	1	none	4F (79 d)
7	1	even	7Bh (123 d)
7	1	odd	5Bh (91 d)
5	2	even	F3h (243 d)

These numbers apply to the Westridge modem. For the Byte-Back, the only difference is that in each case, the number will be one less (bit 0 reset instead of set).

Note that the Xmodem protocol will not work properly with any setting other than 8 data bits, 1 stop bit, and no parity. This corresponds to a value of 79d, which is the value in ZX-TERM*80 as supplied.

MEMORY?:

Further detail on what this does exactly is covered in the next section. For now, you just have to know that "1" is for 16 or 32K systems, and "2" is for 48 or 64K machines (48-64K region has actual memory in it).

INSTALL OTHER OPTIONS?:

If you answer "Y" to this, the program will stop. You can now make any other POKEs that you wish, as described in the "Useful POKEs" section. Re-enter the program after doing your POKEs by entering GOTO 500. NOTE: Never use RUN or CLEAR with this program, or the character sets, translation tables, and other data stored in A$ will be lost.

SAVE CONFIGURATION?:

After you've set up your required parameters, you can save the re-configured program to tape using this option. When later reloading it, you won't have to do the initializing all over again.

MEMORY MAP

Here is how memory is allocated when ZX-TERM*80 is loaded. Addresses are given in hex, with the decimal values in parentheses.

```
0000-1FFF (0-8191)          System ROM
2000-35FF (8192-13823)      High-resolution display file
3600-36FF (13824-14079)     ASCII-to-Sinclair conversion table
3700-37FF (14080-14335)     Sinclair-to-ASCII conversion table
3800-3BFF (14336-15359)     40-column character bit patterns
3C00-3DFF (15360-15871)     60/80-column character patterns
3E00-3E9C (15872-16028)     Redefined keyboard code tables
3E9D-3FFF (16029-16383)     Prompt messages, block 1
4000-407C (16384-16508)     Sinclair system variables
        N.B.: 403C-4060 used for dummy display file whenever in high-res mode.

        UP/DOWNload also uses some of these for temporary variables.

407D-[DFIL] (16509-xxxxx)   BASIC program (or DATA REM)
[DFIL]-[VARS] (xxxxx-yyyyy) Normal (low res) display file
[VARS]-6FFF (yyyyy-28671)   Variables, workspace, and stacks
7000-703F (28672-28735)     ZX-TERM*80 system variables
        N.B.: IY register is revectored to base address=7000h while in  hi-res
        mode
7040-707F (28736-28799)     Input buffer queue
7080-7DE9 (28800-32233)     ZX-TERM*80 main machine code
7EDA-7E0D (32234-32269)     Printer patch routine
7E0E-7E2D (32270-32301)     Entry routine
•7E2E-7FFF (32302-32767)    Prompt messages, block 2

C03C-C060 (49212-49248)     Dummy display file "echo" (16K) or copy (32K+)

FFD9-FFFD (61440-65533)     Dummy display echo buffer area
        (64K only)
```

THE DUMMY DISPLAY FILE

The WRX16 high-resolution display system relies on a "dummy display file," 37 bytes long, for its operation. In ZX-TERM*80, this dummy display file is located in the "printer buffer," spilling over into the first four bytes of the calculator memory. This does not interfere with normal operation, since on return to BASIC this region is restored to normal when the next immediate command is entered. Similarly, the ROM system's use of this does not affect ZX-TERM*80 since its 2040 printer routines does not use PRBUFF, and ZX-TERM*80 does not invoke any floating-point operations.

On a 16K or 32K machine, the hi-res system relies on the high-memory (48-64K "echo" of this dummy display file, which appears because of the 16K pack's incomplete decoding. Note, therefore, that 32K systems which have been fully decoded will not work. 64K packs, on the other hand, WILL work, since there is "real" (as opposed to "phantom") memory in the top 16K. To make the same program work without change on 64K machines, the hi-resolution set-up routine creates a duplicate of the DDF exactly 32K higher (C03C-C060h). Having this in two different places could cause trouble in some applications, however, so I went to a bit of additional effort to make it as user-transparent as possible.

If you have a 16K (only) ZX81/TS1000, or a 32K ZX81/TS1000 or TS1500, there is no problem. With 64K, however, the problem is that this high-memory copy will overwrite whatever else you might have up there. A classic example is Memotext V3, which uses this area for its data tables. Here's how I got around this. If

you select "64K" on start-up, the program will "save" whatever data lies in this region. It does this by transferring the region from C03C-C060 up to the very top of memory (see the memory map). Only then does it create the high-memory copy of the DDF. On return to normal video mode, it automatically "recalls" the data, putting it back where it belongs.

However, if you try to use this option with 16 or 32K packs, the lack of complete decoding will overwrite the top of the ZX-TERM*80 messages file. This is why the start-up option is necessary.

The best way to explain this further is with another relocation example. Let's say that you want to use ZX-TERM*80 with Memotext V3, which uses the region from A870-E942h (including its help file). So, a good place to relocate ZX-TERM*80 is to 9870h (39024d) - A86F.

The procedure is very simple; LOAD Memotext, and use its QIT function to return to BASIC. Now LOAD ZX-TERM*80, and follow its prompts to set it up. Enter 39024 for "relocation address". After getting into the main loop, ESCAPE back to BASIC and enter NEW. Now jump into Memotext's cold boot address, or load a (previously created) set of files and enter Memotext's warm boot address. When your files are as you want them, QIT Memotext and jump back into ZX-TERM*80 to upload your files, or to download other files (using the "Variables" option).

Be aware that when entering Memotext from a cold boot, you MUST use the "Preserve RAMTOP" entry address, else you will zoof ZX-TERM*80.

Here's the moral of the story. Even though the dummy display file lies smack dab in the middle of Memotext's printer table, no harm will be done since the "dummy display buffer" at FFD9 is well out of harm's way. As long as you don't try to use Memotext while in hi-res mode (an obvious impossibility unless you know something I don't), everything will still be copescetic.

A PHILOSOPHICAL DIGRESSION

What makes an excellent program? Is it ease of use, flexibility, user options, advanced features, graphics, or adaptability? Which are the most important? When dealing with the Sinclair/Timex, we must also include memory economy in the list of requirements. Obviously there must be some trade-offs and compromises in any real system.

Perhaps the best that any programmer can do is to write softwa.. that he himself would enjoy using. Yet that isn't enough, since a programmer is often not a "typical" user. So he has to keep his ear to the ground to get an idea of what other users want to see in a program. Even then, how are these requirements to be implemented? Does he assume that the user wants to be led by the hand through mazes of menus and help files? Or does he stray in the other direction and make his work intelligible only to the most hardened hacker?

These and many other questions arose during the development of ZX-TERM*80. The result of trying to answer these questions is a program that is very easy to use, yet has great flexibility for adaptation. The price for all this is that implementing some of these options requires at least a smattering of computer know-how, particularly that peculiar brand of savvy required to make the ZX/TS "sing and dance." I realize that the following sections will be over the heads of many casual users. If it all seems like gibberish, don't worry about it; to don't HAVE to understand this stuff to use ZX-TERM*80 effectively. Do make an effort to comprehend its implications, though; take it slowly, talk it over with other users, gradually it might start to make sense. Experiment! The rewards are great, not only as relates to this particular program, but also generally.

SOME USEFUL POKES

Though I've tried to include the most useful user options in the BASIC loader, there are a number of other changes that you can make to customize ZX-TERM*80 to your needs and tastes. In this section, I'll give some interesting options that are available to you. Each sub-section starts with descriptive text that details how the program works presently, and how it is changed by implementing the POKEs. Then the required POKEs are given; the format is as follows:

POKE address1 (address2), new (old)

"address1" corresponds to the actual run-time location of that byte, in the default location. This is so that you can quit to BASIC, make the POKE(s), and return to ZX-TERM*80 to experiment with its effect.

"address2" (in parentheses) corresponds to the location of that byte within the 0 REM statement in the BASIC loader. If you decide that you like a change, you can re-load ZX-TERM*80 and make the modification to the loader, and save the modified loader to tape. This way, subsequent loads will already have the change implemented. Note that "address2" is always 12144 bytes lower than "address1."

"new" is the value that should be POKEd to implement the change. "old" (in parentheses) is the original value of this byte, as supplied. If you don't like a particular change, you can reverse it by POKEing the original value back in.

IMPORTANT NOTE: Values followed by an asterisk (*) will change if the program is relocated. The best way to circumvent problems is to do your experimentation with the standard location. If you decide to implement a change permanently, re-load the original program and make the POKE(s) to the loader. Then relocate the program; the relocator will automatically take such changes into account.

1: DISABLE CONTROL-CODE ECHO

If you don't want control codes to echo in window 3, use these POKEs:

POKE 30743 (18599), 0 (205)
POKE 30744 (18600), 0 (230)*
POKE 30745 (18601), 0 (117)*

2: AUTO-REPEAT RATE

You can change the rate at which keys auto-repeat. The present rate is what I personally find the most pleasing; however, you can halve the rate (slower), double it, or even quadruple it. Yet faster rates are possible, but not recommended.

POKE 29446 (17302),n (127)

n=255 (half speed)
n=127 (present setting)
n=63 (double speed)
n=31 (quadruple speed)

3: KEYBOARD DEBOUNCE

As supplied, the keyboard sensing routine must find two consecutive identical readings before a keypress is registered. This results in fast action, but may give unwanted multiple characters with bouncy keyboards. Change this by making the following POKE; n can be as high as you like, but no less than 1. Values between 4 and 8 should be adequate.

POKE 29469 (17325),n (2)

4: SAVE OUTPUT IN DATA REM

As supplied, your typed output is NOT saved in the DATA REM when SAVE TOGGLE is ON. In other words, only INPUT (what shows up in window 3) is saved, not the stuff you type into window 2. This is how you will probably prefer it most of the time. Most boards will echo important data (such as menu choices) anyway, whether you want them to or not. Usually, you won't want to eat up memory space with your typed commands, etc. You also don't want to be forever toggling SAVE TOGGLE (SHIFT 1). If you really want your messages, etc. to be "part of the record," you can always have the board repeat them back to you using its LIST or EDIT options.

On the other hand, it can be convenient to have a "Textwriting" mode of operation, as with the original Mini-Xmodem. This would allow you to compose a message (article, etc.) before getting on-line, then transfer it like any other upload and save considerable phone time. You would simply initialize your DATA REM, turn SAVE TOGGLE ON, and type your message. Turn SAVE TOGGLE OFF, and quit to BASIC via the SAVE YES option (to truncate the DATA REM to the actual data length). Jump back into ZX-TERM*80, get on-line, and upload the DATA REM ("Program").

This might also be handy if you are having a two-way conversation with another user in a direct connection.

POKE 30112 (17898),156 (140)*
POKE 30113 (17899), 113 (112)*

5: DISABLE CONTINUOUS PRINTING

To disable continuous printing to your big printer, simply take the printer offline. Alternately,

POKE 28679 (16535),0 (1)

6: DISABLE OUTPUT PRINT

Unlike what goes to the DATA REM with SAVE TOGGLE ON, I set up the program to print EVERYTHING (including your typed output) when in continuous printing mode. This is because (usually) paper is less precious than memory space. However, you can turn off your typed output (what appears in window 2) with the following:

POKE 30109 (17965),240 (234)*
POKE 30110 (17966),116 (125)*

7: MODEM CONTROL DEFAULT (Westridge only)

The program automatically starts up the modem and picks up the phone when entering the main loop by pressing a key after the copyright screen. If you don't want it to initialize the modem on start-up, use this modification:

```
POKE 32295 (20151),0 (205)
POKE 32296 (20152),0 (164)*
POKE 32297 (20153),0 (113)*
```

8: MODEM HANGUP DISABLE (Westridge only)

Conversely, you might not want the program to automatically shut down the modem on return to BASIC. To disable this feature,

```
POKE 30129 (17985),0 (205)
POKE 30130 (17986),0 (180)*
POKE 30131 (17987),0 (113)*
```

9: NEWLINE=LF INSTEAD OF CR ON RECEIVING

As supplied, ZX-TERM*80 recognizes a carriage return (CR = ASCII 13 dec.) as a newline character. If you'd prefer to have the linefeed (LF = ASCII 10 dec.) code signal a newline, make the following changes. Note that you can't change the loader with a POKE, since this data is in string variable A$ (the stuff that gets booted into low memory). As a result, changes to the loader have to be done as shown in the LET statement after each POKE.

```
POKE 13834,64 (67)
LET A$(11)=CHR$ 64 (67)

POKE 13837,67 (64)
LET A$(14)=CHR$ 67 (64)
```

10: SEND LF INSTEAD OF CR

Similarly, the program sends a CR when you press newline. If you want it to send an LF instead, make the following changes (again, change to the loader has to be made using a LET statement).

```
POKE 14144,10 (13)
LET A$(321)=CHR$10 (13)
```

11: RETURN INSTEAD OF ERROR

If you want the program to simply return from the machine-code instead of giving ERROR Q on return to BASIC (SAVE NO), enter the following POKE. This would be handy if you plan on calling ZX-TERM*80 from within a BASIC program, and want the program to keep running after quitting.

```
POKE 30132 (17988),201 (207)
```

12: RETURN TO ZX-TERM*80 AFTER "SAVE YES"

If you want the program to return to ZX-TERM*80 after selecting the "SAVE YES" option, use the following:

```
POKE 31049 (18905),195 (205)
POKE 31050 (18906),98* (43)
POKE 30151 (18907),120* (15)
```

NOTE: the relocator will NOT automatically adjust this location; so if you relocate the program, change the address represented by the second two bytes as required.

13: EXPAND INPUT BUFFER

The input buffer is presently 64 bytes long, located at 7040-707Fh. It can be relocated to any 256-byte block of memory, and at the same time expanded to 256 bytes. This has two significant advantages: 1) It frees up 64 bytes within the program's machine code for other patches, etc. 2) It gives you up to about 8 seconds to adjust windows, etc. without losing any input.

For example, if you have 64K you might want to locate the input buffer to FE00-FEFF. (FF00-FFFF is not allowed, as explained in the section on the dummy display file). FEh is 254 decimal, which is the value you would use for "n" in the 2nd and 5th POKEs below.

```
POKE 29007 (16863),0 (64)
POKE 29008 (16864),n (112)
POKE 29014 (16870),255 (63)
POKE 30168 (18024),0 (64)
POKE 30169 (18025),n (112)
POKE 30172 (18028),255 (63)
```

N.B.: This is the only "Useful POKE" that MUST be made AFTER relocating. Else the relocator will move your new buffer address! So do ALL OTHER POKES as described, then GOTO 500. Relocate as desired, then press BREAK. Now do the above POKEs. Enter RETURN to get back into the BASIC boot program.

As supplied, the default input buffer contains two very useful machine-code subroutines for big-printer users. These are described in Appendix V. If you have more than 16K, it is highly recommended that you do "Useful POKEs #13," so that you can use these routines.

14: NO LF TO BIG PRINTER ON NEWLINE

As supplied, the programs send both a CR and an LF to your big printer. If you get double line feeds, use this POKE:

```
POKE 32267 (20123),0 (10)
```

APPENDICES: HACKER'S DELIGHTS

ZX-TERM*80's has two different character set tables; one for the 40-column characters, and one for the 60 and 80-column characters. Here is how the characters are arranged.

CHR$ 00h through 3Fh are the same as the corresponding "normal" Sinclair characters. As in the Sinclair system, bit 7 is used as an inverse flag, so CHR$ 80h through BFh are the inverses of the normal (capitals) character set, also as in the normal system.

Bit 6 high is no longer illegal. In fact, bit 6 high is used for lower-case and special symbols. In addition, five of the codes (which would correspond to "lower case graphics") are used for system control codes. These are:

40h = carriage return ("Enter" - analogous to the old 76h)
41h = Symbol mode (analogous to "function" = 79h)
42h = Escape (shift space) (analogous to BREAK)
43h = Null (same as ASCII 00h, no Sinclair analog)
44h = ESC (no Sinclair analog; used for printer control)

41h is also used in the keyboard tables, for any undefined keys (symbol mode tokens, etc.) 43h takes a bit of additional explanation. Some boards send "nulls" = 00h (not to be confused with "space" = 20h) at the start of each line, tp accomodate software that requires down-time to scroll its screens (unlike ZX-TERM*80). These are simply ignored by the printing routine. ZX-TERM*80 maps these nulls to 43h, because 00h is of course the Sinclair space.

The codes from 44h through 4Ah are unused. 4B through 5B are the "special" symbols. 5C through 6? ("lower case numbers") are also unused. 66 through 7F are the lower case alphas. Again, bit 7 high is simply the inverse video equivalent.

So there are a total of 17 unused codes, with their inverses. As supplied, these will simply print spaces (as if they are encountered during a program review, etc.). You can, however, create your own graphics characters to correspond to these codes; furthermore, there is no reason why you couldn't change the other characters (as the Sinclair graphics) to suit your own purposes, IBM-compatible graphics can therefore be mimicked, at least in principle. All the data is in the form of tables in low memory, from 3600h through 3E9Bh. The following appendices detail how these tables are arranged, what they mean, and how you can modify them.

The best way to experiment with these is to load ZX-TERM*80, then quit to BASIC. Load up a tool such as Hot Z, a character set designer, or perhaps even just a BASIC PEEKer/POKEr. Make the changes you want to incorporate, then reload ZX-TERM*80. BREAK at the cover screen, then RAND USR 20787. This will boot the modified tables into dimensioned string variable A$(2560). You can then save the modified program by entering GOTO 0, then press S to save.

Similarly, the modified A$ can be uploaded to bulletin boards, or directly to other users using the "Variables" option. As mentioned earlier, we won't mind seeing such modified variables sets ONLY on the boards; we had just better not see the program itself.

APPENDIX I: MAPPING TO ASCII AND BACK

The first two data tables are both 256d bytes long. The first, at 3600-36FF, maps from ASCII to our modified Sinclair characters. The second, at 3700-37FF, maps from our Sinclair set to ASCII. The best way to describe how these work is by detailing the translation routines:

```
74EB 2636     TSNC LD H,36
74ED 6F       TLT* LD L,A
74EE 7E            LD A,(HL)
74EF C9            RET
74F0 2637     TASC LD H,37
74F2 18F9          JR TLT*
```

H is loaded with the base address of the appropriate table. L is loaded with the character to be translated, which gives the offset into the table. A is then loaded with the contents of that location, and the translation is complete.

Another way to look at it is that the low byte of the address is the code to be translated, and the contents of that location is the translation. Very simple, very fast, and very flexible.

You'll have to print out the contents of these tables to see what I've done here. The normal upper and lower alphas, numbers, and symbols are straightforward, and there is no need (or desirability) to ever change them. Most of the rest is pretty arbitrary, and you can do with it as you please.

The two tables are pretty much complementary; however, there are some exceptions. If you think about it for a while, you'll realize some of the reasons. For example, ASCII controls (01 through 1Ah) are mapped as the inverse caps for control purposes (shift 3). See the table at 37A6-37BF and you'll see what I mean. On the other hand, an incoming carriage return (control M) should cause a newline, not print an inverse M; similarly, an incoming linefeed (control J) should simply be ignored. That is why the entries at 3601-361A don't map exactly to the section from 37A6-37BF.

Some of the possibilities may be dawning on you. The bit 7 high ASCII characters (3680-36FF) can be mapped to just about anything you like, including but not limited to the unused Sinclair codes mentioned earlier. Conversely, the matching codes in the 3700-37FF table can go back the other way.

At this point I should mention that whatever changes you make are YOUR responsibility. I did what I could to leave ZX-TERM*80 open to hackery, but we cannot accept responsibility for improper operation as a result. The program works as supplied; if you mess it up as a result of your poking around, that's your problem. If all else fails, you can always load up the original and try again.

APPENDIX II: 40-COLUMN CHARACTERS

The next table runs from 3800h through 38FF, and contains the character bit patterns for the 40-column characters. 3800-3807 is CHR$ 0 (space), 3808-380F is CHR$ 1 (graphic on :`, etc. just like the ROM table at 1E00-1FFF. Unlike this table, though, which is only 6-bit, we are using a 7-bit system to accomodate the lower-case alphas, etc. This means there are a total of 128d characters in this table. If you look at 3A00-3A57, you'll see a bunch of zeros corresponding to the "characters" from 40h-4Ah which are either system controls or unused. Similarly, the "lower case numbers" run from 3AE0-3B2F (characters 5C-65). You can put whatever graphics you want here, presuming that your ASCII table has been appropriately re-mapped to point to these. Similarly, there's no reason why you couldn't change the Sinclair graphics (characters 00-0A) to whatever you like.

Only the first six bits (bits 7-2 = most significant) are used. However, I suggest that you use only the first five, if you want a 1-bit space between characters. The two low-order bits (0 and 1) should always be reset (0).

Remember, bit 7 (codes 80-FFh) is used as an inverse flag. Some characters might look fine in normal video but really goofy if reversed; keep this in mind if you plan on using the inverses.

APPENDIX III: 60/80 COLUMN CHARACTERS

Next up, at 3C00-3DFF, are the 60/80 column characters. The way these are arranged is a little trickier.

Since these use only four bits, it is possible to save 1/2 K of memory by "packing" the bit patterns. Here's how it works. The left (most significant, = bits 7-4) nybble represents the EVEN numbered characters. The right nybble (bits 3-0) are the ODD numbered characters. So, for example, the entries at 3C00-3C07 have CHR$ 0 (space) in the left nybble, and CHR$ 1 (graphic on;) in the right nybble. Similarly, 3C08-3C0F has CHR$ 2 (left) and CHR$ 3 (right), etc.

If you create new patterns for the 40-column table, you should work up corresponding entries for the 60/80 column table so that all modes will still work as intended. This can get a little tricky, especially for the 80-column mode. Here are a few things you should keep in mind:

In 60-column mode, all four bits are printed. In 80-column mode, only the first three bits (most significant) are used. It is therefore highly recommended that you use only the first three bits, keeping the fourth (bits 4 and 0) reset. This way, your characters will still be recognizable, if perhaps scrunched.

This isn't always possible, however. For instance, there's no way to do a believable "2" symbol in only 3 bits. In this case, we just have to take our lumps and love it, in 80-column mode. Similarly, graphics might look stupid if there is a 1-bit space between them; for these, we just have to live with whatever happens in 80-column mode.

APPENDIX IV: THE KEYBOARD TABLES

I swiped and modified the ROM routine 07BD to point to my own keyboard code tables. This allows easy "typewriter action" to be implemented, as well as permitting flexibility in changing them, or assigning other codes.

The first such table is at 3E00-3E26. These are the unshifted codes (lower case). The second table is at 3E27-3E4D, and are the shifted codes (capitals). The third table is at 3E4E-3E74, and are the unshifted SYMBOL-mode keys. Finally, the last table at 3E75-3E9B are the shifted SYMBOL-mode keys.

Note that each of these tables is exactly 39d bytes long, corresponding to the 39 keys (except shift) on your keyboard. Looking over my tables, it may not be obvious how they are arranged. Instead, use as a template the original tables in the ROM, starting at 007E. It simply scans each row of keys, starting with Z-X-C-V and moving clockwise around the keyboard. All groups are five keys except the first, which of course includes "shift" and therefore only has four valid entries.

Looking over my tables, especially the SYMBOL mode tables, you'll find a lot of 41h entries. These represent the unused ("illegal") keycode combinations; in other words, any key that does not have a single symbol on it (the only exceptions are space=N, and shifted space, covered in the next appendix). There's absolutely no reason why you can't define these for other things.

The circle is thus completed. The keyboard tables specify what codes are generated by pressing a certain key in either mode (normal or symbol). The character bit pattern tables specify what these codes look like on your screen. The translation tables define how the other terminal will perceive them, and how you perceive what you receive.

APPENDIX V: PRINTER MAGIC

ZX-TERM*80's flexibility still staggers me sometimes, and I wrote the silly thing. Having a program that can speak ASCII as well as Sinclairese makes for some interesting possibilities with other ASCII-based peripherals such as printers. Its continuous print mode allows your printer to be a natural extension to your modem.

When you send controls to the other modem, you also send it to the printer. This can cause problems, if for instance your control happens to match the printer's code to "go into Japanese character set" or something. So to be safe, take the printer offline before sending controls that could wreak havoc.

On the other hand, there are at least a few controls that have serendipitous results. For example, some boards use XON/XOFF (Cntrl Q and S) to pause or restart transmission. Epson-compatible printers use exactly the same codes to take the printer on or off-line from software! So when you pause the board using Cntrl S, you automatically take your printer offline at the same time.

Similarly, some boards can be told to enable an audible "beep" signal. Your ZX/TS, of course, can't be called into service for this purpose, being dumb if not deaf... but your printer can, using the BEL code (ASCII 07 = Cntrl G). This can be handy, say, after receiving a long message, so you can do something else while all the stuff is coming in. When you hear your printer calling you, come back and continue modemming.

Just about every printer uses the control codes to "shift in" or "shift out" your characters (different character widths, etc.), back-space, tabulate, and so on. So there's no reason why you can't "send" a control character to set up your printer as desired, before actually getting online.

Perhaps serendipitously, perhaps because there is more standardization than is at first apparent, the codes from 1C-1F are unused on most printers; lucky for us, since we can only send Cntrl-A through Z (01-1A).

But what about ESC (=1Bh)? This important control code is used for all kinds of other printer set-up options, like enhanced, NLQ, or double-strike, underlining, and so on. Heh, heh. ZX-TERM*80 has yet another trick up its sleeve. Go to symbol mode (shift enter) then press shift space (for ESC). This will send an ESC code, which you can follow with the "parameters" to implement the desired ESCape control. For example, our Gemini 10 uses ESC E to set up enhanced mode. We simply hit shift enter (symbol mode), then press shift space (ESC), then press E. Follow with a carriage return (enter), and we're in enhanced mode.

Note that there is no guarantee as to how the other terminal will react to such printer controls; for this reason, it's best to set up your printer before getting online with a bulletin board.

On the other hand, this opens all kinds of neat possibilities if you're communicating with other users in a direct connection, or via a local board. As long as you know what the other guy's printer requires, you can remotely do all kinds of fancy stuff and just imagine his jaw dropping while his printer goes through "impossible" gymnastics.

Sending bit-mapped graphics? Should be quite possible. I'll leave it to you enterprising hackers to figure out how to get ZX-TERM*80 to download pictures from databases and dump them to the printer.

Now you might see why I decided against incorporating extensive "big printer" routines. The possibilities are endless, and it would have been impossible to work up a truly comprehensive printer package for all the different printers and interfaces on the market. However, there are at least two routines which will come in handy for most of you; these are "Dump data buffer to printer" and "Translate data buffer to ASCII." These routines are included in the "input buffer queue" as supplied. If you have more than 16K and do the "useful POKE #13", these routines will be accessible to you.

Addresses given are the default location. You will have to re-compute your actual address by adding the relocation offset. This offset is given by (your relocation address)-28672.

DUMP DATA BUFFER TO PRINTER

RAND USR 28736 dumps the entire ASCII file in the data buffer to your big printer. In this case, we DO check for printer-ready status. As supplied, this is for the Tasman Type B interface. The bytes from 28754 to 28761 (16610-16618 in the loader) should be changed to match your interface:

AERCO/OLIGER 2068: 219, 127, 203, 103, 32, 250, 0, 0
BYTE-BACK: 203, 31, 203, 127, 40, 250, 0, 0
OLIGER 1000: 58, 255, 255, 254, 24, 32, 249, 0
ENER-Z: 211, 5, 219, 1, 230, 1, 32, 250
EPROM SERV: 219, 155, 23, 56, 251, 0, 0, 0
MEMOTECH CP: 219, 63, 15, 56, 251, 0, 0, 0
MEMOTECH RS232: 58, 13, 47, 230, 16, 40, 249, 0
TASMAN TYPE C: Poke 2nd byte (16611) with 251

TRANSLATE DATA BUFFER TO ASCII

If your file is in Sinclairese, you will have to translate it to ASCII before printing it out using the previous routine. Do this with RAND USR 28780 (plus your relocation offset). Note that you must call this ONCE ONLY. Else you'll convert your file into pure garbage.

NOTE: to translate the file from ASCII to Sinclair instead, POKE 16648 in the loader (=28972 in the default run-time location) with 54 (36h) instead of 55 (37h).

APPENDIX VI: WESTRIDGE TIPS

Having trouble with your Westridge modem? Does it put out a garbage carrier, or otherwise act up? Chances are, this simple fix will take care of it.

The Westridge modem comes with a gosh-awful long cable. To make matters worse, each wire has a little ferrite bead on it, to help suppress radio/TV interference. Unfortunately, this adds enough inductance in the lines to mess up the data timing. This will especially be a problem with fully expanded systems that already have a lot of bus capacitance. Capacitance + inductance = a tuned circuit, which can oscillate and cause ALL KINDS of trouble.

Cut the cable to about 1/2 its original length, and re-solder it to the expansion card WITHOUT the ferrite beads. We had trouble with both of our modems until we did this; after the change, we never got improper operation or wrong bytes.

Want another tip? If your computer power supply is "beefy" enough, you don't need a separate supply for the modem. Simply run a wire from the tip of the modem's power input jack to edge connector pin 28 (9V). Voila. The modem actually takes very little power; the only real reason for a separate supply is, presumably, the little relay that picks up/hangs up the phone. If your computer supply is borderline, this little relay kicking in could cause a glitch.

ADDENDA for Version 2.2

SYSTEM VARIABLES

ZX-TERM*80 uses its very own system variables. These start at your selected base address (7000h default) and take the first 128 bytes above that. They include the kinds of things (window variables, print positions, mode flags, character buffers, etc.) that you wouldn't want messed with; that's why I put them up there. Whenever in hi-res mode, the IY register is revectored to your base address, to allow indexed addressing to the system variables (more memory-economical). This scheme means that you can quit, NEW, load files/programs for uploading, etc. without affecting the ZX-TERM*80 operating parameters.

However, there are a few system variables in "low" memory (where the normal system variables are). These are used primarily during the Xmodem operations. There is one variable, especially, which can be usefully POKEd. In fact, it's almost essential to know about this if you want to create and upload "custom" files.

This is the two-byte variable I call E_FILE, at 4076h (16502-16503). It specifies where the end of your file is, and is referred to by UPLOAD, VU DATA, and MEMORY REPORT. It reports the virtual end-of-file even if the DATA REM has not been shortened with SAVE YES.

Here's one way you can use this. Let's say that you generated a file to upload, using a word-processor, BASIC program, or whatever. Create a DATA REM, translate your file to ASCII (if necessary or desired), and POKE it (or use the equivalent machine-code) into your DATA REM. POKE E_FILE with the last character in the file, and upload it. This variable is, of course, saved by SAVE. Though some other software titles use this (notable Hot Z in single-step mode), this will not normally pose a problem.

TRANSLATION HINTS

When translating word-processor files to ASCII, do NOT use ZX-TERM*80's translation tables. The reason is that my usage of "bit 6 high" for lower-case is non-standard. Other programs (like Memotext, Word-Sinc, and WORD*) don't use these at all... remember, in the normal ZX/TS system, characters with bit 6 high can't be placed in the display file without crashing. So these usually use "bit 7 high" (inverse) for capitals, bit 7 low for lower-case. I didn't use my "non-standard" approach to be ornery; it was the best way I could think of, of allowing user-customizable characters, retaining inverse of both upper and lower case, and so on. Besides, what does "standard" mean on the ZX/TS, anyway?

The bottom line is that you should use the GENERATING program's translation table instead. All WP's that print to big printers have to have an ASCII translation table somewhere. If all else fails, make up your own table by comparing the ASCII codes (see your printer manual) with the Sinclair codes (see the ZX/TS manual, and the documentation for your WP software). The mechanics of translation has been covered in depth elsewhere; see the SyncWare News Vol. 1 "book," the "TRANSLATIONS" series in SWN Vol. 2, and MEMONOTES.

MORE ON XMODEM -- A SAMPLE DOWNLOAD

Most modern BBS's don't use CP/M at all. Instead, they might use GEM (if Atari-ST-based), MS-DOS (IBM), or God-only-knows. BBS programs for these operating systems are usually easier to use than "good ole" CP/M. Play it by ear. These will usually have plenty of "user help" options available to help you out. On the other hand, you never know just exactly what you'll get when you call a board. Don't worry; you can talk to any of them using ZX-TERM*80. One of my favorite BBS's is appropriately called the "Jolly Roger," and uses as non-standard as possible a mode word, along with other "eccentricities." ZX-TERM*80 handles it with no trouble.

For those of you who can't find the MODEMPRT.001 file referenced on page 16, I uploaded it to the NNN (Nicolson Nightime Network) in Nelson BC. This is an Atari 1040ST-based system, and is online only nights (1800-0900) and Sundays. So call late Sunday night for the lowest rate. Use the standard mode word (8/N/1), and call (604) 354-4666. Initialize your DATA REM with shift 8. Log on, giving your name and password. Fill out the questionnaire (won't take long; it's a very liberal board... but keep it clean!).

Note that most boards will give DEFAULT choices (on NNN, these are indicated in brackets). If you want the default, just press newline.

Note also that responses on this board (and most other modern boards) DO NOT require a newline; think of it as INKEY$ instead of INPUT. You'll probably prefer 1-window mode; but if you still need the menu window, make window 2 as small as possible. 60-column mode is the best readability compromise. After the introductory messages, the board will say: "Main Menu:" ? Pressing newline gives a help file (same goes anytime you see the question mark). Press "f" or "F" (for FILES). You're now in the Files section. Press "D" or "d" to DOWNLOAD. You're asked for which protocol; press "X" or "x" for XMODEM. Then you're asked if your terminal supports CRC. Press "N". You're now asked for the name to download. Type "XMODEM.DOC" then newline. You're then told how long it will take, and to "go to Xmodem to receive file." Press Shift 4, then D, then P. Wait.... Voila! You've just downloaded a datafile, which you can read (shift 5), print, or save.

There are also [L] (list) and [C] options; the LIST option asks for a "search mask." This follows the the CP/M format; for example, to search for only DOC files, enter "*.DOC".

There are also a lot of other neat features on this board, including a dynamite role-playing game (if you can afford the weekly phone call). If you do call them, thank them for me, for the opportunity to use their board every night for weeks (months?) debugging ZX-TERM*80.

USEFUL POKE #15

When using VU DATA, or dumping to big printer using USR 28736 (or whatever relocation address), the program prints a "v" at the end of your file. This is the CHR$ 118 at the end of the DATA REM. This is not normally objectionable; I let the program include this, because sometimes a downloaded file will contain a lot of trailing spaces; the "v" helps to isolate these by indicating the PHYSICAL end-of-file. However, if you find this annoying, quit to BASIC, and PRINT PEEK 16502. Then POKE 16502 with one less. If PEEK 16502 just happened to print 0, then POKE 16502,255 and POKE 16503, PEEK 16503 - 1. Go back into ZX-TERM*80 to VU, or RAND USR .... to send to big printer.

ADDENDA for Version 2.3

USEFUL POKE #16 (version 2.2 and earlier only)

There is a minor bug in ZX-TERM=80, version 2.2. When an ampersand (&) is received, it is printed as an underscore instead. (Oops, sorry about that.) It is sent correctly, but shows up improperly on being echo'd back.

If you have version 2.3, this has already been corrected. However, if you have 2.2, it can be easily upgraded to 2.3 with a couple immediate-mode commands. Load the program, and press BREAK at the cover screen. Then,

LET A$(39)=CHR$ 77 (fixes the bug)
POKE 21920,159 (updates the version reference)

GOTO 0, then press "S" to save the repaired program to tape.

USEFUL POKE #17

An early customer pointed out a valid objection to ZX-TERM=80's keyboard display routine. When typing a "space", you have no on-screen indication that the keypress actually "registered." This will especially be a problem with the stock ZX81/TS1000 or TS1500 keyboards.

One obvious solution would be to have a typing cursor mark the current print location. However, this would require additional code to implement; as it stands, there are exactly three NOPs in the entire program, so this isn't possible without exceeding my self-imposed 4K limit on the program.

Don't despair... there is a easy way to cure the problem, involving only a couple POKEs. What we'll do is cause the "space" character to print a dotted underscore instead of a blank. Though it will make your screens look slightly odd, it WILL assure that you don't accidentally run words together because of insufficient pressure on the space key.

To try it out, quit to BASIC and enter the following POKEs:

POKE 14343,84 (40-col. characters)
POKE 15367,160 (60/80 col. characters)

If you like this change, you can make it permanent by entering the following LET statements during the USER OPTIONS part of the set-up procedure:

LET A$(520)=CHR$ 84
LET A$(1544)=CHR$ 160

MORE ABOUT THE DATA REM

When using SHIFT 8 to create a DATA REM, the program automatically makes as big a REM as it can, for that particular relocation. In the default location, this will be 11,200 bytes (just shy of 11K).

If ZX-TERM=80 is located higher, generating a data REM will automatically use the additional space available... up to a limit of 15.1K. To understand the reason for the limitation, requires a little explanation.

Because of the way that the video system was designed, the 32-48K region of memory has the following property: we can EITHER place the display file above

32K (as on a stock system), OR run machine code there (with ULA MI* decoding). You CAN'T do both, short of doing some very hairy hardware mods.

Since we have to be able to run machine code in this area (ZX-TERM*80), this means that the display file is not allowed in the 32-48K region. If we tried it, we would certainly and irrecoverably CRASH.

ZX-TERM*80 is smart enough to realize this. If you locate it to address 32926 or higher, the generated DATA REM will be fixed at about 15.1 K.

This is not as restrictive as it may seem. 15K represents quite a lot of routine modemming, especially if you use SAVE TOGGLE judiciously to avoid saving garbage you don't really need.

Even with up/downloads, there will rarely be a problem. This is because all commercial programs of which I'm aware, were written with the 16K system in mind. The only exceptions are the occasional "super-long" home-brew program, in which the display file was pushed into the 32-48K region. However, keep in mind that such a program won't even RUN in a machine with MI* decoding, so there's not much point in trying to download such a program anyway. (As a side note, I have yet to see such a super-long home-brew program that could not be condensed to fit into 16K, using memory-saving techniques that have been well-published over the years.)

Note that I'm NOT talking about 16K programs that have been "expanded" to 32K or 64K. An example is "super-long VU-FILE." The program itself has not been expanded; only the variables area (i.e. arrays have been lengthened). Since the variables lie on the other side of the display file, this will NOT be a problem with ZX-TERM*80. If you locate ZX-TERM*80 as high as possible (44-48K), you will be able to up/download a cram-packed 15K program, PLUS almost 12K of variables!

Some files (e.g. Memotext files) are ALL variables; the display file in this case is located at the very start of the BASIC program area (16509). Even if Memotext (V3) and ZX-TERM*80 are both present in memory (as described on page 24 of the manual), you will be able to up/download Memotext files up to about 21K in total length.

MASS-STORAGE CONSIDERATIONS

Using alternate mass-storage (e.g. Compusa, Aerco or Larken disk, or A&J Stringy-Floppy) is possible, but will require some additional hardware work, plus (in the case of the Compusa or A&J) a few software mods.

THE HARD PART

The hardware is the more difficult aspect, but still is nothing "scary." The reason is that all the devices mentioned above use part of the 8-16K region for their DOS EPROM. When running ZX-TERM*80, this entire 8K (except for the very last two bytes, reserved for the Byte-Back MD-2's memory mapped ports) is in use by the program.

So you will need to install switches on both your disk controller, and on your SRAM, so that either can be disabled at will. If you have our SCRAM, half the battle is already won, since this device has a BOARD DISABLE switch.

If you have the Compusa controller, contact The Compusa User Support System (c/o G&C Computers, PO Box 2186, Inglewood CA 90305) for instructions on disabling the controller card.

If you have the Aerco, Larken or A&J, contact the manufacturer for instructions on disabling the controller.

Here's how it works in practise. Disable SCRAM, enable DOS; load ZX-TERM*80. When loaded, return to BASIC (as with A&J) or otherwise be sure you're out of the DOS (other systems). Disable DOS, enable SCRAM. Start ZX-TERM*80 with the appropriate GOTO command. When done, and you want to save your acquired files, ESCAPE ZX-TERM*80, and answer SAVE YES. Give it a name, any name, press ENTER, then BREAK. Disable SCRAM, enable DOS; save the file. Reverse the procedure to load back your files, etc.

COMPUSA OWNERS:. You can simply specify a SAVE NAME in the Compusa format, e.g. A:FILES,P and the system should automatically divert your save to disk instead of tape. (Provided that you disabled SCRAM and enabled Compusa before pressing ENTER.)

OTHER COMPUSA NOTES:

The Compusa uses the first two bytes at RAMTOP to point to its directory. So, you will have to lower RAMTOP by two bytes, and fill them with the desired location of the directory. The best way to illustrate this is with an example. We'll use the relocation given on page 24 for our example.

Once into ZX-TERM*80, quit to BASIC (SAVE NO). ZX-TERM*80 (and RAMTOP) have been relocated to 39024. This represents 112 in location 16388, and 152 in location 16389 (112 + 256*152 = 39024). We want to lower RAMTOP by two bytes, so POKE 16388,110 then NEW. RAMTOP will now be at 39022 (110 + 256*152 = 39022). Now you need to decide where to put the directory. You need about 1.5K somewhere in free memory. The top 1/4K is off limits because of the dummy display buffer area. The 1/4K below that is also off-limits, if you located your input buffer here (used a value of 254 for "n" in Useful POKE #13). So, a good place to put the Compusa directory is at 60K (61440). So POKE 39022,0 and POKE 39023,240 (0 + 256*240 = 61440). You can now LOAD "A,DIR", SAVE "B:FILES,P", etc. without messing up anything else.

A&J NOTES:

Similarly, the A&J needs the first 1K above RAMTOP for its directory. After start-up, you will have to quit ZX-TERM*80, then POKE 16389 (four less than at present) then NEW, then re-enter ZX-TERM*80 or the A&J "EOS". To continue the above example, POKE 16389,148 (=152-4) then NEW. NOTE: ALWAYS use USR 13083 for entering the EOS "directory mode"... NOT 12345, as this will undo all your hard work.

Alternately, use "Economy mode" and avoid the hassles and extra memory required by the directory.

AERCO AND LARKEN

Aside from the hardware mod to disable the controller board, these should not require any software Pokery. With the Aerco, however, you'll probably be limited to the Aerco DOS; unless you do considerable finagling, it's unlikely that you can effectively use BBDOS. The reason is that this and ZX-TERM*80 will be fighting for the same memory space in the 8-16K SRAM. However, enterprising software hackers might be able to figure out a way of designing "auto-boot" routines to make BBDOS and ZX-TERM*80 relatively user-transparent.

IF YOU ARE NEW TO MODEMMING...

Even long-time users of personal computers often are intimidated by the prospect of using a modem (modulator/demodulator) to communicate with other computers and computer systems. We Timex/Sinclair users are especially prone to such intimidation; over the years, we've heard no end to sometimes derogatory, often only condescending remarks about our "toy" machines. We have a bit of an inferiority complex, despite repeated proof that these machines CAN be used for real jobs, and ARE, in the truest sense, REAL computers.

If you're new to all this, here are a few comments that might help out.

* You WILL enjoy modemming.

* You DO own a real computer.

* You DO NOT have to be a computer wizard to use this software.

* You CAN communicate with other computers and computer services on a equal basis.

* Modemming NEED NOT cost you an arm and a leg. There are dozens of free bulletin boards in every major city on the continent. There's even a free board (and a very good one!) in Nelson, BC!

* You WILL NOT be castigated for making errors while learning. Most SYSOPs (system operators) are willing to help, and are tolerant of the inevitable "beginner boo-boos." Remember, the Sysop was once a beginner too!

* Your spouse NEED NOT become a computer widow(er). Even non-computerists enjoy communicating. Include her (him) in your modem activities; you now have something more in common! Remember, a computer is a multi-function machine. Just because your spouse disapproves of your spending hours playing Pac-man doesn't have to mean that the computer has to get the same status as your golf clubs or bridge club.

* You WILL NOT get in trouble with Ma Bell for hooking your computer to the phone line.

* You WILL find new vistas of experience, education, and enjoyment.

* This software WILL NOT lock you into an unchangeable pattern. I've done what I can to make the program easy to use for beginners, but flexible enough to grow with you as you progress.

* You WILL enjoy modemming!

Fred Nachbaur
Founder, Silicon Mountain Computers

Silicon Mountain Computers
C-12, Mtn. Stn. Group Box
Nelson, BC V1L 2J3

November, 1987

ZX-TERM*80 V2.3

ADDITIONS AND ERRATA

UPGRADING TO V2.4

You guessed it - we've found a couple more bugs, and "science fiction" in the ZX-TERM*80 manual. Fortunately they are relatively minor, and easy to fix.

The relocator relocates one location that shouldn't be changed. This can cause bizarre behavior with certain relocations. Fix it as follows:

LOAD "ZXT*80", V2.3. Press BREAK when loaded. Now enter the following POKEs:

```
POKE 22058,112
POKE 22059,47

POKE 21920,160
POKE 20508,160
```

After answering SAVE'YES, the program saves alright but comes back with a strange error code instead of the error "S" reported. Also, the "useful POKE #12" instructions are incorrect. First fix the error code bug with the following:

```
POKE 18899, 205
```

If you want to return to ZX-TERM*80 after "SAVE YES," replace the POKEs given in the manual with the following:

```
POKE 31046 (18902),24 (207)
POKE 31047 (18903),205 (27)
```

These changes upgrade ZX-TERM*80 V2.3, to V2.4. GOTO 0, then press "S" to save the upgraded program to tape.

Finally, the NOTE on page 35 contains an incorrect address. This should be "28792", not "28972" as printed.

SUPPORT FOR ZX-TERM*80

Since ZX-TERM*80 propels us into the electronic information age, it only makes sense that after-sale support should be via the electronic medium. With this in mind, I have arranged to obtain disk space on the "Nicolson Nightime Network." Phone (604) 354-4666. At this writing, the board has an article I uploaded into the files section, detailing how to import files from Memotext V3 into ZX-TERM*80. You can now upload files created with Memotext, completely translated and formatted; in other words, just as it would look if printed to paper. Other similar articles and "helpful hints" are in the works.

At this point, the project is still experimental. By that I mean, if we don't get sufficient interest in the form of calls, I will discontinue the effort. The sysop of the board was kind enough to allow space for ZX-related files, but was quite dubious that enough people would bother calling. Let's prove him wrong.

If it proves popular, I plan on uploading public domain Sinclair programs (some of them never-before-seen! Including high res!). However, it depends entirely on you.

Any files that exist on what I'll informally call "The ZX-TERM Exchange" are in the public domain, and may be uploaded to other boards or services at your discretion. To avoid the trouble of having to set up a separate SIG for Sinclair/Timex ZX/TS users, I have decided that all file names will start with "ZX". This way, all the ZX stuff will be right at the end of the catalog of available files, out of everyone else's way, and yet all in one group for ZX aficionados. If you upload anything to the NNN, please follow this convention. (..DOC or .RLE files of general interest not included, of course). In addition, let's standardize to the following suffixes:

ZX----.DOC - ASCII document files readable by any computer
ZX----.MTX - Memotext files (readable only by ZX/TS users).
ZX----.PGM - ZX/TS1000 programs
ZX----.VAR - Variables associated with a given program.

Note that the name given to program and variables sets should be the same, so that it's obvious that the two go together. Similarly, if there is documentation for the program, use the same name followed by .DOC. Preferably, upload the elements in this sequence: .DOC, .PGM., .VAR (if needed).

For years now, ZX/TS users have been clamoring for support, and rightly so. Here is a golden opportunity to get those questions answered (hopefully, anyway) for free, gain access to free programs, and get what is in essence a free newsletter for ZX-TERM*80. Use the opportunity. All it will cost you is a phone call; and if you call late at night or on Sunday, it will only cost you a few dollars for a half-hour online, regardless of where you are.

The NNN runs at 8 bits, no parity, 1 stop (standard for Xmodem and ZX-TERM*80), and allows Xmodem up/downloading. It operates from 00-0900 every night Monday-Saturday, and all day Sunday. When you log on, THANK THE SYSOP for the courtesy of letting us humble-yet-cheap ZX fanciers use his board for our own special-interest purpose.

Fred Nachbaur

PS - When you first log on to the NNN, go to the Information section from the main menu, and select "ZX-TERM EXCHANGE" for an introductory letter outlining our goals, etc. Have your SAVE TOGGLE on, as it's quite long.

Silicon Mountain Computers
C-12, Mtn. Stn. Group Box
Nelson, BC V1L 5P1

Canada

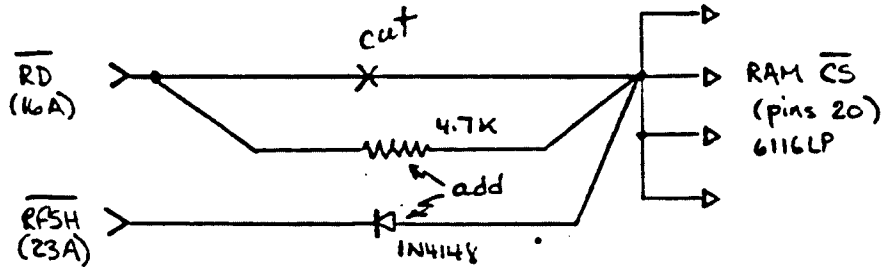
MODIFYING "HUNTER" FOR WRX16

THEORY

The "WRX16" core routine which is the basis for our latest generation of high-resolution software has minimal hardware requirements; specifically, a static RAM in the 8-16K region. The reason it has to be static is because the bit patterns have to be accessed during "refresh" time, which is normally used with dynamic memories to keep their data intact.

The "Hunter" NVM board will work if a simple modification is made. The reason for the modification is to make the board's data accessible during refresh time. This is accomplished by ANDing RD* and RFSH*. In other words, if RD* or RFSH* goes low, the board is enabled; the board is disabled (CS* high) only if both RD* and RFSH* are high.

On the Hunter board, the RD* line is directly connected to the RAM chips CS* (chip select not). To make it usable with WRX16 software, this line only needs to be cut, bridged with a resistor, and one diode added. The schematic below describes the needed modification.



MAKING THE MODIFICATION

Viewing the board from the rear (non-component side) with the edge connector at the bottom, locate the RD* line. This is a narrow trace that connects to edge connector pin 16A (the 13th pin from the slot at the top). The other end of this trace goes to pin 20 of the second 6116 RAM chip. (A trace on the component side of the board connects it with pins 20 of the other RAM chips.) Cut this trace, thereby isolating RAM CS* from RD*.

Clip the leads of a 4.7 K resistor so that it will just fit between the second RAM chip's pin 20 (CS*), and edge connector pin 16A (RD*). Cover it with a piece of "spaghetti," shrink-tubing, or aquarium air-line tubing to prevent shorts. Solder it into place. Note that you are effectively bridging the gap you made by cutting the trace; the board should work exactly as before.

Finally, install an ordinary 1N914A or 1N4148 diode between RAM CS* (pin 20, the same point the resistor was connected to) and edge connector pin 23A (RFSH*). This is that last pin on the top row. Note that the cathode (banded) end of the diode must go towards RFSH*. Again, clip it to the right length and protect against shorts with "spaghetti."

With this modification, the board should again work just as before with all known software. (The exception is "TS*1500 HI*RES", as described in SynaWare News #3:4. If you wish to make your Hunter board usable with this, install a switch in series with the diode.) The modification is therefore "user-transparent." However, it will now be capable of true high-resolution displays using the WRX16 software core.

ZX-TERM*80 V2.5 ADDENDA

MAY, 1988

By F. Nachbaur

I have made some "final" changes to ZX-TERM*80, resulting in version 2.5. This version sports a number of additional refinements, making it even more flexible and useful than before. The changes are summarized in this .DOC file. Feel free to circulate this, the program, and the associated Hot Z "NAMES" file to other legitimate users.

The program is still only 4K long. Virtually every nook and cranny has been filled, and it is most unlikely that further changes will be necessary or desirable.

1: WRX16-V2 CORE - ZXT80 V2.5 uses the new V2 of Wilf Rigter's WRX16 high-res core. This eliminates the need for a copy of the dummy display file in high memory, thereby removing some of the headaches that this causes.

V2.5 should work fine with Memotech 32K RAM packs (without additional 16K pack). It also works with a completely stock TS1500. (In both cases, of course, the all-important 8K static RAM is assumed.)

Finally, there is no need for any data swapping, etc. to make the program work with 64K packs. This frees up a little space for some of the added features. Note also that there is no need to select "memory type" in the BASIC configuration portion of the program.

2: NEW ENTRY POINT - The entry point for V2.5 is ten higher than previous versions. E.g., at the default location, enter V2.5 with RAND USR 32280 instead of 32270 as in previous versions. If you relocate it as high as possible (45056), the entry point is 48664 instead of 48654. Internally, some of the routines have been moved about. This may require changes in auxiliary "patch" programs such as BIGRLE and Gary Robinson's PATCH (dual data buffer).

3: BYTE-BACK MD-2 MODEM: Previous versions required that you use SHIFT 6 (modem control) carefully. They only checked if you were pressing the "A" key (to go to "Answer" mode). Any other key was taken to mean "Originate." This meant that you had to press Shift 6 very quickly to avoid a false reading.

This has been corrected in V2.5. Also note that, unlike before, you MUST hit shift 6 and select either A or O before getting online; the SIO chip is no longer automatically initialized on entry. (See below.)

3: WESTRIDGE MODEM: The program no longer automatically initializes the modem on entry, and does not shut it off when quitting to BASIC. This allows more flexibility in the use of the program.

For instance, you might have a tape fast-save routine in an alternate bank in 8-16K, or you mihgt have a disk controller mapped in this region. You can thus download a file, quit to BASIC to save it, and re-enter ZX-TERM*80 to continue modemming, download another file, etc. The connection will not be broken by quitting to BASIC or re-entering. (So don't forget to hangup after logging off.) Similarly, you could download a program, test run it to see if it's something you wnat to keep, and jump back in to download something else if you so decide, else log off.

4: INPUT BUFFER QUEUE EXPANDED: The input buffer has been permanently expanded to 256 bytes, much as if you had employed "useful POKE #13." The buffer address (high byte) is stored in the second byte in the program (generally, this will be the first address beyond RAMTOP). Since most users seem to have 64K, the default is at 65024 (254=FEh * 256). This can be changed by modifying location 16529 as desired.

If you are using 16K, a little finagling will be necessary. The first time through, answer "Y" to "INSTALL USER OPTIONS?" and POKE 16529,111. Enter GOTO 500, answer relocate No, and save the modified program to tape. After saving (or subsequent reload), enter ZX-TERM*80 as usual but DO NOT do any modemming yet. Quit to BASIC using ESC, the POKE 16389,111 and NEW. This will lower RAMTOP by 256 bytes, reserving space for the input buffer. Re-enter ZX-TERM*80 and modem to your heart's (and budget's) content.

5: TEXTWRITER MODE: If your save toggle is on, your typed input will be stored in your capture buffer ("Save" buffer). This is as if you had installed "useful POKE #4). The reason this was done is to give you the option of fast message uploading using the ASCII TRANSFER feature, dis-- cussed next.

6: ASCII UPLOADING: Some boards don't support XMODEM, but allow uploads/ downloads using the "ASCII protocol." This really isn't a protocol at all, since it does no error checking; it simply sends the data, assuming that the other terminal is there catching it all.

ASCII downloads can be done with all versions of ZX-TERM*80. Simply open your capture buffer, and press a key (usually SPACE). Uploading, however, was not directly supported until V2.5.

Since the main menu keys were all used up, this and the other new features were placed in the FUNCTION mode. When you press FUNCTION (shift enter), you will now get a FUNCTION sub-menu. The symbol keys are used as before for symbols, but you now also have (D)ump, (H)ardcopy and (T)ransmit options. .The first two are covered in the next section. The (T)ransmit option proceeds sending the contents of your capture buffer (or what ever else is in the program area) to the other terminal in a continuous stream. The output is shown in window 2 as it is sent. Pressing SPACE or ESC will prematurely terminate the output stream.

Bit 7 is NOT masked, so this is actually a little more than just an "ASCII" uploader. In theory, at least, you could therefore use it for 8-bit data, even Sinclair programs. However, since there is no error-checking, use Xmodem whenever possible for such uploads.

With many boards, you can use this to prepare a long message, then ship it to the board in one go (saving time and money). I noticed, though, that on the NNN, the board doesn't have enough time to process each line as it is received. As a result, messages longer than 4 or 5 lines get garbled. In such cases, you might consider uploading your long message to the files area, and leave a short note to the sysop to ask him to transfer it to the message area.

7: BIG PRINTER OPTIONS: The program's handling of big-printer operation has been considerably refined. Previously, there was not much processing of the line-feed (LF) and carriage-return (CR) codes. This caused headaches with some printers, causing to print double-spaced lines in some circumstances, while refusing to line-feed at all in others. V2.5 now does the following. When printing, it completely ignores any LF codes that may be received. The CR code alone is used to signal a newline. When a CR is sent, an LF is automatically sent right afterwards. So set up your printer for the "LF + CR" mode, your printed output will always be spaced correctly.

You can now turn the printer on or off using FUNCTION H (Hardcopy toggle). Window 2 will show the present printer ON or OFF status. This was added because some printers continue to accept data into their buffer, even if taken offline. Having software control eliminates the need to take the printer offline, or shut it off completely to clear its buffer.

Finally, the (D)ump option can be used to dump the entire contents of the capture buffer to the big printer. This obsoletes the DUMP DATA BUFFER TO PRINTER option in Appendix V of the manual. As with the (T) option, pressing SPACE or ESC terminates the dump. Also as with the (T) option, window 2 will display the data as it is sent. This option can be used either while online, or after logging off.

8: COSMETIC TOUCHES: Finally, some cosmetic changes were made that you may or may not notice. For instance, the main menu now includes options SHIFT 9 (MEMory report) and SHIFT Ø (DELete). Delete, incidentally, sends a BS (ASCII Ø8) instead of DEL (7Fh), since some boards don't like the DEL code (including the NNN) whereas virtually all will respond to BS. (That is NOT a value judgement!)

Incidentally, you can send a % symbol with "FUNCTION ." and an apostrophe with "FUNCTION SHIFT ." - these were present in previous versions, but were omitted from the table on page 6.

USEFUL POKES MODS: Since some routines were moved, some of the useful POKES have to be changee. Others are now obsolete.

#1: POKE 30705 (18561), 0 (196)
30706 (18562), 0 (210)
30707 (18563), 0 (117)

#2, #3: OK as published.

#4: Implemented. OK to reverse.

#5: Obsolete

#6: POKE 30109 (17965), 240 (226)*
30110 (17996), 116 (125)*

#7, #8, #9, #10: Obsolete

#11: Not recommended. Ignore.

#12: Not possible. RAND USR (address) to return.

#13: Obsolete

#14: POKE 32267 (20120), 0 (10)

#15: OK as published.

#16: Obsolete

#17: OK as published.