UltraMON51

When I was five years old, I decided what my home really needed was a swimming pool. Undeterred by the fact that I couldn't swim I set out one summer day to dig the pool myself. But even with my trusty "Planet of the Apes" combination lunch box/shovel I never quite managed to finish the job. I did however make a hole satisfactorily deep enough to fill with water and in the process of all this make a big muddy mess in the backyard.

Years later I was learning how to program in assembly language. Fortunately by that time I had learned the valuable lesson of having the right tool for the task at hand, and purchased a "monitor" to aid in my study. The purpose of a monitor is to make assembly language programming easier, and is invaluable as an instructional tool as well as a debugging tool. A *really good* monitor can save hours of debugging time and also help flatten that steep "learning curve" associated with mastering the intricacies of the microcontroller or microprocessor you might be working with.

Some may say "Why bother? Isn't assembly language programming a dead art? Why, simply everyone knows it's useless knowledge and in order to really do anything the source must be written in 'C' so that blah blah blah [here's where I usually go cross-eyed]". The fact is your assembly code will be superior in speed, efficiency, and compactness than any higher level language equivalent, and that's including todays politically correct programming language "C". So when the code really matters I write the source in assembly, simple as that. Also, learning assembly language is the best way to gain an intimate understanding of your computer system's internal operations and is well worth the time investment.

There seems to be a tremendous amount of electronic and computer hobbiest interest in microcontrollers these days, and rightly so. With microcontroller processor prices as low as they are and their accessibility so high, it's never been a better time to integrate microcontrollers into your designs to replace a lot of discrete logic and turbocharge your gadgets by giving them "intelligence."

Certainly the most popular and widespread microcontroller is the *8031*, generically referring to all the 100+ IC variants of the "8031" hardware architecture sharing a common instruction set. To say that the *8031* has an unusual instruction set is an understatement, and unfortunately I believe that many of the people interested in the topic are intimidated and discouraged because of the difficulty in working with the only "built-in" programming language available to them, assembly language. The "erase, burn, crash, cuss" EPROM based assembly debugging drudgery is not exactly conducive to learning *8031* assembly programming or debugging your programs in general.

There are of course alternatives to EBC^2 assembly language development, but their applicability to you personally varies with your interest in the topic and the size of your wallet. Look through any magazine with microcontroller coverage and you will see plenty of advertisements for hardware in-circuit emulators and PC based software emulators. Usually the exorbitant price tags are left out of the ad as being incidental to the product's wonderful features. To an entrepreneur with a vague new consumer gadget idea, an engineering student who wants to add voice synthesis to the dormitory toilets as a gag, or someone with only a passing interest the expense cannot be justified. It's for these people UltraMON51 was written.

UltraMON51 is a first-class monitor program with features rivaling those of the expensive *8031* hardware ICE and PC based software emulators, while offering some unique features not commonly available in those products. If you are putting together or customizing your own *8031* based single board computer ("SBC"), UltraMON51 will prove valuable in the troubleshooting of the hardware in your system. If you are just learning *8031* assembly language, UltraMON51 is tremendously valuable as an educational real-time tutor. Should you be an experienced *8031* programmer, UltraMON51 can assist in the debugging of your tricky software algorithms, or help out with the deciphering of someone else's object code. In short, you'll wonder how you ever got along without it!

Some of the more interesting features of UltraMON51 include XMODEM-CRC data transfers to and from program ROM and external RAM which allows for generic data to be transferred without pre-formatting into any specific arrangement such as Intel Hex, Motorola Hex, etc. The data can be ASCII text, data tables, or executable object files which opens up the possibility of EPROMless code development. Also included is a line oriented semi-symbolic assembler and dissembler, perfect for writing and modifying code on the fly, or learning the oddball *8031* instruction set. The single-step execute mode includes the relevant instruction disassembly with SFR/register dumps tremendously simplifying the elimination of those elusive programming bugs. Appendix A gives an example UltraMON51 run and illustrates the utility of this monitor program.

GETTING STARTED

This section describes how to get UltraMON51 running on your *8031* SBC in the least amount of time. However, to get the most out of UltraMON51 be sure to thoroughly read and understand the next section "TECHNICAL."

The following conventions are followed to avoid confusion throughout the remainder of this manual:

1.	Output from UltraMON51 to the terminal screen is displayed in **bold type**.
2.	Input from the user is displayed in *italic* with special keystrokes sent to UltraMON51 by the user as directives and non-printable characters are displayed as *[bracketed italic]*.

Include the UltraMON51 EPROM into your SBC making any necessary circuitry modifications to accommodate it, and verify that it is addressed from $0000 to $1FFF. The object code for UltraMON51, ready to be programmed into a 2764 EPROM as well as preprogrammed parts, are available from the author.

Since UltraMON51 communicates with the user via the *8031* built-in RS232 3-wire serial I/O UART, verify that UltraMON51 can serially communicate with your PC/terminal program or dumb terminal. Make sure the terminal is communicating with the following settings: 8 data bits, no parity, 1 stop bit, full duplex. No special terminal emulation mode such as "ANSI" or "VT100" is required, or for that matter supported by UltraMON51.

Upon power-on or RESET, UltraMON51 is interrogating the serial port waiting for data to arrive so that it can auto-detect the baud rate being generated on the terminal. Therefore you need to hit a key on the terminal before UltraMON51 can do anything! Upon reception of serial data, UltraMON51 will calculate the timer 1 reload values, initialize timer 1 to generate the necessary baud rate, and print a message similar to the following:

On-line at 19200 baud
UltraMON51 V1.0

Shadow RAM: □

UltraMON51's auto baud rate detect feature supports the following baud rates with an 11.0592MHz clock: 300, 600, 1200, 2400, 4800, 9600, 14400, and 19200. Should your SBC be using a different clock speed refer to appendix B to learn how UltraMON51 accommodates this.

Upon setup of the baud rate generator, UltraMON51 prompts you for the address to store the shadow RAM as discussed in the following "TECHNICAL" section. Enter a hex two byte external RAM address and hit *[CR]*:

Shadow RAM: *3E80[CR]*

> □

Users who consistently enter the same values for shadow RAM address and baud rate may wish to skip the auto baud rate detection and shadow RAM address query and go straight into the monitor. Refer to appendix B for information on how to accomplish this.

The monitor prompt is ">" and this informs you that the monitor is ready to accept a command. All commands are activated with a single key press which are echoed to the terminal. It is not necessary to have the "CAPS LOCK" engaged as UltraMON51 will convert lower case ASCII to upper case automatically. Unrecognized commands will echo "?" as will errors involving invalid address ranges, invalid hex number input, etc.

Upon reception of a valid command UltraMON51 will echo to you those parameters, if any, required to complete the command freeing beginners from the nuisance of having to remember all the various command parameters. For example the "$" command, convert hex to decimal, requires a two character hex number input:

>*$* xx
A9 **= 169**

In general, it is not necessary to hit *[CR]* to complete a command because UltraMON51 is smart enough to know when valid parameters have been completely entered. The exceptions to this rule are those commands accepting strings of input to be entered such as with the "modify SFRs and registers" command. Also, extra characters such as dashes and commas which denote address ranges and multiple command parameters are automatically inserted by UltraMON51 as an added assistance to the user.

That's all there is to it! I recommend following the sample run in appendix A on-line to get a good feel for all the various commands, and you should be making maximum use of all of the UltraMON51 features in no time.

TECHNICAL

Due to the general purpose nature of the *8031* and the multitude of family members with many different features, there exist an almost infinite number of configurations a SBC can take based on this processor architecture. This of course makes this processor family useful in a wide range of applications but made developing a general purpose monitor program difficult. Every effort was made to be flexible and not make assumptions about the user's SBC setup, but since UltraMON51 is actually a program executing on the host *8031* processor limitations do exist as certain processor resources are tied up by the monitor itself. These limitations are kept to a minimum through the use of several key mechanisms by UltraMON51. The importance of this section cannot be understated - "read it know it live it"!

A certain familiarity with the *8031* hardware architecture is assumed in this section to be able to divulge the necessary technical information in a succinct fashion. Since reams of information exist on the basic *8031* architecture I will not waste the space to regurgitate a bunch of redundant hardware facts, but simply recommend browsing through back issues of magazines with *8031* coverage such as "ComputerCraft" for readers not familiar with the topic. In addition, I would like to mention that a copy of the Intel Embedded Controller Handbook Volume I is probably the best source of information for anyone interested in *8031* microcontrollers. It is available from Intel Literature Sales, 1-800-548-4725, for a reasonable price.

UltraMON51 will exist in your *8031* software development system as the master control program, and the UltraMON51 object code (usually in an 8K 2764 EPROM package external to the *8031* processor) must be addressed from $0000 to $1FFF ($ indicates a hex digit) so that it is executed upon power-up or at reset. This address is the only absolute requirement for a minimalist UltraMON51 based development system. However, in order make maximum use of UltraMON's varied features the system should have external RAM somewhere in the *8031* memory map. In addition, the organization of the entire memory space should ideally be "Von Neumann" which simply means that the master OE' read strobe for both ROM and RAM is the ANDed result of RD' (external RAM read strobe) and PSEN' (external program ROM read strobe). This gives a linear, non-overlapping program ROM/external RAM memory map and also allows for the execution of code from external RAM. In addition program ROM may be read with the "MOVX" instruction, usually employed by most of the UltraMON51 commands as a generalized memory read mechanism, instead of the more cumbersome "MOVC" instruction.

If your *8031* SBC does not have external RAM, the UltraMON51 commands that modify values in RAM ("edit external RAM", "fill external RAM", etc.) will obviously not perform their intended function. If your memory map is not arranged in the previously mentioned fashion you will not be able to execute code from external RAM ("walk execute", "break execute", and "go execute" commands) but will be able to execute and debug code from program ROM.

Clearly this memory issue is a complicated subject given the fact that the *8031* architecture splits the memory map into two separate entities each with a different read strobe. VERY IMPORTANT: For a memory read (program ROM or external RAM), UltraMON51 always uses a MOVX instruction with the exception being the "disassemble" routine. For the two commands that perform or include a disassembly of memory ("disassemble" and "walk execute") the MOVC instruction is used for obvious reasons. Use these facts to determine if a particular command will function within the context of your particular SBC memory configuration. When it doubt let OE'=RD'•PSEN' for the whole memory map and make your life a lot simpler!

Figure 1

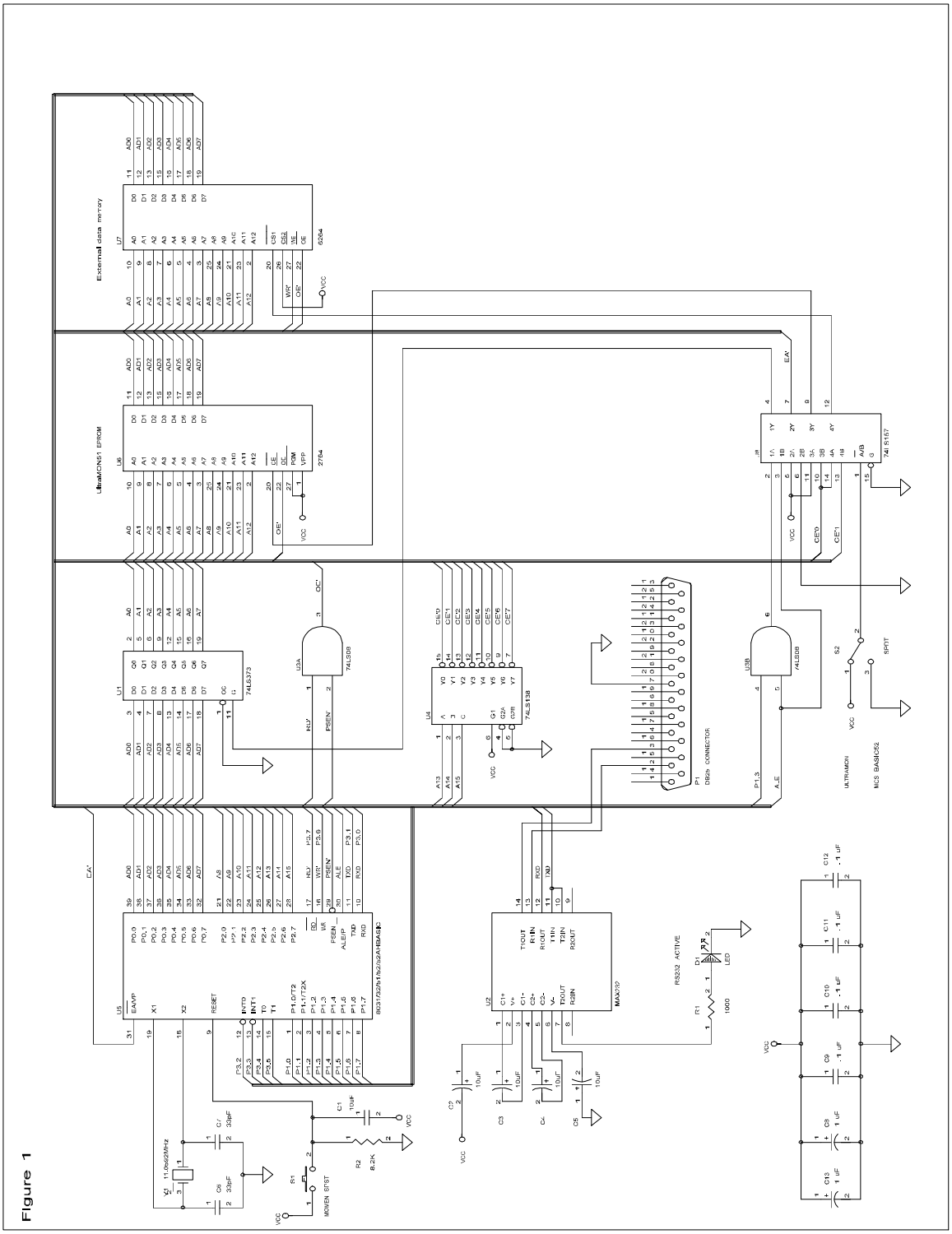
External data memory
UltraMCN51 EPROM
MCS BASIC52
ULTRAMON
DB25 CONNECTOR
RS232 ACTIVE
MOMEN SPST
SPDT
MAX202
74LS373
74LS138
74LS08
74LS157
2764
6264
11.0592MHz
33pF
10uF
8.2K
1000
LED
.1 uF
1 uF

Figure 1 is provided for illustration purposes and details a flexible *8031* SBC that was constructed and used in testing UltraMON51. {the author can provide for the price of a SASE printed circuit board layouts for this SBC} There's nothing very startling about it and predominately consists of the standard *8031* glue. Notice that RD' and PSEN' are ANDed at U3A to synthesize the OE' read strobe for both program ROM (the UltraMON51 EPROM in this case) and external RAM (the 6264 8K static RAM). With this SBC and an 8052AHBASIC processor, toggling S2 will manipulate the necessary signals causing this processor to work double duty supporting both assembly code development with UltraMON51 and also MCS BASIC-52 code development (the "best of both worlds").

Previously mentioned was the fact that UltraMON51 is itself a piece of code executing on the SBC, and therefore ties up the vast majority of the lower half page of internal data RAM for its own operating requirements as well as most of the SFRs. UltraMON51 does not require or use any memory from the upper half page of internal data RAM should your SBC be *8032* based and have RAM up there, and this memory is completely protected for user object code utilization.

UltraMON51 handles the lower part of internal RAM through a "shadow RAM" mechanism, effectively giving back to the user this internal RAM and in addition some of the SFRs tied up by the monitor that are not an integral part of proper timer operations. This perhaps confusing explanation requires some further details, but is a powerful hidden feature of UltraMON51 and is important for users to understand thoroughly to avoid problems.

When first executed UltraMON51 requests you to enter a two byte hex address, corresponding to a location in external RAM, for "shadow RAM" storage. Consecutively stored here are 128 bytes of "internal RAM" plus seven bytes for "SFR" storage. Commands that alter "internal RAM", the "SFRs", or the "registers" such as "edit internal RAM" and "modify SFRs and registers" do not actually modify the internal RAM of the *8031* but instead modify a corresponding shadow RAM memory location. The lower 128 bytes of "internal RAM" are stored here, in addition to the following *8031* "SFRs": SP, DPH, DPL, P1, A, B, and PSW. The "registers" R0 through R7 are mapped to their logical locations in "internal RAM" depending on the register bank currently selected in the PSW "SFR".

Upon entry into any of the execute instructions ("go execute", "walk execute", and "break execute") the values from shadow RAM are copied into the actual lower 128 bytes of the *8031* internal RAM plus the seven shadowed SFRs are copied to their respective internal RAM addresses. The time delay required to accomplish this should be almost imperceptible. Upon return to the monitor from a "walk execute" or "break execute" command the values are re-copied to shadow RAM for storage where they may be viewed or modified before the next subsequent execute command cycle. This swapping process is repeated for as many executes and returns to monitor operations that are initiated. This is a one directional one time only copy for the "go execute" command.

For all intents and purposes you may think of commands such as "dump internal RAM" and "modify SFRs..." as actually editing/displaying the values in the *8031* internal RAM since most of the time the swapping process from shadow RAM to internal RAM will be completely transparent and error-free. However, it is important to keep the mechanism by which this is accomplished in the back of your mind and know how the actual process works to avoid problems. Again, editing memory above address $80 with the "edit internal RAM" command is a modification of the actual location of internal RAM in an *8032* CPU; this memory is completely protected for user object code utilization.

The "walk execute" and "break execute" commands implicitly require five actual bytes of internal RAM storage for proper operation. These locations must remain constant for both the execution of UltraMON51 and the user object code being debugged/executed. Locations $7B through $7F are tied up during these commands and therefore must not be modified by the user object code being executed. Failure to heed this requirement may result in catatonia or worse yet erroneous operation. As a result of this possibility "internal RAM" locations $7B through $7F will not be copied from shadow RAM before a "walk execute" or "break execute" to prevent this from occurring. The remainder of internal RAM is free for use by the user object code. During the "walk execute" command, there also must exist sufficient user stack space to accommodate 32 stack pushes and pops required by the "disassemble" command, invoked as an integral part of the "walk execute" command.

In practice it may be a good idea to set the SP to some sufficiently large number to accommodate these stack operations (be sure the stack can't overwrite locations $7B through $7F!) and any user object code stack operations, effectively protecting the area underneath the stack for data/variable storage from being accidentally overwritten. See the example run in appendix A for an example of how to accomplish this.

It is safe to assume that the "walk execute" and "break execute" commands will tie up for proper internal operations any of the SFRs that are not "shadowed", for example timer 0 and timer 1 relevant SFRs such as TCON, TMOD, TL0, TH0, TL1, TH1. In general, do not attempt to "walk execute" or "break execute" through any object code that:

1. modifies an SFR that is not shadowed (SP, DPH, DPL, P1, A, B, and PSW).
2. directly or indirectly modifies internal RAM locations $7B-$7F.
3. overwrites internal RAM locations $7B-$7F with a stack operation.

A few details about the "break execute" command need to be mentioned. Since UltraMON51 uses a unique software controlled timer overflow mechanism for trapping each instruction as it is executed, it can break at instructions in either program ROM or external RAM. However this does slow down the execution time of the user object code slightly, and therefore execution is not at maximum CPU speed. As previously mentioned, five bytes of internal RAM are tied up by the "break execute" routine and should not be modified by user object code. In addition, the 32 bytes of user stack space required by the "walk execute" command is not required by the "break execute" command since it does not include an integral instruction disassembly. Upon exit, 16 bytes of user stack space is required for the register dump output routine, integral to a "break execute". If you are debugging a piece of code that uses a considerable amount of internal RAM storage it might be best to avoid the "walk execute" command and stick with the "break execute" command. Keep these facts in mind to make sure UltraMON51 itself does not accidentally overwrite any important value in internal RAM you want preserved.

Initiation of the "go execute" command will perform a complete copy of shadow RAM, including locations $7B-$7F and the seven shadowed SFRs, and unconditionally pass control to the user object code. At this point the user code takes full-speed complete CPU control and is free to execute any bit of code possible. UltraMON51 should only be re-entered from a full speed execution with a LJMP $0000 to avoid hang-ups and erroneous operation. It may be desirable to leave timer 1 setup for serial operations so that it may be used for terminal output during full speed execution. Many routines within UltraMON51 are extremely handy for debugging routines at full speed execution and output information to the terminal, assumed by these routines to be setup and communicating properly, and can be incorporated into the user object code to perform such useful things as register dumps, variable value output, etc. Some of the more useful UltraMON51 routines are listed in appendix C.

COMMAND DISCUSSION

The following section lists all of the UltraMON51 commands and describes them in detail. Usage examples of each command can be found in appendix A. As an aid, the key that initiates the command is displayed in capitalized bold typeface. The following command parameter conventions are followed:

1. "xxxx", "yyyy", "zzzz" indicate a two byte address.
2. "xx", "yy", "zz" indicate a one byte hex number.
3. "abc" indicates the ASCII name of a register like "R4" or a shadowed special function register such as "DPH".
4. "ddd" indicates a decimal number.

A	*xxxx*	**A**ssemble

This command invokes the one line assembler for modifying or composing a program on-line to begin at address *xxxx*. "One line" is in reference to the way the instructions are assembled into external RAM immediately upon hitting *[CR]*. Therefore no labels are allowed and all operand references must be absolute. Hex numbers are prefixed with a "$", and tabs or spaces are acceptable as delineating the mnemonic from the operand. Refer to appendix E for a list of the valid SFR bit and byte names supported by the "assemble" command.

The current address will be displayed, and you may enter the instruction to be assembled terminated with a *[CR]*. If the instruction was valid it will be assembled into external RAM, and the next resultant address displayed. Enter another instruction or *[CTRL-C]* to return to the monitor prompt. If the instruction is not valid a "?" will be echoed and the address re-displayed. Hit the backspace key to correct any typing errors.

B	*xxxx,yyyy*	**B**reak execute

This command starts a breakpoint execution beginning at address *xxxx* and ending at *yyyy*. The instruction at *yyyy* is not executed. The ending address must be at the first byte of an instruction (the opcode) or the breakpoint will never be encountered, resulting in non-stop execution. Should this happen the processor must be reset. Upon encountering the breakpoint a register dump will be performed before returning to the monitor prompt.

C	*xxxx-yyyy,zzzz*	**C**opy external memory

This command copies external memory from *xxxx* to *yyyy* inclusive to the external RAM destination address beginning at *zzzz*. The values to be copied are not verified upon write. Be careful when copying absolute object code to other areas of memory as address references are not resolved. Also, copying blocks of memory with a destination address that is within the copy range will result in errors in the data copied.

D	*xxxx-yyyy*	**D**isassemble

This command performs a disassembly of the memory range beginning at *xxxx* and ending at *yyyy* inclusive. All branches and addresses are displayed as absolute references. Whenever possible, UltraMON51 will substitute an SFR bit or byte name. See appendix E for a listing of the supported names. Hit a key on the terminal to pause the listing, and hit another key to resume. *[CTRL-C]* cancels the listing and returns to the monitor prompt. Since the *8031* instruction set includes 255 valid opcodes, disassembling memory that does not contain "real" object code will not be readily apparent.

E	*xxxx*	edit **E**xternal RAM

Use this command to modify values in external RAM, beginning at address *xxxx*. The address is displayed and then the current value contained there. Enter a two character one byte hex number to change the value, *[CTRL-C]* to cancel returning to the monitor prompt, "/" or *[CR]* key to skip forward through memory, and "\" to skip backward through memory. Wrap-around to $FFFF occurs when skipping backward from $0000, and from $FFFF to $0000 when skipping forward.

F	*xxxx-yyyy,zz*	**F**ill external RAM

Use this command to fill a range of external RAM beginning at *xxxx* and ending at *yyyy* inclusive with the value *zz*. The value *zz* is not verified upon write.

G	*xxxx*	**G**o execute

This command starts a full speed execution beginning at address *xxxx*. Entry back into UltraMON51 should only be accomplished with a LJMP $0000. Refer to appendix C for a list of internal UltraMON51 subroutines useful for debugging at full speed execution.

I	*xx*	edit **I**nternal RAM

This command is used to modify values in internal RAM, beginning at address *xx*. The address is displayed and then the current value contained there. Enter a two character one byte hex number to change the value, *[CTRL-C]* to cancel returning to the monitor prompt, "/" or *[CR]* to skip forward through memory, and "\" to skip backward through memory. Wrap-around to $FF occurs when skipping backward from $00, and from $FF to $00 when skipping forward.

M	*abc[CR]*	**M**odify SFRs and registers

This command is used to modify the values contained in the following SFRs and registers: SP, DPH, DPL, P1, A, B, PSW, and R0 through R7. Enter the name of the SFR or register to be modified (from the list above) and hit *[CR]*. Use the backspace to correct any typing errors. The current value contained in the SFR or register is displayed, and you may enter a two character one byte hex number for the new value. Multiple register banks, selected with RS0 and RS1 of PSW, are supported and R0 through R7 are modified properly.

N	*xx-yy*	dump i**N**ternal RAM

Use this command to display a range of internal RAM beginning at *xx* and ending at *yy* inclusive. A maximum of sixteen bytes of memory per line are displayed. Hit a key on the terminal to pause the listing, and hit another key to resume. *[CTRL-C]* cancels the listing and returns to the monitor prompt.

R	*xxxx,zz*	**R**eceive XMODEM-CRC

This command is used to receive data from a host PC to external RAM beginning at address *xxxx*. The parameter *zz* is a timing value, and a value of "04" should work well for the standard 11.0592MHz clock frequency. Refer to appendix D for more detailed information on XMODEM-CRC should the SBC be operating at a different clock frequency or problems are experienced with transfers

When first initiated the "receive XMODEM-CRC" command will begin printing the character "C" across the terminal screen. This is the XMODEM-CRC sync character. Sixty such characters will be printed before the transfer times out, and the transmit or "upload" XMODEM-CRC should be started on the PC sometime before this. The transfer will begin and end, and UltraMON51 will output to the terminal whether the transfer was successful or not as well as the address of the last byte received plus one. XMODEM-CRC always transfers data in blocks of 128 bytes therefore the length of the data received will always be an even multiple of 128. Depending on the terminal program, a short last block (<128 bytes) will be padded to a length of 128 bytes with ASCII SUBstitutes or some other value.

S		dump **S**FRs and registers

This command is used to display the values contained in the following SFRs and registers: SP, DPH, DPL, P1, A, B, PSW, and R0 through R7. Multiple register banks, selected with RS0 and RS1 of PSW, are supported and displayed properly.

T	*xxxx-yyyy,zz*	**T**ransmit XMODEM-CRC

This command is used to transmit the data from address *xxxx* to *yyyy* inclusive to a host PC terminal program. The parameter *zz* is a timing value, and a value of "04" should work well for the standard 11.0592MHz clock frequency. Refer to appendix D for more detailed information on XMODEM-CRC should the SBC be operating at a different clock frequency or problems be experienced with transfers.

Upon initiating the "transmit XMODEM-CRC" command you will have about sixty seconds to begin the receive or "download" on the PC. The transfer will begin and end, and UltraMON51 will output to the terminal whether the transfer was successful or not as well as the address of the last byte transmitted plus one. XMODEM-CRC always transfers data in blocks of 128 bytes, and a short last block (<128 bytes) will be padded to a length of 128 bytes with ASCII SUBstitutes automatically during the transfer. The length of the block specified as input does not by necessity have to be an even multiple of 128 bytes.

V	*xxxx-yyyy*	**V**erify external RAM

Use this command to perform a memory test of external RAM beginning at *xxxx* and ending at *yyyy* inclusive. A bit test is done to determine if the external RAM range specified is good, and upon a failure outputs what address and what bit within that address failed. The contents of the external RAM memory range will be preserved.

W	*xxxx*	**W**alk execute

This command starts a single step execution beginning at address *xxxx*. The current contents of the registers and SFRs are displayed, followed by a disassembly of the instruction to be executed. Hit *[CR]* to execute the instruction, *[CTRL-C]* to cancel the execution and return to the monitor, "S" to skip the execution of the instruction displayed, or "C" when the instruction is a subroutine call (ACALL, LCALL) and it's desired to completely execute the subroutine and return to the calling routine. This is useful for rapidly executing a subroutine which performs some known function, and amounts to a "break execute" through the subroutine.

X	*xxxx-yyyy*	dump e**X**ternal memory

Use this command to display a range of memory beginning at *xxxx* and ending at *yyyy* inclusive. A maximum of sixteen bytes of memory per line are displayed along with the ASCII representation of the memory. Should a byte not be within the printable ASCII character range 32-127, periods (".") corresponding to the non-printable byte will be echoed preferentially. Hit a key on the terminal to pause the listing, and hit another key to resume. *[CTRL-C]* cancels the listing to return to the monitor prompt.

$	*xx*	hex to decimal conversion

This command converts a hexadecimal number *xx* to decimal. A two character one byte hex number is expected as input, and its decimal equivalent is printed immediately following the hex number that was input.

#	*ddd*	decimal to hex conversion

Use this command to convert a decimal number *ddd* to hex. The valid decimal range is 0 - 999. Hit *[CR]* to convert a one or two digit decimal number to avoid entering the leading zeros. Conversion is automatic upon entering the third digit.

HINTS ON DEBUGGING

This sections includes some useful hints about debugging with and using UltraMON51. Many such suggestions could be made and no doubt you will be able to add a few of your own to this list as you dig deeper into using this monitor.

1. The "verify external RAM" command is not only useful for performing a memory test, but also very handy for testing the memory decoding logic of your SBC. When you add memory or change your memory map run this command to see if you have RAM where you "think" you have RAM!

2. When single-stepping through code with the "walk execute" command, be sure to mind the stack pointer and all your stack operations. For example, if you exit a subroutine that was called from some other routine you single-stepped through, remember that the SP is going to reflect the appropriate amount of PUSHes and POPs that occurred during the execution. If your intent is not to "go/walk/break execute" to return to the subroutine you exited from then be sure to reset the SP with the "modify SFRs..." command before starting another "go/walk/break execute" elsewhere. This way unruly UltraMON51 hangups are avoided. UltraMON51 trusts you enough to not crash it!

3. The XMODEM commands are useful for not only transferring object code, but also for generic data in general. For example, if you are debugging a routine that does alot of data manipulation, use them to setup the RAM required by the routine for "what if?" scenarios. Also extremely useful is setting up and saving the "internal RAM" and "SFRs". Since these are located in external RAM, transfers to/from shadow RAM are extremely useful during debugging for setting up the registers to a predetermined condition. From the start of your shadow RAM location, 128 bytes corresponding to "internal RAM" are consecutively stored followed by seven consecutive bytes of "SFR" storage arranged in the following order: SP, DPH, DPL, P1, A, B, and PSW.

4. Some locations within the UltraMON51 EPROM from $0000-$002F are used for the usual jump vectors. Most vector locations are left unreserved (are RETIs in the object code of UltraMON51) and so the object code of UltraMON51 can be modified to accommodate the debugging of routines that may use undedicated vectors. The following vectors (3 bytes each) are in use by UltraMON51:

$0000-Reset vector
$0006-normally unused; UltraMON51 internal vector
$000B-Timer 0 overflow vector
$000E-normally unused; UltraMON51 internal vector

For example, the vectors for external interrupts INT0' and INT1' are free for use. Since UltraMON51 kills all interrupts not required by itself to ensure proper operation, trapping and single-stepping through interrupt routines will not be possible. However, you could single-step through the routine itself (not under interrupt control), or in real-time do a full-speed "go execute" to a routine to enable the interrupt after having modified the appropriate vector within UltraMON51. The next bullet gives a method to modify these vectors.

5. I found it very convenient to run UltraMON51 from nonvolatile RAM as opposed to EPROM. This allowed me to modify vectors and the startup-initialization memory (see appendix B) with UltraMON51 itself on the fly. This was accomplished with the SBC in figure 1 in the following manner: First, the 6264 8K*8 static RAM was replaced with the Dallas Semiconductor DS1225 8K*8 NVRAM, a pin-for-pin compatible nonvolatile replacement for the 6264. Second, a "copy external memory" command was used to copy the UltraMON51 object code to the DS1225. Lastly, the DS1225 was moved to the socket where the UltraMON51 EPROM was, the necessary circuitry modifications made to accommodate it, and the 6264 replaced. And that's it! Of course, care was taken to not overwrite the UltraMON51 object code during its operation and while executing code being developed with it.

APPENDIX A - Example UltraMON51 execution

The following is an example UltraMON51 execution illustrating all of the commands. Comments to this listing are displayed within {braces} and special points of interest are underlined. This output was generated on the SBC depicted in figure 1 with an 8052AHBASIC processor and crystal frequency of 11.0592MHz.

{power on}
*[space]***On-line at 9600 baud** {hit key to set the baud rate}
UltraMON51 V1.0

Shadow RAM: *3E80* {put shadow RAM at high end of test SBC's external RAM}

>*S*
SP:08 DPH:00 DPL:00 P1:FF A:00 B:00 PSW:00
R0:00 R1:00 R2:00 R3:00 R4:00 R5:00 R6:00 R7:00

>*M* **abc[CR]**
*SP[CR]***:08** *40* {stash away some internal data RAM for safe keeping}

>*M* **abc[CR]** {switch to register bank 1}
*PSW[CR]***:00** *08*

>*S*
SP:40 DPH:00 DPL:00 P1:FF A:00 B:00 PSW:08
R0:00 R1:00 R2:00 R3:00 R4:00 R5:00 R6:00 R7:00

>*I* **xx**
00
00: 00 *03*
01: 00 *[/]* {skip over the next few bytes}
02: 00 *[/]*
03: 00 *[/]*
04: 00 *[/]*
05: 00 *05*
06: 00 *[CTRL-C]*

>*N* **xx-yy**
00-7F {dump lower half page of internal RAM}
00: 03 00 00 00 00 05 00 00 00 00 00 00 00 00 00 00
10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

>*M* **abc[CR]**
*R0[CR]***:00** *FF*

>*N* **xx-yy** {R0 is mapped to internal RAM location $08 w/register bank 1}
00-0F
00: 03 00 00 00 00 05 00 00 FF 00 00 00 00 00 00 00

>*X* **xxxx-yyyy** {notice how shadow RAM is equal to internal RAM}
3E80-3E8F
3E80: 03 00 00 00 00 05 00 00 FF 00 00 00 00 00 00 00 ....

>*A* **xxxx**
2000 {compose a small test program on-line}
2000: *MOV A,$00[CR]*
2002: *LCALL $1BBA[CR]* {print decimal number in A (UltraMON51 subroutine)}
2005: *DEC[BS][BS][BS]INC $00[CR]* {backspacing is allowed here for edits}
2007: *MOV A,$00[CR]*
2009: *CJNE A,R7,$2002[CR]***?** {OOPS not a valid instruction}
2009: *CJNE A,$05,$2200[CR]***?** {OOPS can't branch that far}
2009: *CJNE A,$05,$2002[CR]*
200C: *NOP[CR]*
200D: *[CTRL-C]*

>*D* **xxxx-yyyy**
2000-200A
2000: E5 00 MOV A,$00
2002: 12 1B AF LCALL $1BBA
2005: 05 00 INC $00
2007: E5 00 MOV A,$00
2009: B5 05 F4 CJNE A,$05,$2002

>*W* **xxxx**
2000
A:00 PSW:08 B:00 DPH:00 DPL:00 R0:FF R1:00 R2:00 R3:00 R4:00 R5:00 R6:00 R7:00
2000: E5 00 MOV A,$00*[CR]*

A:03 PSW:08 B:00 DPH:00 DPL:00 R0:FF R1:00 R2:00 R3:00 R4:00 R5:00 R6:00 R7:00
2002: 12 1B AF LCALL $1BBA*[C]* {execute this subroutine and return}
3 {this number was output by above subroutine located within UltraMON51}
A:33 PSW:08 B:00 DPH:00 DPL:00 R0:FF R1:00 R2:00 R3:00 R4:00 R5:00 R6:00 R7:00
2005: 05 00 INC $00*[CR]*

A:33 PSW:08 B:00 DPH:00 DPL:00 R0:FF R1:00 R2:00 R3:00 R4:00 R5:00 R6:00 R7:00
2007: E5 00 MOV A,$00*[CR]*
{notice parity change in PSW}
A:04 PSW:09 B:00 DPH:00 DPL:00 R0:FF R1:00 R2:00 R3:00 R4:00 R5:00 R6:00 R7:00
2009: B5 05 F6 CJNE A,$05,$2002*[CR]*
{notice carry is now set}
A:04 PSW:89 B:00 DPH:00 DPL:00 R0:FF R1:00 R2:00 R3:00 R4:00 R5:00 R6:00 R7:00
2002: 12 1B AF LCALL $1BBA*[C]*
4
A:34 PSW:09 B:00 DPH:00 DPL:00 R0:FF R1:00 R2:00 R3:00 R4:00 R5:00 R6:00 R7:00
2005: 05 00 INC $00*[S]* {skip this instruction}
2007: E5 00 MOV A,$00*[CR]* {but execute this one}

A:04 PSW:09 B:00 DPH:00 DPL:00 R0:FF R1:00 R2:00 R3:00 R4:00 R5:00 R6:00 R7:00
2009: B5 05 F6 CJNE A,$05,$2002*[CR]*

A:04 PSW:89 B:00 DPH:00 DPL:00 R0:FF R1:00 R2:00 R3:00 R4:00 R5:00 R6:00 R7:00
2002: 12 1B AF LCALL $1BBA*[C]*
4
A:34 PSW:09 B:00 DPH:00 DPL:00 R0:FF R1:00 R2:00 R3:00 R4:00 R5:00 R6:00 R7:00
2005: 05 00 INC $00*[CTRL-C]* {quit walking-this instruction not executed}

>*I* **xx**
00
00: 04 *02*
01: 00 *[CTRL-C]*

>*B* **xxxx,yyyy**
2000,*200C* 234
A:05 PSW:08 B:00 DPH:00 DPL:00 R0:FF R1:00 R2:00 R3:00 R4:00 R5:00 R6:00 R7:00

>*T* **xxxx-yyyy,zz** {save this routine on host PC for later use}
2000-*200C*,*04*
Ready to upload XMODEM...
0:Successful,exit: $2080

>*#* **ddd**
72[CR]=**0048**

>*$* **xx**
48=**72**

>*V* **xxxx-yyyy**
0000-0001 {program ROM will always fail since is read only}
Testing...0000 Failed:01!=02

>*V* **xxxx-yyyy**
2000-3FFF {test SBC had external RAM addressed here}
Testing...OK

>*W* **xxxx** {can walk execute through program ROM also}
0000
A:05 PSW:08 B:00 DPH:00 DPL:00 R0:FF R1:00 R2:00 R3:00 R4:00 R5:00 R6:00 R7:00
0000: 02 00 30 LJMP $0030*[CR]*

A:05 PSW:08 B:00 DPH:00 DPL:00 R0:FF R1:00 R2:00 R3:00 R4:00 R5:00 R6:00 R7:00
0030: 75 81 08 MOV SP,#$08*[CTRL-C]*

>*R* **xxxx,zz**
3E80,*04*
Ready to download XMODEM...
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
1:Unsuccessful {timeout after about 60 seconds}

>*R* **xxxx,zz**
3E80,04 {setup "internal RAM" to some saved conditions}
Ready to download XMODEM...
CCCCCCCCCCCCCCCCCC{XMODEM-CRC upload started on host PC at this point}
0:Successful,exit: $3FFF

>*F* **xxxx-yyyy,zz**
2000-200F,41

>*C* **xxxx-yyyy,zzzz**
2000-200F,2020

>*X* **xxxx-yyyy**
2000-202F
2000: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
2010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
2020: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA

>*G* **xxxx**
*10[CR]***?** {no backspacing allowed when entering hex numbers}

>*G* **xxxx** {simulate a hardware reset to change the baud rate}
0000
*[CR]***On-line at 19200 baud** {host baud rate changed to 19200}
UltraMON51 V1.0

Shadow RAM: *3E80*

>*S* {all SFRs and lower half page internal RAM at reset condition values}
SP:08 DPH:00 DPL:00 P1:FF A:00 B:00 PSW:00
R0:00 R1:00 R2:00 R3:00 R4:00 R5:00 R6:00 R7:00

>*E* **xxxx**
3F00 {notice that SFRs are shadowed here}
3F00: 08 *01* {SP}
3F01: 00 *02* {DPH}
3F02: 00 *03* {DPL}
3F03: FF *04* {P1}
3F04: 00 *05* {A}
3F05: 00 *06* {B}
3F06: 00 *07* {PSW}
3F07: 00 *[CTRL-C]*

>*S*
SP:01 DPH:02 DPL:03 P1:04 A:05 B:06 PSW:07
R0:00 R1:00 R2:00 R3:00 R4:00 R5:00 R6:00 R7:00

> □

APPENDIX B - Default baud rate/shadow RAM address

When developing UltraMON51 it was realized that in some user SBC systems a clock frequency other than the standard 11.0592MHz would be used. Therefore UltraMON51 sets aside a small amount of program ROM (32 bytes) within the UltraMON51 8K object area to be used for self-supplied user initialization routines. Since UltraMON51 occupies practically the entire 8K space allotted for it, the amount of room set aside is very small however should prove sufficiently large to contain the object code intended to occupy it.

Just after power-on or reset, the UltraMON51 examines location $1FE0 to see if anything other than $FF (indicating a blank EPROM cell) is found. If $FF is found, the auto-baud rate detect routine is called, immediately followed by the shadow RAM address location query before displaying the ">" prompt. If anything other than $FF is found, UltraMON51 will execute a LCALL $1FE0 and make the assumption that object code to initialize timer/counter 1 for serial communications as well as code to initialize internal RAM locations $7C and $7D (shadow RAM high byte and shadow RAM low byte respectively) is found at $1FE0. Upon RETurn from the called subroutine, the auto-baud rate detect routine and shadow RAM address location query is skipped to go straight to the ">" prompt. For example, the following bit of code will initialize timer 1 for 19.2K baud and setup the shadow RAM location as $3E80:

```
>D  xxxx-yyyy
    1FE0-1FF4
1FE0:  75  98  52     MOV SCON,#$52      {8 bit UART enabled}
1FE3:  75  89  20     MOV TMOD,#$20      {8 bit auto-reload}
1FE6:  75  8D  FD     MOV TH1,#$FD       {9600 baud @ 11.0592}
1FE9:  43  87  80     ORL PCON,#$80      {...and double to 19200}
1FEC:  D2  8E         SETB TR1           {start timer 1}
1FEE:  75  7C  3E     MOV $7C,#$3E       {shadow RAM high byte}
1FF1:  75  7D  80     MOV $7D,#$80       {shadow RAM low byte}
1FF4:  22             RET                {return from user init}
```

It will not be possible to only skip the auto-baud rate detect routine but keep the shadow RAM address query, or vice-versa. The user supplied routine must initialize both.

It is extremely handy to customize your copy of UltraMON51 in this fashion for a couple of reasons. First, if your SBC system is static with respect to clock frequency and memory map, supplying a user initialization will save the repetitive entry of the same keystrokes every time you enter the monitor. Second, you can prevent some potential confusion in the following way: since UltraMON51 must only be re-entered from a full speed execution at $0000 and since the auto-baud rate detect routine waits for a keystroke to begin communications without any terminal display to let you know UltraMON51 is successfully back on-line, the difference between a "hang-up" of the code you are working on and a successful return to UltraMON51 may not be immediately apparent. The following illustrates this point:

```
$2000   {code starts here}
 .....
 .....
 .....
 .....      {code ends here}
$2100   LJMP $0000
```

Executing with the "go execute" command at $2000 gives two possibilities: the code could hang somewhere between $2000 and $2100, or the code could successfully complete and return back to the monitor. Including an initialization routine at $1FE0 would immediately display the ">" prompt on the terminal to let you know your code has finished executing and the monitor is now back on-line, but the auto-detect routine would sit patiently, displaying nothing on the terminal, until a key is pressed. The difference between the two scenarios may seem minor, but in the heat of debugging battle the difference between your code hanging up or not would be immediately apparent.

There are many ways to add your user initialization code to the UltraMON51 EPROM. Probably the easiest method is to download the UltraMON51 object code with an EPROM programmer to a PC, edit the object file or hex file to include your initialization routine, and reprogram a different part to reflect the changes. The section "HINTS ON DEBUGGING" gives another possible method.

APPENDIX C - Useful UltraMON51 Subroutines

Many of the subroutines within UltraMON51 are useful when it comes time to debug at full speed execution. The following is a table of a few including their absolute addresses and any parameters required by the routine/subroutine along with its function. Disassemble or "walk execute" through the subroutine to determine what registers are utilized during operation. These routines assume that the *8031* serial UART has been initialized and is functioning properly.

$0000 RESET
A LJMP here should be the only method employed to re-enter the monitor.

$0628 REGSDUMP
A LCALL here will perform a non-destructive SFR/register dump. No registers or flags are affected by the subroutine but sufficient stack space must exist to support ten stack operations..

$1C15 OUTSPACE
A LCALL here will output to the terminal an ASCII space.

$1C0D OUTCRLF
A LCALL here will output to the terminal an ASCII line-feed and carriage return.

$1C17 OUTCHAR
A LCALL here will output to the terminal the current value in the accumulator as an ASCII character. A key press on the terminal during transmission will stop the output and another key press will continue it.

$1BF8 OUTHEX
A LCALL here will output to the terminal the current value in the accumulator as a hex number.

$1BF1 OUTDPTR
A LCALL here will output to the terminal as a two byte hex number the current value of the DPTR.

$0859 OUTSTRING
A LCALL here will output to the terminal an ASCII string. The ASCII string to be output must immediately follow the LCALL and must be terminated with a 00 value. Execution will continue with the instruction proceeding the 00.

$1BBA OUTDECIMAL
A LCALL here will output to the terminal as a decimal number the value currently contained in the accumulator.

$1C8F INCHAR
A LCALL here will return into the accumulator the ASCII value of a character input from the terminal. When called, this subroutine will wait indefinitely for a key press.

$1C41 INHEX
A LCALL here will return into the accumulator the value of a hex number input. If the one byte hex value entered was valid, the carry flag will be clear and ACC will contain the value. If a syntactically invalid hex number was input the carry flag will be set. See INCHAR subroutine.

APPENDIX D - Further notes on XMODEM-CRC

XMODEM-CRC is a method for transferring information between computers and enjoys an almost universal support in PC terminal programs; its inclusion in UltraMON51 is a bit of a departure in monitor programs. Originally the XMODEM-CRC subroutines were written as part of a remote data acquisition project which needed to be able to transfer information back and forth over the phone lines, and to insure the data transferred was complete and error free. XMODEM-CRC lends itself very well to inclusion in a monitor program such as UltraMON51 due to its non-dependency on formatted data. Eight bit data of any sort can be transferred, irregardless of its nature. In addition, with the proper interface circuitry and additional software, UltraMON51 could be used gather, transmit, receive data, or allow for the complete modification of relevant object code over the phone lines with a modem interface in real-time. If interested, the source code as well as related technical information on XMODEM-CRC for the *8031* was recently published in the May 1993 issue, pp.61, of *ComputerCraft* magazine.

Both the "receive" and "transmit XMODEM-CRC" commands require a timing parameter to be entered before the transfer is begun. For a clock frequency of 11.0592MHz the best value to use here is "04" and works well in most of the PC terminal software I have tried it on, however it's possible you may experience problems with this value with a fussy terminal program. The problem usually manifests itself by not allowing the transfer to even begin. Try experimenting first by increasing the value, and then by decreasing the value to see if the problem goes away. I have observed that some terminal programs have a "relaxed XMODEM timing" setting. If so then set it. Some PC terminal programs aren't fussy about XMODEM timing and some are; hence the inclusion of this variable parameter in UltraMON51.

Should your SBC be operating with a different clock frequency a different timing parameter should be used, and is easily experimentally determined in the following way. First, initiate a "receive XMODEM-CRC" command" and start with the timing parameter "04". With a stopwatch, determine the time interval between the display of successive "C"s and let UltraMON51 time-out the transfer (60 "C"s will be displayed before time-out). If your clock frequency is less than 11.0592MHz, the time interval will be something greater than about one second (decrease the timing parameter). If your clock frequency is greater than 11.0592MHz, the time interval will be something less than about one second (increase the timing parameter). Keep initiating the "receive" command with a different timing parameter and clocking the "C"s with a stopwatch until the interval between "C"s is about one second. In most cases it's best to stay on the slightly higher side of one second. Jot the value down and use this timing parameter with both the "receive" and "transmit XMODEM-CRC" commands.

APPENDIX E - Supported SFR bit and byte names

The following is a list of all the byte and bit addressable SFR names supported by the assemble and disassemble commands. With the assemble command, you may refer to the SFR by its hex address or by its name. The disassemble command always substitutes the SFR name for an direct address or bit address when possible. These SFR names in UltraMON51 conform to the naming convention found in the Intel Embedded Handbook Volume 1.

Byte addressable SFR names:

P0	SP	DPL	DPH	PCON	TCON	TMOD	TL0
TL1	TH0	TH1	P1	SCON	SBUF	P2	IE
P3	IP	T2CON	RCAP2L	RCAP2H	TL2	TH2	PSW
ACC	B						

Bit addressable SFR names:

P0.0	P0.1	P0.2	P0.3	P0.4	P0.5	P0.6	P0.7
IT0	IE0	IT1	IE1	TR0	TF0	TR1	TF1
P1.0	P1.1	P1.2	P1.3	P1.4	P1.5	P1.6	P1.7
RI	TI	RB8	TB8	REN	SM2	SM1	SM0
P2.0	P2.1	P2.2	P2.3	P2.4	P2.5	P2.6	P2.7
EX0	ET0	EX1	ET1	ES	ET2	EA	
P3.0	P3.1	P3.2	P3.3	P3.4	P3.5	P3.6	P3.7
PX0	PT0	PX1	PT1	PS	PT2	CPRL2	CT2
TR2	EXEN2	TLCK	RCLK	EXF2	TF2	P	OV
RS0	RS1	F0	AC	CY			
ACC.0	ACC.1	ACC.2	ACC.3	ACC.4	ACC.5	ACC.6	ACC.7
B.0	B.1	B.2	B.3	B.4	B.5	B.6	B.7

Note that with bit addressable SFRs that can be referred to by "SFR.bit" or by "bitname", UltraMON51 supports the more descriptive bitname. For example, "PSW.5" is not recognized but "F0" (the name of bit 5 in PSW) is assembled correctly. There was not sufficient space to support both within UltraMON51.

Instead of shortening "POP ACC" and "PUSH ACC" to "POP A" and "PUSH A" UltraMON51 will assemble and disassemble these instructions as the former because they are not instructions inherent to the accumulator as implied by the latter, but actually two byte direct addressing mode only instructions. Intel names the SFR for the accumulator (direct address $E0) "ACC" not "A".

APPENDIX F - UltraMON51 Command Quick Reference

A	*xxxx*	**A**ssemble
B	*xxxx,yyyy*	**B**reak execute
C	*xxxx-yyyy,zzzz*	**C**opy external memory
D	*xxxx-yyyy*	**D**isassemble
E	*xxxx*	edit **E**xternal RAM
F	*xxxx-yyyy,zz*	**F**ill external RAM
G	*xxxx*	**G**o execute
I	*xx*	edit **I**nternal RAM
M	*abc[CR]*	**M**odify SFRs and registers
N	*xx-yy*	dump i**N**ternal RAM
R	*xxxx,zz*	**R**eceive XMODEM-CRC
S		dump **S**FRs and registers
T	*xxxx-yyyy,zz*	**T**ransmit XMODEM-CRC
V	*xxxx-yyyy*	**V**erify external RAM
W	*xxxx*	**W**alk execute
X	*xxxx-yyyy*	dump e**X**ternal memory
$	*xx*	hex to decimal conversion
#	*ddd*	decimal to hex conversion

Notes:

1. CTRL-C to cancel a listing and return to ">" prompt.

2. Any key but CTRL-C to pause/continue a listing.

3. Backspace permitted only with "assemble" and "modify SFRs and registers" commands.

4. During any "edit" command, "/" or "CR" (carriage return or enter key) will skip to next location, "\" will back skip to previous location.

5. During "walk execute" command, "C" will execute a subroutine completely at "break execute" speed, "S" will skip displayed instruction, "CR" will execute displayed instruction, "CTRL-C" to return to monitor.

6. For "receive/transmit" commands, "zz" is a timing value-04 works well for 11.0592MHz crystal frequency.