MTU
UPDATE
3RD QUARTER '82
Vol. 1 No. 2

CP/M Now Available for MTU-130 Users!

Due to popular demand, MTU has undertaken the development of another slave processor board, this time with a Z-80 microprocessor. When equipped with a Z-80 slave processor board, the MTU-130 will be able to run most of the vast quantity of CP/M based software that is available in the marketplace. Such software includes dozens of business programs, database managers, word processors, and proofreading programs. Also available are scientific packages for numeric analysis and symbolic math and all kinds of language processors. With this board, MTU can provide the environment needed by these commercially available CP/M programs.

Keep in mind though that CP/M programs are typically written for the most rudimentary level of operator interaction (to achieve compatibility with a wide variety of systems) and therefore will not use the MTU-130's graphics, sound, light pen, function keys, legends, and other advanced features. Thus MTU developed software will continue to be CODOS/6502 based and optimized for use on the MTU-130.

The board itself consists of a 4MHz Z-80A processor and 64K of dual-port memory. The MTU-130's 6502 microprocessor can access all 64K of the memory in either Bank 2 or Bank 3 (jumper selectable) and in fact the Z-80 board acts as a memory expansion for the 6502 when the Z-80 is not running. The Z-80 can only access its on-board 64K. In addition, the 6502 can interrupt the Z-80 on two levels and the Z-80 can interrupt the 6502.

Besides processor and memory, the board will be equipped with a Centronics compatible parallel printer port and a full serial port with programmable baud rate and modem controls (6551 chip). With the printer port, the board can double as an intelligent 64K printer buffer to free up the '130 during long print runs. The serial port can optionally drive a serial printer or be used by CP/M to drive an external terminal directly.

This will be a small board that can plug into any MTU-130 slot. The board is presently in the prototype stage and should be available for shipment along with CP/M by the end of the year. ◆

On the Inside

From the President

As you can see by our new format, some changes are starting to take place at MTU.

(continued p.2)

A forum for the exchange of ideas and information exclusively for MTU-130 users
by MICRO TECHNOLOGY UNLIMITED, a Division of Consolidated Sciences, Inc.

(continued from front cover)

Like many companies today, we have been in a period of evaluation of just where we fit into the marketplace. With our research completed we feel comfortable and are moving forward. First let me give you a synopsis of MTU history.

In our early years we built add-on products for the KIM, SYM, AIM, and PET marketplace. Most customers in this area were ultimately concerned with price and less concerned with performance. But, MTU is a performance company dedicated to low price. We have been successful in marketing our expansion products to those persons who had very low budgets and needed the high performance expansion we supplied.

As the microcomputer market matured, it became more and more obvious that single-board systems were being phased out or used "as is" with less and less expansion. Those who wanted a "full" system were buying one in the first place. This is reflected in the growth of APPLE and PET in 1980-81. The marketplace is now innundated by full systems and the demand for inexpensive systems is being filled by TI, VIC, ATARI, etc. which do not lend themselves to MTU style performance and expansion.

In 1980 we reviewed our position as a peripherals supplier tied to large companies not sharing the same goals for customer support and system performance. We felt that it was time for MTU to design its own complete system. To develop a product as complex as the MTU-130 requires significant investment of both people and money resources. This investment has slowed our growth for the last 2 years compared to what we had been achieving. Now that the product exists, we are rapidly producing expansion peripherals. Because system expansion was planned for, the amount of effort necessary to design a significant add-on function is a fraction of what we used to expend. This allows us to do a better job and do it faster than before.

Our direction for the MTU-130 is to firmly establish it as the ELITE SYSTEM for descriminating individuals who desire and need something more than the mass market micros. We will continue to improve our image through print and advertising to properly represent the system as it should be seen.

We now have many dealers and system houses interested in carrying the MTU-130 since it is no longer an "unproven" product. You have helped us prove it, and now they believe it! It is difficult to sell any complex product such as the MTU-130. It is necessary to see an actual demonstration by someone knowledgeable about the system. Thus, we need others in contact with our potential customers who can give a demonstration. Since the special introductory period of the MTU-130-2D system is over, the price reverts back to that established in 1981 of $4995. This allows us to offer commissions on sales.

We expect to be using dealers, but who should be an MTU dealer? We have worked with computer "dealers" for 4 years now on our other products. The results have been enlightening. As you know, the MTU-130 is a very flexible system offering few restrictions. Most microcomputers sold by dealers have restrictions which are easily found, limiting the need to understand the system in depth. We need dealers who understand technology and how to apply it to problems in the real world. We need your recommendations for dealers we should contact in your area (or elsewhere if applicable) who you feel would properly represent MTU and you. We will be doing our own search, but any direct feedback from you would save us hours of time and even false starts that could otherwise be prevented.

In addition to dealers, we feel that you, our existing customers, can help immensely in "getting the word out" about your system. For this reason, we have taken a unique approach to marketing which we call:

USER REFERRAL MARKETING

This plan is designed to allow you to earn purchase credits toward any MTU product or make money by simply doing what you already

(continued p.3)

(continued from p.2)

do, demonstrating and talking about your system to others. To participate, read on!

The prime requirement is that you must own an MTU-130 series system, or an MTU-100 upgraded to 130 status. The basic concept is that as an MTU-130 user, you are capable of and willing to demonstrate your system to others. In so doing, you generate new contacts for MTU which can be turned into sales. You may obtain names of potential customers in any way you desire as long as the names submitted are truly qualified.

There are no territorial limitations on your referrals; you may refer any person anywhere. For example, you may demonstrate to a friend visiting from out-of-state. You must qualify the potential customer prior to sending in the Referral Form. Qualification consists of the following:

*Demonstrate your MTU-130 computer to the potential customer or,
*Talk to the potential customer in person or by phone,
*And the person wishes to be contacted further by MTU or an MTU dealer.

If you are interested, request copies of the MTU USER REFERRAL FORM from us. This is a very simple form which documents who you are and who your referral customer is. When submitting a qualified customer name, you simply send this filled out form to MTU and keep a copy for yourself. The referred customer must purchase MTU-130 related products within one (1) year of the date on your Referral Form for commission to be paid. If the customer was a properly qualified sales lead he will usually buy within several months of learning of the MTU-130.

If you are interested in receiving leads from MTU for you to call or demonstrate, (which you are credited for if they purchase) call or write us for the separate Approved User Form. We will not disclose your name to someone calling us for a nearest user unless you have agreed to participate in this second program. This protects you and your privacy. Commissions on qualifying sales are accrued by MTU-130 user serial numbers. This is you if the MTU-130 you use to demonstrate is your personal property, or your department at work if the MTU-130 is owned by your company, university, etc. If there are any conflict of interest problems with your employer, please resolve them prior to referring persons who might be in conflict.

When a sale is made to someone you referred, MTU will credit your account with 4% of the total initial retail sale value less shipping and any applicable taxes. If MTU supplied the referral name initially to you as a lead, the commission will be 3%. For example, a retail sale of a single MTU-130-2D system only with a 4% credit would be $200 credit. Credits in your account can be taken equally as cash or credit toward hardware or software. We feel that this plan offers something for all of you who wish to earn credits or money using your computer. We are trying to establish a mechanism where those of you who have expressed an interest in this type of plan can participate. If you are not interested, please, do continue telling your friends and associates about your system. All referrals are kept confidential.

As always, I welcome any feedback you may have on our image approach or the USER REFERRAL MARKETING plan. Please write or call me (sometimes I'm hard to reach by phone). Your feedback is appreciated. To those of you who have already been working with me on these plans, thank you again. ♦

DAVID B. COX, President

New Products

CODOS ON THE 68000?

Well, yes; at least virtually. DMXMON is a "cross monitor" program that in conjunction with the Datamover-256 board makes the MTU-130 appear to the user and programmer as if it were in fact a 68000 based system! DMXMON was designed to make all of the MTU-130 facilities that are normally available to the 6502 programmer

(continued p.4)

(continued from p.3)

also available to the 68000 programmer. DMXMON is just like CODOS with all of its power and convenience so there is no need to obtain and learn a new operating system just to tap the power of the 68000 microprocessor. The best part is that DMXMON is included FREE with the Datamover-256 board!

For the user, DMXMON offers the complete array of CODOS file maintenance commands such as FILES, RENAME, DISK, KILL, TYPE, and others. These operate exactly like their CODOS counterparts and in fact are implemented by "passing the command through" to CODOS for execution. 68000 programs are executed simply by typing their name along with any required arguments just as with 6502 programs. Job files can also be used to automatically execute DMXMON commands, run 68000 programs, and even switch back and forth between DMXMON for 68000 related operations and CODOS for 6502 stuff.

For the 68000 programmer, DMXMON offers an extensive array of over 90 "supervisor calls" (SVC) that allow access to all of the CODOS and MTU-130 I/O functions. All of the applicable CODOS SVCs are provided as well as SVCs for every major entry point into the standard text and graphics routines available on the '130. Additional SVCs allow a 68000 program to read/write any 6502 memory location, call 6502 subroutines, and control timers. There is even a group of SVCs that permit any portion of the 68000's memory to be displayed as graphics on the MTU-130 screen. Huge graphic arrays (such as 1000x1000 pixels) may be easily manipulated and displayed through a 480x256 movable "window" on the '130 screen.

For the debugger, DMXMON offers an extensive array of memory manipulation commands, execution error messages, and 68000 program breakpoints. All of the standard CODOS memory manipulation commands such as SET, FILL, HUNT, DUMP, COMPARE, etc. are provided--the only difference being that they operate on 68000 memory. Full 24 bit addresses are recognized and 32 bit arithmetic is used in expressions. When the 68000 microprocessor recognizes an error condition such as an illegal instruction, DMXMON will report it with an English error message showing the problem location and full register contents. Up to 4 breakpoints may also be set to facilitate debugging.

Thus the Datamover-256 user receives a powerful (and familiar) disk operating system and a full array of I/O device driver subroutines right along with the board at no extra charge. Customers who have already purchased their Datamovers have also received a preliminary version of DMXMON. The final version is presently running in-house and will be sent to these people shortly. If you have not yet purchased a Datamover and would like to know more, the two manual set (Datamover hardware manual and DMXMON user's manual) is only $10.00 with a prepaid order. ◆

THE WORDPIC™ WORD PROCESSOR IS HERE!

The best way to describe WORDPIC is to show you so we've included a sample printout here with some of the highlights. WORDPIC files can be easily processed with the MTU-Editor, and can be read or written by BASIC and other programs.

The WORDPIC word processor is priced at $395. The WORDPIC package presently supports the NEC 8023 printer. We also plan to extend support to other printers. In addition, the complete assembly language source code for the printer-dependent part of WORDPIC is included on disk. Knowlegeable users can alter this to drive their own printers.

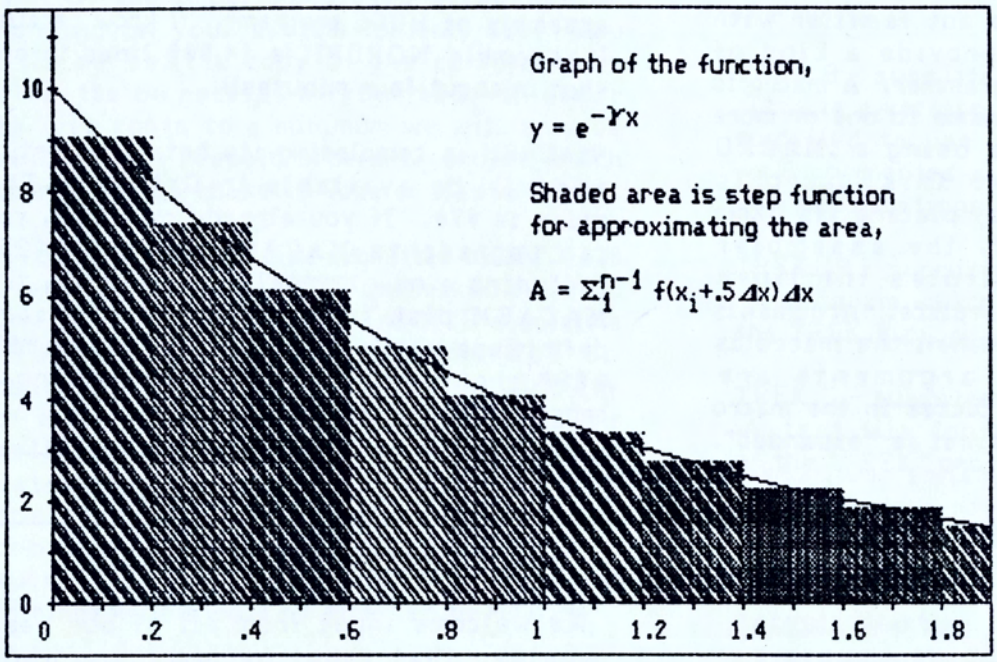
Besides the word processor and print program, you also get a program called WORDSHELL. WORDSHELL provides a menu-driven environment for word processing, and eliminates the need to learn anything about CODOS before using the word processor. WORDSHELL displays a sorted list of word processing files currently available on the disk in drive 0. You can pick a file for editing or printing using the cursor and function keys. Extensive error checking is provided and error messages include suggestions for how to correct the problem. ◆

WORDSHELL is a trademark of Micro Technology Unlimited.

WORDPIC™

WORD/PICTURE PROCESSOR FOR MTU-130 COMPUTERS

Micro Technology Unlimited brings a whole new dimension to Word Processing with the introduction of WORDPIC for the MTU-130 computer. For the first time, you can print high-resolution graphics information mixed with your text! What other word processor can do this:



Using WORDPIC you can print high-resolution graphics images of up to 1280 x 2048 pixels, intermixed with text. You can print text beside graphics, text over graphics, multiple pictures side by side or superimposed, etc. Graphics images created by BASIC programs, MTU's Graphic Editor, or other programs can be directly merged with the text. CRT screen images and even digitized photographs can be used. Graphic images can be inserted, deleted, and moved by WORDPIC just as easily as ordinary text. The graph at the left was created by an 8 line BASIC program.

Despite its sophistication, WORDPIC is one of the easiest word processors to use. The MTU-130's green phosphor CRT monitor displays 80 sharp characters by 24 lines of text, and can edit lines up to 160 columns wide. Upper and lower case characters with descenders plus a full set of special characters including Greek and math symbols (π, Σ, Δ, ∞, etc.) are displayed on the CRT just as they will be printed on the hardcopy. An interrupt-driven keyboard assures no lost keystrokes no matter how fast you type. Full screen editing is used with "infinite scrolling" which allows you to view and edit any part of a file of up to 1,000,000 characters. Cursor control keys which repeat when held down make positioning on the screen a snap, and the INSERT and DELETE keys make editing fast and easy. You dont' have to memorize a list of complex "control key" sequences to edit your text as you would with other word processors. Instead, you control WORDPIC by simply pressing function keys, each of which has a displayed legend. Press the JUSTPARA key and the paragraph justifies itself between the current margins, right before your eyes! Press CENTER and the line is instantly centered. It's that easy!

Write or call MTU for more detailed information on these other features:

- Automatic "word wrap", even during inserts.
- Automatic margins, variable within the document.
- Automatic tabstops, variable within the document.
- Global or local search or search-and-replace.
- Repeat and "veto" options for search-and-replace.
- Searches may be limited to line and column ranges.
- Multiple searches can be combined in a single command.
- Block DELETE, MOVE, COPY, SAVE.
- Can recall deleted block.
- External files may be merged into text.
- Block write to external file(s).
- Pica, Elite, Proportional & Compressed print pitches.
- True proportional spacing for "typeset" appearance.
- Flush justification.
- Subscripts and superscripts (or both: Σx_i^2).
- Underline and w i d e print.
- 6 or 8 lines per inch; 1, 1.5, 2, or 3 line spacing.
- **overstrike** and **multipass** print.
- Print multi-column text one column after another.
- Print all or selected pages from document.
- Print text-on-graphics, text-beside graphics, etc.
- Print extended fonts such as Greek and math symbols.

WORDPIC GENERATED THIS ENTIRE PAGE!

If a picture is worth a thousand words, how many conventional word processors is WORDPIC worth?

6502 MACRO ASSEMBLER

All you assembly language hackers take note! A new resident 6502 macro assembler is now available for the MTU-130. MACASM is upward compatible with ASM, the previous assembler. If you are not familiar with the concept, macros provide a kind of "shorthand" for the programmer. A macro is defined by assigning a name to one or more lines of program code using a .MACRO directive. Whenever this macro is subsequently invoked by placing its name in the opcode field, the assembler automatically substitutes the lines previously defined in its place. Arguments may also be specified when the macro is invoked, and these arguments are substituted at selected places in the macro body automatically when it is "expanded". For example,

```
DISPLAY .MACRO      ;DISPLAY 'MESSAGE'
        SVC    2
        .BYTE  2,!1,0
        .ENDMAC
        ...
SHOW    DISPLAY  'Hello'
```

defines a macro called DISPLAY and invokes it with an argument of 'Hello'. The assembler will expand the invocation as follows:

```
SHOW    DISPLAY  'Hello'
        SVC    2
        .BYTE  2,'Hello',0
```

Macros may have up to 36 arguments of any length. A macro may invoke other macros or even invoke itself recursively. Local labels are supported, so that a macro with labels in its definition can be invoked any number of times without creating a duplicate label.

In addition to macros, the improved assembler supports seven new conditional assembly directives, including the ability to test for null arguments in a macro invocation. Also included is a .FILL directive for creating pre-initialized block data, and a .DATE directive which embeds the date of assembly into the object code.

MACASM automatically "sniffs" for the presence of expansion RAM in bank 3 (such as a DATAMOVER), and if present, the cross reference list and macro definitions are stored there instead of in the regular symbol table. This allows room for assembly of HUGE programs. (I know because I assemble WORDPIC's 14,000 lines in one shot in about four minutes!)

MACASM is completing its beta-site testing and will be available in October. The price is $90. If you already own ASM, you can upgrade to MACASM for only $25, including a new manual. As a bonus, the MACASM disk includes a set of macro definitions which allow MACASM to assemble 8080 programs. This is like getting a cross-assembler free! ◆

The MTU User Group

USER INPUT INFORMATION

We welcome input from all of our users whether that input be programs, hints, music, pictures or any other item which you think may be useful and/or interesting to other MTU-130 users.

We would prefer that your input be provided on single sided diskettes--see below for information on their return. Programs should be accompanied by a separate text file containing a one paragraph abstract of the program suitable for publishing in the next newsletter as well as specific operating instructions for your program. Articles or other text should be written using the MTU Editor with 43 columns maximum width. We use one space before the first word in the paragraph and a line between paragraphs.

All programs submitted to MTU will be considered in the public domain and the user submitting the program is requested to include a statement regarding the original author if other than the submitter as well as any known previous publication history.

For those submitting material on diskette we will make a copy of that material and a copy of the then current user group

(continued p.7)

(continued from p.6)

diskette for return to you in the same packing in which we received it. ♦

HOW TO GET YOUR USER DISKETTE

All of the programs listed below can be running on your system for only $15! You will be sent a copy of the current user diskette on receipt of your $15. In order to keep costs to a minimum we will only be able to fill prepaid orders or orders which charge the purchase to Visa or MasterCard.

Programs on this disk submitted by MTU-130 users are provided AS IS, unchecked by MTU. The MTU-130 User Group and MTU disclaims responsibility for any direct or consequential damages incurred in using any of these programs. User Group programs are believed but not verified to be free of patent and copyright restrictions. Any operational questions about User Group programs should be directed to the submitter. Any questions about copyright infringement should be directed to MTU. ♦

USER GROUP DISK NUMBER II CONTENTS

AN1.B - AN510.B (90 programs) - submitted by Jerry Froelich.
These are all of the programs from the book, SOME COMMON BASIC PROGRAMS 3rd EDITION, plus a few extras. Obtaining a copy of this book is almost essential for using these programs. The programs range from one that calculates the future value of an investment, to one that does multiple linear regression. You may obtain the book from your local book store or from OSBORNE/ McGraw Hill, 630 Bancroft Way, Berkeley, CA 94710.

MENU1.B - submitted by Jerry Froelich
A menu program to let you run the above programs from a menu.

HELP.B - submitted by Jerry Froelich
A "help" program to be used with MENU1.B above.

ADP.J - submitted by Jerry Froelich
A job file to start up the MENU1.B program.

CONVERT.B - submitted by Jerry Froelich
Converts decimal to hex, octal, or binary. Also converts hex and octal to decimal. This is the same program as on the last User Group Disk, but with a few bugs corrected.

TREK.B - submitted by Dale Hedman
This is a version of "TREK" which has been modified to use the VGL and IGL. It is well commented so the users may add their own modifications.

TREKI.B - submitted by Dale Hedman
A program which prints instructions for the TREK.B program.

TREK.T - submitted by Dale Hedman
A text file containing brief description of the "TREK" programs.

RENUMBER.C - submitted by Dale Hedman
A machine language utility program which renumbers the BASIC program in memory. Accepts starting line number and increment as arguments. Corrects all line number references in GOTO's, GOSUB's, etc.

RENUMBER.T - submitted by Dale Hedman
Instructions for using the RENUMBER.C program.

DIRLINKTAB.J - submitted by Dale Hedman
A job file which prints a memory dump showing the Block Allocation Table and Directory of the disk in drive 0. It uses the DISKETTE Utility (found on your Distribution Disk, not supplied on this User Disk) to read the appropriate sectors into memory, then dump them to the printer. This program may be used in conjunction with the CODOS DISK ACCESS article in this newsletter.

MAZEMAKER.B - submitted by David Maynard
A program to generate mazes on the MTU-130 display screen using a weighted and slightly modified version of a spanning tree algorithm published in the December 1981 issue of BYTE.

PASSWORD.C - submitted by David Maynard
A program which, when inserted in the STARTUP.J file, prevents a person from

(continued p.8)

(continued from p.7)

gaining access to CODOS during a BOOT operation unless he or she enters a user-definable password from the console.

PASSWORD_DOC.T - submitted by David Maynard
Description and operating instructions for PASSWORD.

PCM8023.C - submitted by David Maynard
A machine-language program which allows rapid setting of the NEC PC-8023A printer's various printing modes and formats. The program is written using simple, easily modifiable programming techniques to allow anyone with the MTU Assembler to adapt the program for their own printer.

PCM8023.A - submitted by David Maynard
Source code for PCM8023.

PCM8023_DOC.T - submitted by David Maynard
Documentation and operating instructions for PCM8023 Printer Configuration Monitor.

UNLIST.C - submitted by Hal Chamberlin
A Utility program which converts an assembly listing file produced by the ASM program (or MACASM Macro-Assembler, see announcement in this newsletter) into a source file that may be reassembled. This can be very handy if you accidentally delete your assembly source file.

UNLIST.A - submitted by Hal Chamberlin
The assembly language source code for the UNLIST program.

UNLIST.T - submitted by Hal Chamberlin
Instructions for using the UNLIST program.

SHEPARD.T - Submitted by Ralph Erickson
SNOTRAN source statements for a demonstration of the Shepard tones. This is an infinite series of tones in which each successive note seems to be higher in pitch than its predecessor!

SHEPARD.M - Submitted by Ralph Erickson
Compiled version of SHEPARD.T. To play the Shepard tones, GET SHEPARD.M and then SPLAY. (SPLAY may be found on your distribution disk.)

AIMTP.C - submitted by Charles Davis and Albert Markhart Department of Horticulture, Univ. of MN, St. Paul, MN 55108.
This program reads AIM-65 format cassette tapes through the cassette input port and stores the information onto a disk file.

AIMTP.A - submitted by Charles Davis and Albert Markhart.
This is assembly language source code for AIMTP.C

AIMTP.T - submitted by Charles Davis and Albert Markhart.
Detailed description of how to use AIMTP.C program. It also describes how the program works.

SKIPOVER.B - submitted by Tom Wason
This program sets the VFU verticle forms control in the NEC 8023 printer to automatically skip over the perforations in fanfold paper. It allows you to specify the number of lines to be skipped.

SKIPOVER.T - submitted by Tom Wason
Instuctions for the SKIPOVER.B program. ♦

Notes from the Factory

EDITOR BUG

There is a minor bug in the MTU-130 EDITor on Distribution Disk 1.4. The symptom is that occasionally you may erroneously get the message "BACKUP IS SAME FILE" after you specify a backup file name when entering the Editor. The following procedure will correct the problem:

```
GET EDIT
DUMP 31A0
```

This should display the value in address 31A0 as 03. If it doesn't, you don't have the latest version of the Editor, and should not proceed. If it does, enter:

```
SET 31A0=1
RESAVE EDIT=700 ABD0 AD7F 0700 3293
```

This completes the modification. ♦

HOW DOES YOUR BAUD RATE?

The transmission speed (baud rate) of the MTU-130's serial port is controlled by a separate oscillator which uses a "resonator" (inexpensive crystal). The proper frequency for this resonator is 3.68MHz but a few of the early machines were shipped with 3.58MHz resonators due to late delivery of the correct, customized frequency. The resulting 2.7% error is generally not significant when communicating with serial printers or the DC Hayes Smartmodem but it can lead to excessive error rates when using other modems or communicating with the serial port of a SYM-1 single-board computer. If you have difficulty in communicating with some serial devices and have had your machine for some time, you may wish to enter the following program which will measure the exact baud rate of the serial port. Simply enter the following CODOS commands exactly as shown:

```
SET 700 D8 38 66 EE 00 0C 8D C9
SET 708 BF A9 16 8D CB BF A9 0A
SET 710 8D CA BF A9 00 85 B0 85
SET 718 B1 A9 C8 85 B2 A9 40 8D
SET 720 DB BF A9 50 8D D4 BF A9
SET 728 C3 8D D5 BF 8D C8 BF AD
SET 730 C9 BF 29 10 F0 09 8D C8
SET 738 BF E6 B0 D0 02 E6 B1 2C
SET 740 DD BF 50 EB AD D4 BF C6
SET 748 B2 D0 E4 A9 00 8D C9 BF
SET 750 8D DB BF A0 00 00 0B A2
SET 758 02 00 07 00 02 02 20 42
SET 760 41 55 44 0D 00 60
SAVE BAUDCHECK 700 765
```

To run the program, just type its name, BAUDCHECK (be sure to turn off any device that may be connected to the port). After 10 seconds, it will print the actual baud rate when 300 was selected. If the correct resonator is installed, it should print either 299 or 300. If you have the old 3.58MHz part, it will likely print 291 instead. Other rates can be verified by changing the byte at $70A (see page U1-2 notes 1 and 2 in the Utilities section of the MTU-130 manual for a table of values). Note that as the rate increases, the sensitivity of the test also increases and small errors will show up better. Even with the correct resonator, the rates will be about 0.17% slower. If you find that you indeed have the old resonator, a free upgrade kit is available on request. ♦

BUZZING MUSICAL BUG

Have you ever defined your own instrument sound using the SNOTRAN system and found that no matter what you do, it buzzes at you? Well if it looks like a bug, smells like a bug, and sounds like a bug, then perhaps it is a bug. That in fact is the case and the bug is in the SPLAY music interpreter that goes with the SNOTRAN compiler. Both programs came on your distribution disk. The bug is most likely to show up when very complex waveforms are specified and the component harmonic amplitudes add up to substantially more than 100 such as they did in the Shepard Tones experiment written up elsewhere in this issue.

To correct the bug in SPLAY, enter the following CODOS commands (disk with SPLAY assumed to be in drive 0):

```
GET SPLAY
SET 8FB E6 B6 D0 02 E6 B7 06 B6 26 B7
RESAVE SPLAY 700 BC3 18 97 1000 13FF
```

Do this on every disk you have that has a copy of SPLAY (except the distribution disk itself). Now, any bugs that remain should be of the crawling variety. ♦

ANNOTATE BUG FOUND

If you have received a copy of the User's Group diskette and run the ANNOTATE program, then you are probably aware of a rather strange bug. When the program is first loaded and run, it usually (but not always) will fail to work properly. However, if it is interrupted (with a CTRL/C) and run again, it runs perfectly every time!

Fortunately an alert (and experienced?) reader has found the problem and it is easy to fix. The difficulty stems from the fact that a critical data array is

(continued p.10)

(continued from p.9)

set up in line 2 (the GOSUB 9900) before the needed libraries are LIBed in! Thus the data array tends to get overwritten by the libraries when they are loaded the first time.

To fix the bug, merely swap the two statements that appear in line 2 of the ANNOTATE.B file and re-save the file. ANNOTATE will now run the first time, every time! ♦

ERROR IN OP68000.A FILE

Along with the DATAMOVER-256 board is supplied a diskette with a file called OP68000.A recorded on it among others. This file contains equates for the bit patterns of all of the 68000 machine language op codes which can be used to assemble 68000 programs with MTU's 6502 assembler. Unfortunately, one of the bit patterns is wrong and it even corresponds to a frequently used instruction.

To correct the error, insert a disk with the OP68000.A file and start the Editor with: EDIT OP68000.A. Next use the FIND command to locate "MOVEA.W". Following MOVEA.W you will find the bit string "0001000001000000". This should be changed to "0010000001000000" (i.e., two zeroes before the first 1 and 6 after it). Then QUIT the Editor. That's all! You should now be able to "assemble" a legal MOVEA.W instruction. ♦

LOOSE BITS

The following are a collection of miscellaneous items which should assist you in obtaining maximum performance from your MTU-130. They are short techniques or tips which have been found useful by others. Please let us know of your favorite "loose bits".

EDITOR on distribution diskette 1.4 and later allows use of designated column in the FIND command.

...

FIND 'WRITE' (C1) goes to first occurrence of WRITE starting exactly in column 1.

...

FIND 'WRITE' (C8-17) goes to first occurrence of WRITE starting in columns 8 through 17 inclusive.

...

FIND 'WRITE' (C35-) goes to first occurrence of WRITE from column 35 to the end of the line.

...

BASIC - each integer variable uses 5 bytes in memory. Elements in an integer array use only two bytes for each number. Integer variable and integer array elements only use two bytes each when stored on diskette.

...

BASIC - FOR ... NEXT will not accept an integer variable as the index. (i.e. FOR X%=1 to 10 is not valid.)

...

TERM - When using with a large computer check that computers requirements - it may not want a line feed and null after a carriage return.

...

ASSEMBLER Manual - On page 32 add:

0949 Set to $00 to eliminate forms feed in listing.

...

BREAKER - Before running the BREAKER program on User Diskette Number 1 execute the following:

SET 7F96 0 0

♦

Inputs & Outputs

"Y-DEVICE" REVISITED

by James K. Butler
Raleigh, NC

The following patch is my implementation of a "Y-device" driver, as discussed in MTU-130 Newsletter #1 May, 1982. It provides complete logging of CODOS> prompts, command input, and CODOS output to the printer.

```
SET D27D 20 E6 CB   Patch at STARTUP.J time
SET 030 7D D2       Enable print. (ONKEY 7...)
SET 030A E6 CB      Disable printer. (ONKEY 8...)
```

♦

CIRCULAR LOGIC

by Bruce D. Carbrey
MTU

Since the first User Group Disk was released, several customers have asked how I programmed various aspects of the Pool program. One particular question is "how do you draw all those circles so fast?" Everybody knows that to draw a circle you've got to compute the sine and cosine of the angle for each point to be plotted. Or do you? Below is a program which plots one eighth of a circle using only addition and subtraction. The remaining 7 "octants" of the circle can be plotted by symmetry with no further computation, but this is left as "an exercise for the student". The actual program was written in machine language instead of BASIC, and can compute each point on a circle in about 40 microseconds.

```
1000 REM SUB DRAWS 1ST OCTANT (0-5 DEG)
1005 REM OF CIRCLE WITH CENTER (X0,Y0), &
1010 REM RADIUS R.  REQUIRES IGL.
1020 REM
1030 X=R: Y=0: S=-R : SMOVE X0+X, Y0+Y
1040 SDRAW X0+X, Y0+Y
1050 S=S+Y+Y+1: Y=Y+1
1060 IF S >= 0 THEN S=S-(X+X)+2: X=X-1
1070 IF Y <= X GOTO 1040
1080 RETURN
```

♦

DYNAMIC LIBING IN BASIC

by Larry Isaacs
MTU

The FRELIB and LIB commands were implemented such that it is possible to dynamically change libraries during the execution of a BASIC program. To do this, only a couple of simple facts about these commands must be taken into account.

First, the LIB command will try to leave all current variables undisturbed. If the LIB command does not have to move BASIC's top of memory pointer down to make room for the library, all variables are left intact. If the top of memory does move down to make room, the LIB command clears all variables since some of the string data gets overwritten.

Second, if the FRELIB command is executed from a running program, it does not move the top of memory pointer back up to its original position. By not moving the top of memory, all variables may remain undisturbed.

The trick to changing libraries during the execution of a program is to load in the library, or combination of libraries, which take up the most space at the very beginning of the program. This will move the top of memory pointer to its lowest value. At this point you may change libraries by executing a FRELIB command followed by an appropriate LIB command. All variables will remain intact since the top of memory shouldn't need to be moved down during the LIB command.

The advantage of changing libraries on the fly is that it allows more memory for use by your program. If the libraries don't all have to be linked in at one time, then you only use enough space for the largest combination. The following program gives an example of switching between the CIL and the combination IGL and VGL.

```
 10 FRELIB: LIB "CIL":REM THE LARGEST
 20 REM GENERATE SOME GRAPHICS DATA
 30 ...
100 FRELIB: LIB "VGL","IGL"
110 REM DRAW THE IMAGE
120 ...
200 FRELIB: LIB "CIL"
210 REM WRITE THE GRAPHICS DATA TO DISK
220 ...
```

This program brings in the CIL Library first because it takes up more space than the combination of VGL and IGL. The program later switches to the VGL and IGL for the purpose of drawing a image specified by some graphics data. Once this image is drawn, the program switches back to the CIL for the purpose of writing the graphics data to disk. This would give you about 5900 extra bytes for the program or data. The only penalty is the extra time it takes to load the libraries. ♦

STORING AND RESTORING LEGENDS

by David Maynard
Winston-Salem, NC

Every 130 user has run across the problem of having application programs redefine the function key legends and substition strings so that when control returns to CODOS, the system legends that were set have been lost. While the best remedy is to have the application programs store and restore the legends, this feature is difficult to implement from BASIC. I suggest storing the legends and strings as a loadable CODOS file with the SAVE command. By typing "(RE)SAVE <filename> = 315 5C0 5FF 400 4FF" you create a file that can call up a menu simply by typing "filename". This command can be inserted into the STARTUP.J or other job files and will execute much faster than eight ONKEY commands. ◆

STARTING BASIC PROGRAMS FROM A JOB FILE

by Larry Isaacs
MTU

As most of you already know, MTU BASIC can be started from a job file. For example, a file containing the commands:

```
ASSIGN 3 P
BASIC
```

will assign channel 3 to the printer device then enter MTU BASIC. Because BASIC takes its input from a channel, BASIC will first try to read its commands from the job file. If any are found, they will be executed. This continues until the end of the job file is reached. When BASIC discovers that it has reached the end of the job file, it will automatically assigns the input channel to the console device. At this point, future commands will be read from the console. For example, if the job file contains:

```
BASIC
RUN "MYPROGRAM"
```

then after BASIC is started up, the RUN command will be read and executed. This will load and run the BASIC program MYPROGRAM.B on the default drive. It is important to note that the RUN command in the job file satisfies BASIC's request for the input of a command. This means that BASIC doesn't know if there is anything following the RUN command in the job file. This also means that when the BASIC program begins running, BASIC's input channel is still assigned to the job file.

If a BASIC program requires user input from the console, then it may be necessary to take this last fact into account if you want to run the program from a job file. A simple example will illustrate what happens in this situation. Let's assume the job file contains:

```
BASIC
RUN "GETFILENAME"
```

And that the GETFILENAME program contains:

```
10 INPUT "ENTER FILE NAME ";NM$
20 PRINT "FILE NAME = ";NM$
```

Once the GETFILENAME program begins running, it will execute the INPUT statement. BASIC will try to read the requested string from the input channel which is still assigned to the job file. BASIC will then discover that there is no more data in the job file to be read and automatically assign the input channel back to the console. Now comes the critical point. What should BASIC do next?

When implementing MTU BASIC we had a choice of two possibilities. We could make BASIC automatically try again to read some input, which would require the user to enter some data on the console. Or, we could let BASIC return a null string, and let the program control whether to retry the INPUT statement. Actually, both are implemented. If the BASIC's input channel is channel 1, the normal input channel, BASIC will try again to input a string, this time from the console. If BASIC's input channel had been changed with a POKE

(continued p.13)

(continued from p.12)

statement, BASIC would not try the input again, but simply return a null string. However, it would automatically set the input channel back to channel 1 after freeing the file which it had been accessing.

It is important to note that BASIC will try again to input from the console only if no characters were received from the job file. If any characters are received, then BASIC will return these characters as the input once channel 1 has been assigned back to the console. Also, the GET statement will cause reassignment of the input channel back to the console when end-of-file is encountered. However, the GET statement will always return a null character, and not retry the GET.

In the example above, the job file ends with the RUN command. This would force the requested file name to come from the console, which is the desired result. A simple mistake, however, could foul this up. If a null line (i.e. a line consisting of just a carriage return) followed the RUN command, then BASIC would receive the carriage return as an input character before detecting the end of the job file. BASIC would return this null line as the input and not try to read anything further. In this example, the program would abort at this point due to null input to an INPUT statement.

Fortunately, it is very simple to use the Editor to check for null lines at the end of the job file. First enter the EDIT program specifying the desired job file as the file to edit. Once you are in the Editor, move the cursor to the last line in the file. You can tell if you are on the last line if the cursor down key fails to move the cursor down or make the screen scroll. Now if the cursor is on a blank line below the last command in your job file, then you have some null lines on the end. Use the SHIFT/DELETE function to delete the blank line(s) until the cursor will not move below the last command. Press the QUIT function to exit the Editor and your job file will be all set. ◆

CODOS DISK ACCESS

by Eric J. Hedman
Dale E. Hedman

The CODOS disk operating system is very simple from the user's viewpoint but for the curious, for the unfortunate user that gets a CODOS error message "Unformatted disk or irrecoverable read/write error" or for one who inadvertently deletes a file, knowledge of the CODOS disk layout is helpful. Anyone writing assembly language or FORTH programs is vulnerable to software damage of a disk sector or track and partial recovery from this state requires use of the "DISKETTE" utility program and knowledge of the CODOS disk layout. While this article does not explain how to recover from such failures, the information here should be helpful for those interested in experimenting with such a recovery attempt.

CODOS stores data in BLOCKS and keeps a linked list of these blocks for disk 0 at location $E400. A similar table for drive 1 is at $E300. This table is referred to as the "Block Allocation Table" and is also saved on the disk as discussed below. You may wish to dump the memory block $E400-$E4FF on your system for reference. The byte at $E400 is not used and is always $00. The next 247 bytes show all allocated blocks. The block identifiers are:

$00=> free block
$FC=> last block in a series
$FE=> system overlay block
$FF=> an unusable block
<$other>=> link to the next block for the file

For instance the $02 at $E401 links block 1 to block 2. With blocks linked in this way the blocks of data do not have to be contiguous on the disk and as you save files and release blocks of data, CODOS will seek out $00 locations for disk file storage.

Again for the curious, the last 6 bytes $E4FA-$E4FF contain:
$E4FA-E4FB = VSN (disk volume number)
$E4FC = always 0

(continued p.14)

(continued from p.13)

$E4FD = number of files on disk
$E4FE = always 0
$E4FF = last block number allocated

DISK DIRECTORY

The block allocation table and the directory are stored on the first 18 ($12) sectors of track 12 ($C) of the disk. The first and last sectors of this storage are copies of the block allocation table and the remaining 16 sectors are available for directory entries. Each directory entry is made up of 16 ($10) bytes, the first being a $01, the next 14 the file name, and the last byte is a pointer to the block allocation table. CODOS accesses a file by reading this directory from the disk and finding the first block entry point with all remaining blocks of data found from the linked list on the block allocation table. When a directory entry is deleted the first character of the file name is set to $00 and the linked list of allocated blocks are all set to $00.

DISK TRACK-SECTOR-BLOCK RELATIONSHIPS

At the lowest level of organization the disk is divided into tracks and sectors. There are 77 tracks on an 8 inch disk, with each track subdivided into 26 sectors for a single-sided disk and 52 sectors for a double-sided disk. A sector of 256 bytes is the smallest block that CODOS recognizes internally and any read/write operation, therefore, accesses a sector of 256 bytes at a time. Because of the small size of the sectors, the disk is organized on a higher level, namely BLOCKS.

A BLOCK holds 2K bytes of information on a single-sided disk and 4K bytes on a double-sided disk. The block, which is the smallest unit of disk that can be allocated for a file by CODOS, is a much more convenient size element to work with than sectors. If smaller blocks were used, a single byte could not be used to link blocks in the Block Assignment Table, because there would be more than 256 blocks on the disk. Using larger blocks also means that files do not tend to become fragmented into many little pieces all over the disk, which would adversely effect peformance. A data file containing $500 bytes will be allocated 2K of storage on a single-sided disk. CODOS can access blocks numbered 1 through 247; i.e., 494K of storage on a single-sided disk or 988K of storage on a double-sided disk. When CODOS executes a disk operation, it first gets the block number of the file, converts this to a track and sector number and takes the data off the disk with the process repeated if the file is larger than 1 block. The sequence of blocks for each file is indicated in the block allocation table.

CODOS maintains a directory file on the disk which normally cannot be accessed by the user directly. This file contains all references necessary for CODOS to maintain the disk. As stated above, the directory occupies the first 18 ($12) sectors of track 12 ($C) but this directory is not included in the users block allocation. This directory is interleaved into the user blocks on the disk with block 39 ($27) ending at the last sector of track 11 ($B) and block 40 ($28) beginning on track 12 ($C) sector 18 ($12). As seen in the block allocation table this hole is not apparent and the block allocations seem to be continuous to the user. In reality however, there is a mean, nasty directory monster hiding in the depths of track 12. Track 12 is used instead of a more "sensible" location like track 0 in order to reduce the average seek time between the directory track and a random file on a disk which is filled an average amount.

CONVERSION BLOCK --> TRACK,SECTOR

There is a simple way to convert a block number to a track and sector number. To do this you must know the number of sectors per track and the number of sectors per disk.

Single-sided disk - 26 sectors/track
8 sectors/block
Double-sided disk - 52 sectors/track
16 sectors/block

Now if we define:
BS= sectors /block
TS= sectors /track,

(continued p.15)

(continued from p.14)

we can calculate the track and sector for any block as

Track= Int((block-1)*BS/TS)
Sector= ((block-1)*BS) MOD TS

This simple conversion would work all the time if the block allocation were continuous on the disk but we must add a correction for the directory on track 12. That is we must add 18 ($12) sectors to the track/sector location for any block greater than 39.

Example 1 (Single-sided disk)
Find the track and sector for block 11.

Track= Int(10*8/26)= 3
Sector= 10*8 MOD 26 = 2

Block 11 is before block 39 so the calculation is correct. Block 11 begins on track 3 sector 2.

Example 2 (Single-sided disk)
Find the track and sector for block 47.

Track= Int(46*8/26) = 14
Sector= (46*8) MOD 26 = 4

Block 47 follows block 39 so we add 18 to the sector count so block 47 begins at track 14 sector 22.

Example 3 (block 49 SS disk)

Track= Int (48*8/26) = 14
Sector= 48*8 MOD 26 = 20

But 49 > 39 so add 18 sectors to get block 49 begining at track 15 sector 12. (Note the carry when sector exceeds 26.)

If you use DISKETTE to actually look at the disk contents, remember that the default values for diskette are in HEX and the numbers calculated above are in DECIMAL.

CODOS FILES

Each file recognizable by CODOS has a 64 byte header on the disk. This header contains loading and directory information but is not loaded into user memory when the program is read by the user. The user file actually starts with the 65th byte on the disk file and this is where CODOS starts reading (or writing). This 65th byte is accessed at position $000000 of a file.

The first 16 bytes ($00 - $0F) of this "invisible" header contains the directory entry for the file exactly as it is found in the directory itself; i.e., the name of the file and the file starting block number. The 16th byte ($10) is a flag byte, $80 for normal read/write files and $A0 for locked files. The next three bytes ($11, $12, and $13) are the length of the file with the low order byte first. The length includes both the actual file data and the 64 byte ($40) header so, file length = total length - 64 ($40). The next two bytes ($14 and $15) are the relative pointer for the directory file which points to the first byte of that file's directory entry. The first file, for instance, would have $101 for a pointer, the second would be $111 and so on. (Note that the first sector of the directory is the "Block Allocation Table", $100 bytes, and each directory entry is $10 bytes.) The next 10 bytes ($16 - $1F) are the date on which the file was created. This date is not changed if the file is updated but is stored only when the file is first written onto the disk. The remaining bytes are reserved for future upgrades.

CODOS reserves the blocks on the first track of the disk because the system hardware loads this track when the system is booted up. In addition, a section of the disk (blocks 40 and 41 for single-sided disks and block 40 for double-sided disks) are reserved for system overlays. If the system overlay is destroyed many of the system commands will not operate. These blocks plus the directory are required for the system to operate properly. ✦

SHEPARD TONES

by Ralph O. Erickson
Media, Pennsylvania

In his book about Godel, Escher and Bach, p. 717, Douglas Hofstadter discusses the Shepard tones, and Bach's endlessly Rising Canon. The Shepard tones are strange tones which can be arranged into an ascending scale, such that each note sounds higher than the preceding one, but after an octave has been played one is back to the first

(continued p.16)

(continued from p.15)

tone. When I had read about them I was anxious to hear them. Hofstadter's suggestion to try them on the piano did not sound practical so I looked up Roger Shepard's article (1964, Journal of the Acoustical Society of America 36:2346-2353). Then I sat down at my MTU-130 and wrote the following program for the simplified NOTRAN compiler. After a few trials, I got it to play. The effect is there, but there are some flaws in my program. There is some noise which I suppose I can eliminate with a little effort. (Editor's note: the noise is due to a bug is SNOTRAN, see the required patch elsehwere in this issue!) I think the effect would be much better if the program were an infinite loop, but I have not seen how to program this; perhaps I haven't read closely enough. And I am not sure that I have handled the changing amplitudes right; I can hear a distinct new lower component come in, on E, I think.

```
% Generate the Shepard tones using the the MTU-130
% NOTRAN compiler.
% 14 May 1982 - Ralph O. Erickson
% 15 Aug 1982 - Slightly modified by H. Chamberlin
NVOICES 3
% Define a waveform for each note of the chromatic
% scale, each waveform to consist only of octaves.
WAVE 5 100 H1,0; H2,50; H4,87; H8,100; H16,87; H32,50
WAVE 6 100 H1,4; H2,54; H4,89; H8,100; H16,84; H32,46
WAVE 7 100 H1,9; H2,57; H4,91; H8,100; H16,82; H32,42
WAVE 8 100 H1,13; H2,61; H4,92; H8,99; H16,79; H32,38
WAVE 9 100 H1,17; H2,64; H4,94; H8,98; H16,77; H32,34
WAVE 10 100 H1,22; H2,68; H4,95; H8,98; H16,74; H32,30
WAVE 11 100 H1,26; H2,71; H4,97; H8,97; H16,71; H32,26
WAVE 12 100 H1,30; H2,74; H4,98; H8,95; H16,68; H32,22
WAVE 13 100 H1,34; H2,77; H4,98; H8,94; H16,64; H32,17
WAVE 14 100 H1,38; H2,79; H4,99; H8,92; H16,61; H32,13
WAVE 15 100 H1,42; H2,82; H4,100; H8,91; H16,57; H32,9
WAVE 16 100 H1,46; H2,84; H4,100; H8,89; H16,54; H32,4
%
TEMPO 1/2=750
% START 1 OCTAVE OF SCALE HERE
ASSIGN 5 5 5 0
PLAY 5
ASSIGN 6 6 6 0
PLAY 6
ASSIGN 7 7 7 0
PLAY 7
ASSIGN 8 8 8 0
PLAY 8
ASSIGN 9 9 9 0
PLAY 9
ASSIGN 10 10 10 0
PLAY 10
ASSIGN 11 11 11 0
PLAY 11
ASSIGN 12 12 12 0
PLAY 12
ASSIGN 13 13 13 0
PLAY 13
ASSIGN 14 14 14 0
PLAY 14
ASSIGN 15 15 15 0
PLAY 15
ASSIGN 16 16 16 0
PLAY 16
% END 1 OCTAVE OF SCALE HERE
% DUPLICATE THE SCALE AS MANY TIMES AS DESIRED
% (DUPLICATE AT LEAST 3 MORE TIMES)
ENDCMD
%
MAXVOICE 3
SEGMENT 5
    1G1,1/2; 2C2,1/2; 3E2,1/2
    R,1/4
ENDSEG
SEGMENT 6
    1G#1,1/2; 2C#2,1/2; 3F2,1/2
    R,1/4
ENDSEG
SEGMENT 7
    1A1,1/2; 2D2,1/2; 3F#2,1/2
    R,1/4
ENDSEG
SEGMENT 8
    1A#1,1/2; 2D#2,1/2; 3G2,1/2
    R,1/4
ENDSEG
SEGMENT 9
    1B1,1/2; 2E2,1/2; 3G#2,1/2
    R,1/4
ENDSEG
SEGMENT 10
    1C2,1/2; 2F2,1/2; 3A2,1/2
    R,1/4
ENDSEG
SEGMENT 11
    1C#2,1/2; 2F#2,1/2; 3A#2,1/2
    R,1/4
ENDSEG
SEGMENT 12
    1D2,1/2; 2G2,1/2; 3B2,1/2
    R,1/4
```

(continued p.17)

(continued from p.16)

```
ENDSEG
SEGMENT 13
    1D#2,1/2; 2G#2,1/2; 3C3,1/2
    R,1/4
ENDSEG
SEGMENT 14
    1E2,1/2; 2A2,1/2; 3C#3,1/2
    R,1/4
ENDSEG
SEGMENT 15
    1F2,1/2; 2A#2,1/2; 3D3,1/2
    R,1/4
ENDSEG
SEGMENT 16
    1F#2,1/2; 2B2,1/2; 3D#3,1/2
    R,1/4
ENDSEG
END
♦
```

HOW TO PEP UP THE EDITOR FOR LARGE FILES

by Larry Isaacs
MTU

If you haven't used the EDIT program to edit large files (i.e. over 100K), then you probably haven't experienced the slowdown that occurs when the file goes over approximately 120K bytes. The slowdown can be rather substantial, but is easy to remedy.

Actually this slowdown can occur in other situations as well. It appears first in the EDIT program because the Editor is overwriting an existing file. Since it can occur in other situations, it would be wise to explain the cause of the slowdown. To do this we must first explain what sector interleaving means.

All data transfers from disk to memory, or memory to disk, are buffered by the CODOS Operating System. The buffers that CODOS uses are 256 bytes long, which is exactly the same size as each sector on the disk. When, for example, a program asks CODOS to read a large block of data from disk into memory, CODOS must transfer the data in 256 byte chunks. CODOS reads a sector into a buffer, transfers the data from the buffer to the proper location in memory, reads the next sector into the buffer, transfers this data, and so forth. This continues until all of the requested data has been moved into memory or end-of-file is reached.

When trying to optimize performance, an important question comes up once the data has been transferred from the buffer into its proper destination. This question is: "Where is the next sector to be read in relation to the read/write head of the disk drive?" If we want the best performance, the next sector to be read should be the next sector to pass under the read/write head. What we especially don't want is for the next sector to be read to be the one that has already started passing under the read/write head. At this point it's too late to begin reading this sector and we have to wait for a whole revolution of the disk before we get another chance read it. If this were always the case, it would take 26 revolutions of the disk to read all the 26 sectors that make up one track. This would equate to an average transfer rate of only 1536 bytes per second. This is a rather poor rate, so we want to avoid missing sectors like that. You've probably guessed by now that this is the cause of the slowdown.

If CODOS wrote the data to disk in physically adjacent sectors, we would miss sectors all of the time. This is because the next physical sector starts passing under the read/write head before CODOS can transfer all of the data out of the buffer. So for better performance, we want to interleave the desired sectors so their sequence to be read is different from their physical sequence. By skipping one or more sectors between each sector written during a write operation, we give CODOS more time between sectors during a read operation. If we skip enough sectors, then CODOS will have the necessary time to empty the buffer and not miss the next sector to be read.

In the typical case, CODOS needs to skip only one sector between each sector it writes. This is expressed as an interleaving ratio of 2, and is the default ratio used in the FORMAT and BACKUP program. With an interleaving ratio of 2,

(continued p.18)

(continued from p.17)

it takes 2 revolutions to read a track, which results in an average transfer rate of 19968 bytes per second. This is why we like to use a ratio of 2.

However, with an interleaving ratio of 2, CODOS has only a very small amount of extra time before the next sector to be read starts under the read/write head. If something should cause this extra time to be used up, CODOS would start missing sectors. As it turns out, the time necessary to calculate the next sector increases as the sectors get farther into a file. Also, if CODOS is overwriting a file, a little extra time is used up. Thus, if a file gets long enough, CODOS will start missing sectors towards the end of the file. As stated above, the EDIT program encounters the slowdown once the file size reaches approximately 120K.

Fortunately there is a simple way to increase the interleaving ratio. The FORMAT program accepts an argument which specifies the ratio. To pep up the EDIT program, use the command:

FORMAT S=3

to specify an interleave ratio of 3. This will format a new disk using a ratio of 3. You can then copy the system files or any other files over to this disk. This is the best way to convert a disk with an interleave ratio of 2 to one with a ratio of 3. (You can't reFORMAT an exiting disk without destroying the old data.) The BACKUP program will also accept an argument just like FORMAT to specify the interleave ratio. However, it factors the new ratio into it method for reading the original. If you are backing up a disk with a ratio of 2 to one with a ratio of 3, the backup will be very slow. It would be faster to FORMAT the new disk and use COPYF to copy all the files. It would make sense to use BACKUP when the original has an interleave ratio of 3. However, you must again specify the "S=3" argument if you want the backup to also have interleave ratio of 3.

On a disk with an interleave ratio of 3, you can edit much larger files without encountering the slowdown. We have EDITed files as large as 600K and not encountered a slowdown. This information also applies to files created by WORDPIC(tm), our new word processor. Use an interleaving ratio of 3 if files are expected to exceed 120K.

Increasing the interleave ratio does affect the average transfer rate. For a ratio of 3, the average transfer rate becomes 13312 bytes/second. However, since most programs load in only a couple of seconds, the slower transfer rate would add less than a second to the load time. This won't be that noticeable. The main thing that will be noticeable is that the slowdown no longer occurs when editing large files.

The COPYF utility can also experience the slowdown. On a single-sided disk, the slowdown starts to occur when the file goes over approximately 300K. It is twice this for double-sided disks. If this slowdown is a problem, it is better to switch from a single-sided disk to a double-sided disk before resorting to increasing the interleaving ratio.

As one final note, the interleaving ratio affects only the format of the disk. It in no way involves any changes in the CODOS operating system. CODOS isn't even aware of the interleaving ratio on the disks. It is therefore perfectly okay to "mix" disks with different interleave ratios. The only thing the ratio affects is the time it takes to load and access files. ♦

THE '130'S HIDDEN PARALLEL PORT

by Hal Chamberlin
MTU

Surprise! The MTU-130 has an additional parallel port hidden inside! If you're a hardware hacker type and have studied the circuit diagrams in the MTU-130 manual, then this is probably not news to you. But judging from how often we are asked about the availability of a few extra I/O lines, this fact may not be common knowledge.

(continued p.19)

(continued from p.18)

If you have ever lifted the cover of your '130, you may have noticed two connectors on the CPU board that do not connect to a cable. The larger connector has 14 pins (less 1 key) and was intended for connection to a miniature tape cartridge drive in a low-cost entry-level system. The smaller connector has 10 pins (again less 1 key) and was intended for connection to a local network interface board (a-la Ethernet). Since all MTU-130s so far have gone out with disk, it is safe to assume that the "MTUtape" connector is free for use. And since the network transceiver is not yet available, the "MTUnet" connector can be used for the time being as well.

The MTUtape connector is the most useful of the two. Its 13 pins give you the low 6 bits of a 6522 parallel port, the CB1 and CB2 handshaking signals of the same 6522, and DC power. The pinout of this connector is shown below (it can also be found on page 23 of the General Hardware manual):

PIN	SIGNAL
1	+5 Volts regulated (100MA limit)
2	+12 Volts regulated (100MA limit)
3	-12 Volts regulated (100MA limit)
4	Ground
5	-polarize- (that means missing!)
6	No connection
7	SYS2 CB2
8	SYS2 PB5
9	SYS2 PB4
10	SYS2 PB3
11	SYS2 PB2
12	SYS2 PB1
13	SYS2 PB0
14	SYS2 CB1

All of the parallel port bits connected to the MTUtape connector are part of the SYS2 6522 I/O chip. This chip uses addresses $BFF0 - $BFFF. The CODOS operating system does nothing with the B side of this chip (which all of these lines are connected to) so you are perfectly free to manipulate the port B data and direction registers at will and they will stay put.

The 6 bits of data and two handshaking lines provided are sufficient for many applications. If a seventh data line is required (such as for representing ASCII data), it may be found on one pin of the MTUnet connector described below. The eighth bit is not readily accessible since it is connected to the cassette port read amplifier. The CB1 and CB2 handshaking lines give access to many of the 6522's advanced functions and can be used as interrupt inputs, pulse output, programmable oscillator (through timer 2), and parallel/serial converter with the shift register feature. Refer to pages 23-36 in the Monomeg manual for information on programming the 6522 (remember to use the SYS2 addresses instead of the rear panel parallel port addresses given).

The smaller MTUnet connector can also be useful. Its 10 pins provide a single data line, 3 handshaking lines, and also DC power. The pinout of this connector is shown below (it can also be found on page 23 of the General Hardware manual):

PIN	SIGNAL
1	+5 Volts regulated (100MA limit)
2	-12 Volts regulated (100MA limit)
3	+12 Volts regulated (100MA limit)
4	Ground
5	-polarize-
6	No connection
7	SYS2 PB6
8	SYS2 CA1
9	SYS1 CB2
10	SYS1 CB1

The data line (pin 7) makes up the seventh bit of SYS2 port B. You also have another pair of CB1 - CB2 handshaking lines which are connected to the SYS1 6522 addressed at $BFE0-$BFEF. The third handshake line is CA1 of SYS1 which is useful only as an interrupt input. Be careful when manipulating CA1, CB1, and CB2 of SYS1 with your program because the other sections of SYS1 control a number of critical system functions such as extended memory addressing.

(continued p.20)

(continued from p.19)

If you use the DC power voltages on either of these connectors, be sure not to draw too much current. These connectors are supplied by the same regulators that power the CPU board. If you draw enough current to put the regulators into current limit (even for as little as 10 or 20 microseconds), the 6502 CPU (and your program) will very likely crash!

Connecting to the MTUtape and MTUnet connectors is relatively simple. You can either obtain the mating connectors (AMP part # 1-640440-4 for the 14 pin or 1-640440-0 for the 10 pin connector or nearly any kind of single row, .10" spacing, .025" square pin female connector), or wire directly to the pins with a standard wire-wrap tool. There is plenty of room inside the MTU-130 case for a little interface board if needed. Connection to the outside world can be made through a connector of your choice mounted to one of the removable plates on the back panel.

For I/O expansion greater than what the "hidden ports" provide, there is always the soon to be available "MultI/O" board that was briefly described in newsletter issue #1. Or, if you don't mind a little electronic construction work, you can obtain an MTU K-1020 prototyping board and build the example circuit (a full 6522 dual parallel port) given on page 45 of the Monomeg manual. Happy porting! ♦

In the Queue

68000 CROSS ASSEMBLER

Work is proceeding on a cross assembler for the 68000 microprocessor on the Datamover board. DMXASM (for DataMover cross ASseMbler) will actually be an adaptation of MTU's highly acclaimed 6502 assembler which means that it will be FAST! Since it will run entirely on the 6502, the Datamover will not be necessary to edit and assemble programs.

Plans call for a "95% compatible subset" of the extremely rich assembly language Motorola has defined for their baby. DMXASM will recognize all of the 68000's mnemonics and will perform extensive automatic selection of the most favorable instruction type and addressing mode to fit the situation. Expressions will be evaluated to 32 bit precision and programs may be located anywhere in the 68000's 24 bit address space. Macros will also be included. Extensions to Motorola standards include labels up to 31 characters and the ability to create and read external symbol files. The object file format will be directly compatible with DMXMON (see the DMXMON description elsewhere in this issue) thus allowing one-step assemble-and-go operation. Compromises include absolute assembly and omission of the "structured assembly" feature of the Motorola standard.

In essence, DMXASM will be an excellent high-speed tool for preparing 68000 programs with a minimum of fuss and cost. For more details, please contact MTU. ♦

WE'LL "C" YOU IN SEPTEMBER!

If you follow the computing industry journals and magazines, you know that the "in" language right now is C. Although the C language has been around for a number of years, it has only recently fallen into fashion outside the hallowed halls of Bell Telephone Laboratories, where it has been a mainstay since the mid seventies. There is good reason for this new interest. Rapid technology advances in LSI design and the accompanying foreshortening of technological lifespans for computer architectures has made development of portable software a necessity. The C language is relatively easy to bring up on new computers and generates fairly efficient code, since it is not as far removed from the hardware as many other high level languages. It provides powerful data and control structures, suitable for implementing both system and applications software.

We are extremely pleased to announce that MTU will be shipping C language development software for the MTU-130 beginning in late September. This package was developed

(continued p.21)

(continued from p.20)

under contract by Manx Software, a company well versed in the ins and outs of C compiler design. The C software package will run on any MTU-130 computer and does not require any additional memory or other hardware or software. Included is a compiler, a P-code assembler, a 6502 relocating assembler, a linker, and a library. The C compiler accepts virtually the full C language, not just a "tiny" or "small" subset. This includes support for character, integer, long, pointer, float and double (double precision floating point) data types. The full gamut of powerful control structures are provided including IF-ELSE, WHILE, BREAK, CONTINUE, DO-WHILE, and SWITCH. About the only thing not supported is the seldom used "bit-field" data type. Functions can be separately compiled and linked to library functions and to 6502 assembly language routines. Once linked, C programs are executable modules which can be run from CODOS just like any other machine language program. C programs use standard CODOS terminal and file I-O. Program source code can be prepared using the familiar MTU Editor.

Compiled C programs run much faster than BASIC programs. How much faster depends on the operations performed. Typically, speed improvements of 300 to 1500 percent can be expected! The larger and more complex the program, the better the speed improvement. C programs also require less memory than their BASIC counterparts. Unlike BASIC, which always requires about 12K of memory for the interpreter to be present in memory, C programs only need whatever functions they actually use. Most programs will use a "core" of standard functions from the library (such as input-output) which occupy about 6K bytes. MTU C compiles to a very compact pseudo-object code usually occupies even less memory than assembly language for the same function, so you can run very large and complex programs on the MTU-130's.

If you are contemplating selling or distributing software written in C, you will be pleased to know that unlike many other licenses, there is NO ROYALTY on compiled object code from MTU-130 C. This means that you can sell or give object programs written in C to other MTU-130 owners, and the programs will run even if they do not have the C package. We are looking forward to receiving a lot of programs for the User Group Disk written in C in the near future.

How hard is C to learn? Well, its not as easy to learn as BASIC, but its a lot easier than FORTH or assembly language. If you've ever used Pascal you will probably have no trouble adapting to C (and will be pleased to learn that C can do all the annoying little things Pascal can't!). If you'd like to learn a little more about C, the August 1981 BYTE and May 1980 Dr. Dobbs' Journal contain good introductions. The C "bible" is Kernighan and Ritchie's book, The C Programming Language, published by Prentice Hall and available at many college bookstores or from MTU. You should definitely obtain this book if you are going to use MTU-130 C.

We are now accepting orders for MTU-130 6502 C. The price is $199. We plan to offer subsequent upgrades and libraries for a modest upgrade charge. Notify the factory to have your name placed on the mailing list for additional information. ♦

MTU UPDATE

Volume 1 Issue 2 3rd Quarter'82

MTU UPDATE is a quarterly publication for MTU-130 users. Please address all correspondence to:

Newsletter Editor
Micro Technology Unlimited
P.O. Box 12106
2806 Hillsborough Street
Raleigh, N.C. 27605 U.S.A.

PRICE LIST

Micro Technology Unlimited

Effective September 15, 1982

FULL SYSTEMS

200010	MTU-130-1D	FULL 1 DRIVE SYSTEM LESS MANUAL #270000	$4400 ¥
200020	MTU-130-2D	FULL 2 DRIVE SYSTEM LESS MANUAL #270000	$4945 ¥
200040	MTU-130-4D	FULL 4 DRIVE SYSTEM LESS MANUAL #270000	$7245 ¥

PARTIAL SYSTEMS

FOR APPLICATIONS REQUIRING SEPARATE BOARDS, OR
PARTIAL OR CUSTOM SYSTEMS, PLEASE CALL MTU DIRECTLY.

BUSINESS APPLICATIONS

210000	WORDPIC	WORD PROCESSOR LESS MANUAL #280010	$385

EDUCATIONAL APPLICATIONS

214001	SIMPLEMUS	6502 4 VOICE SIMPLE WAVEFORM MUSIC	STANDARD
214002	INSMUS-8	6502 INSTRUMENT SYNTHESIS MUSIC	$80
214003	INSNOTRAN	6502 COMPILER FOR INSMUS-8	$50
214100	SLIDESHOW	6502 BASED SLIDE CREATION/PRESENTATION	CALL

GAME APPLICATIONS

216001	POOL	MACHINE LANGUAGE POOL GAME	STANDARD

OPERATING SYSTEMS AND UTILITIES

220000	CODOS 2.0	6502 CHANNEL-ORIENTED OPERATING SYSTEM	STANDARD
220001	MTU-EDITOR	FULL SCREEN, INFINITE SCROLL EDITOR	STANDARD
220002	AUTOTERM	INTERACTIVE & UNATTENDED COMMUNICATIONS	STANDARD
220003	DISKEX	DISK EXCHANGE UTILITY LESS MANUAL #280020	$140
220004	USERSHELL	SOURCE CODE	$25
221000	DMXMON	68000 CHANNEL-ORIENTED OPERATING SYSTEM	STD W/DATAMOVER
222000	CP/M 2.2	Z-80 OPERATING SYSTEM	$150

LANGUAGES

230001	MACASM	6502 MACRO ASSEMBLER LESS MANUAL #280030	$80
230002	DMXASM	68000 MACRO CROSS ASSEMBLER LESS MANUAL #280031	$190
230101	MTU-BASIC	INCLUDES VGL, IGL, CIL LIBRARIES	STANDARD
230102	MTU-BASIC 1.5	REVISION 1.5 UPGRADE	$50
230103	KGL	ADDITIONAL GRAPHIC LIBRARY	$50
230201	P-SYSTEM	SOFTECH P-SYSTEM	$475
230301	P-PASCAL	SOFTECH P-SYSTEM PASCAL	$375
230401	P-FORTRAN	SOFTECH P-SYSTEM FORTRAN	$375
230501	MTU-C	6502 FULL C LANGUAGE LESS MANUAL #280100	$190
230601	MTU-FORTH	6502 FULL FORTH LANGUAGE	$150

¥ MEANS CUSTOMER PAYS SHIPPING. MTU PAYS SHIPPING IN USA ON OTHER ITEMS.

DISTRIBUTION AND USER DISK SOFTWARE

240001	USER DISK 1		$15
240002	USER DISK 2		$15
240100	DIST 1.4	CODOS DISTRIBUTION DISKETTE 1.4 UPGRADE	$15
240200	10 DISKETTES	BLANK DYSAN DOUBLE-SIDED, DOUBLE-DENSITY	$75

MTU DESIGNED HARDWARE PRODUCTS

250000	DATAMOVER	68000 BOARD LESS MANUAL #270010	$1490
250005	PROGRAMMOVER	4MHZ Z-80A BOARD LESS MANUAL #270012	$440
250100	MULTI-O	EXPANDED LAB I/O LESS MANUAL #27011	$890
250110	PROTOTYPE	WIRE WRAP PROTOTYPE BOARD	$50
250200	MULTIDISK	FLOPPY AND RIGID DISK BOARD LESS MANUAL	CALL
250300	DIGISOUND-8	AUDIO SOUND SYSTEM	CALL
250401	FDISK MODULE	DUAL 8" FLOPPY EXPANSION MODULE	$2300 ¥
250500	PARACABLE2	PARALLEL DEVICE CABLE WITH 2 CONNECTORS	$65
250501	PARACABLE3	PARALLEL DEVICE CABLE WITH 3 CONNECTORS	$85
250502	SERIALCABLE2	SERIAL DEVICE CABLE WITH 2 CONNECTORS	$40
250600	GROUP NUMBER	CUSTOM REQUESTED GROUP NUMBER IN ROM	$50

PRODUCT MANUAL AND BOOKS

HARDWARE

270000	MTU-130 M	MTU-130 USER/SERVICE MANUAL	$50
270010	DATAMOVER M	DATAMOVER HARDWARE MANUAL	$10
270011	MULTI-O M	MULTI-O MANUAL	$10
270012	PROGRAMMOVER M	PROGRAMMOVER HARDWARE MANUAL	$10

SOFTWARE

280010	WORDPIC M	WORDPIC WORD PROCESSOR MANUAL	$10
280020	DISKEX M	DISKEX MANUAL	$10
280030	MACASM M	6502 MACRO ASSEMBLER MANUAL	$10
280031	DMXASM M	68000 MACRO CROSS ASSEMBLER MANUAL	$10
280100	DMXMON M	DMXMON AND 68000 REFERRENCE MANUAL	$25
280200	MTU-C M	MTU-C USER MANUAL	$10
280201	C-BOOK	THE C PROGRAMMING LANGUAGE BY KERNIGHAN & RITCHIE	$18

EXPANSION PERIPHERALS FROM OTHER MANUFACTURERS

300001	SMARTMODEM	DC HAYES WITH SERIALCABLE2(250502)	$345 ¥
300002	SMARTMODEM 1200	DC HAYES WITH SERIALCABLE2(250502)	$695 ¥
300100	NEC 8023A	PRINTER WITH PARACABLE2 (250500)	$650 ¥
300200	QUME DT8	QUME DATATRACK 8 FLOPPY DRIVE	$895 ¥

¥ MEANS CUSTOMER PAYS SHIPPING COST. MTU PAYS SHIPPING IN USA ON OTHER ITEMS.

OVERVIEW

DON'T MISS the details of the User Referral Plan being announced in this issue by MTU's President, Dave Cox. MTU UPDATE is also introducing several new products: the Z-80 slave processor board for the CP/M on the MTU-130, the cross monitor program DMXMON, a wordprocessor plus called WORDPIC, and the MTU-130 macro assembler, MACASM.

From all reports, our first User Group Disk was both useful and fun, so be sure to look over the contents of our second User Group Disk. In addition, UPDATE's Inputs & Outputs section features articles from MTU-130 owners James Butler, David Maynard, Eric and Dale Hedman, and Ralph Erickson. (Many thanks to all those who contributed!)

In this issue we give you fixes to bugs and ways to increase your MTU-130 capabilities without spending a cent. If there is something you want to hear more about or something we haven't covered, call (919) 833-1458 or write us!

Growing MTU-130 Acceptance!

At deadline (September 24) sales for the month already had broken all records.

MTU President David Cox told UPDATE that much credit for "this new spurt in acceptance" should go to a growing pattern of MTU-130 users informally exposing their elite-of-systems to associates and other professionals.

Micro Technology Unlimited
P.O. Box 12106
2806 Hillsborough Street
Raleigh, N.C. 27605 U.S.A.
(919) 833-1458

RALEIGH OCT-7'81 N.C. U.S. POSTAGE ≡ 0.54 RB.METER 1537326

First Class Mail