MTU UPDATE

1st QUARTER '83 Vol. 1 Issue 3

MULTI-O BOARD EXPANDS MTU—130 APPLICATIONS

This multi-function input/output board will add a variety of digital and analog I/O ports to the MTU-130 computer. It plugs directly into the MTU-130 bus and gets all necessary operating power from the bus. All additional I/O connectors are contained in a small box which easily mounts to the '130's rear panel. All cables between the board and these connectors are hidden under the box and enter the '130 case through one of the existing rectangular cutouts. Below is a summary of the many hardware functions present on the MULTI-O board.

PARALLEL I/O PORTS: A full 6522 chip including both timers and shift register is made available. This gives two independent 8 bit parallel ports plus two handshaking lines on each. Besides the digital signal lines, +5, +12, and -12 volt regulated power is provided for external interfaces. The ports are terminated in a 36 pin D-type ribbon connector that is fully compatible with the MTU-130's standard parallel connector.

SERIAL I/O PORTS: Two independent serial I/O ports are provided. One of the ports has a full set of modem control signals (CTS, DSR, DCD, DTR, and RTS) while the other has just transmit and receive data lines. These ports use type 2651 serial interface chips which allow synchronous protocols as well as the usual asynchronous format. The baud rate is under software control (or external for synchronous) and can range from 50 to 19,200 baud. Although the 2651 is programmed somewhat differently than the 6551 used for the '130's standard serial port, it can do anything the 6551 can and then some. Each port has its own 25 pin D-type connector.

12 BIT A-TO-D CONVERTER: This is a high accuracy, high speed, multi-channel analog input interface that bridges the gap between the MTU-130 and sophisticated laboratory data acquisition applications.

(Continued on p.2)

ON THE INSIDE

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)

It is equipped with 8 input channels which may also be configured as 4 differential inputs. Each input has a bias current less than 100PA and is overload protected up to ±200 volts. Channel selection is under software control. The internal 6522 interface IC has two timers that may be programmed for auto-sampling on an interrupt basis. Total conversion time including channel selection, track-and-hold, and the actual conversion is less than 70uS. Low drift high accuracy components are used throughout including a temperature regulated oven reference zener.

12 BIT D-TO-A CONVERTER: To complement the A-to-D converter is a single channel 12 bit D-to-A converter. It has a low impedance short-circuit protected output that settles to .01% in less than 25uS. Both the A-to-D and D-to-A terminate in a 15 pin D-type connector. There are also 2 interrupt inputs and 2 digital control I/O lines that terminate in a 9-pin D-type connector. These are separate from the parallel I/O port.

IEEE-488 BUS INTERFACE: This is a full GPIB implementation including talker, listener, and controller functions. It uses the Texas Instruments TMS9914 interface chip which handles most of the IEEE protocol in hardware. Data rates up to 80K bytes/second can be handled in a burst or up to 20K bytes/second continuous to disk. This port terminates in an IEEE standard 24 pin D-type ribbon connector.

CLOCK AND CALENDAR: This battery-powered software readable clock and calendar is positively protected against power glitches and program crashes. The time and date may be read at any time and a special utility program is provided for setting the time and date. Timing accuracy is within 30 seconds/month and interrupts every second or every hour may be programmed. Two standard low cost penlite cells mounted in the connector box for easy access will power the clock for over a year.

The MULTI-O board also comes with a full complement of demonstration, utility, interface, and diagnostic software. Below is a brief description of this software:

DEMONSTRATIONS: A digital voltmeter and storage oscilloscope demonstration of the analog inputs. A programmable voltage source and waveform generator demonstration of the analog output.

UTILITIES: TIMESET for setting the time and date. AUTODATE for automatically reading the date when used in the STARTUP.J file. WAITUNTIL for waiting until a designated time before returning. Versions of SYSGENPRINTR and AUTOTERM for using the parallel and serial ports on the MULTI-O board.

INTERFACE SUBROUTINES: A driver that allows an IEEE device to be assigned to a CODOS channel. A BASIC library and assembly subroutines for general IEEE bus operations. A BASIC library and assembly subroutines for handling analog I/O, parallel I/O, and the clock/calendar.

DIAGNOSTIC PROGRAMS: Looparound tests for the parallel and serial ports. Looparound tests for the analog input and output ports. Clock/calendar function test. IEEE-488 interface test.

The MULTI-O board is presently in PC layout and will be available in mid February. The price for the board is $980 and the manual is $10. ◆

Volume 1 Issue 3 1st Quarter 1983

MTU UPDATE is a quarterly publication for MTU-130 users. Please call (919) 833-1458 or write:

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

Copyright, 1983, Micro Technology Unlimited

NOTES FROM THE FACTORY

DISTRIBUTION DISK

Distribution Disk 1.5 is now available. Compared with 1.4 and earlier versions, several bugs in various programs have been fixed and some programs have been improved. The most significant improvements are the replacement of the TERM utility with the much more powerful AUTOTERM and substantial improvements to BROWSE. Computers shipped after November 18, 1982 include this disk. Others may order Distribution Disk 1.5 for $15.00. The following is a description of all of the changes made with Distribution Disk 1.5.

AUTOTERM - Replaces the original TERM utility. Now has automatic call and answer and verified transparent file transfer capability.

SVCPROC.Z -Fixed the SVC-13 bug where a GET into a bank other than 0 will crash the system. This fix is of particular benefit to DMXMON users.

BROWSE - Several minor bugs have been fixed. It now displays in a 20 line window similar to EDIT. Includes a WRITE command that can be used to print selected portions of a file or create a new file from one or more pieces of an existing file.

DIR and FILES - Will now function correctly when channel 2 is assigned to a file on drive 0 and files on drive 1 are being listed.

COPYF and COPYF1DRIVE - Will now copy the maximum line length attribute of WORDPIC files.

DISKETTE - Will now print the correct sector number if an error occurs on side 1 of a double-sided diskette.

BACKUP - Will now automatically sense the interleave factor of the source disk and format the copy disk the same way unless overridden with an argument. Will also print the correct sector number when a read error occurs on side 1 of a double-sided diskette.

VMDUMPNEC - The various spacing options between screen dumps will now work. Also, the alignment of plotted vertical lines has been improved.

FILECRC - This program has been added to simplify the task of positively confirming that two files are identical. See a description elsewhere in this issue.

CODOS.Z - The standard disk parameters are appropriate for the Qume disk drives currently being shipped (3MS track-to-track and 40MS head load). Also, the "P" device has been predefined.

STARTUP.J - Now loads and initializes PRINTDRIVER.Z.

GRAPHDRIVER.Z - The missing bytes at the end of the light pen driver have been added (see MTU UPDATE #1).

FORMAT - Now copies GRAPHDRIVER.Z and PRINTDRIVER.Z automatically as part of the minimum system files.

EDIT - Lines are now correctly displayed in landscape mode (width greater than 80 characters). Also the problem with a CHANGE that inserts at column 80 has been fixed.

TERM.A - This file has been deleted. The source for AUTOTERM is too large to fit on the distribution disk but is available separately on request. ◆

AUTOTERM—LET'S USE IT!

Perhaps many of you have used the standard TERM terminal emulator utility to talk to timesharing computers, bulletin board systems, or other MTU-130 users and wished that it could do more. Well, TERM has been greatly expanded. Called AUTOTERM now,

(Continued on p.4)

(Continued from p.3)

it is much easier to change the communication parameters such as baud rate and break duration. Instead of patching the program, pressing a function key will present a series of questions about the desired parameter settings. If you wish, the questions can be answered from a file rather than the keyboard. Echoing of transmitted characters is also controlled by a toggling function key and you can change all of these parameters while on-line without a glitch.

A major new feature is a "transparent binary file" send and receive mode. With this capability you can send ANY kind of file to another '130 user over phone lines with positive verification of transmission accuracy and automatic retry in case of difficulty. In operation the designated file is broken up into blocks of 256 bytes and each block is sent and verified with a CRC check. If the receiver detects an error, it asks for a retransmission. At the end, the CRC of the entire file is also checked. All of this happens automatically; the only requirement is that both the sender and the receiver be in the binary mode.

The most significant feature however is the ability to operate unattended. You can instruct AUTOTERM to place a call to a specified telephone number at a specified time and send a file, either in text or transparent binary mode. You can even instruct it to retry a designated number of times at designated intervals until the file is successfully transmitted. You can also have AUTOTERM automatically answer incoming calls and append what is received to a designated text file or receive any number of files in binary mode.

Other improvements include easier to understand control of the CON, RDR, and PCH files, more informative messages, a special beep when the Break key is pressed, and an "escape" key to send the escape sequence to a Smartmodem. The only special hardware required to use the automatic calling and receiving feature is a D. C. Hayes Smartmodem which is available from MTU. All other features are operable with any kind of asynchronous modem or direct RS-232 connection. AUTOTERM is available now on version 1.5 of the MTU-130 Distribution Disk. (See the article on Distribution Disk 1.5 in this issue for a complete list of the improvements in this new version.) ◆

FANCY SCREEN CLEAR

Have you ever seen an Apple computer clear its screen while in graphics mode? If so, you may have noticed that it just doesn't wipe from top-to-bottom like the MTU-130 screen does. Instead, it resembles a set of venetian blinds closing. The reason is that display memory in the Apple is not addressed linearly but instead is addressed in a somewhat scrambled fashion. Thus when a simple clear routine scans linearly through display memory, the screen pattern produced is scrambled. It has been said that the Apple designers did that scrambling (which has been the bane of many an Apple graphics programmer) merely for that "venetian blind" effect! Although thats a little hard to accept, the pattern really is more interesting than a simple wipe.

User Group Disk #3 has the source and object code for a simple assembly language routine that will clear the MTU-130 screen in an endless variety of ways. You can clear the screen in diagonal stripes, horizontal bars, seemingly at random, or even backwards. The sequence in which the screen is cleared is governed by a 16 bit argument you pass in the A and X registers. When calling from BASIC, you poke the argument into memory and then do an MCALL to clear the screen. The argument may be any odd number giving over 32,000 ways to clear the screen! To put real spice in your game programs, you could even use random arguments. The routine is only 65 bytes long, is "position independent" (will run anywhere you load it in memory), and takes less than a second to clear the screen regardless of the argument. ◆

CRT MONITOR INTERFERENCE

The MTU-130's CRT monitor can seriously interfere with proper operation of the disk unit under some conditions. The culprit is the high voltage transformer (and to a lesser extent the width coil, horizontal linearity coil, and deflection yoke) in the monitor which generates a strong high frequency magnetic and electrostatic field that extends out several inches from the transformer itself. If the disk unit is positioned so that these fields reach the read/write head, sufficient noise can be induced into the windings to interfere with reliable operation. The fields are not strong enough to affect the diskettes themselves however so you may continue to lay them on top of or to the side of the monitor.

The Zenith monitor used in the '130 has this transformer mounted in the left rear corner of the case near the base. If you have arranged your system so that the monitor sits somewhat left of center (to make the displayed legends line up with the function keys) and the disk unit sits to the left close to the '130 case, then the interference is probably occurring. The effect is usually an unexplained delay in loading a program or an excessive incidence of the "UNFORMATTED DISK OR UNRECOVERABLE READ/WRITE ERROR" message when using disk drive 1, particularly when the inside tracks are being accessed. If you are experiencing these symptoms, try moving the disk unit or monitor around and see if they change or disappear.

The best solution is to place the disk unit on the right side of the keyboard unit which completely solves the problem (be sure to leave an inch or two of clearance for the fan intake). If the disk must be to the left, try to keep it at least 6 inches from the keyboard and move the monitor as far right as possible. If these measures are unacceptable, you can shield the heads by inserting a 14.5" wide by 7.5" high piece of thin sheet steel (the material used to make heating ducts is ideal) between drive 1 and the right hand plastic panel in the disk case. If you are using your own monitor and disk reliability has been a problem, you may want to determine the location of the flyback transformer and try rearranging your system if it is on the same side of the monitor as the disk unit. ◆

MACRO ASSEMBLER PATCHES

If you have MACASM 1.1 and don't wish to upgrade to MACASM 1.2 (see page 19) at this time, here is a way to patch or avoid all known bugs in the macro assembler (which are minor). These changes apply only if your assembler "signs on" with the message:

MTU 6502 MACRO ASSEMBLER 1.1 (21-OCT-82)

The patches below fix the following symptoms:

1. If MACASM discovers that the listing file already exists, it may overwrite it with garbage before exiting with the message "OUTPUT FILE EXISTS".

2. The command line option E=P will not divert the error summary to the printer.

3. For directives which do not permit a forward reference in the operand (such as *=, .IF, etc.), a reference to a symbol which is defined by an equate to a forward reference is not detected as an error, and an undetected phase error may occur.

To correct these bugs, carefully execute the following CODOS commands:

```
GET MACASM
SET AEB=D
SET 214B=E6
SET 1354=EA EA
RESAVE MACASM 700 3386 3F00 4309
```

A side effect of the above changes is that MACASM will no longer permit a forward reference in the operand of an equate, so you should strike out the paragraph in the middle of page 9 of the MACASM 1.1 manual which indicates that forward references are permitted for equates.

(Continued on p.6)

(Continued from p.5)

Finally, a bug may cause incorrect conditional assembly when using nested conditionals. Since it is not practical to patch this bug you should either get the upgraded MACASM 1.2 which corrects this problem or not use nested .IFs. ◆

DON'T CHOKE YOUR '130

Fully expanded, your MTU-130 packs an unprecedented amount of power in a keyboard style package. But with this power comes "waste heat" which must be removed to keep the system operating properly. This is accomplished by a small fan mounted on the right side panel. If the fan opening is blocked by the disk unit or a box of diskettes, normal air circulation is cut off and overheating can occur, particularly if the system includes a DATAMOVER 68000 or MULTI-O board.

Usually the first symptom of overheating is a jittery, unstable display. If you see this, check the right side panel for at least 1" clearance in all directions. Sometimes a sheet of note paper or the MTU-130 Quick Reference Card can be sucked up against the fan opening. Simply remove the obstruction and the display should clear up in a couple of minutes. If the problem continues, feel for airflow around the fan intake and around the exit holes near the light pen connector. If there is none, a paper clip or other obstruction may be keeping the fan from turning. ◆

LINGERING SVC PROC. Z BUG

The new SVCPROC.Z file on the new Distribution Disk 1.5 doesn't completely fix the bug it was intended to correct. The bug occurs when you use SVC 13 to execute a CODOS "GET" command which loads a file into a bank other that bank 0. The result is that the program crashes when the SVC 13 returns to the calling program.

To patch a pre-Distribution 1.5 version of the SVCPROC.Z use the following sequence of CODOS commands:

```
GET SVCPROC.Z
SET DF2E 4C 8F DF
SET DF8F 85 DB A9 00 8D CE E6 8D CF E6 60
RESAVE SVCPROC.Z DD20 DF99
```

To patch the version on the Distribution Disk 1.5, you may use the commands:

```
GET SVCPROC.Z
SET DF96 8D CF E6 60
RESAVE SVCPROC.Z DD20 DF99
```

The cause of the crash is simple but tough to figure out at first. Whenever CODOS loads a "SAVEd" file into memory, CODOS sets the User Program Bank and User Data Bank variables to the bank in which the first block of the file gets loaded. Since the GET command loads a "SAVEd" file, the User and Data Bank variables will be set as a result. In addition, these variables, and other user register variables are used by the SVC Processor to save the current status of the 6502. This is done so that the proper status may be restored after CODOS has been called to perform the desired function.

If that function was a GET command, then the User Bank variables will have been changed, and could be restored to undesired values. Such is the case when the GET command fetches a "SAVEd" graphics file to the display RAM. The SVC 13 returns with Program and Data bank set to 1 causing the program to crash. The patch above modifies the routine which handles SVC 13. This routine is made to unconditionally set the User Program and Data bank variables to bank 0 before these variables are used to restore the state of the 6502. The result is that SVC 13 will always return to the calling program with program and data bank set to bank 0. ◆

If you are using the SOURCE or COMPUSERVE networks and would like to contact other MTU-130 owners, please let us know your ID number. We will publish your name and ID number in a subsequent issue of UPDATE. If sufficient interest develops, we will start a telecommunications column in UPDATE. ◆

CRT MONITOR INTERFERENCE

The MTU-130's CRT monitor can seriously interfere with proper operation of the disk unit under some conditions. The culprit is the high voltage transformer (and to a lesser extent the width coil, horizontal linearity coil, and deflection yoke) in the monitor which generates a strong high frequency magnetic and electrostatic field that extends out several inches from the transformer itself. If the disk unit is positioned so that these fields reach the read/write head, sufficient noise can be induced into the windings to interfere with reliable operation. The fields are not strong enough to affect the diskettes themselves however so you may continue to lay them on top of or to the side of the monitor.

The Zenith monitor used in the '130 has this transformer mounted in the left rear corner of the case near the base. If you have arranged your system so that the monitor sits somewhat left of center (to make the displayed legends line up with the function keys) and the disk unit sits to the left close to the '130 case, then the interference is probably occurring. The effect is usually an unexplained delay in loading a program or an excessive incidence of the "UNFORMATTED DISK OR UNRECOVERABLE READ/WRITE ERROR" message when using disk drive 1, particularly when the inside tracks are being accessed. If you are experiencing these symptoms, try moving the disk unit or monitor around and see if they change or disappear.

The best solution is to place the disk unit on the right side of the keyboard unit which completely solves the problem (be sure to leave an inch or two of clearance for the fan intake). If the disk must be to the left, try to keep it at least 6 inches from the keyboard and move the monitor as far right as possible. If these measures are unacceptable, you can shield the heads by inserting a 14.5" wide by 7.5" high piece of thin sheet steel (the material used to make heating ducts is ideal) between drive 1 and the right hand plastic panel in the disk case. If you are using your own monitor and disk reliability has been a problem, you may want to determine the location of the flyback transformer and try rearranging your system if it is on the same side of the monitor as the disk unit. ◆

MACRO ASSEMBLER PATCHES

If you have MACASM 1.1 and don't wish to upgrade to MACASM 1.2 (see page 19) at this time, here is a way to patch or avoid all known bugs in the macro assembler (which are minor). These changes apply only if your assembler "signs on" with the message:

MTU 6502 MACRO ASSEMBLER 1.1 (21-OCT-82)

The patches below fix the following symptoms:

1. If MACASM discovers that the listing file already exists, it may overwrite it with garbage before exiting with the message "OUTPUT FILE EXISTS".

2. The command line option E=P will not divert the error summary to the printer.

3. For directives which do not permit a forward reference in the operand (such as *=, .IF, etc.), a reference to a symbol which is defined by an equate to a forward reference is not detected as an error, and an undetected phase error may occur.

To correct these bugs, carefully execute the following CODOS commands:

```
GET MACASM
SET AEB=D
SET 214B=E6
SET 1354=EA EA
RESAVE MACASM 700 3386 3F00 4309
```

A side effect of the above changes is that MACASM will no longer permit a forward reference in the operand of an equate, so you should strike out the paragraph in the middle of page 9 of the MACASM 1.1 manual which indicates that forward references are permitted for equates.

(Continued on p.6)

(Continued from p.5)

Finally, a bug may cause incorrect conditional assembly when using nested conditionals. Since it is not practical to patch this bug you should either get the upgraded MACASM 1.2 which corrects this problem or not use nested .IFs. ◆

DON'T CHOKE YOUR '130

Fully expanded, your MTU-130 packs an unprecedented amount of power in a keyboard style package. But with this power comes "waste heat" which must be removed to keep the system operating properly. This is accomplished by a small fan mounted on the right side panel. If the fan opening is blocked by the disk unit or a box of diskettes, normal air circulation is cut off and overheating can occur, particularly if the system includes a DATAMOVER 68000 or MULTI-O board.

Usually the first symptom of overheating is a jittery, unstable display. If you see this, check the right side panel for at least 1" clearance in all directions. Sometimes a sheet of note paper or the MTU-130 Quick Reference Card can be sucked up against the fan opening. Simply remove the obstruction and the display should clear up in a couple of minutes. If the problem continues, feel for airflow around the fan intake and around the exit holes near the light pen connector. If there is none, a paper clip or other obstruction may be keeping the fan from turning. ◆

LINGERING SVC PROC. Z BUG

The new SVCPROC.Z file on the new Distribution Disk 1.5 doesn't completely fix the bug it was intended to correct. The bug occurs when you use SVC 13 to execute a CODOS "GET" command which loads a file into a bank other that bank 0. The result is that the program crashes when the SVC 13 returns to the calling program.

To patch a pre-Distribution 1.5 version of the SVCPROC.Z use the following sequence of CODOS commands:

```
GET SVCPROC.Z
SET DF2E 4C 8F DF
SET DF8F 85 DB A9 00 8D CE E6 8D CF E6 60
RESAVE SVCPROC.Z DD20 DF99
```

To patch the version on the Distribution Disk 1.5, you may use the commands:

```
GET SVCPROC.Z
SET DF96 8D CF E6 60
RESAVE SVCPROC.Z DD20 DF99
```

The cause of the crash is simple but tough to figure out at first. Whenever CODOS loads a "SAVEd" file into memory, CODOS sets the User Program Bank and User Data Bank variables to the bank in which the first block of the file gets loaded. Since the GET command loads a "SAVEd" file, the User and Data Bank variables will be set as a result. In addition, these variables, and other user register variables are used by the SVC Processor to save the current status of the 6502. This is done so that the proper status may be restored after CODOS has been called to perform the desired function.

If that function was a GET command, then the User Bank variables will have been changed, and could be restored to undesired values. Such is the case when the GET command fetches a "SAVEd" graphics file to the display RAM. The SVC 13 returns with Program and Data bank set to 1 causing the program to crash. The patch above modifies the routine which handles SVC 13. This routine is made to unconditionally set the User Program and Data bank variables to bank 0 before these variables are used to restore the state of the 6502. The result is that SVC 13 will always return to the calling program with program and data bank set to bank 0. ◆

If you are using the SOURCE or COMPUSERVE networks and would like to contact other MTU-130 owners, please let us know your ID number. We will publish your name and ID number in a subsequent issue of UPDATE. If sufficient interest develops, we will start a telecommunications column in UPDATE. ◆

USER GROUP DISK III CONTENTS

Tools and Utilities

BITPAD.C - submitted by Ralph O. Erickson
Routine to accept digitized data from a Summagraphics BIT PAD ONE, and place them in BASIC integer variables, for use in a BASIC program.

BITPAD.A - submitted by Ralph O. Erickson
Source code for BITPAD.C. This source file also contains information on how to connect the bitpad to the MTU-130.

BITPAD.B - submitted by Ralph O. Erickson
Demonstration program, to illustrate how BITPAD digitizing program is used.

CFP.C - submitted by Dennis A. Lang
This program is like the CODOS TYPE command except that it can process embedded commands which display specified parts of memory in the output file. The result is text file which shows the status of various locations in memory.

CFP.A - submitted by Dennis A. Lang
The source file for CFP.A. This file contains the list of commands handled by the CFP.C program.

GETOPT.A - submitted by Dennis A. Lang
Part of the source code for CFP.C. This routine can be used separately for processing command line arguments. Some of the features include the ability to append different default extensions and drive numbers on various file names in the command line.

CFP.T - submitted by Dennis A. Lang
Documentation for using the CFP program. (See CFP.A of list of commands processed by the CFP program.)

DISK_ERROR.T - submitted by Dennis A. Lang
A control file for the CFP program which displays locations in memory where CODOS logs disk errors.

DEVICE_LIST.T - submitted by Dennis A. Lang
A control file for the CFP program which displays the current device table in CODOS along with the entry point addresses for each device.

CROSSREF.B - submitted by Bill Smith
CROSSREF.B is a BASIC program which generates a cross-reference listing of all program variables and line numbers referenced by GOTO, GOSUB, IF ... THEN, and RESTORE commands in any BASIC program stored as an ASCII file.

CROSSREF.T - submitted by Bill Smith
Documentation for the CROSSREF.B program.

DKSRCH.C - submitted by Dennis A. Lang
DKSRCH.C is used to help recover files that have been accidently deleted. It will display the CODOS file header (not normally seen by the user) for each possible file block. This will help find where the deleted file is found on disk.

DKSRCH.T - submitted by Dennis A. Lang
Documentation for the DKSRCH.C program.

EPSONFONT - submitted by Ralph O. Erickson
Lower case descenders are formed for the Epson MX-70 printer, using bit image mode. The program can be modified to form special characters on the MX-70 and MX-80 printers.

EPSONFONT.A- submitted by Ralph O. Erickson
Source code for EPSONFONT.

FANCYCLEAR.C - submitted by Hal Chamberlin
A Utility program which can clear the screen in unusual patterns. Also see the associated article in this newsletter.

FANCYCLEAR.A - submitted by Hal Chamberlin
Source code for FANCYCLEAR.C.

FANCYCLEAR.T - submitted by Hal Chamberlin
Documentation on how to use the FANCYCLEAR program.

(Continued on p.8)

(Continued from p.7)

FDUMP.C - submitted by Larry Isaacs
FDUMP is a program which dumps the contents of a disk file in a format similar to the CODOS DUMP command. The part of the file to be dumped may be specified by a starting and ending file position. (This program was written in MTU-C)

FDUMP.T - submitted by Larry Isaacs
The source code for the FDUMP program written in MTU-C.

FDUMP_DOC.T - submitted by Larry Isaacs
Documentation for using FDUMP program.

FILECRC.C - submitted by Hal Chamberlin
The FILECRC program is used to compute a CRC check value on the specified file. It can be used to determine if two files are identical.

FILECRC.A - submitted by Hal Chamberlin
Source code for FILECRC.C program.

FILECRC.T - submitted by Hal Chamberlin
Documentation for using the FILECRC.C program.

GETGRAY.C - submitted by Hal Chamberlin
The GETGRAY.C program is used to help generate gray-scale displays. A gray-scale image may be drawn using the normal drawing routines and using the left and right halves of the display as if they were two separate bit planes. Once the image is drawn in this manner, the GETGRAY program may be used to rearrange the bits in the display RAM as required by the gray-scale hardware to obtain the grey-scale image. (See USING GRAY SCALE article in this issue.)

ONGRAY.C - submitted Hal Chamberlin
The ONGRAY.C program is used set the appropriate hardware register to turn on the gray-scale display mode.

OFFGRAY.C - submitted by Hal Chamberlin
The OFFGRAY.C program is used to set the appropriate hardware register to turn off the gray-scale display mode.

GETGRAY.T - submitted by Hal Chamberlin
Documentation on how to use the GETGRAY.C, ONGRAY.C, and OFFGRAY.C programs.

HELP.C - submitted by Dennis A. Lang
A utility program which processes a special "help" file called HELP.H. It accepts a command line argument which is a string containing one or more indentifiers. The HELP.H file will be searched and all information relevant to the specified indentifiers will be displayed.

HELP.A - submitted by Dennis A. Lang
The assembly language source file for the HELP.C program. This source file contains most of the documentation for using and updating the HELP programs. Further information on using the HELP program may be obtained from the HELP program itself. Enter the command: HELP HELP

HELP.H - submitted by Dennis A. Lang
The "help" text file used by the HELP.C program.

MACLIB6502.A - submitted by Hal Chamberlin
This is a library of assembly language macros which implement various convenience functions for use in assembly language programs. These macros are used as if they were new 6502 instructions. Appropriate 6502 code will be generated to accomplish the desired function. These functions include such things as conditional jumps and 16-bit loads and moves.

MACLIB6502.T - submitted by Hal Chamberlin
Documentation for the MACLIB6502.A file.

MDO.C - submitted by Hal Chamberlin
The MDO.C program is an enhanced version of the CODOS DO command. It allows multiple jobs to be executed out of a single job file. An individual job is executed by specifying the job file and name of the job as arguments to the MDO.C program. This can save a lot of disk space if a large number of small jobs need to be present on a disk.

MDO.A - submitted by Hal Chamberlin
Source code for the MDO.C program.

(Continued on p.9)

(Continued from p.8)

MDO.T - submitted by Hal Chamberlin
Documentation on how to use the MDO.C program.

SCDUMPNEC.C - submitted by Roy Kerth
A modification to VMDUMP utility program to allow the user to select a portion of the screen to be dumped. The area is specified by use of the GRIN cursor.

SCDUMPNEC.A - submitted by Roy Kerth
Source code for SCDUMPNEC.C.

SCDUMPNEC.T - submitted by Roy Kerth
Instructions for the SCDUMPNEC.C program.

SYSDATE.C - submitted by Dennis A. Lang
The SYSDATE.C program is an enhanced version of the CODOS DATE command. It will read an existing date from the SYSDATE.T file and allow you to modify it or leave it unchanged. The date is set accordingly, and the SYSDATE.T file updated.

SYSDATE.A - submitted by Dennis A. Lang
Source code for the SYSDATE.C program. Also, instructions for using the SYSDATE.C program are contained at the beginning of the source code.

SYSDATE.T - submitted by Dennis A. Lang
This file is used by the SYSDATE.C program to hold the most recently used date.

Applications

ADM.C - submitted by Roy Kerth
A modification to TERM to emulate a Lear Siegler ADM3 terminal.

ADM.A - submitted by Roy Kerth
Source code for ADM.C.

ADM.T - submitted by Roy Kerth
Instructions for the ADM.C program.

NAMES.B - submitted by Joseph A. Carr
This is a BASIC program which manages a file of names, addresses, telephone numbers and free form information.

NAMES.J - submitted by Joseph A. Carr
A job file which starts up NAMES.B.

NAMESDOC.T - submitted by Joseph A. Carr
Documentation for the NAMES.B program.

NAMESDATA.T - submitted by Joseph A. Carr
A sample data file for the NAMES.B program.

Entertainment and Demo

BLACKBOX.B - submitted by James Clarkson
A BASIC game where you try to determine the positions of atoms in a box. This is done by shooting rays through the box and observing how the rays are affected by the atoms.

BULLBUZZ.B - submitted by Bill Cadwallender
An official DOD (Department of Defense) buzzword generator. Helps you to include systematized nonfunctional officialese (SNO) in your reports.

HAMMURABI.B - submitted by James Clarkson
A BASIC game where you are Hammurabi, ruler of Acient Sumeria. You must decide how to allocate last year's harvest for the coming year. Will the peasants prosper under your rule, or will they have to revolt?

HIDE_LIN_REM.B - submitted by O. Asakawa
HIDE_LIN_REM.B is a BASIC demo program which will graph math functions in three dimensions with hidden line removal. The angle of view, selections of the portions of the function to graph as well as a choice of plotting with dots, lines or mesh are provided. This program requires the KGL library.

HIDE_LIN_REM.T - submitted by Bill Smith
Documentation for HIDE_LIN_REM.B.

HIDE_LIN_REM.G - submitted by Bill Smith
This is a graphics file which illustrates the effect of certain variables on the image plotted by HIDE_LIN_REM.B.

HURKLE.B - submitted by James Clarkson
A BASIC game where you try to find the location of the Hurkle.

(Continued on p.10)

(Continued from p.9)

OTHELLO.B - submitted by Bill Smith
A BASIC game which plays OTHELLO, you versus the computer.

PARTINBOX.B - submitted by Douglas Eadline
A BASIC program that plots the solutions to the simple (?) quantum mechanical problem of a particle confined to a two dimensional box. The program allows the user to input quantum numbers and calculates the energy, wave function, and PSI squared for the particle. The plots will look interesting even if you don't understand them.

PARTINBOX.T - submitted by Douglas Eadline
PARTINBOX.T contains a quick tutorial on the quantum mechanics needed to understand the program. The tutorial is written with a touch of humor, so you should find it entertaining as well as informative.

QUEST.B - submitted by James Clarkson
A simple adventure game written in BASIC. The object of the game appears to be to find the Pirate's treasure.

TURTLE.C - submitted by Dennis A. Lang
TURTLE.C is a demo program of some simple turtle graphics written in MTU-C.

TURTLE.T - submitted by Dennis A. Lang
The C source code for the TURTLE.C demo program. ◆

ORDERING USER GROUP DISKS

All of the programs listed in the User Group Disk III Contents can be running on your system for only $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 submitted by MTU-130 users on this disk 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. ◆

SUBMITTING ARTICLES AND PROGRAMS

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 a maximum width of 43 columns, one space before the first word in the paragraph, and a blank 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 disk for return to you in the same packing in which we received it. ◆

CORRECTION TO USER DISK II

The file referred to as ADP.J in the USER DISK II CONTENTS in MTU UPDATE #2 should read ABP.J. Also, the last command in this file should be:

```
RUN "MENU1"
```

instead of RUN "MENU". ◆

INPUTS & OUTPUTS

THE 68000—A NEW EXPERIENCE

by Dale E. Hedman
Schenectady, NY

I have been interested for some time in the new 16-bit microprocessors so I decided to buy the DATAMOVER, having faith that MTU would provide a good system with software support. I purchased the DATAMOVER to enhance the speed of BASIC and to take this opportunity to learn the machine language for the 68000. While I have had the DATAMOVER for only a few weeks my experience with the system may be of interest to others. Therefore I will comment on the machine language programming tools supplied with the DATAMOVER.

For me the big problem came when I tried to relate to the 68000 machine language commands and to the use of the 6502 assembler as an aid in the "hand assembly" of 68000 code as described in the MTU documentation. The problem was difficult because of my lack of familiarity with the 68000 mnemonics but was compounded by the new set of mnemonics supplied by MTU as the "op code equates" required to build the 68000 code using the 6502 assembler.

For those who are familiar with 6502 machine or assembly language coding, the 68000 is quite a different experience. The 68000 operands can be 8,16, or 32 bits long and are referred to as Byte, Word, or Long word operands. In addition there are 16 registers, 8 data registers and 8 address registers, each 32 bits long. There are up to 14 addressing modes available for the 68000 including register direct, absolute, program counter relative, register indirect, and immediate. Indexed and automatic incrementing are available for some of the addressing modes. In addition relative addressing with 8 bit and 16 bit offset are available for branching operations. The 68000 op code or instruction, being 16-bit, generally contains part or all of the addressing information, register and operand length as well as other information. The instruction may also be followed by several bytes of information, with the total instruction being from one to 5 16-bit words long. Obviously the best approach to program development on the 68000 would be to use a commercial assembler, but I prefer to use the development tools supplied with the DATAMOVER to save a bit of money at least until I get my feet on the ground. (By the way, a full scale cross assembler for the 68000 which will run on the MTU-130 is now available from MTU.)

Most of the 68000 op codes use source and destination designators in the following way:

(Source) <operation> (Destination) =>
Destination

Generally the source and destination are registers or the effective address, <ea>. The effective address is an important concept for the 68000 addressing because this is a standard part of the instruction word and refers to the 12 addressing modes available for each instruction. Two of the 14 addressing modes, quick immediate and implied register, are accomodated by the op codes and are not part of the effedtive address selection. Also you should note that not all addressing modes are available for each op code as is shown in TABLE IV. The registers refer to the 8 data registers, Dn, and the 8 address registers, An, where A7 is also the stack pointer, SP.

Because of the lack of books and articles describing the assembly language programming of the 68000, I had to develop my understanding from the documentation supplied by MTU and the sample programs that were delivered with the DATAMOVER. I found the going rather tough for a couple

(Continued on p.12)

(Continued from p.11)

of days because of the two different mnemonics used as mentioned above. The MTU mnemonics were developed to aid in the "hand assembly" of machine language programs using the standard 6502 assembler where the assembler becomes a bookkeeping device and allows the use of labels and equivalents to be combined to form instructions. The instructions, which are 16 bit words, are constructed by adding elements together. The elements used in this construction are the "op code equates", where each element is a bit pattern which comprises part of the required instruction. In addition addresses and labels used in the assembly are combined into the desired instruction. Thus you will see that the final MTU op codes will typically be sums of elements.

The primary purpose of this article is to comment on the different mnemonics used by MTU and Motorola. I had to develop a systematic comparison of the MTU and Motorola mnemonics to be able to understand the MTU documention and to relate this to the Motorola Users Manual supplied with the DATAMOVER. This was done by sorting the op codes into groups with similar characteristics and to place the Motorola and MTU op codes side-by-side in the following tables.

The addressing mode mnemonics for the Motorola op codes and the MTU "op code equates" are shown in TABLE I with the extension words for the addressing modes shown in TABLE III. TABLE II lists the symbols used in the assembler syntax op codes used by Motorola and MTU. The op codes are presented in three tables for convenience in reading. TABLE IV lists all op codes using multiple modes of addressing, TABLE V lists op codes by groups using similar syntax, and TABLE VI lists op codes which cannot be grouped by similar syntax.

There are a few Motorola op codes which are not included in the MTU "op code equates":

```
ANDI #xxx,CCR
ANDI #xxx,SR
EORI #xxx,CCR
EORI #xxx,SR
ILLEGAL
ORI #xxx,CCR
```

If any user finds it necessary, it should be no problem to add these to the "op code equate" file supplied by MTU.

Just so you get a feel for the differences between the Motorola and MTU code, look at TABLE VI. This is an example of a subroutine program to clear memory locations $20000 to $40000. Notice that the MTU code requires a .DBYTE statement to generate a two byte word which becomes the 68000 instruction. From this example it should be clear that it would be nice to have an assembler for program preparation, but it is possible to develop machine code with the 6502 assembler without an unreasonable amount of work. ♦

Editor's Note: See page 19 for a description of the 68000 assembler using standard mnemonics and syntax available from MTU.

TABLE I
DATA ADDRESSING MODE SYMBOLS

Mnem.=> No.	Motorola (ea)	MTU Source Add. MEA	MTU Dest. Add. DEA
1	Dn	DRD	DMDRD
2	An	ARD	---
3	(An)	ARI	DMARI
4	(An)+	ARII	DMARII
5	-(An)	DARI	DMARI
6	d(An)	ARIO	DMARIO
7	d(An,Xi)	ARIOX	DMARIOX
8	Abs.W	ABS.W	DMABS.W
9	Abs.L	ABS.L	DMABS.L
10	d(PC)	REL	----
11	d(PC,Xi)	RELX	----
12	imm	IMM	----

(Continued on p.13)

(Continued from p.12)

TABLE II
SYMBOLS USED IN ASSEMBLER SYNTAX

XXX = Op code mnemonic
Y = B,W,L as allowed

USED IN MOTOROLA SYNTAX

An = Address register
CCR = Condition code register
Dn = Data register
d = displacement
#(data) = Data as required
#(disp.) = Displacement
(ea) = Effective address
(label) = Program label or address
(mask) = Register list
Rn = Register
SR = Status register
#(vector) = vector number
#(xxx) = Immediate data

USED IN MTU SYNTAX

(address) = Program label or address
(bit no.) = Bit pattern
(count) = Bit count for shift or rotate
(data) = Data as required
DEA = Destination EA = DEA+Szz
(disp.) = Displacement
(dm) = Destination extension word
EA = Effective address = MEA+Szz
(hd),(ld) = Long operand extension
(high data),(low data)
(index reg) = Dn, used in index addressing
(m) = Extension word used by some addressing modes.
(mask) = Register list
(offset) = Offset to address
(sm) = Source extension word
(ud),(ld) = (high data),(low data)
(vector#) = vector number
(xxx) = Immediate data
zz = Dn or An registers (Preceded by S for some oper.)

TABLE III
EXTENSION WORDS USED BY MTU ADDRESSING MODES

MTU Source Address MEA	MTU Dest. Adderss DEA	(m)
ARIO	DMARIO	(offset)
ARIOX	DMARIOX	(index reg)¥4096+(offset)
ABS.W	DMABS.W	(address)
ABS.L	DMABA.L	(address)
REL	DMREL	(address)
RELX	--	(index reg)¥4096+(offset)
IMM	--	(data) (for B or W) (hd),(ld) (for L)

NOTE: The MTU "op code equates" restrict the Indexed Address Register Indirect With Offset and the Relative Indexed Addressing to the use of the data register and word length offsets in the register. The 68000 instructions allow address registers and long word offsets in the register. The user can easily create the added "op code equates" if required.

(Continued on p.14)

(Continued from p. 13)

TABLE IV
OP CODES USING MULTIPLE MODES OF ADDRESSING

Motorola Syntax	MTU Syntax	Operand Length	Addressing modes
ADD,AND,OR,SUB			
XXX.Y (ea),Dn	XXX.ER.Y+EA+Szz,(m)	B,W,L	1,2,3,4,5,6,7,8,9,10,11,12
XXX.Y Dn,(ea)	XXX.RE.Y+EA+Szz,(m)	B,W,L	-,-,3,4,5,6,7,8,9,--,--,--
ADDA,CMPA,MOVEA,SUBA			
XXX.Y (ea),An	XXX.Y+EA+Szz,(m)	B,W,L	1,2,3,4,5,6,7,8,9,10,11,12
ADDI,ANDI,EORI,ORI,SUBI	((data)=) L requires (data)= (hd),(ld))		
XXX.Y #(data),(ea)	XXX.Y+EA,(data),(m)	B,W,L	-,-,3,4,5,6,7,8,9,--,--,--
ASL,ASR,LSL,LSR,ROL,ROR,ROXL,ROXR	(1 bit word shift or rotate only)		
XXX.Y (ea)	XXX.EA+EA,(m)	W	-,-,3,4,5,6,7,8,9,--,--,--
ADDQ,SUBQ	((data)= 0 to 7 =>3 bits long)		
XXX.Y #(data),(ea)	XXX.Y+EA+[sX(data)],(m)	B,W,L	1,2,3,4,5,6,7,8,9,--,--,--
BCHG,BCLR,BSET,BTST			
XXX.Y Dn,(ea)	XXX.R.Y+EA+Szz,(m)	B,-,L	1,-,3,4,5,6,7,8,9,--,--,--
XXX.Y #(data),(ea)	XXX.I+EA,(bit no.),(m)	B,-,L	1,-,3,4,5,6,7,8,9,--,--,--
CHK,CMP,DIVS,DIVU,MULS,MULU			
XXX.Y (ea),Dn	XXX.Y+EA+Szz,(m)	B,W,L	1,-,3,4,5,6,7,8,9,10,11,12
CLR,NEG,NEGX,NOT,TST			
XXX.Y (ea)	XXX.Y+EA,(m)	B,W,L	1,-,3,4,5,6,7,8,9,--,--,--
NBCD,SCC,SCS,SEQ,SMI,SNE,SPL,SGE,SGT,SLE,SLT,SLS,SHI,SVC,SVS,SF,ST,TAS			
XXX.Y (ea)	XXX.Y+EA,(m)	B	1,-,3,4,5,6,7,8,9,--,--,--
	Op codes added by MTU	SHS= SCC	SLO=SCS
JMP,JSR			
XXX (ea)	XXX+EA,(m)	B,W,L	-,-,3,-,-,6,7,8,9,10,11,--
CMPI.Y #(data),(ea)	CMPI.Y+EA,(data),(m)	B,W	1,-,3,4,5,6,7,8,9,--,--,--
	CMPI.Y+EA,(ud),(ld),(m)	L	1,-,3,4,5,6,7,8,9,--,--,--
EOR.Y Dn,(ea)	EOR.R.Y+EA+Szz,(m)	B,W,L	1,-,3,4,5,6,7,8,9,--,--,--
LEA (ea),An	LEA+EA+zz	L	-,-,3,-,-,6,7,8,9,--,--,--
MOVE (ea),CCR	MOVTOCCR+EA,(m)	W	1,-,3,4,5,6,7,8,9,10,11,12
MOVE SR,(ea)	MOVFMSR+EA,(m)	W	1,-,3,4,5,6,7,8,9,--,--,--
MOVE (ea),SR	MOVTOSR+EA,(m)	W	1,-,3,4,5,6,7,8,9,10,11,12
MOVE.Y (ea),(ea)	MOVE.Y+EA+DEA,(sm),(dm)	B,W,L	
	Use EA for source modes		1,2,3,4,5,6,7,8,9,10,11,12
	Use DEA for destination modes		1,-,3,4,5,6,7,8,9,--,--,--
MOVEM.Y (mask),(ea)	MOVEM.RM.Y+EA,(mask),(m)	W,L	-,-,3,-,5,6,7,8,9,--,--,--
MOVEM.Y(ea),(mask)	MOVEM.MR.Y+EA,(mask),(m)	W,L	-,-,3,4,-,6,7,8,9,10,11,--
For predecrement addressing use mask:		D)	0,1,2,3,4,5,6,7 0,1,2,3,4,5,6,7 (A
For all other addressing use mask:		A)	7,6,5,4,3,2,1,0 7,6,5,4,3,2,1,0 (D
PEA (ea)	PEA+EA,(m)	L	-,-,3,-,-,6,7,8,9,--,--,--

(Continued on p. 15)

(Continued from p. 14)

TABLE V
OP CODES USING SIMILAR SYNTAX

Motorola Syntax	MTU Syntax	Operand Length	Comments
ABCD,ADDX,SBCD,SUBX			
XXX.Y Dy,Dx	XXX.RR.Y+zz+Szz	B,W,L	(zz is source, Szz is dest. reg.)
XXX.Y -(Ay),-(Ax)	XXX.MM.Y+zz+Szz	B,W,L	(zz is source, Szz is dest. reg.)
ASL,ASR,LSL,LSR,ROL,ROR,ROXL,ROXR			
XXX.Y Dx,Dy	XXX.Y+zz+[s%(count)]	B,W,L	((count)= 3 bits long
XXX.Y #(data),Dy	XXX.R.Y+[s%(count)]+zz	B,W,L	(count)=0 for shift of 8)
BCC,BEQ,BCS,BGE,BGT,BHI,BLE,BLS,BLT,BMI,BNE,BPL,BRA,BSR,BVC,BVS			
	(Op codes added by MTU	BHS= BCC	BLO= BCS BRN= No operation)
XXX (label)	XXX+[(-%-2+(address)]	B	
	XXX ,-%-2+(address)	W	(For greater than 128 byte disp.)
DBCC,DBCS,DBEQ,DBGE,DBGT,DBHI,DBLE,DBLS,DBLT,DBMI,DBNE,DBPL,DBRA,DBVC,DBVS			
	(Op codes added by MTU	DBHS=DBCC	DBLO=DBCS)
XXX Dn,(lable)	XXX+zz,-%-2+(address)	W	

TABLE VI
UNGROUPED OP CODES

Motorola Syntax	MTU Syntax	Operand Length	Comments
CMPM.Y (Ay)+,(Ax)+	CMPM.Y+zz+Szz	B,W,L	(zz is source, Szz is dest. reg.)
EXG Rx,Ry	EXG.DD+Szz+zz	L	
	EXG.AA+Szz+zz		
	EXG.DA+Szz+zz		(For EXG.DA use Szz for the data reg.)
EXT.Y Dn	EXT.Y+zz	W,L	
	(W= Sign extended from low byte to word of data register)		
	(L= Sign extended from low word to long word of data register)		
LINK An,#(displ.)	LINK+zz,(displ.)		
MOVE USP,An	MOVTOUSP+zz	L	
MOVE An,USP	MOVFMUSP+zz	L	
MOVEP.Y Dx,d(Ay)	MOVEP.RM.Y+Szz+zz,(disp)	W,L	(Szz= data reg. zz= add. reg.)
MOVEP.Y d(Ay),Dx	MOVEP.MR.Y+Szz+zz,(disp)	W,L	(Szz= data reg. zz= add. reg.)
MOVEQ #(data),Dn	MOVEQ+Szz+(data)	L	(imm. data= -127 to 128)
NOP	NOP		
RESET	RESET		
RTE	RTE		
RTR	RTR		
RTS	RTS		
STOP #xxx	STOP,(xxx)		
SWAP Dn	SWAP+zz	W	
TRAP #(vector)	TRAP+(vector #)		
TRAPV	TRAPV		
UNLK An	UNLK+zz		

TABLE VI
EXAMPLE PROGRAM COMPARISON OF MOTORLA AND MTU SYNTAX

---- Motorola Syntax ----			---------- MTU syntax ----------			-- Comment ----------
CLRVDSP	MOVEA.L	#$20000,A0	CLRVDSP	.DBYTE MOVEA.L+IMM+	SA0,$2,$0000	LOAD $20000 INTO A0
CLRVD1	CLR.L	(A0)+	CLRVD1	.DBYTE CLR.L+ARII+	A0	CLR. LG. WORD & INC. A0
	CMPA.L	#$40000,A0		.DBYTE CMPA.L+IMM+	SA0,$4,$0000	TEST IF DONE
	BNE	CLRVD1		.DBYTE BNE+[(-%-2+	CLRVD1]	LOOP IF NOT
	RTS			.DBYTE RTS		RETURN

USING GRAY SCALE

by Hal Chamberlin
MTU

As you are probably aware from published specifications and the demonstration disk, the MTU-130 has two different display modes. The normal mode is 2-level black-and-white with a resolution of 480 dots wide by 256 dots high. In this mode, each bit in display memory (there are 122,880 of them) controls one dot on the display. The grayscale mode gives 4 brightness levels (black, dim, normal, and bright) and a resolution of 240 dots wide by 256 dots high. In grayscale mode, two bits in display memory control each pixel. Since there are half as many pixels, the total size of the display memory remains the same at 15K bytes.

Except for the GRAYCHECKER demonstration program, there is really no standard software support for the grayscale mode. This article will describe how you can produce and control grayscale images with the standard graphics and text software and three simple programs available on User Group Disk #3.

Successful use of the grayscale mode requires two things; creating the proper bit pattern in display memory and enabling the grayscale mode. Of these, the latter is the easiest. Insert a work disk in drive 0 and enter the following commands (if you have User Group disk 3, just insert it instead and skip the commands):

```
SET B400 38 66 EE AD E0 BF 29 EF 8D E0 BF
        00 01
SAVE ONGRAY B400 B40C
SET B400 AD E0 BF 09 10 8D E0 BF 60
SAVE OFFGRAY B400 B408
```

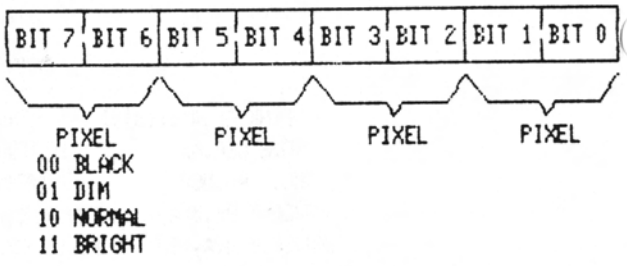
Now when you enter the command ONGRAY, the grayscale mode will be turned on. Likewise, OFFGRAY will restore the normal black-and-white mode. Go ahead and try the commands and note the effect of setting grayscale mode on patterns intended for normal mode. ONGRAY works by clearing bit 4 at address $BFE0 to zero while OFFGRAY works by resetting it to 1. From BASIC you can use:

```
Ongray:   POKE 49120,PEEK(49120)AND239
Offgray:  POKE 49120,PEEK(49120)OR16
```

Be very careful when manipulating the register at $BFE0 or 49120 because it also controls memory bank switching and other system functions and could lead to a crash if abused.

One complicating factor is that some CODOS operations will automatically restore normal display mode. These include normal returns from machine language programs with an RTS (note that ONGRAY uses an SVCRTS to return which will preserve grayscale), and most disk operations. Therefore a BASIC or machine language program may have to turn the grayscale back on after a disk operation.

Producing the bit pattern in display memory for grayscale images is conceptually quite simple. Each byte in display memory controls 4 grayscale pixels as in the drawing below:



It is fairly simple to build up a gray scale image one dot at a time. First, you assume that you are plotting on a grid that is only 240 dots wide. To plot a dim dot at x,y, you would actually plot a dot at X=2*x+1 and Y=y where X,Y are the coordinates you pass to the SDOT routine (use SMOVE X,Y: SRDRAW 0,0 in BASIC IGL). Normal brightness is plotted at X=2*x, Y=y while high brightness will require both of the point plots mentioned above.

While point plotting directly on the grayscale screen is easily done with the standard routines, line and character drawing is not. The logical solution is a

(Continued on p.17)

(Continued from p.16)

set of new routines or an upgrade of the old ones to support grayscale. A more easily implemented alternative is the GETGRAY program on User Group disk #3. The idea is to divide the normal black-and-white screen into a left half and a right half. The left half has X coordinates from 0 to 239 and the right half goes from 240 to 479. As before, you restrict your x coordinates to 0-239. To plot a dim point or line, you simply use X=x+240 and Y=y. To plot normal brightness, use X=x and Y=y. To plot bright, you simply plot twice; dim and normal. For characters you restrict to 40 columns and use COL=col+40 for dim, COL=col for normal, and draw the character in both columns for bright. For the IGL LABEL function, translate x as for graphics.

Once the image has been created in this "double vision" mode, the GETGRAY program can be used to convert it into the bit pattern required by the display hardware. If you execute the GETGRAY command with no arguments, the current screen content is converted. If an argument follows, it is expected to be the name of a .G graphic screen dump file and GETGRAY will load the file first and then convert it. Note that GETGRAY does not turn the grayscale mode on; you will have to use ONGRAY or poke the system control register at $BFE0 to do that (do not use ONGRAY as a SYSTEM command in BASIC to turn grayscale on, use the POKE given earlier). After conversion, you can resave the screen and skip the GETGRAY step when it is redisplayed. GETGRAY is also useful for doubling the width of a standard mode screen image for better legibility by a group. ◆

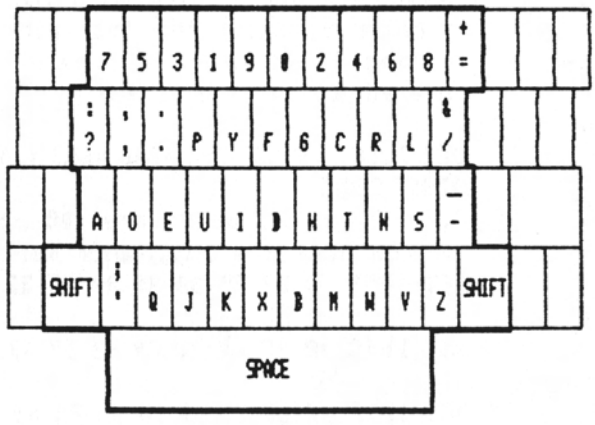
DVORAK KEYBOARD FOR THE '130

by Hal Chamberlin
MTU

As a kid, you undoubtedly asked your parents or teacher why the letters on typewriter keyboards were scrambled up so. The answer you got probably went something like "They're like that so your fingers won't have to travel so far to reach the most commonly used letters and you can type faster."

The truth is that while the usual "QWERTY" layout is probably better than "ABCDEF", it is substantially worse than an arrangement known as the Dvorak layout (that's a man's name, not the first 6 keys). In fact, the QWERTY layout was designed in 1873 by Christopher Sholes to slow down people's typing speed so that the early typewriters didn't jam up so much! Quite a number of controlled tests comparing Dvorak with Sholes on electronic keyboards have concluded that substantial improvement in accuracy and speed can be expected after only a few weeks of practice when an average typist switches to the new layout. In fact, the world typing speed record was achieved on a Dvorak keyboard. Unfortunately old traditions die hard although one can occasionally find a typewriter with an optional Dvorak keyboard available.

Below is a drawing of the "official standard" Dvorak keyboard layout, sometimes called the "American Simplified Keyboard" or ASK. As you can see, the letter, number, and common punctuation keys are affected. Also note that many of the special ASCII symbols are not accounted for. Nevertheless, it is actually quite simple to convert your MTU-130 to the Dvorak layout because the keyboard is completely software driven.

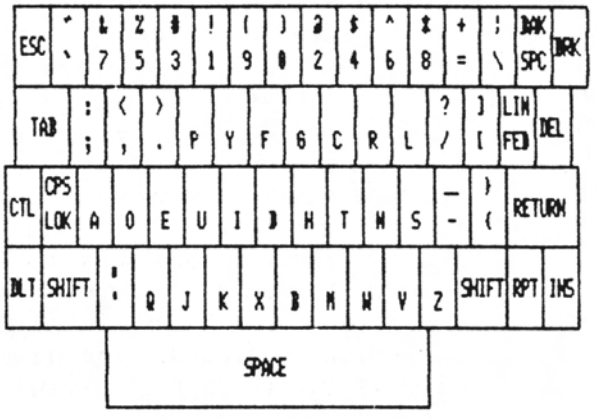


In order to convert to the Dvorak layout, two things must be done. First, the keyboard scan software must be modified

(Continued on p.18)

(Continued from p.17)

and second, the caps for many of the keys must be shuffled around. Since the keytops are stepped rather than "sculptured", they can be reshuffled without problems. In the discussion that follows, a slightly modified version of the Dvorak layout will be used because full compliance would require the upper/lower case pairings of some of the characters to be changed and thus new keytops would be needed. The modified layout is shown below:



Nearly all MTU-130 software relies on the IODRIVER.Z keyboard subroutine. Modifying the translate table in this routine and re-saving IODRIVER.Z is all that is needed in most cases. There are however a few MTU programs that have their own keyboard routines embedded. These are currently: WORDPIC, AUTOTERM, and DMXMON. If you use any of these programs, the same table will have to be modified in them as well.

Assuming that a disk with IODRIVER.Z is in drive 0, enter the following commands to modify it (each command should be entered all on one line):

```
GET IODRIVER.Z =700 =306 =6E0 =15B0
DUMP CB5C
 The first few bytes are: 00 1B 31 32 33
 if you have the applicable version.
SET 1B5E 60 37 35 33 31 39 30 32 34 36 38
        3D 5C
SET 1B6E 3B 2C 2E 70 79 66 67 63 72 6C 2F
        5B
SET 1B7F 6F 65 75 69 64 68 74 6E 73 2D
SET 1B8F 27 71 6A 6B 78 62 6D 77 76 7A
SAVE IODRIVER_ASK.Z =30F 700=200 73D 306
   395 6E0 6FF 15B0=C5B0 2272 FD50:1 FFEF
```

IODRIVER_ASK.Z

The Dvorak layout is now in effect. Now enter the following command:

GET GRAPHDRIVER.Z

When you are satisfied that your patches are correct, you may either delete the original IODRIVER.Z and rename IODRIVER_ASK to IODRIVER.Z or edit the STARTUP.J file so that it loads IODRIVER_ASK instead of IODRIVER.Z.

The procedure for WORDPIC is similar. Enter the following commands to modify WORDPIC:

```
GET WPEDIT
DUMP B9D2
 The first few bytes are: 00 1B 31 32 33
 if you have the applicable version.
SET B9D4 60 37 35 33 31 39 30 32 34 36 38
        3D 5C
SET B9E4 3B 2C 2E 70 79 66 67 63 72 6C 2F
        5B
SET B9F5 6F 65 75 69 64 68 74 6E 73 2D
SET BA05 27 71 6A 6B 78 62 6D 77 76 7A
SET BA34 7E 26 25 23 21 28 29 40 24 5E 2A
        2B 7C
SET BA44 3A 3C 3E 50 59 46 47 43 52 4C 3F
        5D
SET BA55 4F 45 55 49 44 48 54 4E 53 5F
SET BA65 22 51 4A 4B 58 42 4D 57 56 5A
SAVE WPEDIT_ASK 700 550E AE00 B2C9 B400
B710 B740 BAC1
```

To modify DMXMON enter (be sure you have a functioning Datamover board in the system):

```
GET DMXMON
DUMP 2A32
 The first few bytes are: 00 1B 31 32 33
 if you have the applicable version.
SET 2A34 60 37 35 33 31 39 30 32 34 36 38
        3D 5C
SET 2A44 3B 2C 2E 70 79 66 67 63 72 6C 2F
        5B
SET 2A55 6F 65 75 69 64 68 74 6E 73 2D
SET 2A65 27 71 6A 6B 78 62 6D 77 76 7A
SET 2A94 7E 26 25 23 21 28 29 40 24 5E 2A
        2B 7C
```

(Continued on p.19)

(Continued from p.18)

```
SET 2AA4 3A 3C 3E 50 59 46 47 43 52 4C 3F
        5D
SET 2AB5 4F 45 55 49 44 48 54 4E 53 5F
SET 2AC5 22 51 4A 4B 58 42 4D 57 56 5A
SET 0 0 0
SAVE DMXMON_ASK 700 945 B00 3949 FC00:1
     FC38 0=BFBE 1 0:2 15F 700:2 DC9
```

To modify AUTOTERM enter:

```
GET AUTOTERM
DUMP 39FA
```

The first few bytes are: 00 1B 31 32 33 if you have the applicable version.

```
SET 39FC 60 37 35 33 31 39 30 32 34 36 38
        3D 5C
SET 3A0C 3B 2C 2E 70 79 66 67 63 72 6C 2F
        5B
SET 3A1D 6F 65 75 69 64 68 74 6E 73 2D
SET 3A2D 27 71 6A 6B 78 62 6D 77 76 7A
SAVE AUTOTERM_ASK 700 4016
```

After the keyboard driver code tables are modified, the last task is to rearrange the keytops. Removing and replacing a keytop is quite simple but care must be exercised. Each keyswitch on the MTU-130 keyboard consists of three parts: the keytop, the plunger, and the body which includes the switch contacts. These parts are held together by friction. The goal is to pry off just the keytop while leaving the plunger behind in the body.

To remove a keytop, use a moderate sized short-handled screwdriver as a prying tool. Start with a keytop at the edge of the array such as the /? key. Slowly pry the cap up while rocking it back and forth to release the friction fit between it and the plunger. The cap should gradually slide up and off leaving the plunger behind. If you feel it snap off, the plunger has come loose from the body instead. In this event, go ahead and pull the cap and plunger out. Then separate the plunger from the cap and CAREFULLY re-insert the plunger into the body IN THE SAME ORIENTATION THAT IT CAME OUT. The horizontal bar inside the plunger MUST FIT BETWEEN the two sets of contacts. Watch this carefully to avoid crushing the contacts.

Remove all of the keytops that are affected by the new layout starting at the edge of the array and working toward the center. Then re-install them in the their new Dvorak positions. You should now be able to practice typing "Gone With the Wind" in a couple of evenings. ◆

NEW PRODUCTS

68000 CROSS ASSEMBLER

The DATAMOVER Cross Assembler, DMXASM, is complete and being shipped. This powerful assembler will run on any MTU-130 either with or without a DATAMOVER 68000 board installed and makes programming the 68000 in assembly language even easier than programming the 6502. DMXASM is over 90% compatible with the Motorola Exormacs assembler and uses exactly the same mnemonics. Automatic op-code and address mode selection is supported as well as conditional assembly and macros. Yet even with the more complex Motorola syntax and greater number of 68000 instructions, DMXASM still assembles programs at over 1500 lines per minute including a listing and sorted cross-reference map. A more complete description of DMXASM appeared in MTU UPDATE #2. The price is $190 for the software (includes source for an improved HATCOMP) and $10 for the manual. ◆

MACASM 1.2

An improved version of the 6502 macro assembler, MACASM, is now available. This new version has a number of bugs corrected including the two mentioned earlier in this issue. In addition, nesting of conditional assembly (.IF, .ELSE, .ENDIF) now works properly. Improvements include a more specific error message when either the listing or object file already exists, and a correctly formatted assembly summary (amount of symbol table used, etc.) when the errors file has been redirected to a file or the printer. If you have already purchased MACASM, you may upgrade to MACASM 1.2 for $15. If you have not yet

(Continued on back cover)

(Continued from p.19)

purchased MACASM but do own the original ASM, you may upgrade to MACASM 1.2 for $25 including manual. If you own neither ASM nor MACASM, the price is $90 including manual. ◆

IN THE QUEUE

8-BIT AUDIO A-TO-D CONVERTER

As an MTU-130 owner you have undoubtedly heard the speech output program on the standard demonstration disk and thought aloud "Sounds great but how do I enter phrases for my own application?". Well, the DIGISOUND-8 product currently under development will answer just that question and more. DIGISOUND-8 will be a hardware device for microphone input of your own voice and a set of comprehensive software for creating and playing speech files.

The hardware will be a small, separate box with cable that plugs into the '130's parallel I/O port where it receives both interface signals and operating power. Since it will connect to Port A, it may be used simultaneously with a parallel printer provided a "Y adapter cable" is used. The A-to-D converter will have jacks for direct microphone input, input from a tape recorder or other audio source, and a direct unfiltered input for special applications. A tone switch and automatic gain control insure high quality digitized speech under less than ideal recording conditions.

The software will consist of an interactive speech file recording program, a phrase library creation/maintenance program, and a greatly improved SPEAK routine. By putting all of the words and phrases associated with a particular application in one file and using CODOS's direct access provision, phrases may be found almost instantaneously and combined seamlessly. The improved SPEAK will also be usable as a BASIC command library.

The price for DIGISOUND-8 will be about $200 and should be available this Spring. ◆

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

RALEIGH FEB24'83 N.C. U.S. POSTAGE ≡ 0.17 RB.METER 1537326

RALEIGH FEB22'83 N.C. U.S. POSTAGE ≡ 020 RB.METER 1537326

FIRST CLASS MAIL