M T U - 1 3 0

S E T U P A N D I N S T A L L A T I O N

MODULE INTERCONNECTION
FIRST TIME POWER-UP PROCEDURES
SOFTWARE TUTORIAL

February, 1982 REV. B

TABLE OF CONTENTS

CHAPTER 1

INITIAL SYSTEM SETUP

Congratulations! You are about to use the Micro Technology Unlimited MTU-130 Desktop Computer, the most advanced 6502-based computer available. After performing a brief unpacking and setup procedure, you will become acquainted with various software components of the system. Following this you will be free to explore the MTU-130's vast resources in your own programs.

To minimize the possibility of trouble or misunderstanding in this critical setup and "get aquainted" phase, it is recommended that you at least skim Chapters 1 and 2 of this manual before actually doing anything. Then follow the instructions closely but use common sense. If you don't understand a step thoroughly, read ahead a little and it should become clearer in the overall context of the operation. If a serious question still remains, consult someone experienced in small computers or call MTU at 919-833-1458.

UNPACKING

Your complete MTU-130 system was shipped in four separate cartons. The smallest of course contained this manual which you are now reading. Of the other three, the largest contains the disk unit, the lightest contains the display monitor, and remaining one contains the computer itself. If you did not buy a complete system, either the disk unit or the display monitor or both will be missing.

1. Check for Damage

First check each carton for obvious shipping damage such as severely crushed corners, holes ripped in the sides, or deep puncture holes. If such damage is evident, don't unpack the carton! Instead, notify the transportation company who may wish to see the carton damage firsthand. It is more difficult to prove shipping damage after the carton has been opened. Remember that the transportation company is responsible for damage claims, not MTU.

2. Computer Carton

The computer carton should be unpacked first. Use a knife to cut the tape and open the top flaps. Remove the bags of cables and set them aside. Then carefully remove the computer itself, which looks like a large keyboard, and set it on a tabletop. Now take an inventory of the contents using the list below:

___A. MTU-130 Computer

___B. A.C. Power cord

___C. Video cable (has Phono plugs on both ends)

___D. Light pen (includes fiber-optic cable and connector)

___E. Bag containing spare fuses

Check the computer for shipping damage, particularly the keyboard keys and the connectors on the rear panel. If any serious damage is found, notify the transportation company.

3. Monitor Carton

Next unpack the monitor carton. Remove the monitor manual from its wrapping and insert it into the Hardware Documentation section of the System Notebook. Check the monitor case carefully for cracks or other evidence of shipping damage. Look at the CRT screen, it should be uniform in color. If a large dark spot is seen in the middle of the screen, the tube has been broken and you must not apply power to the monitor or it may be damaged further. Now set the monitor on the table with the computer and proceed.

4. Disk Unit Carton

This is the largest and heaviest carton. It should be opened while on the floor to minimize the possibility of dropping it while unpacking. Inside you should find the disk unit manual, a bag with spare fuses, and the disk unit itself. Insert the manual into the System Notebook. Check the disk unit for shipping damage, particularly around the disk drive doors. Remove the white shipping protectors in the disk drives. Set the disk unit aside as it won't be used until later.

DRAWINGS OF MAJOR COMPONENTS

You should now become familiar with the various parts and controls of the computer, monitor, and disk units. This will make it easier to understand the connection and operation instructions later. Examine the drawings below and refer to them whenever necessary.

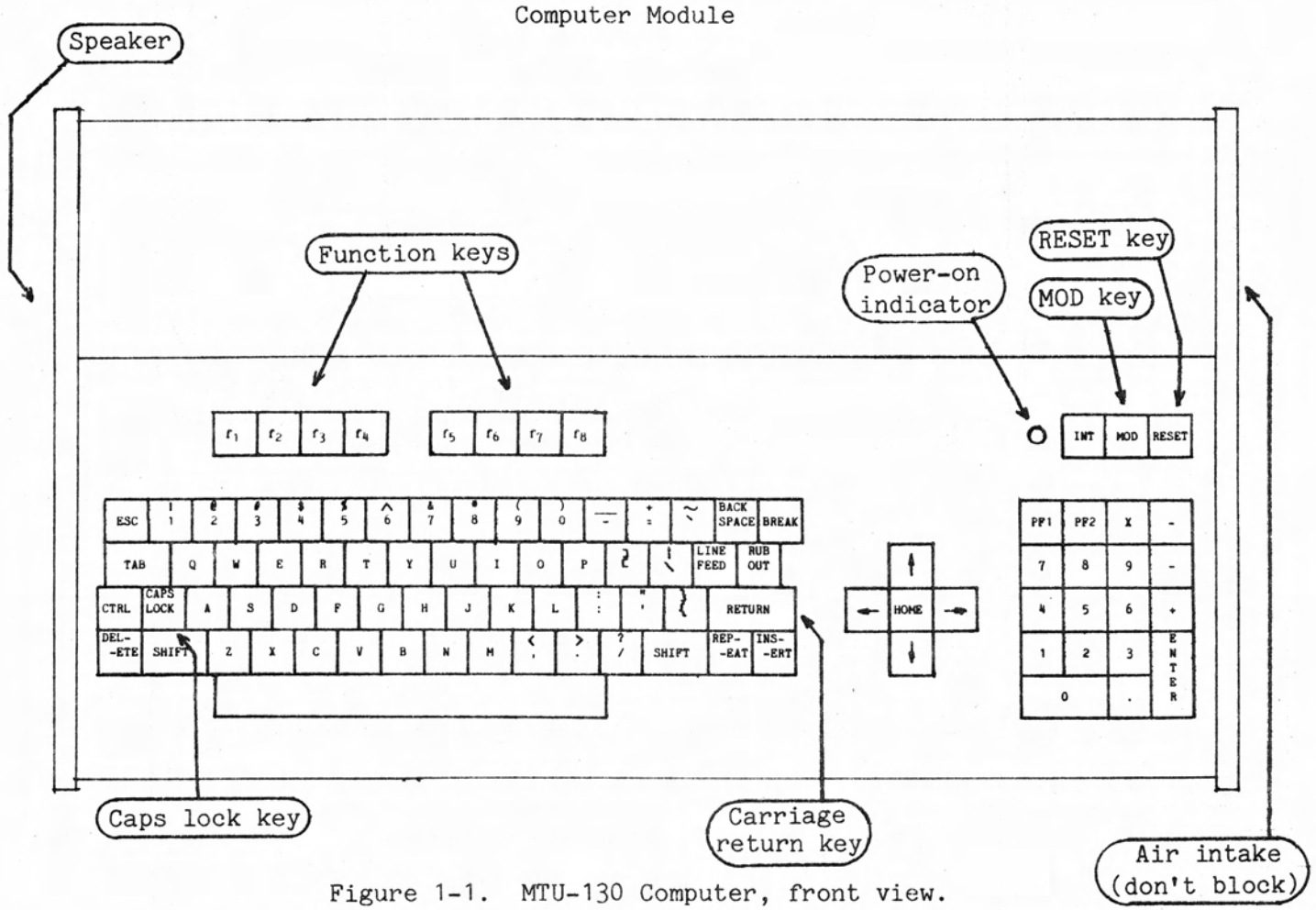


Figure 1-1. MTU-130 Computer, front view.

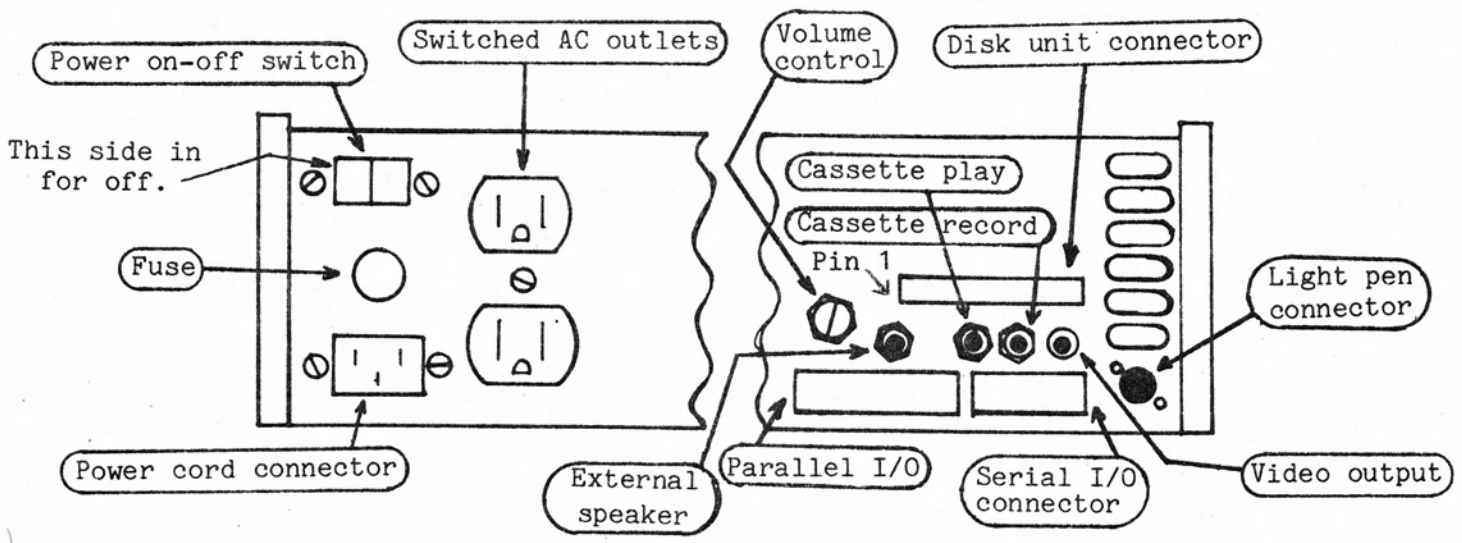


Figure 1-2. MTU-130 Computer, rear view.

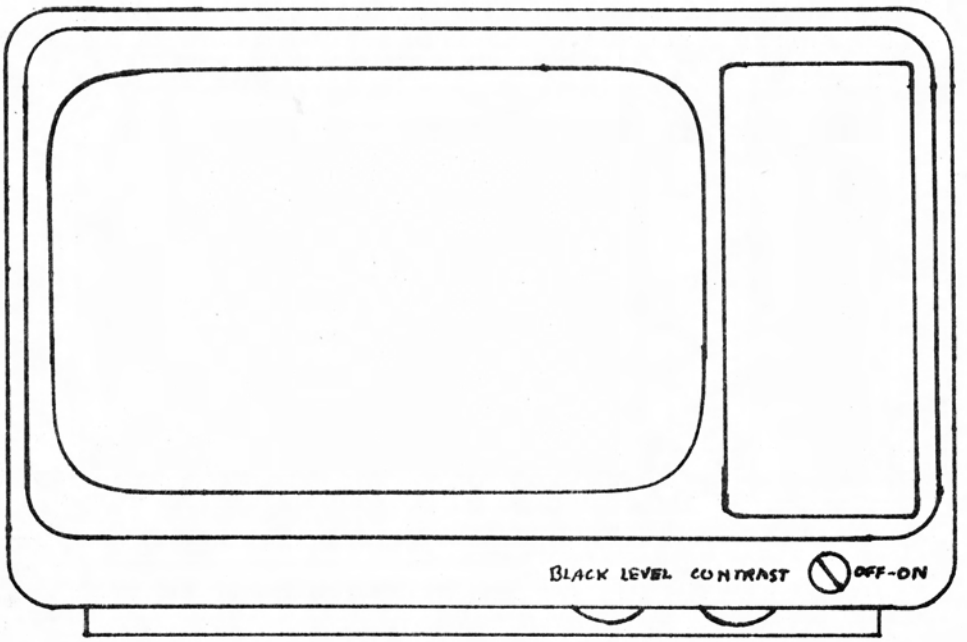


Figure 1-3. Display Monitor, front view.

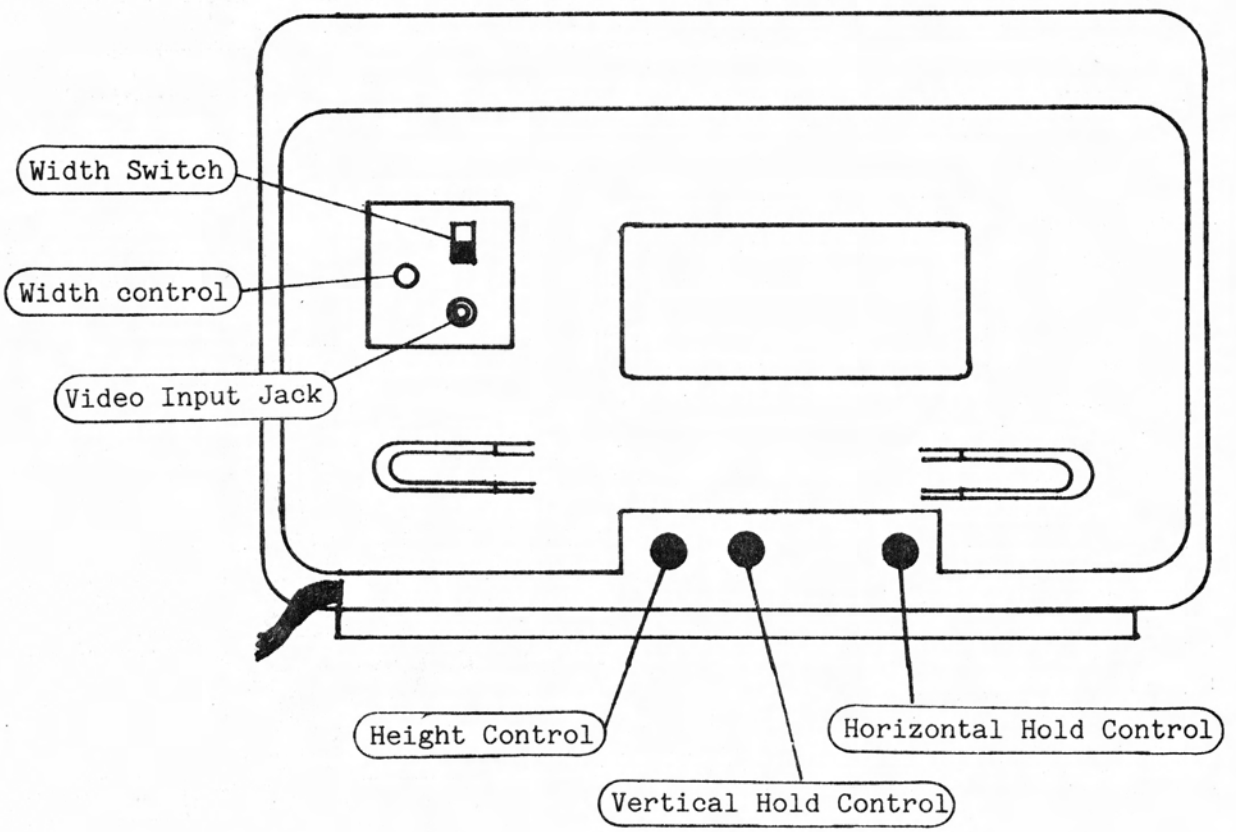


Figure 1-4. Display Monitor, rear view.

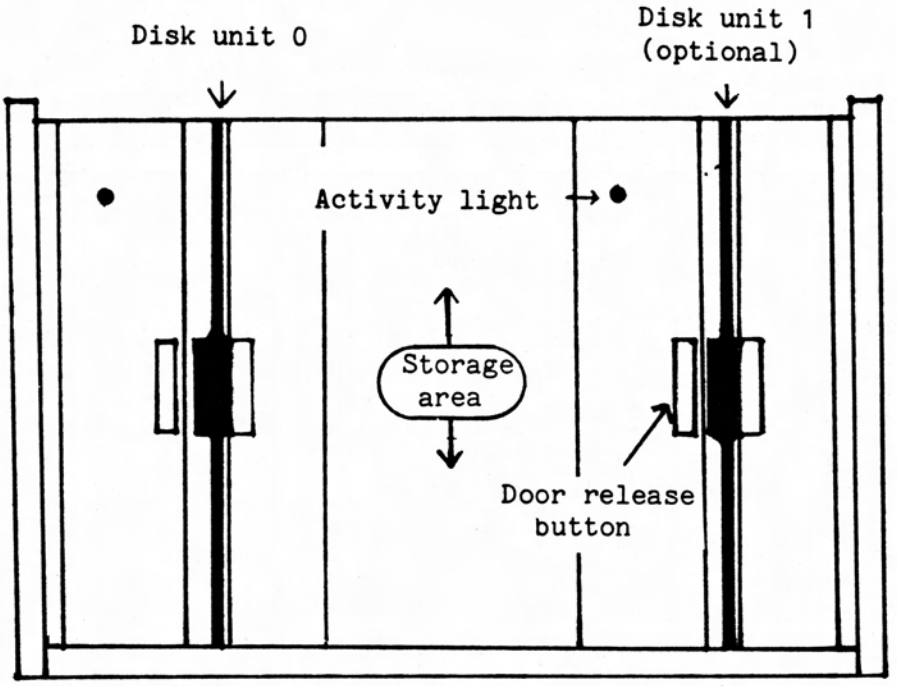


Figure 1-5. Disk Unit, front view.

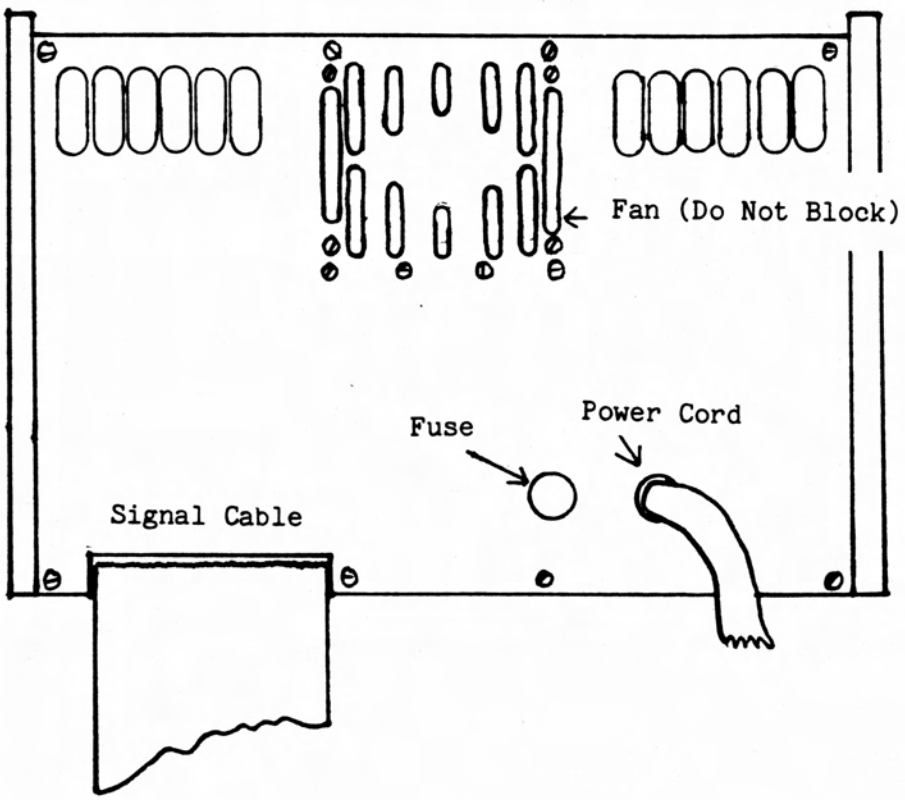


Figure 1-6. Disk Unit, rear view.

CONNECTING THE DISPLAY MONITOR

You will now connect the display monitor to the computer and turn on both units to make preliminary monitor adjustments. If you have purchased a complete system with monitor, there should be very little difficulty in connecting and adjusting it. If you are using your own monitor, be sure to carefully read each paragraph of this section. Remember that you will be looking at the display most of the time you are using the computer so a well adjusted, crisp image is important.

1. Monitor Requirements

If you have an MTU monitor, it already meets these requirements and you may skip to paragraph 2.

A. The monitor must accept 1 to 2 volt amplitude composite video input with negative sync. Either a 75 ohm or high impedance input is acceptable when using the MTU supplied video cable.

B. The horizontal oscillator must be able to synchronize to a 16.129KHz signal.

C. The vertical sweep retrace time should be .8 millisecond or less.

D. The rated video frequency response should be at least 10MHz for good sharpness but a usable image can be obtained with a 6MHz monitor.

E. The screen should have a white (type P4) or fast response green (type P31) phosphor if the light pen is to be used. P39 is too slow for the light pen.

F. To use the MTU supplied video cable, the monitor should have an RCA Phono plug input jack.

2. Monitor Connection

Locate the video cable supplied with the MTU-130 computer. (It has Phono plugs on each end.) Now connect one end to the video input jack on the monitor. Be sure the connection is tight to avoid an intermittent ground connection. Plug the other end of the cable into the video output jack of the MTU-130. Make sure it is plugged in far enough so that the connector shell makes good contact with the outside metal tube of the video output jack.

3. Initial Test and Adjustment

You will now apply power to the MTU-130 and the monitor in order to perform initial monitor adjustments. This will simplify the real power-up test later.

A. Turn the monitor power switch ON (turns clockwise on the MTU monitor).

B. Plug the monitor power cable into one of the switched AC outlets on the MTU-130

C. Make sure the MTU-130 power switch is OFF (the side of the switch nearest the edge of the back panel is in).

D. Locate the MTU-130 power cord. Plug the female end into the MTU-130 and the male end into the wall (European customers please refer to the note attached to the power cord).

E. Turn the MTU-130 power switch ON. Immediately look at the pilot light near the upper right corner of the keyboard. If it does not light, IMMEDIATELY turn the power switch OFF and see the "In Case of Trouble" section at the end of this chapter.

F. After a few seconds, the monitor screen should light up. If necessary, adjust the contrast and black level control to see the image clearly. Since the disk unit is not yet connected, the image being generated is semi-random data in the display memory. When the monitor is adjusted, it will most likely appear to be a number of somewhat broken up vertical bars.

G. If the image is rolling vertically or rapidly flickering, adjust the Vertical Hold control until the image is stable.

H. If you see diagonal bars on the screen instead of a rectangular image, adjust the horizontal hold control until the overall image becomes rectangular. If you are using your own monitor and you cannot achieve horizontal synchronization, refer to the "In Case of Trouble" section.

I. If necessary, adjust the black level and contrast controls until the background outside the image rectangle just disappears.

J. Now use the Horizontal Hold control to center the image horizontally. If the side of the image rectangle starts to bend while doing this, back off a little.

K. Use the Vert size control to reduce the height of the image until it just fills the center 2/3 of the screen. If the image starts to roll while doing this, use the Vertical Hold control to stop the rolling.

L. Initial monitor adjustment is now complete and should be good enough to read the CODOS greeting message that will be displayed later after the disks are connected. Final touch-up can be performed with the Checkerboard program to be described later.

M. Turn the MTU-130 power switch OFF. Notice that the monitor was also turned off since the MTU-130 AC outlets are controlled by the switch. Now unplug the MTU-130 power cord and continue with the next section.

CONNECTING THE DISK UNIT

You will now connect the disk unit to the computer. Since the MTU-130 reads all of its system software into memory from disk, a properly connected and functioning disk unit is required for the system to do anything at all. If you have purchased a complete system with disk, there should be very little difficulty in connecting it. If you intend to use your own disk drives or a disk unit purchased elsewhere, you should refer to the Disk Controller manual in the Hardware Documentation section before going any further.

1. Disk Unit Requirements

If you have an MTU disk unit, it already meets these requirements and you may skip to paragraph 2.

A. The disk unit must accept standard 8 inch, 77 track, soft sectored diskettes.

B. It must have its own D-C power supply and draw less than 2.0 amps from the A-C power line.

C. It must be able to connect to a standard 50 pin flat ribbon cable connector using Shugart standard signals and pin assignments.

D. Double sided drives must use the separate Side Select signal, not the Direction signal to select diskette sides.

E. All disk drive jumpers must be set up for Daisy Chain operation and the track seek stepper motor must be powered at all times.

F. Head Load must not be activated by Drive Select alone.

2. Disk Unit Connection

Follow these instructions for connecting the disk unit to the MTU-130:

A. Remove the white shipping protectors in the disk drives before powering up the the disk drives. Position the disk unit close to the computer unit. It should normally be placed on the side of the keyboard opposite the one on which you usually put papers you are typing from.

B. Locate the Disk Signal Cable. It is part of MTU supplied disk units. Also locate the disk cable receptacle on the rear of the computer.

C. The end of the cable has two rows of 25 receptacles and could possibly be plugged into the computer backward. Therefore there is a raised plastic boss on the bottom side of the cable connector and matching keyway slot on the computer's connector to make backward insertion difficult.

D. Carefully but firmly plug the disk cable into the computer making sure it is plugged in straight. The locking tabs on the computer connector will snap out when it is plugged in all the way. If the cable must ever be removed, press the tabs toward each other and gently pull the connector straight out.

E. If there is a wire and ringlug hanging from the cable, loosen the screw on the computer next to the connector, slip the lug under the screw, and retighten.

F. Plug the disk unit power cord into the second A-C outlet on the computer.

POWER UP

It is now time to power up the entire system and load the operating system software from disk. Follow these instructions carefully because a mistake here may delay your getting the system operational for several days.

1. Double-check the connections between the disk unit and the computer unit.

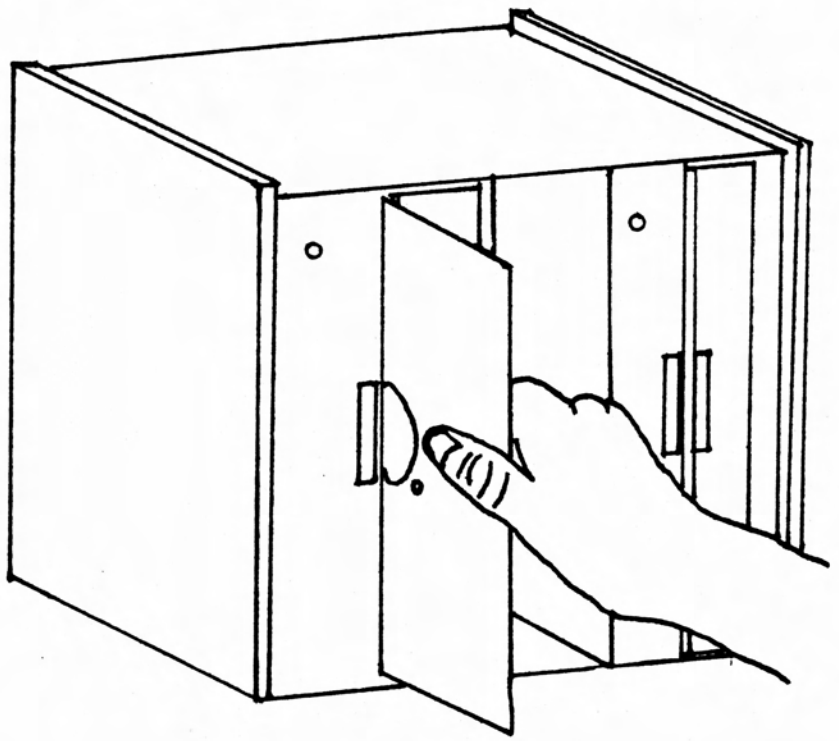
2. Plug the MTU-130 power cord into the wall and turn the power switch ON.

3. You should be able to hear the disk drive motors come on (the motors of 8 inch drives run continuously for quick access and accurate speed) and feel air flow around the intake grille on the disk unit.

4. The monitor should warm up and show an image similar to that seen earlier.

5. Find the diskette marked "CODOS 2.0 Distribution Disk" which should be in a pocket in the front of this notebook.

6. Open the disk drive door by pushing the button to the right of the disk slot. Insert the diskette into Drive 0 (the left drive on MTU disk units) and close the door. Refer to the drawing below if you are unfamiliar with 8" diskettes:



7. The disk drive should immediately click and buzz for about 4 seconds while it loads several programs into memory. Then the display should clear and the following message should appear at the top of the screen along with a flashing cursor:

   ```
   CODOS 2.0 (C) 1981 MTU
   PLEASE ENTER DATE (EXAMPLE:04-JUL-76)?=
   ```

8. If absolutely nothing happened when the door was closed, reach over and press and hold the MOD keyboard key. While holding MOD down, press RESET, hold for one second, and then release RESET while continuing to hold MOD down. When you hear the disk unit click, you may release the MOD key. If nothing happens within 3 seconds of releasing RESET, refer to "In Case of Trouble".

9. If the disk did click when the door was closed but then stopped without displaying the CODOS signon message, open the disk drive, remove the diskette, shake it a couple of times to reposition the disk inside the jacket, and re-insert it into the drive. If nothing happens at this point, go back to step 8.

10. Type in today's date as shown and then press Carriage Return. You may use the BACKSPACE key correct errors you spot before pressing Return.

11. The MTU-130 should respond with: "CODOS>". Now go to chapter 2 for some initial housekeeping chores and then on to enjoy your new MTU-130 system!

IN CASE OF TROUBLE

If you did not experience any problems while assembling and starting up your system, you need not read this section; go on to Chapter 2. If you did have problems, thoroughly try the suggestions below before calling MTU.

1. Pilot light fails to light when power is applied.

A. Make sure the MTU-130 is plugged into a live outlet. If not absolutely positive, check with a lamp or other electrical appliance.

B. Remove the fuse on the MTU-130 back panel and examine the element. If any break at all is visible, insert the spare fuse included with the computer (don't confuse with the disk unit spare fuse) and try again. If the fuse blows a second time, verify that the line voltage is between 105 and 125 volts 50 or 60Hz (USA machines) or between 210 and 250 volts 50 or 60 Hz (European machines). The MTU-130 will not operate on 25Hz. If no cause for fuse blowing is found, there may have been internal shipping damage.

C. If the fuse is intact, turn the MTU-130 on again and wait a few seconds. If the monitor lights up then the MTU-130 is definitely getting power. If an image (other than just a raster) appears, try to complete the monitor adjustments as given.

2. Cannot achieve horizontal synchronization.

A. The horizontal hold control may not have sufficient range to lock to the MTU-130's slightly higher than normal horizontal sweep frequency. To correct this, you will have to adjust the Horizontal Frequency control inside the monitor. You can get a TV technician to do this for you (tell him that you want a 16.129KHz sweep) or you can do it yourself if you can locate the control (most likely to be a tuning slug in a coil). First count the number of the darkest diagonal bars (or listen to the pitch of the monitor's squeal) then turn the monitor off and turn the slug 1/4 turn (try counter-clockwise first). Then turn the monitor back on and count the diagonal bars again. If there are fewer bars (or the pitch is higher), then you turned in the right direction. Try the horizontal hold again to see if synchronization can be achieved and perform another iteration of the whole procedure if not.

B. Check that the video cable has been properly connected at both ends and is not loose.

C. Verify that your monitor does indeed accept composite video with negative sync, not modulated RF or something non-standard.

D. If there is a control marked Line Level or something similar, try rotating that in conjunction with vertical and horizontal hold controls.

3. No disk unit response to Reset key.

A. Make sure the diskette was inserted properly. The jacket edge closest to the oval hole should go into the slot first and the label on the diskette should face the movable part of the disk drive door.

B. Make sure the disk unit power cord is properly plugged into the MTU-130 and is receiving power.

C. Turn the computer off and remove the fuse on the disk unit back panel and examine the element. If any break at all is visible, insert the spare fuse included with the disk unit (don't confuse with the computer spare) and try again. If the fuse blows a second time, verify that the line voltage rating of the disk unit is the same as that of the computer. The MTU disk unit will not operate on 25Hz. If no cause for fuse blowing is found, there may have been internal shipping damage.

D. If the fuse is intact and you heard motor noise from the disk unit, check that the large flat ribbon cable has been properly installed.

4. Operating system software will not load.

A. Make sure the frequency rating of the disk unit (either 60Hz or 50Hz) matches the power line frequency in your area. If the frequency does not match, you will have to obtain matching motor pulleys for the disk drives.

B. Try the Demonstration diskette (also in a pocket in the system notebook) to see if it can be read any better. If not, it is likely that disk drive 0 was knocked out of alignment during shipping. Realignment usually requires return to the factory.

CHAPTER 2

FIRST TIME SYSTEM OPERATION

In this chapter you will be instructed to read the System Identification Numbers, fill out the Registration Card, and copy the CODOS Distribution Disk. These are necessary operations to properly register your system and protect yourself from loss or damage of the operating system diskette.

READING THE SYSTEM IDENTIFICATION NUMBERS

It is assumed that you have just turned the MTU-130 system on and have successfully loaded the CODOS operating system into memory as described in the previous chapter. On the screen you should see:

```
CODOS 2.0 (C) 1981 MTU
PLEASE ENTER DATE (EXAMPLE:04-JUL-76)?=dd-mmm-yy

CODOS>
```

where dd-mmm-yy represents the date you typed in earlier. A flashing cursor should be immediately to the right of the >. Now perform the following to read the System Identification Numbers:

1. Press the key marked CAPS LOCK in the lower left corner of the keyboard so that it clicks and stays down. Since all system commands and most programming use upper case letters, you will usually want the CAPS LOCK mode. Note that CAPS LOCK only affects the letters; you still must use the SHIFT key to get the special characters above the number keys.

2. Type the following exactly as shown below:

   CPUID

   You may use the backspace key to correct any typing errors. When you are sure you have spelled it correctly, press the RETURN key.

3. Immediately the disk unit should click and then the display will show:

   VENDOR NUMBER - 00000 _____

   GROUP NUMBER - 00000 _____

   USER NUMBER - 0034D _____

   Of course the letters and numbers you see will be different but each one will be 5 characters. All zeroes for the Vendor and Group numbers is normal.

4. Get the registration card (bound in the second page of this notebook) and write these numbers in the blanks. Also write them in the blanks above so they will be readily available when you need them.

5. What you have really done is give the CODOS operating system a command to read the program called "CPUID" from disk and run it. The program in turn read your unique system identification numbers from an internal register and printed them on the screen. The CODOS> prompt and cursor indicates that the CPUID program has completed and that the system is ready for another command.

DUPLICATING THE DISTRIBUTION DISK

The diskette you have been using so far is referred to as the "Distribution Disk". Besides the CODOS operating system and the CPUID program, it has recorded on it dozens of other valuable programs. Unfortunately diskettes, like records and tapes, are subject to damage from accident or misuse. Therefore it is strongly recommended that you immediately make a copy of this valuable diskette using the instructions below. Not only will you be protecting yourself from loss, you will be getting acquainted with some of the fundamental CODOS operating system commands.

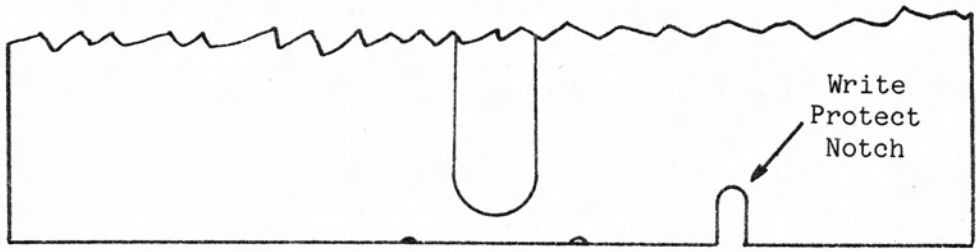
IMPORTANT - If your system has only one disk drive, skip to the next section.

Instructions for Systems With 2 or More Disk Drives

1. The console display should be showing the CODOS½ prompt and a flashing cursor. If not, follow the instructions in Chapter 1 for connecting and powering up the system.

2. You will need an 8 inch soft sectored diskette to make the copy. For minimum error incidence and maximum reliability, diskettes rated for double density are preferred. Satisfactory results can be obtained however with any high quality diskette. If you have an MTU-130-2D system, you may use a double-sided diskette for the copy if you wish.

3. If the new diskette (not the Distribution Disk) has a write protect notch (illustrated below), cover it up with a write permit tab (usually supplied with boxes of diskettes) or with opaque tape to permit writing on this particular diskette. Now insert the diskette into disk drive 1.



4. Type in the following command (be sure to type in upper case):

   FORMAT

   and then a carriage return when you are sure it is spelled right.

5. Disk drive 0 will be accessed momentarily and then the following warning will be displayed:

   ```
   WARNING: FORMAT WILL IRREVOCABLY
   ERASE EVERYTHING ON DISK IN DRIVE 1.
   ARE YOU READY (Y/N)?=
   ```

6. Verify that the disk you want to copy to is in drive 1 and then type a Y and a carriage return. If you want to abort the copy at this point for any reason, type an N (or anything else) and a carriage return.

7. For the next 30 seconds (1 minute for double-sided diskettes), new timing marks will be written onto the disk in drive 1. After this, the following message will be displayed:

   ```
   WANT TO TEST FOR BAD SECTORS (Y/N)?=
   ```

8. Reply Y and carriage return in order to thoroughly test the copy diskette. Testing will take about 3 minutes to complete (6 minutes with double sided diskettes). After you have gained experience with the brand of diskettes you have selected and the system in general, you can omit this testing step.

9. You will now be asked:

    DISK VOLUME SERIAL NO. (VSN)?=

    Type in the 4 digit number you wish to identify this diskette by. Although not required, it is often desirable to use a different number for each diskette in your collection.

10. The next question is:

    WANT TO COPY DRIVE 0 SYSTEM (Y/N)?=

    Since the purpose here is to make a complete copy of the Distribution Disk, type a Y and carriage return. Later on when you are formatting new disks for use with the MTU-130 you may want to maximize the diskette's capacity by not copying the operating system (you gain only about 5% though and such a disk can not be used in disk drive 0).

11. Both disk drives will now show considerable activity as the CODOS operating system program and several other essential files are copied from drive 0 to drive 1. You will see the CODOS ⟩ prompt when this "first round" of copying is complete.

12. Now type in the command COPYF and a carriage return. The name of each file on the Distribution Disk will be displayed and then the copy disk in drive 1 will be checked to see if that file already exists (i.e., copied there by the FORMAT program). If it does not exist, then it is copied. There are ways (described in Chapter 3 and in the CODOS manual) to instruct the COPYF program to only copy one file or a group of related files but here you don't specify so all of the files are copied.

13. Your copy of the CODOS Distribution Disk is now complete. We will refer to this copy disk later on as the Secondary Master. Now enter the following command:

    CLOSE 0 1

    This tells the CODOS operating system that you are about to remove the diskettes in drives 0 and 1. You must always inform CODOS of your intention to remove a diskette so it can make sure that the diskette is properly updated with information that may still be in memory.

14. Prepare a label for the Secondary Master diskette and then attach it. It is good practice to write on the label <u>before</u> attaching it to avoid possible damage to the diskette by writing pressure. Put the Distribution Disk in a safe place. You may never need it again but it is nice to know it is available in case you do need it.

15. Now turn to the section titled "Using the Demonstration Diskette" to become aquainted with the many advanced features of the MTU-130. Or if you prefer, you can go directly to one of the "Getting Aquainted" chapters.

Instructions for Systems With Only 1 Disk Drive

Unlike many disk operating systems, CODOS allows effective system usage with only one disk drive. Copying from one diskette to another however is not as convenient as it is on a multiple drive system. Since the task to be performed here is making a complete copy of the CODOS Distribution Disk which has quite a large number of files, the process may seem tedious. You will find however that virtually all other disk operations on the MTU-130 are nearly as convenient on a single drive system as they are on a multiple drive system.

1. Since you will be handling the diskettes a lot more than in a multiple drive system, please read over the first section of Chapter 3 which describes how to properly care for diskettes.

2. The console display should be showing the CODOS⟩ prompt and a flashing cursor. If not, follow the instructions in Chapter 1 for connecting and powering up the system.

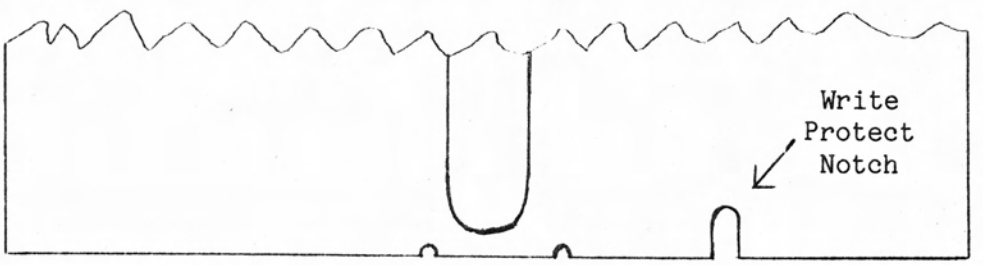
3. The version of CODOS on the Distribution Disk is set up for two disk drives. Before it will format a disk properly on a single drive system, CODOS must be modified for single drive operation. Type in the following commands exactly as they appear with a carriage return at the end of each line:

```
UNPROTECT
SET E74F 01
SET E7C2 0 10 0 A0
PROTECT
```

   These commands have temporarily modified CODOS in memory for single drive operation. In a later step you will be instructed how to permanently modify CODOS on disk.

4. You will need an 8 inch soft sectored diskette to make the copy. For minimum error incidence and maximum reliability, diskettes rated for double density are preferred. Satisfactory results can be obtained however with any high quality diskette. If you have an MTU-130-1D system, you may use a double-sided diskette for the copy if you wish.

5. If the new diskette (not the Distribution Disk) has a write protect notch (illustrated below), cover it up with a write permit tab (usually supplied with boxes of diskettes) or with opaque tape to permit writing on this particular diskette.



6. Now, with the Distribution Disk in the disk drive, type in the following command (be sure to type in upper case):

```
FORMAT
```

   and then a carriage return when you are sure it is spelled right.

7. Disk drive 0 will be accessed momentarily and then the following warning will be displayed:

```
WARNING: FORMAT WILL IRREVOCABLY
ERASE EVERYTHING ON DISK IN DRIVE.
PUT DISK IN DRIVE.
ARE YOU READY (Y/N)?=
```

8. Remove the Distribution disk, put it in its jacket, and set it aside. Then insert the diskette to be copied to into the drive and close the door. Finally type a Y and a carriage return. If for any reason you want to abort at this point, type an N (or anything else) and a carriage return.

9. For the next 30 seconds (1 minute for double-sided diskettes), new timing marks will be written onto the disk. After this, the following message will be displayed:

```
WANT TO TEST FOR BAD SECTORS (Y/N)?=
```

8. Reply Y and carriage return in order to thoroughly test the copy diskette. Testing will take about 3 minutes to complete (6 minutes with double sided diskettes). After you have gained experience with the brand of diskettes you have selected and the system in general, you can omit this testing step.

9. You will now be asked:

```
DISK VOLUME SERIAL NO. (VSN)?=
```

10. Type in the 4 digit number you wish to identify this diskette by. Although not required, it is often desirable to use a different number for each diskette in your collection.

11. CODOS will now print the following message:

```
TO COPY SYSTEM
PUT SOURCE DISK IN.
ARE YOU READY (Y/N)?
```

In the following dialog, the "SOURCE" disk is the CODOS Distribution Disk and the "DEST." disk is the copy disk. Therefore, remove the copy disk, and lay it aside in its jacket, insert the Distribution Disk, and type Y.

12. The disk drive will click momentarily and then CODOS will print:

```
PUT DEST. DISK IN.
ARE YOU READY (Y/N)?
```

13. Swap disks again and type Y. Never let a bare diskette touch the tabletop; either stand it upright or slip it immediately into its jacket.

14. The disk will click again and then ask you to put the source disk in. This will be repeated about a dozen times until the CODOS operating system and the essential system files have been copied. When this first round of copying is complete, CODOS will print:

```
NEW DISK IS NOW OPEN.
CODOS >
```

15. To see which files have been copied by the FORMAT program, type in FILES and a carriage return. You should see (the order may be different):

```
CODOS.Z
COMDPROC.Z
SYSERRMSG.Z
SVCPROC.Z
STARTUP.J
DIR.C
IODRIVER.Z

CODOS>
```

16. To copy the remainder of the files from the Distribution disk, type in CLOSE and a carriage return. Now swap disks so that the Distribution Disk is in the drive and type an OPEN command (all CODOS commands are followed by a carriage return).

17. Now type in the command FILES to see a listing of all of the files on the distribution disk. Since there may be more file names than will fit on the display at once, hold down the CTRL key and type an S to temporarily halt the display when file names have nearly filled the screen.

18. Write down all of the file names you see that are not in the list above. To resume the files listing (if you halted it with CTRL/S), type any character. When finished, you will have a list of all of the files you still need to copy.

19. Type in the command COPYF1DRIVE and a carriage return. you will see the message:

```
PUT SOURCE DISK IN
FILE (OR CR IF DONE)?=
```

Again, the source disk is the Distribution Disk and the destination disk is the copy. Now for each file on the list do the following:

20. A. Enter a file name from your list. You can use backspace to correct typing errors. When the file name is right, type a carriage return.

B. The disk drive will click momentarily and then the message

```
PUT DEST. DISK IN,
CR WHEN READY.?=
```

will be displayed. Swap diskettes so the copy diskette is in the drive and type carriage return.

C. The disk drive will click several times and the message

```
PUT SOURCE DISK IN
FILE (OR CR IF DONE)?=
```

will be displayed. Swap the diskettes back and check off the file name on your list. Now go back to step 20-A and repeat until all of the files have been copied.

21. When there are no more files to copy, simply enter a carriage return in response to the request for a file name. The Distribution Disk should be in the drive at this point.

22. Now enter the command CLOSE and remove the Distribution disk from the drive. When under control of CODOS (not the FORMAT or COPYF1DRIVE programs), CODOS must always be informed of your intention to remove a diskette so it can make sure that the disk is properly updated with information that may still be in memory. Put the Distribution Disk in a safe place. You may never need it again but it is nice to know it is available in case you do need it.

23. Put the copy disk in the drive and enter the command: OPEN. Now enter the command: FILES and check the displayed list against your written list to be sure that no files were omitted.

24. These final steps will permanently modify CODOS on the copy diskette for single disk drive operation. These steps (as well as step 3) will not be necessary for subsequent duplications of this disk.

25. Enter the command: SYSGENDISK. You will see the following displayed:

```
THIS UTILITY MODIFIES DISK ATTRIBUTES
FOR CODOS ON DRIVE 0 DISK
DEFAULTS SHOWN IN ( ).
WANT TO PROCEED (Y)?=
```

26. Type a Y and a carriage return. The program will respond:

```
# OF DRIVES (2)?=
```

27. Type a 1 and a carriage return. The program will now ask:

```
# DISK BUFFERS (6)?=
```

28. Simply enter a carriage return unless you wish to change the number of disk buffers (see CODOS manual Chapters 4 and 10). The program will now ask:

```
TRACK STEP RATE ($08 MS)?=
```

29. Simply enter a carriage return unless you wish to change the disk drive stepping speed (see CODOS manual Chapter 10). The program will then ask:

```
HEAD LOAD TIME ($32 MS)?=
```

30. Simply enter a carriage return unless you wish to change the head load time (see CODOS manual Chapter 10). The disk will click the program will display:

```
SYSTEM MODIFIED.
SUGGEST YOU LOCK CODOS.Z

CODOS⟩
```

31. Enter the command: LOCK CODOS.Z which will prevent you from inadvertently deleting the CODOS Operating system.

32. Your copy of the CODOS Distributon Disk is now complete. We will refer to this copy disk later on as the Secondary Master. Prepare a label for the Secondary Master diskette and then attach it. It is good practice to write on the label <u>before</u> attaching it to avoid possible damage to the diskette by writing pressure.

33. Now turn to the section titled "Using the Demonstration Diskette" to become acquainted with the many advanced features of the MTU-130. Or if you prefer, you can go directly to one of the "Getting Acquainted" chapters.

USING THE DEMONSTRATION DISKETTE

Besides the CODOS Distribution Disk, you have been supplied with a separate Demonstration Disk. On this diskette have been recorded a number of programs designed to show off various aspects of the MTU-130 system. While few of these demo programs have any significance as applications themselves, they do serve to illustrate system hardware capabilities, the manner in which they may be utilized, and the associated programming techniques. Use of the Demonstration Disk is only summarized here. More detailed information about many of the demonstration programs may be found in the Demonstrations and Games section of the System Notebook. Some of the demo programs are written BASIC. BASIC, IGL, CIL and VGL are already on the demo diskette for those who receive BASIC. The demo disk is configured to run an NEC8023A printer. You must rename the appropriate version of SPRINT to SPRINT.Z in order to use it on the demo diskette. The VMDUMP utility may be left as is or renamed to VMDUMP.C.

Loading the Demonstration Diskette

If the MTU-130 system power is off, you can insert the Demonstration Disk into disk drive 0 and switch the power on to automatically load the demonstrations. If you have been running CODOS previously with another disk in drive 0, you should first close the disk you have been working with by entering ing the command: CLOSE and a carriage return. Next remove the disk from drive 0 and insert the Demonstration disk. Now press the RESET and MOD keys simultaneously, hold them for a full second, then release RESET while continuing to hold MOD down. After disk activity starts, you can release the MOD key. In either case, once the loading process is finished, you will be able to run the various demos by simply pressing function keys.

The Main Menu

Regardless of the startup method used, you will be presented with a function key menu at the bottom of the screen similar to that shown below:

AUTODEMO FONTEDIT GRAPHICS PICTURES MUSIC SPEECH GAMES PRINT

Each box and word corresponds to one of the 8 function keys on the topmost row of the keyboard. When you press a function key, either a program will be executed or you will be presented with a "second level" menu as described below:

AUTODEMO - Starts an automatic sequence of demonstrations that will continue to run until the INT key is pressed.

FONTEDIT - Permits the user to alter the font table used for display of characters on the CRT monitor.

GRAPHICS - Presents another menu of graphics oriented demonstration programs.

PICTURES - Presents another menu of choices of digitized high resolution images.

MUSIC - Presents another menu of musical selections.

SPEECH - Presents another menu of phrases you may play through the speaker.

GAMES - Presents another menu of choices for game programs.

PRINT - Presents another menu of printer demonstrations. The Demostration disk contains a copy of PRINTDRIVER.Z which is automatically loaded. This printer driver is suitable for a printer with a parallel interface accepting a positive strobe pulse and returning an active high busy. Refer to Section 10 in the CODOS manual for instructions on preparing the proper cable.

GRAPHICS Menu

The GRAPHICS function key will present a menu of graphics related demonstration programs as shown below:

SWIRL CHECKER GRAYCHEK LIFE LITE PEN VGLDEMO GRF EDIT EXIT

Note that MTU BASIC and the standard libraries will have to be copied from the appropriate distribution disk onto the Demonstration Disk before the "LITE PEN", "VGLDEMO", and "GRF EDIT" selections will work.

SWIRL - Plots a series of 10 spiral and spiderweb-like patterns. The parameters that influence the shapes are determined randomly. CTRL/S will temporarily suspend the sequence, CTRL/Q will restart it, and CTRL/C will abort it and re-display the graphics menu. Note that the keyboard is only tested between patterns.

CHECKER - Displays a series of 10 checkerboard patterns. The parameters that influence the height and width of the squares are determined randomly. CTRL/S will temporarily suspend the sequence, CTRL/Q will restart it, and CTRL/C will abort it and re-display the graphics menu. Note that the keyboard is only tested between patterns.

GRAYCHEK - Similar to CHECKER except the gray scale display mode is used. You may have to adjust the monitor's black level and contrast controls to clearly see the 3 levels of gray and black.

LIFE - Performs 20 generations of Life on whatever is presently on the screen. It is most interesting if a checkerboard pattern with moderately large squares (from 5 to 15 accross the screen) is on the screen. CTRL/S will temporarily suspend the sequence, CTRL/Q will restart it, and CTRL/C will abort it and re-display the graphics menu. Note that the keyboard is only tested between patterns.

LITE PEN - Loads and runs the VGLPENDRAW BASIC program. The program will fill the screen with white (or green) and wait for you begin drawing with the light pen. While drawing, you may pull the pen away from the screen and begin drawing in a new location. If you pull the pen away for more than 5 seconds, light pen input will stop. The screen will be cleared and a new menu is displayed. The selection give you a choice of what size to redraw what you drew with the light pen. The F8 key returns you to CODOS. Execute the CODOS command: DO GRAPHICSMENU.J to bring back the GRAPHICS menu.

VGLDEMO - Loads and runs the VGLDEMO BASIC program. This program demostrates some of the capabilities of the VGL Library. The program will draw a box and an image of the top of the MTU computer within the box. Pressing the F1, F2, and F3 keys will cause the box and image within it to change in size or orientation. The F5, F6, F7, and F8 keys will cause size of the computer image within the box to change in size. The box does not change. Pressing the F4 key will return you to CODOS. Execute the CODOS command: DO GRAPHICSMENU.J to bring back the GRAPHICS menu.

GRF EDIT - Loads and runs the GREDIT BASIC program. This program provides a simple graphics data base editor. Instructions on its operation may be found in the Demonstrations and Games section of the System Notebook.

EXIT - Will return to the main menu display.

PICTURES Menu

The PICTURES function key will present another menu of choices for pre-digitized pictures. Each picture is simply a file of data on disk that loads directly into the display portion of MTU-130 memory. As the display screen represents 15K of memory, the loading process is a graphic illustration of the speed of the MTU-130's disk system.

HATLOGO GRAPH FACE WALLPAPR VMREV VMDUMP EXIT

HATLOGO - An overlay of MTU's logotype and a solid of revolution math plot.

GRAPH - An example of application oriented plotting with dashed lines.

FACE - A digitized photograph. Grey scale is represented by varying densities of black and white dots.

MTU-130 - Another digitized photograph.

WALLPAPR - An example of the wallpaper-like designs that may be created by running LIFE on a checkerboard pattern (see Graphics menu).

VMREV - Produces a negative image of whatever is on the screen. Useful prior to printing a digitized photograph.

VMDUMP - Prints out current screen image dot for dot on a printer.

EXIT - Will return to the main menu display.

MUSIC Menu

The MUSIC function key will present menu of musical selections. Note that there may be a delay of several seconds between key depression and when sound is heard. There is a volume control on the MTU-130 rear panel. Use of a large external speaker or amplifier and speaker will materially improve the sound quality over that obtained with the built-in speaker.

SPEECH Menu

The SPEECH function key will present a menu of of spoken phrases. Each phrase is actually a file of digitized speech which is read from disk into memory and then played through the digital-to-analog converter. This process is really more like playing a digital recording than speech synthesis but the results are the same. A hardware/software package will become available that will not only perform the playback more efficiently than is done in this demo but will also allow you to produce your own vocabulary as easily as making a tape recording.

GAMES Menu

The GAMES function key will present a menu of available games and diversions. Details on playing these games are found in the Demonstrations and Games section of the System Notebook. Please note that games written in BASIC require that MTU BASIC be copied from your Secondary Master disk onto the Demonstration Disk.

PRINT Menu

The PRINT function key will present a menu of print related functions. Note that you must have installed a text printer driver program for the type of printer on your system before this key will function properly (see CODOS 2.0 manual Chapter 10). The name of the text printer driver should be PRINTDRIVER. If the printer is capable of bit graphics output, the graphics driver should be named VMDUMP (see the Utility Programs manual).

SCREEN FILES BLURB EXIT

SCREEN - Will print in dot graphics from whatever is on the screen.

FILES - Will print a listing of all files on the Demonstration Disk.

BLURB - Will print a news release announcing the MTU-130.

EXIT - Will return to the main menu.

What System Files Should Be Kept on a Working Disk?

The MTU-130 Distribution Disk contains about 60 files. Chapter 2 of the Setup and Installation Manual explains how to make a backup copy of this disk. However, you will NOT want to keep all (or even most) of these files on your "working disks" which you use for day to day operations. Most of these files will seldom if ever be needed. After you have used your MTU-130 for a while it will become clear what files you will want to keep on your working disks. Initially though, this will not be obvious. We suggest you start with the following as a "working set" of files, and prune or add to the list as you develop confidence with the system:

CODOS.Z	COMDPROC.Z	SYSERRMSG.Z	SVCPROC.Z	STARTUP.J
DIR	IODRIVER.Z	GRAPHDRIVER.Z	FORMAT	BACKUP
COPYF1DRIVE (1 drive system)			COPYF (2drive system)	
KILL	EDIT			

Assuming you have BASIC, you will also want:

PRINTDRIVER.Z and SPRINT.Z

If you have the optional Assembler/Disassembler, you will also want:

ASM and DISASM

This configuration (the 23 files above) will give you about 360K of free space on a single-sided disk or about 820K on a double-sided disk.

CHAPTER 3.

GETTING AQUAINTED WITH CODOS

The heart of MTU-130 system software is the Channel Oriented Disk Operating System, usually called CODOS. The "channel oriented" part of the name derives from the way that data flow among input/output devices is performed. The "disk operating system" part of the name indicates that CODOS is a program for operating the entire system and that it uses disks for its permanent storage of data. Thus CODOS is the "central command" program that ties user programs and all other programs in the system together into one smoothly functioning unit.

Very briefly, CODOS performs the following functions:

1. Operates the input/output devices of the MTU-130 such as the disk drives, keyboard, and display.

2. Accepts and interprets commands from the user to run programs; create, delete, and copy disk files; and direct the flow of data among the various input/output devices connected to the system.

3. Maintains an index of data and programs stored on diskettes.

All other programs in the system including the MTU BASIC interpreter, the Editor, the Assembler, and all user programs whether in machine language or high level language are subservient to CODOS. By subservient it is meant that all of these programs are called into action by CODOS commands and that they use the facilities of CODOS to accomplish the three function classes listed above.

In the sections to follow you will become familiar with the most frequently used CODOS commands and functions. That which is covered will be only the tip of the iceberg, however, since CODOS has many more commands and capabilities than can be described or indeed absorbed in one sitting. The CODOS 2.0 manual contains descriptions of these other commands as well as more complete descriptions of the ones that will be covered here. It is strongly recommended, however, that even experienced computer users read and do the experiments in this section in order to become familiar with the "philosophy" of CODOS 2.0 and indeed the entire MTU-130 system. Many things are done differently (and we think better) than on conventional systems so even experts will find something to learn.

CARE AND HANDLING OF DISKETTES

Full sized (8 inch) floppy disks are used for storing programs and data in the MTU-130 system. Consequently, the system is easy to use and has large storage capacity compared to systems using tape cassettes or mini-disks for storage. However, the half-million or one million character storage capacity (equivalent to a typical paperback novel) means that any given disk may contain a great deal of valuable data. Responsibility for the safety of this data rests with the user. The way floppy disks are handled and stored will materially affect their lifetime and reliability. To insure that you receive the reliability and performance the MTU-130 system is capable of, follow the rules below religiously:

1. Always keep the diskette in its protective envelope. Get in the habit of removing a disk from the drive directly to the paper envelope. Dust particles look like a boulder to a recorded bit!

2. Do not touch the exposed recording surface of the disk. Fingerprints are a killer, too.

3. Do not bend the disk. It's called a flexible disk, but you may damage it if you try to prove it!

4. Do not write on the disk directly with pen or pencil. Use only a soft-tip marker, and write only in the label area, or you may damage the magnetic surface underneath. Better yet, write on the label while the backing paper is still in place and then apply it to the diskette.

5. Avoid exposure to harsh environments such as extreme heat or cold. Storage in a locked car on a hot day is a killer!

6. Keep the diskette away from strong magnetic fields such as speakers and magnetic note hangers (the top surface of the disk cabinet or monitor is OK however).

7. Keep frequently used diskettes in the storage area of the MTU disk cabinet and infrequently used ones in the boxes they were shipped in. Diskettes strewn about the tabletop are subject to damage from heavy books, dropped items, and coffee spills.

8. The X-ray machines and metal detectors used at airports are <u>not</u> a hazzard to diskettes. They are safer in your briefcase than they are in checked luggage which may be subjected to rough handling, temperature extremes, and low atmospheric pressure.

If you follow these simple procedures, don't be surprised if you <u>never</u> get a read/write error. Other disk systems don't always work like that, but we're sure you won't mind doing without the disk errors!

WHAT KIND OF DISKS SHOULD BE USED?

Any <u>quality</u> soft-sectored 8 inch floppy disk may be used. We recommend Dysan double density disks for maximum data integrity (type 3740/1D for single-sided drives or type 3740/2D for double-sided drives). However, satisfactory results can usually be obtained even with diskettes rated only for single density if they are of good quality, due to the exceptionally high-quality data separator used in the MTU-130 double-density controller, and the automatic error recovery software built into CODOS. The FORMAT Utility will automatically record the proper double density timing marks on any soft-sectored disk.

GENERAL INFORMATION ABOUT FLOPPY DISKS

Floppy disks are normally sold in boxes of 10, and can be purchased from almost any computer or business supply house. Figure 3-1 illustrates the important parts of the floppy disk, which are:

1. Manufacturer's permanent label.

2. User's label and MTU Copyright notice. These blank labels are provided in several colors with the box of disks. Fill out the label <u>before</u> affixing it to the disk, and be sure to include the CODOS Copyright notice on all disks which are to contain a copy of the system.

3. Index hole. There is a hole in the disk surface and a hole in the jacket. While the disk rotates in its jacket inside the drive, a beam of light shines through the hole. Once each revolution the holes line up and the light passes through, triggering timing circuits in the drive. Double-sided disks have their index hole slightly further off center than single-sided disks, as shown in Figure 3-2. This permits the system to automatically identify double-sided disks. Soft-sector diskettes have only one index hole in the disk itself, but hard sectored disks (which cannot be used with the MTU-130) have 33.

4. Drive spindle hole. When the disk is in the drive, the drive inserts the spindle into this hole and clamps on the exposed disk surface, spinning the disk inside its jacket.

5. Head slot. This portion of the disk is exposed for access by the read/write head.

6. Write-protect notch. This notch should be covered with a small gummed label provided with the box of diskettes in order to write on the disk and should be removed to prevent writing on the disk. Don't use ordinary masking tape for this purpose as it is transparent to infrared light. Not all diskettes have this notch. (You can punch your own with a handheld hole punch using the Distribution Disk as a position and depth guide).

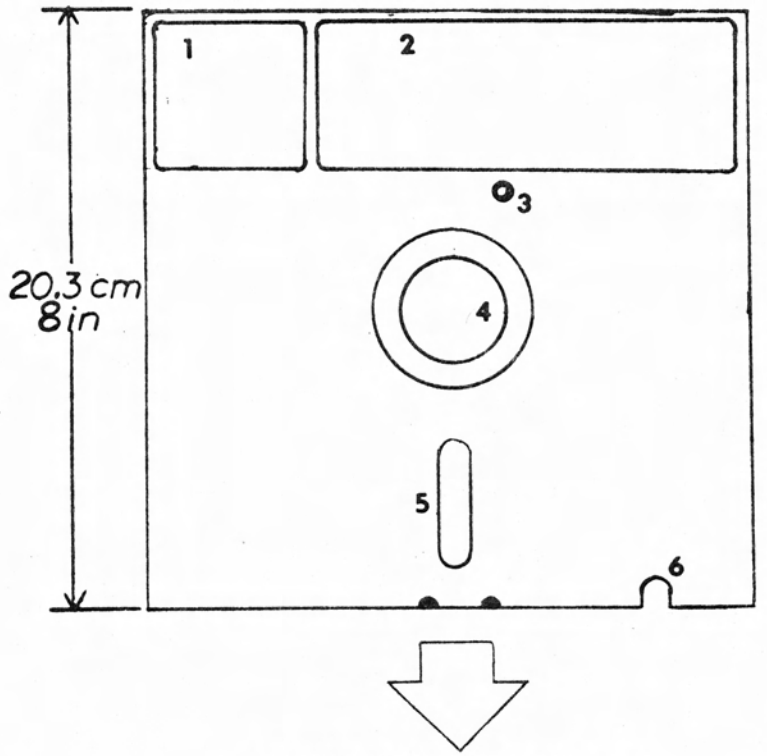


FIGURE 3-1. FLOPPY DISK

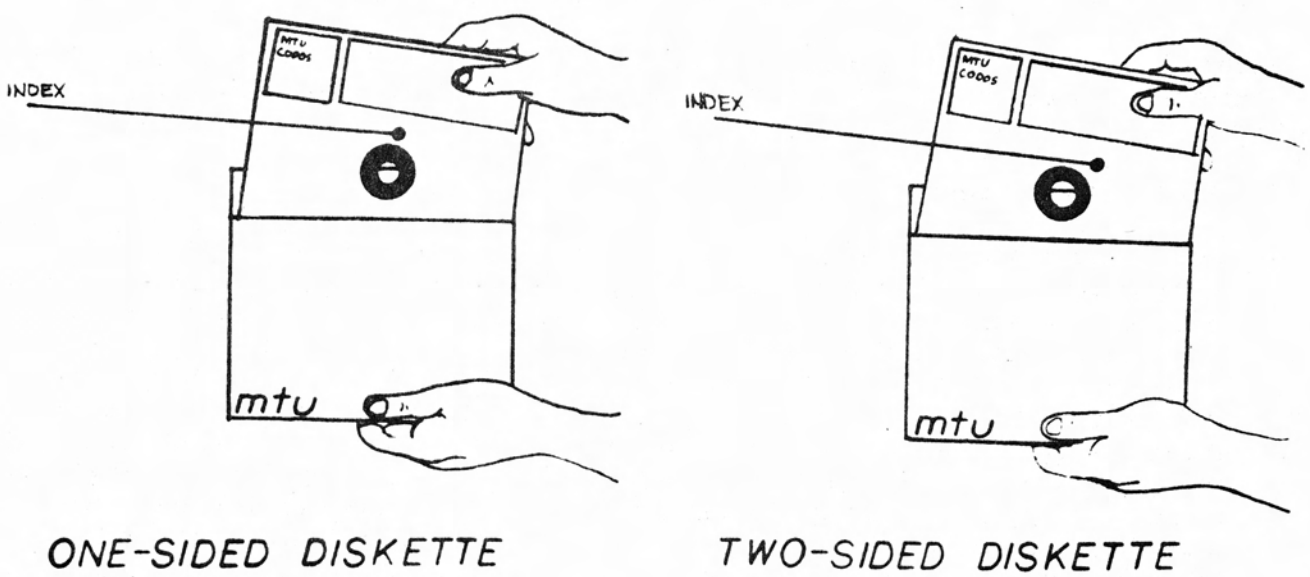


FIGURE 3-2. DISTINGUISHING DOUBLE-SIDED DISKS

FILES, NAMES, AND THE DIRECTORY

FILES

Information is stored on floppy disks in units called files. A file may be a program, a quantity of text, an array of numbers, or a block of specialized binary data. The best way to visualize diskette data storage is to consider the diskette as a drawer in a file cabinet. Into this drawer are placed individual file folders which correspond to the files on the disk. Inside each folder may be a list of computer instructions (a program), manuscript pages (text file), columnar pages of numbers (a numerical array), or perhaps some kind of compactly encoded information (the binary data). A file folder may even have different kinds of information mixed such as text and numbers.

In CODOS there is really only one kind of file; the light brown manila type. We may frequently refer to text files or BASIC program files or loadable files but the folders are all the same. The distinction is the kind of data stored in the file. Actually, at the lowest level all file data looks alike to the computer, so the real distinction is made by the program that reads and writes the data. If the data in the file does not make sense to the program, then it is the wrong kind of file. Similarly, medical records written in Japanese would not make sense to a U.S. doctor but would make perfect sense to a Japanese-born nurse.

NAMES

Every file on the disk has a different name. This can be likened to the tab on a file folder. The file name has two parts. The main part can have from two to twelve characters. Less than 2 or more than 12 characters is an improper name and will be rewarded with an error message. Additionally, the first character of a file name must be a letter. The other characters must be either letters (upper case required), numbers, or the underline character (_). The reason for these restrictions is to prevent confusion between file names, device names, and channel numbers by the computer. Since the first part of the name can have a number of characters, it is usually used to describe the file contents in a way that is meaningful to the user.

A file name also has a second part called an extension because it extends the meaning of the first part of the name. The extension is a single character which must be either a letter or a number. All files have this second part but if you don't say, CODOS will assume a C. The extension is separated from the first part by a period. The extension is usually used to identify the kind of data in the file. For example, a .T identifies a file of text, a .B identifies a file containing a BASIC program, .Z identifies an operating system file, and .J identifies a file of CODOS commands. The .C, which you get when you don't say, usually identifies a file that contains a machine language program. CODOS does not enforce any particular convention for choosing the extension but the ones mentioned above are in common usage (see Table 2-3 in the CODOS manual for a complete list of standard extension names).

Listed below are some perfectly legal file names. When creating a file you can choose anything you want for a name as long as it obeys the rules outlined above but it should make sense to you and your associates.

A2	AR_MAY_3_81.D	DISKETTE.C
YANK.A	Z1.2	STARTUP.J
MY3RDFILE.3	OLD_XY_DATA.C	IGL.Z
OUR_STUFF.L	CODOS.Z	CPUID
Z0123456789Q.T	BASIC.Z	UPLOAD

The following is a list of illegal file names. Obviously you will never see an illegal name actually stored on a disk but you might accidentally try to instruct CODOS to create or find an illegal filename.

A.T	Only one character in first part.
3TODAY.J	Doesn't start with a letter.
March_1	Not all upper case letters.
PAY ROLL	Blanks are not allowed inside a file name.
PROGRAM#3.A	Character included that is not a letter or a number.
REALLYLONGNAME.3	More than 12 characters long.
PROGRAM1._	Underline not allowed as an extension.
PARTA,B	A period must be used before the extension.
DOIT.COM	Only single character extensions allowed.

At this point lets try an experiment with file names. It is assumed that the MTU-130 is turned on and that CODOS has been loaded from your Secondary Master disk and is showing the CODOS> prompt and a flashing cursor. Now type in the following:

```
REALLYLONGNAME.3
```

CODOS will respond with:

```
CODOS ERROR #0C
REALLYLONGNAME.3
'
MISSING OR ILLEGAL FILE NAME.
CODOS>
```

which we already know is the case. What you really did was to instruct CODOS to search the disk for a file called REALLYLONGNAME.3, interpret the data as a machine language program, then load the file's data into memory and run the program. With an illegal file name however CODOS didn't even get to first base. Now try typing in this illegal file name:

```
PAY ROLL
```

CODOS will respond with:

```
CODOS ERROR #01
PAY ROLL
   ^
COMMAND NOT FOUND.
CODOS>
```

This is a little different. The blank between PAY and ROLL caused CODOS to misinterpret the name as just PAY (with an extension of .C) which is a legal name. You can easily tell that this has happened because the "error pointer" is pointing to R in ROLL indicating that it has "successfully" interpreted PAY as a file name. It then searched the disk but could not find a file called PAY.C and responded with the "COMMAND NOT FOUND" message. Standard CODOS terminology refers to machine language programs as commands. Finally, try typing in this:

```
CPUID
```

CODOS will look for the file called CPUID.C, which should be on your Secondary Master, will find it, load it into memory, and run it. This program will then proceed to print out the identification numbers for your MTU-130. When done, it will return control to CODOS which is evidenced by the CODOS> prompt and cursor.

Perhaps the most important function of CODOS is keeping track of all of the file names on each disk in the system. This is done efficiently by keeping a list called the directory on the disk. Thus each diskette has a directory of the files on that disk, itself stored as a special CODOS file. In the directory is the name and location of each file. Also associated with the directory is the disk ID number and a map of free space on the disk. Other information about each file, such as its length and creation date, are kept with the file itself.

The simplest command for displaying the directory is the FILES command. Go ahead and enter the command FILES followed by a carriage return. You will see a simple but rapid listing of all of the file names on the disk on drive 0. Notice that the names don't seem to be in any particular order. Actually the listing is the order files appear in the directory which is approximately the order in which they were created. The FILES command is most useful for a quick glance at the entire contents of a disk. There are more powerful commands for listing closely related names and finding the exact spelling of a file name.

Now let's explore a more powerful command for listing the file names on a diskette. Enter the command DIR and a carriage return. You will now get a listing of not only the file names but the creation date, the file size, and the protection status as illustrated below:

```
                                          64K 4K  K
CODOS.Z          :0 L    11/8/81      $001883
COMDPROC.Z       :0 L    11/8/81      $00050C
STARTUP.J        :0 -    11/13/81     $0001D1
CPUID.C          :0 -    *UNDATED*    $000083
```

The :0 indicates that the file resides on the disk in drive 0 (all will in this example) and the L indicates that the file is locked to prevent inadvertant deletion. The date field tells when the file was written onto the disk. If it says *UNDATED* it means that the request for today's date was ignored back when the file was created. The rightmost column gives the exact file length in bytes in hexadecimal. If you can't read hexadecimal you can still get a quick idea about how big the file is by remembering that the third digit from the right is quarters of a K (1K=1024 characters), the fourth is 4K, and the fifth is 64K. Also the letter A stands for 10, B for 11, up to F for 15. Thus the CODOS.Z file is roughly 8 x 1/4K plus 1 x 4K = 6K long.

You probably noticed that the DIRectory listing was bigger than the screen so the lines that came out first just scrolled off and disappeared. You can easily halt the listing temporarily by holding down the CTRL key on the keyboard and then typing an S (for Stop). The listing will remain stopped until you hold down CTRL and type a Q. You can also prematurely terminate a program's execution (DIR is simply a program to list the directory) by holding CTRL down and typing a C. This will generally work with any program provided it is outputting new text to the display at the time CTRL/S or CTRL/C is typed.

The DIR command also has some convenience features that make it easy to find filenames, particularly if the directory has hundreds of names in it. Try entering the following command: DIR *.Z and a carriage return. You will see a listing of only those file names that have an extension of Z which are usually just the operating system files. Similarly, DIR *.J gives just those names with a .J extension. Now try DIR C* . You will get all of the file names that start with C. Or will you? Note that CODOS.Z was not listed even though it starts with C. In fact you just got those that start with C and have an extension of C. As you learned earlier in the discussion about file names, if you don't specify an extension, then .C is assumed. Now try DIR C*.* which should give all of the filenames that start with C regardless of their extension. Finally, try DIR COPYF. Since you have specified a complete file name (along with the assumed .C), you will get a listing of that one file name only if that file indeed exists on the diskette. Now try DIR QWERTY which won't list anything since there isn't a file called QWERTY.C anywhere on the disk.

Lets try to generalize from these experiments. Just a plain DIR without any arguments (arguments are additional qualifier fields such as a file name or drive number) will simply list all of the files on drive 0. If an argument is given, it is a file name pattern and all of the file names that match the pattern will be listed. An * in the pattern will match any string of characters of any length in file names. Seems simple enough doesn't it? Try this command: DIR *S.* . You may expect that all file names that end in S would be printed but in fact there are no matches at all. Unfortunately, the computer's totally logical mind matches the entire first part of the file name with the * and then doesn't have anything left over to match to the S. Therefore as a practical matter, you can only use the * at the end of the pattern.

Another character that can be used in file name patterns is the ?. The ? will match any single character. Try this command: DIR ???.? . You will get a listing of every file name that is exactly 3 characters long. Since the extension is always one character, it doesn't make any difference whether you use an * or a ? to specify match anything. Pattern matching has other capabilities too which are completely described in Chapter 4 of the CODOS manual.

Using DIR with a completely specified name (no pattern characters) will quickly tell you if that file is indeed on the disk. Use of pattern matching can quickly give you the exact name spelling when you only know the approximate spelling. With a little planning when choosing your file names coupled with the pattern match capability of the DIR command (and others), you can get CODOS to automatically do quite a bit of grouping and selecting of names automatically.

DIRECTORY MAINTENANCE

Like a file drawer, you will want to add new file folders, discard old ones, add information to existing files, discard old information in existing files, rename files, split and combine files, copy files, "lock" important files, and ascertain how much space is left in the drawer. CODOS has commands for doing many of these functions directly while the others can be done indirectly through a program.

The most fundamental command is the DISK command. Try typing in DISK and a carriage return. You will see displayed in parentheses the ID number of the disk in drive 0 and the amount of free space available for new files or additions to old ones. A "K" is a little over 1000 characters so 350K of free space is a lot of space. A completely empty disk has about 500K of free space while a typical disk with essential operating system files has about 450K free. If you are using double-sided diskettes, these numbers will be approximately doubled.

You can easily rename files as well. There should be a file on your disk called UPLOAD.C . Lets rename it to UPANDUPLOAD.C . Try typing in the command: RENAME UPLOAD UPANDUPLOAD and a carriage return. The disk will feverishly move its head for a second or so and the CODOS> prompt will reappear. Now type DIR UP* . You should see in the listing the new name UPANDUPLOAD.C but not UPLOAD.C. That's all there is to the RENAME command; just type RENAME, a space, the file name to be changed, a space, and then the name you want it to be changed to. Now before continuing, rename the file back to UPLOAD and verify that you have done it correctly.

One of things CODOS does very well is to make copies of files. Try entering the following command: COPYF DOWNLOAD EXPERIMENT1 . The disk will click for awhile and then stop. Now use DIR or FILES to see that a new file called EXPERIMENT1.C has been added to the disk and that DOWNLOAD.C is still there. The DISK command should tell you that there is slightly less free space than there was previously. The format of the COPYF command is similar to the RENAME command, just COPYF, a space, the file name to be copied, a space, and the name of the duplicate file. Although COPYF is usually used to make copies onto another diskette, it obviously can make a copy onto the same diskette as well. Before continuing, make another copy of DOWNLOAD and call it EXPERIMENT2.

Use DIR EXPERIMENT* to list all of the information about the new files you just created. Note that there is a dash right after the :0 in both files. This means that these files may be very easily erased by the DELETE or KILL commands to be described shortly. Now enter the command: LOCK EXPERIMENT2 and then do another DIR EXP* . You will now see an L after the :0 for EXPERIMENT2.C but EXPERIMENT1.C will still have the -.

Now lets try the DELETE command. Enter DELETE EXPERIMENT1 and a carriage return. In an instant the EXPERIMENT1 file is erased without warning and you are greeted with the CODOS> prompt. Now try deleting EXPERIMENT2. Since EXPERIMENT2 is locked, you will be chided with a LOCKED FILE VIOLATION message and the file will remain intact. Now unlock the file by entering UNLOCK EXPERIMENT2 and use DIR to see that it has indeed been unlocked. Finally, type in the command KILL EXPERIMENT2. This time CODOS will be nice enough to ask you if you really want to delete the file first. Reply with a Y to delete the file. In general you should always use the KILL command to delete files since it gives you a second chance. DELETE is provided only for people (or programs) who never make misteaks.

This covers the basic directory housekeeping functions provided by CODOS and are all that are really necessary. Combining, splitting, adding to and truncating from files are more advanced functions usually performed by application programs. See the Multiple Disk Drives and Shortcuts sections for additional commands that streamline routine directory maintenance and make use of multiple disk drives.

USING MULTIPLE DISK DRIVES

Although disk based computers can operate very effectively with just one disk drive, most systems will have two drives and some may have 3 or 4. If your system has only one disk drive it is still worthwhile to read this section even though you won't be able to perform the experiments.

In a multiple drive system, the disk drives are numbered starting with zero (0). Except when actually changing disks, you will always have a disk in drive 0 but the other drives may be empty such as they are now. In addition, the disk in drive 0 must always have a copy of the operating system files because many CODOS commands are loaded from disk rather than staying in memory all the time.

For the first experiment, insert the Demonstration disk into drive 1. Now enter the DISK command. Note that CODOS does not yet recognize that there is a disk in drive 1. You must inform it of this fact by entering the command: OPEN 1 and a carriage return. Now when you enter the DISK command, you will see the status of both disks. Enter this command: FILES 1 and a carriage return. You will see a listing of all of the demo disk files (and there are a lot of them!). Now enter FILES 0 to see the files on the secondary master which is still in drive 0. Thus the FILES command will accept a drive number argument which specifies which drive you want the files listing from. Furthermore, if you don't say, drive 0 is assumed. Such assumptions by CODOS are frequently handy, particularly after you become an experienced user but if in doubt, always say explicitly.

Before you remove a disk from a drive, you should inform CODOS first with the CLOSE command. Enter the command: CLOSE 1 and a carriage return. You may now remove the demo disk from drive 1. Now try the FILES 1 command again. You should get an error message telling you that the drive is not open. OPEN and CLOSE are used the same way when changing disks in drive 0 or any other drive for that matter. If you should ever forget to CLOSE a drive before removing the disk, always put the disk you failed to close back into the drive and give the CLOSE command before executing any more commands. Then pull it back out and continue with what you were doing. Most other systems require you to open and close every file you use; CODOS only requires you to open and close entire diskettes.

As a final experiment with OPEN and CLOSE, enter the command CLOSE 0 and remove the disk from drive 0. Now enter a FILES 0 command (or just about any other command). You will see an ERROR #03 message because the drive you wanted the files listing from is not open but there will be no English explanation of the error. The reason is that the error explanations are actually stored in a system file (called SYSERRMSG.Z), but since drive 0 is not open, the file cannot be read. You always want a disk with the operating system files to be in drive 0. Now reinsert the secondary master into drive 0 and open it before continuing.

For the next set of experiments, get another blank disk, put a write permit tab on it, and insert it into drive 1. Just for kicks, try to open drive 1. After about 10 seconds of feverish attempts to read the blank disk, an UNFORMATTED DISK OR IRRECOVERABLE READ/WRITE ERROR message will be given. With proper care of your diskettes, this is probably the only time you will ever see this message. Now enter the FORMAT command and answer all of the questions with Y. Get into the habit of assigning a unique volume/serial number to each disk and write it in the corner of the permanent disk label. In a system with two or more disk drives, FORMAT always uses drive 1; you have no choice. Note that FORMAT unconditionally erases everything on the disk. If it is an old CODOS disk rather than a fresh blank one, it still erases everything including any locked files.

FORMAT copies only the essential CODOS system files onto the new disk in drive 1. Many of the utility commands (programs) are not copied. Let's get a little more familiar with the COPYF command. First enter a FILES 1 command to see the files that are currently on the disk in drive 1. Now enter the command: COPYF SERLDR . Both drives 0 and 1 will click. Finally enter a FILES 1 command again which should show that SERLDR has been added to the disk in drive 1. This usage of the COPYF command is a little different than before because only one file name was specified. In this case COPYF assumes two things. First, it assumes that that you want the duplicate file to have the same name as the original. Second, it assumes that you want the duplicate on the opposite disk drive from the original. When you don't say which drive the original is on (we'll see how later), CODOS assumes it is on drive 0. Since drive 1 is clearly the opposite of drive 0, the duplicate was thus written onto drive 1.

COPYF will also accept patterns for file names just like DIR does. Try entering this command: COPYF *.Z . This will cause all of the .Z files on drive 0 to be copied onto drive 1. In addition, since a pattern was specified, COPYF gives a report of exactly what it has done. Notice that those files that were already on drive 1 are listed as ALREADY EXISTS and are not copied again.

Frequently, it is necessary to specify which disk drive a file name is on. For example, you might want to specify that a particular file is to be read from drive 1. This is done by appending the characters :1 right after the file name. Thus SERLDR:1 specifies the file named SERLDR.C on drive number 1. When reading a name such as SERLDR:1, think "S-E-R-L-D-R on one". Likewise, STARTUP.J:0 specifies the file STARTUP.J on drive 0. As before though, if you don't say, drive 0 is assumed.

Enter the command DIR SERLDR and look at the date associated with that file. Now enter the command DIR SERLDR:1 . Note that the date is different. In fact it will have today's date because you just created it a few minutes ago. Now try this: DIR *.Z:1 . You will see all of the .Z files on drive 1 which will also have today's date. Thus, a drive number may even be appended onto a pattern as well as an explicit file name. Finally, try this: DIR 1 . You would expect to see a directory listing of all of the files on drive 1 but instead you see an error message. What happened is that DIR tried to interpret the 1 as a file name which is clearly illegal. If you want a full directory listing of drive 1, you will have to use a pattern, so the proper command would be: DIR *.*:1 .

Before going on to other topics, lets experiment with COPYF a little more. Enter the following command to create an entirely new file on drive 1: SAVE JUNK.X:1 1000 4FFF . This command will be detailed later but essentially you have created a file called JUNK.X and then written a 16K portion of the MTU-130's memory into the file (the .X extension is often used to designate temporary "junk" files). Now let's make a copy of that file onto drive 0. With what you already know about the COPYF command and drive number specification you should be able to guess that COPYF JUNK.X:1 would do the trick. Go ahead and try it then use DIR to verify that the copy was made. Now try this command: COPYF JUNK.X:1 MOREJUNK.X . This will not only make another copy of the JUNK file onto drive 0, but it will also give the copy a different name. Finally, lets clean up the mess on drive 0 with the command: KILL *.X . This has the potential to delete all of the .X files on drive 0. Reply Y to JUNK.X and MOREJUNK.X but N to any others (there shouldn't be any). The DELETE command will not accept a pattern for a file name (that would be awfully dangerous).

In summary then, CODOS assumes drive 0 when you don't say. Some commands (like FILES) just need a number to specify the drive they are to act on. Any place where a file name can be used you can append a :d (where d is the drive number) onto the name to specify the drive.

TEXT FILES AND JOB FILES

A CODOS text file is simply a file that contains strings of ASCII characters that make sense when printed out and read by a human or are read directly by a language processor program. CODOS itself has a fair number of commands for handling text files although the MTU Editor is normally used for extensive manipulation of text files.

Probably the most fundamental thing you can do with a text file is to look at it. The TYPE command is provided for that purpose (and others). Lets use TYPE to look at a short text file on your Secondary Master. Enter: TYPE STARTUP.J . You should see several lines of CODOS commands printed, a blank line, and the CODOS½ prompt. Those CODOS commands are the actual text of this file and are executed whenever you load CODOS (such as when power is switched on). Such a file is called a Job file and will be studied later.

Let's try looking at another text file. Enter: TYPE UPLOAD.A and a carriage return. This is a much longer text file and is actually the assembly language source for the UPLOAD utility program. As before, you can make the display pause by typing CTRL/S, resume with CTRL/Q, or cancel with CTRL/C. Try these and then go ahead and cancel the listing (it is quite long). Finally, try this command: TYPE UPLOAD . Since this is not a text file (it is a binary machine language program), you will see garbage displayed (there may be some coherent message text interspersed however).

Now lets create a short text file. Normally you would use the MTU Editor but for instructional purposes we will use the TYPE command. Enter: TYPE C TESTTEXT.T and a carriage return. Note that a CODOS> prompt does not appear, just the flashing cursor. From now on what you type will be written onto the new file you just created. Unlatch the CAPS LOCK key and type in the lines below. If you make a mistake in a line, backup with the backspace key and correct it. Once you press the carriage return key to end a line, however, it's in the file.

```
This is a test of the CODOS TYPE command as used to create a text file.
I would normally use the Editor to do this but I am learning how CODOS
works by reading this manual and trying the experiments it describes.
```

After pressing carriage return to enter the third line, hold down CTRL and type Z to designate the end of the text file (Z is the end of the alphabet too). The disk will click to finish up the file and then the CODOS prompt will re-appear. Now do a DIR *.T to see that your text file is really there. Its length should be about $0000D5. Finally do a TYPE TESTTEXT.T which should display exactly what you typed in.

At this point we should describe the TYPE command further since the same command seems to be able to do opposing functions (typing out and entering in). TYPE is a generalized "source" to "destination" command. It reads text from some "source" and transfers it to some "destination". Sources and destinations can be all kinds of things in the MTU-130 system. A source could be the keyboard, a disk file, or some kind of accessory added to the computer. A destination can be the display, a disk file, a printer, or some other accessory. The TYPE command normally accepts two arguments, the first specifying the source and the second specifying the destination. If you only give it one argument (as in TYPE TESTTEXT.T), it takes that as the source and assumes that the display is the destination.

Thus TYPE TESTTEXT.T transfers from the file called TESTTEXT.T to the display, that is, displays the file. TYPE C TESTTEXT.T transfers from the keyboard to the file TESTTEXT.T. The C stands for Console and when used as a source means the keyboard portion of the console. When C is given for the destination, it means the display (try TYPE TESTTEXT.T C). If a file is given as the destination and it does not exist on the disk, it is created. Try this command: TYPE TESTTEXT.T TESTTEXT1.T . The disk will click for a while as TYPE effectively makes a copy of your TESTTEXT.T file. This is not a very efficient way to make a copy but it does work and further illustrates the way TYPE functions. (Do not use TYPE to copy non-text files because TYPE will insert a carriage return character every 192 bytes if none is found in the source file.)

In CODOS terminology, the console is a device and C is its device name. There may be other devices connected to the system; a printer for example would be the P device. Thus TYPE TESTTEXT.T P would print the file on the printer if the system has one and the "software driver" for the printer has been "installed".

CODOS also has something called channels. Channels have numbers from 0-9. The channel itself is not physical but it can be assigned to something physical such as a device (console, printer, etc.) or a file. After the channel is assigned, you can use the channel number alone and the effect will be the same as if you had given the device or file name. The ASSIGN command is used to perform this assignment. The command ASSIGN 5 TESTTEXT.T will "connect" the file TESTTEXT.T to channel 5 while ASSIGN 6 C will connect channel 6 to the console. Data can flow either way through a channel, the actual direction depends on how the channel is used.

Go ahead and enter the two ASSIGN commands described in the previous paragraph. Note that CODOS responds to the first one by saying that TESTTEXT.T is an "old file" which means that it already exists. If it did not exist, it would have been created and a "new file" message given. To check the channel assignment, enter ASSIGN with no arguments. You will get a listing of all active channels. Channels 1 and 2 are used by CODOS and will be described later. Channels 5 and 6 should be assigned to TESTTEXT.T and C respectively (channels 9 and 10 may also be assigned; just ignore these for now). Now enter the command: TYPE 5 6 and a carriage return. You should get a listing of the file just as before. Now enter ASSIGN again to check the channel assignment. Five and 6 are no longer assigned; the TYPE command "frees" them when it is finished.

The real power of channels lies not in cutting typing effort but instead in their ability to "redirect" data flow in the MTU-130. In the assignment listing you noticed that channels 1 and 2 seem to always the assigned to the console. The fact is that CODOS (and most other programs) read their command input from channel 1 and print their results and messages on channel 2. Since these are normally connected to the console, communication is directly to the user through the keyboard and display. If they are assigned to something else, commands and results can come from or go to that something else!

Enter the following command: ASSIGN 2 FILESLIST.T and a carriage return (don't worry about the lack of a CODOS prompt afterwards). Now enter a FILES command. The disk will click a few times and the cursor will reappear. Now enter ASSIGN 2 C to get things back to normal. A DIR FILESLIST.T will show that a new file called FILESLIST.T has been created and has several hundred characters in it. Now do a TYPE FILESLIST.T. What you will see is everything that CODOS would have displayed but which was redirected to the FILESLIST.T file while channel 2 was assigned to that file. If you had a printer attached to the MTU-130, you could have assigned channel 2 to P and you would have a hard copy of CODOS's console output.

Now what would you expect if channel 1, CODOS's command input channel, was redirected to a file? The answer is that CODOS would read its commands automatically from the file rather than the keyboard! To try it, create a text file called TESTJOB.J using the TYPE command and enter the following into the file:

```
MSG 2 THIS IS A TEST OF CODOS'S JOB FILE CAPABILITY
DIR *.T
TYPE TESTTEXT.T
DUMP 0 2F
MSG 2 END OF TEST
```

Be sure to terminate the text file after the carriage return of the last line with a CTRL/Z. Use the TYPE command to review the file to be sure it is right. If there is a mistake in the file, KILL it, and try again.

Next let's test the job file. Enter the command: ASSIGN 1 TESTJOB.J . Immediately the MTU-130 will seem to be "running itself" as the commands in the file are executed automatically. After the last command is executed, CODOS automatically reassigns channel 1 to the console and waits for a user command as evidenced by the flashing cursor. You can get the same effect by using the DO command. Try entering: DO TESTJOB.J . The effect is exactly the same as ASSIGN 1 TESTJOB.J except it is easier to remember and type.

The MSG command in the first line above is used to send a message to a channel. The number right after MSG is the channel number and the remainder of the command line is the message text. The DUMP command is explained in the next section.

Now it should be apparent what the purpose of the STARTUP.J file is. Whenever the system is "booted up" such as when power is turned on, CODOS initially reads its commands from this file. These include commands to load other programs (such as I/O driver programs), perhaps make "patches" to the operating system, or even print informative messages. There is nothing special about the STARTUP.J file (except its name) so you can prepare your own to replace the one supplied if you want to. For example, you could insert a patch to turn the keyboard sound off or you could insert a command to immediately load BASIC or a command to automatically load and execute one of your application programs. The uses for this capability should be obvious, particularly in turnkey systems set up for operators who may be unfamiliar with CODOS.

Before going to the next section, let's clean up the Secondary Master disk. Enter: KILL *.T and a carriage return. You should reply Y to the files you have created during the experiments in this section and N to any others. It is good practice to explicitly reply Y to kill the file or N to keep it; don't just hit carriage return to kill the file. You should also kill TESTJOB.J .

MACHINE LANGUAGE EXPERIMENTS

CODOS has many commands that make the development, debugging, and maintenance of machine language programs and binary data straightforward, if not simple. If you have no interest in machine language, you might want to skip to the next section but at least some knowledge of machine language concepts is very valuable in the long run.

The true heart of any microcomputer system is its memory. Developing and maintaining machine language programs usually focuses on the contents of memory. Consequently CODOS has a number of commands for examining and manipulating the MTU-130 memory. Perhaps most fundamental is the DUMP command which displays the contents of specified memory locations. Enter this: DUMP 300 and a carriage return. CODOS will respond as below (the exact contents may be different):

```
      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
0300  4C 53 E8 4C 03 E6 4C 75  C9 4C E6 CB 4C 36 CA 4C  LS.L..Lu .L..L6.L
```

Leftmost is the address of the first location dumped followed by 16 locations printed as hexadecimal numbers. To the right is the same 16 locations interpreted as ASCII characters. Note that only values which represent printable ASCII characters are interpreted; any others are represented by dots. The top line of the dump is a "guide strip" that makes determining the exact address of any of the values printed easier. Try also the following: DUMP 205 . The listing will now start at location 205 and go through 214 with the guide strip adjusted appropriately.

DUMP will also accept two arguments such as DUMP 200 2FF which will print all of the memory contents between these two limits, 16 values per line. The ability to see an entire "page" of memory at one time is often valuable. You can also follow the high limit address with a channel number or a device name (such as P for printer) if you wish to redirect the dump to another device or file.

Complementing DUMP is the SET command which is used to manually enter data into memory. Free memory in the MTU-130 extends from 0700 through BDFF (which is about 46K) so you should restrict your SETting to this area until you become familiar with the system in detail. First dump location 700. Then try the following command: SET 700 A9 99 0 0 followed by a carriage return. The first argument is the address in hexadecimal of the first location to set. The following argument or arguments are the data to set. Thus SET can be used to set just one location or dozens. In fact, you can overflow the command line and fill up the next one for setting up to about 50 bytes with one command. The values you set are not actually written into memory until you press return so you can use the backspace key freely to correct errors.

Now, dump 700 again which should show that those locations were indeed changed. If you have a lot of memory to set at once, you will need more than one SET command. Each SET command will require a new starting address; CODOS does not keep track of where the last SET ended. Like other commands, SET can be placed into a JOB file (even STARTUP.J) for automatically making "patches".

If you have a lot of memory to fill with the same value, the FILL command can be used. First dump locations 800 through 840 to see what they currently are. Now enter: FILL 800 840 99, a carriage return, and then a DUMP 800 840. You should see that memory between addresses 800 and 840 inclusive has been set to to 99 (since DUMP always displays full lines, locations 841-84F, which were not changed, are also displayed).

Now try this command: FILL C000:1 EFFF FF . You should see most of the display turn green (or white). What you have actually done is to write all binary ones (which represent white display dots) into most of the MTU-130's display memory. The :1 after the C000 starting address specifies that you want to fill starting at C000 in memory bank 1 which is the beginning of the display memory. In general you can specify a bank number with every memory address; if you don't say, bank 0 is assumed. Also note that only the first address of a range is allowed to have a bank number; the end of the range is assumed to be in the same bank. Every MTU-130 has all of bank 0 and from C000-FFFF in bank 1 filled with active memory (the display only shows up through FBFF:1). A memory expansion board is available to fill banks 2 and 3.

Another handy command is COPY. Its function is to copy from one area of memory to another and should not be confused with COPYF which copies disk files. COPY takes 3 address arguments. The first two specify the starting and ending addresses for the source memory block and the third is the starting address of the destination block. To make a copy of what we had SET into memory earlier into location 800, enter the following command: COPY 700 703 800 . Now dump both 700 and 800 to see that the first 4 bytes af each are the same. COPY can also copy from one memory bank to another. Try this: COPY D000:1 DFFF 1000:0 which copies a portion of display memory into regular memory at 1000. Now enter COPY 1000 1FFF C000:1 which will recopy back to the dsiplay but higher up on the screen. Note that if you don't specify a destination bank number, the command assumes the same bank as the source, not bank 0.

COMPARE is like COPY except that it compares two memory blocks to see if they are the same and prints out the address of the first discrepancy, if any. Try COMPARE 700 710 800 . You will probably see: 0704 printed which indicates that 700-703 is the same as 800-803 (which they should be since you just copied there) but that 704 is different from 804. If you instead enter: COMPARE 700-703 800, you will see a SAME report. COMPARE can also compare between banks like COPY.

Finally there is the HUNT command that can be used to find a specific byte or sequence of bytes. Try this: HUNT 0 FFF A9 99 0 0 . CODOS will print out: 0700 and 0800 indicating that the sequence A9 99 0 0 was seen in those two places between addresses 0000 through 0FFF. HUNT can also hunt for single bytes and hunt in other banks. The maximum allowable sequence length is 19 bytes.

CONTROLLING MACHINE LANGUAGE PROGRAMS

Now let's see how you save, get, run, and debug machine language programs under CODOS. The 4 bytes we have been working with so far (A9 99 00 00) are really a very short machine language program which loads the A register with hex 99 and then "breaks" to end. To run this program, enter: GO 700 and a carriage return. The program will be run (it only takes about 5 millionths of a second) and then the following will be printed out (some of the numbers may be different):

```
BRK, P=0702:0/0 (000007) A=99 X=00 Y=00 F=A5 S=FB
```

The BRK indicates that the program terminated with a BReaK machine language instruction and that's how CODOS was re-entered. The "P=" tells where the program ended, in this case the BRK instruction at 0702. The :0/0 indicates that the program was running in bank 0 (the :0) and that the data bank was also 0, which is the usual case. The (000007) simply gives the next 3 bytes in memory at the P= address (00 00 07). You can use a DUMP 702 to verify this. The remainder of the line shows the contents of the CPU A, X, Y, flags, and stack-pointer registers respectively. You can see that the value 99 was indeed loaded into the A register by the program.

Now let's try the REG command which is used to display and change the CPU registers. If you just enter REG you will get a register printout just as before but less the BRK at the beginning. New try REG A=0 which should set the A register to 0. Verify that it did with another plain REG command.

Ending a program with a BRK certainly gets the job done but is a bit messy on the screen for a "finished" program. A better way to return to CODOS at the end of a program is the the RTS instruction which has a hex code of 60. Use a SET 702 60 to modify the program for this convention. Now run the program with a GO 700. This time you will simply get the CODOS> prompt as a signal that the program is finished. To see that it did indeed run, enter a REG command to show that A has been changed to 99. You will notice that P= still shows 0700. P is not updated when a program terminates with an RTS instruction.

Now lets enter a little more complicated program which will be used for the remainder of this section. The listing is shown below:

```
ADDR  CONTENTS          ASSEMBLY LANGUAGE STATEMENT

0700  A9 20       CHRSET LDA  #32      ; START WITH ASCII BLANK
0702  A2 02              LDX  #2       ; SPECIFY CHANNEL 2 FOR OUTPUT
0704  38                 SEC           ; SET SUPERVISOR CALL ENABLE FLAG
0705  66 EE               ROR  $00EE
0707  00 04       PRINT   SVC  4        ; OUTPUT BYTE SUPERVISOR CALL
0709  18                 CLC           ; INCREMENT CHARACTER CODE
070A  69 01              ADC  #1
070C  10 F9              BPL  PRINT    ; REPEAT UP THROUGH $7F
070E  60                 RTS           ; RETURN TO CODOS
```

Enter this program like this: SET 700 A9 20 A2 2 38 66 EE 0 4 18 69 1 10 F9 60

Let's see how the BP (BreakPoint) command can be used in debugging. Enter the following command: BP 704 and a carriage return. This will temporarily insert a breakpoint into the program at location 704. Now do a GO 700 . The program should run up to the breakpoint and then re-enter CODOS which will print:

```
BP, P=0704:0/0 (3866EE) A=20 X=02 Y=00 F=04 S=FB
```

You can see that the program interruption was due to a breakpoint and that the LDA and LDX instructions did what they were supposed to do. Also, when the breakpoint was hit, it was cleared. CODOS will keep track of up to 3 different breakpoints.

Now type NEXT which will let the program continue from the breakpoint and run to completion. What it does is print the ASCII character set over Channel 2 which is normally assigned to the console. Several features of this very simple program are significant and pretty much indicative of what programming the MTU-130 in machine language under CODOS is like. The 2 loaded into register X is used later to specify where the output is to be sent. The SEC and ROR efficiently set bit 7 of location 00EE to a one which "enables" the Supervisor Call facility of CODOS.

It is the following SVC instruction that sets the MTU-130 and CODOS apart from other systems. The SVC 4 instruction tells CODOS that the program wants to do an "output byte over channel" operation which is SVC code number 4. The channel number is expected to be in the X register which it is. Channel 2 was specified in this case which, as we have seen before, is usually assigned to the console. Actually the SVC is really a standard 6502 BRK instruction with the SVC code number following it. Since the SVC enable flag is on, the CODOS BRK processor (BRK causes a system interrupt) interprets the BRK as an SVC rather than a program halt. The remaining instructions increment through the ASCII codes and then cleanly return to CODOS with an RTS instruction.

SAVING AND LOADING MACHINE CODE ON DISK

Since the preceeding was such an incredibly useful program, it is desirable to save it as a disk file. This can be accomplished very simply with the SAVE command. Enter: SAVE CHARSET 700 70E which will save the program as a disk file. Then clear that area of memory with a FILL 700 7FF 0 so you can prove that the program was saved. Now enter: GET CHARSET then DUMP 700 to see that the program was reloaded. Finally, do a GO 700 which should run it as before. Since the CHARSET file is a machine language program, you can GET and run it in one operation by just typing the file name (CHARSET) and nothing else. Try it. The most powerful command in CODOS is no command at all!

One perennial problem on most systems is determining where in memory unknown machine language programs load and run. In CODOS this is very simple. Try entering: GETLOC CHARSET . You should see the following printout:

CHARSET.C =0700 0700 070E

The CHARSET.C is the full program name and the =0700 is the starting address for execution or entry point of the program. The next two addresses are the first and last addresses occupied by the program. GETLOC will work on any machine language program or SAVEd memory image but text files or BASIC programs will give an error because the GETLOC (and GET for that matter) command cannot interpret the data in these files. Now try a GETLOC IODRIVER.Z . You will see a name and entrypoint as before but you will also see several sets of start-end addresses. This program thus consists of several separate blocks of memory (one is even in memory bank 1) which are all loaded at once when a GET or just the file name is used. We will see how to create this kind of file a little later.

Now lets do two things to the character set program in memory. First, use the COPY command to make a duplicate of the program starting at 800. Now lets modify the duplicate so that it outputs over channel 5 instead of channel 2. Do this by entering: SET 803 5 . Finally, the program really should send a carriage return character after it is finished with its output. Do this by adding the following code at location 80E with the SET command: A9 0D 00 04 60 . This added code simply sends a carriage return character ($0D) before it returns.

Before running the modified program, let's save the original and the modified version as two blocks on disk. Let's also replace the previous version of the program. You can do this with: RESAVE CHARSET 700 70E 800 812 . The RESAVE command is the same as the SAVE command except it is designed to effectively delete the old file and replace it with a new one of the same name in one operation. By putting another start-end address pair in the command, the second block was created. Use GETLOC to verify this.

The program entry point is assumed to be at the beginning of the first block when you don't say. If we wanted the entrypoint to be at 800 instead, we could have used: RESAVE CHARSET=800 700 70E 800 812 where the =800 designates the desired entry point. Go ahead and try this too and use GETLOC to verify that the entrypoint has been redefined.

Now let's run the modified program. First, channel 5 will have to be assigned to something. Enter this: ASSIGN 5 CHARSET.T . This will create a new file called CHARSET.T and assign it to channel 5. Now run the program by typing CHARSET and a carriage return. The ASCII character set followed by a carriage return will be written to the file. Now TYPE the file to prove that it is really there. A machine language program to do what this one just did would be much more complicated on other systems, but on the MTU-130 under CODOS, its simple!

SHORTCUTS FOR THE EXPERIENCED

CODOS has a large number of "shortcuts" available that streamline its operation for the experienced user. Not all or even most of these convenience features are covered here but the simplest and most generally useful ones are. All are covered in detail in the CODOS manual. Keep in mind that knowledge of little if any of the following is necessary for successful system operation. If you have a patient, meticulous personality, you might want to skip this section for now.

DEFAULTS

Defaults are assumptions that CODOS makes when you don't explicitly say what you want. You have already seen that the default disk drive number is 0, the default file name extension is .C and the default memory bank number is 0. Of these, the default disk drive number is easily changed with the DRIVE command. If you enter DRIVE 1, the default disk drive number becomes 1 and you have to explicitly specify when you want drive 0. Keep in mind though that many of the "commands" that have been discussed are actually just ordinary programs stored on disk. If the default drive is changed to 1, then a simple DIR, for example, will be fetched from drive 1. If a data disk without a copy of DIR is in drive 1, you would get a "COMMAND NOT FOUND" error. The correct command then would be DIR:0 . You still need to have the operating system files on the disk drive 0 even if the default drive has been redefined. Changing the default drive is most beneficial in a job file which may need to be flexible with respect to which drive it uses.

The default file name extension is .C. However when under the control of MTU BASIC, it is temporarily changed to .B and when BASIC Libraries are being accessed, the default is .Z. There is no user command to change the default extension but it is possible that an abnormal exit from BASIC or other languages may leave the default altered. This is evidenced by "COMMAND NOT FOUND" messages when some commands, such as DIR, are entered. To correct the situation, the program should be re-entered and then exited by normal means or you can press (and hold for a second) the RESET key. Temporarily, you can add the .C to disk-resident commands until the default extension is fixed.

NUMBER BASES AND ARITHMETIC EXPRESSIONS

In the section on machine language, single hexadecimal numbers were used for all addresses and byte values. You can use decimal numbers if you wish by preceeding the number with a decimal point (period). Thus .100 is decimal 100 and is equivalent to hexadecimal 64. You can also use actual ASCII characters for byte values if desired. Thus SET 1000 "A" would put the ASCII code for A into location 1000. SET 1000 "That's all folks" would store the entire ASCII string "That's all folks" into memory starting at 1000. Note that the quotes used to enclose the string can be either single (') or double (") quotes but must be the same on both ends. This allows the opposite kind of quote to be embedded in the string without problems. SET will also allow mixed operands such as:

```
SET 800 38 66 EE 0 2 2 'THIS IS AN INLINE MESSAGE' D 0 .96
```

Go ahead and try it with a GO 800, its a real program!

You can also use arithmetic expressions in address and data values. For example, to save 1000 decimal bytes starting at location 800 hex, you could enter: SAVE MESS 800 800+.1000 where an arithmetic expression is used to calculate the ending address. Or you could look into the 100th position of a translate table that starts at 85B4 with DUMP 85B4+.100 . Besides addition; subtraction, multiplication (*), divison (/), and modulus (\) can be performed. The order of evaluation is strictly left to right and no intermediate results can exceed +65535 or -32768 (decimal). Also, integer arithmetic is used throughout the calculation.

COMMAND EDITING

If you are like most people, you will make numerous trivial little mistakes while entering commands. While you can always use backspace to backup and correct any mistake, CODOS has a number of command line editing features that make correcting many kinds of mistakes faster than simply backing up and retyping.

One simple fact thats helpful to know is that CODOS sees what you see on the command line when you press RETURN regardless of where the cursor is. So if you discover that the command name, for example, was mispelled after you had entered the entire command but before you press return, you can backspace to the error, correct it, and then press return right away without moving back to the end.

You can also insert and delete characters in the middle of the line with the INSERT and DELETE keys (located near the shift keys). The DELETE key merely deletes the character the cursor is covering and moves everything that follows left one position to fill the hole. Pressing the INSERT key puts the keyboard in insert mode. In insert mode, when you enter a character, text under and to the right of the cursor moves right one position and the character entered goes into the resulting hole. While in insert mode, the RUB OUT key "unenters" the last character. Backspace and the cursor keys still perform their function and then cancel the insert mode. Probably the most frequent use of insert will be to go back and insert a forgotten drive number or file name extension.

Another useful function is the ability to recall a previously entered command line for editing. To recall the previous line, hold down CTRL and hit B (for Backup) and the previous line will appear. This is most useful right after CODOS has detected an error on the line (like a misspelled file name). You would just hit CTRL/B and the line is redisplayed ready for editing to correct the error instead of having to type it in all over again. As a matter of fact, you can recall additional commands back into the "history" of your session with the MTU-130 by continuing to hit CTRL/B. If you have been doing the experiments in the previous sections, go ahead and try it. There is a 256 character limit to the "archive", however, but that is enough to hold 8 to 15 average commands.

Other command line editing functions are available as well but these are the most commonly needed ones. Section 2 of the CODOS manual details all of the command line editing functions.

CHAPTER 4.

GETTING AQUAINTED WITH BASIC

The standard high-level language used on the MTU-130 computer is BASIC. MTU BASIC is an enhanced version of the industry standard Microsoft BASIC used by virtually all desktop and personal computers. Most likely, if you have had experience with BASIC on another microsystem, then the transition to MTU BASIC should be a painless, rewarding experience. If your prior experience has been with a non--standard minicomputer or mainframe BASIC or some other algebraic high level language (such as FORTRAN), there will still be far more similarities than differences. In either case, a working knowledge of programming concepts and terms such as variables, loops, arrays, and strings is assumed. The purpose of this chapter is to aquaint you with the operation of MTU BASIC so that you can immediately start to do useful things with it and be able to answer specific questions about its operation by consulting the MTU BASIC Reference Manual.

STARTING MTU BASIC

Unlike many desktop systems, the MTU-130 is a general purpose computer, not just a BASIC computer. The MTU BASIC interpreter is in fact just another program (albeit a very large one) that must be loaded into memory from disk and executed before it can be used. Thus MTU BASIC actually runs under the CODOS disk operating system which was described in the previous chapter. When you are finished with BASIC, control is returned to the operating system.

Starting MTU BASIC is actually very simple. Assuming that the MTU-130 has been turned on, your Secondary Master diskette (or a copy of it) is in drive 0, and the CODOS operating system is displaying the CODOS> prompt, you simply type:

```
BASIC
```

In a couple of seconds, you will see:

```
 34797 BYTES FREE

MTU-130 BASIC V1.0

READY.
```

and a flashing cursor (the number of bytes free may be slightly different). BASIC is now ready for you to enter commands or program statements. Unlike CODOS, BASIC does not print a prompt before the cursor. The fact that the cursor is displayed at all when under BASIC's control is an indication that it is ready for input.

Normally BASIC accepts only BASIC commands. Most CODOS disk commands, such as FILES, are not among these. If you need to use the services of CODOS while in BASIC, you can temporarily exit BASIC with the BYE command. Go ahead and enter:

```
BYE
```

You will now see the CODOS prompt and can enter CODOS commands. When you wish to re-enter BASIC (assuming that you have not overwritten BASIC with some other program while in CODOS), enter:

```
GO 0
```

BASIC should respond with a READY and a cursor. BYE is also used when you wish to permantly exit BASIC.

SIMPLE BASIC PROGRAMMING

In this section we will get aquainted with many of the characteristics of MTU BASIC by means of simple examples and experiments. Only the most fundamental points are covered here but they should be enough to get started.

MTU BASIC is an interactive implementation of the BASIC language. What this means to the user is that an illusion of the computer "directly executing BASIC statements" is created very effectively. In particular, its operation is conducive to experimentation and "incremental" program construction. BASIC appears to operate in two modes. If just a bare statement without a statement number is typed in, it is executed immediately when carriage return is hit. But if it is preceeded by a statement number, the statement is stored away in memory without being executed. A RUN command is necessary to execute statements that are stored away.

STATEMENTS AND PROGRAM EDITING

Lets try a few examples using the PRINT statement. Type in the following:

```
PRINT 2+2
```

BASIC will print a 4 and say READY again. You can give it a complex algebraic expression as well such as:

```
PRINT SQR(SIN(3.14159265/7))
```

which is the square root of the sine of PI over 7 and should be .658698519. You can use the backspace key to correct any errors you might make while typing. Thus MTU BASIC can be used directly as a powerful scientific calculator without any programming at all! The calculations performed by MTU BASIC are normally accurate to 9 digits which is better than most scientific calculators.

Now let's put a statement number ahead of the line. Enter:

```
10 PRINT 2+2
```

This time there was no 4 printed and the cursor simply moved down to the next display line. Since it had a number, the statement has been stored away in memory. Now type:

```
RUN
```

The stored statement is executed and the 4 is printed. Now let's add a second statement. Type:

```
20 PRINT 9*9
RUN
```

This time you will get the 4 from statement 10 and an 81 from statement 20. In the absence of instructions to the contrary (such as GOTO statements), BASIC executes statements in order of ascending line numbers. The exact magnitude of the line numbers is not really significant as long as the ascending order of numbers is the desired execution order of the statements. Actually it is good practice to leave gaps in the number sequence, such as numbering by 10's, so that new statements can be inserted between existing ones if desired. The highest statement number allowed is 65535 and you must not use commas in the statement number.

You can get a printout of your stored BASIC program by using the LIST command. Enter:

```
LIST
```

BASIC will respond with a listing of your entire program which in this case is just two statements. You can also follow the LIST command with a blank and a statement number such as:

```
LIST 10
```

In this case BASIC will just list statement number 10. LIST will accept a range specification as well. For example:

```
LIST 10-100
```

will list all statements with numbers between 10 and 100 which in this case would be the whole program.

Now let's say that you wanted to change statement number 10. This is accomplished by simply typing in another statement number 10. For example, try entering:

```
10 PRINT 1/7
```

Do a LIST to confirm that statement 10 has been updated and a RUN to confirm that the program has indeed been altered. The rule is that when BASIC is given a numbered statement, it first searches its memory to see if a statement with the same number is already there. If so, it replaces the old statement in memory with the new one. If the number is not found, the new statement is added to memory without disturbing the statements already there. You can effectively erase a statement by simply entering its number and a carriage return. The satements are always kept in memory in ascending order regardless of the order in which you enter them. You should note that numbered statements are <u>not</u> checked for validity as they are stored into memory. They are only checked when they are executed.

You can enter more than one statement on a line if you separate them with colons(:). Try entering the following statement:

```
PRINT 4*ATN(1): PRINT EXP(1): PRINT 2^2^2^2^2
```

You should see printed:

```
3.14159266
2.71828183
65536.0001
```

Using multiple statement lines can make a program more "human readable" by grouping related statements together and reducing the overall number of program lines.

VARIABLES

Like all high level programming languages, you can use variables in BASIC to represent numerical values. Variable names in MTU BASIC must start with an uppercase alphabetic character. Succeeding characters may either be uppercase letters or digits. The name may be of any length but only the first two characters are looked at by the computer. Thus the variables FREQUENCY1, FREQUENCY2, FREQ1, and FR would all be the same to BASIC. It is good practice to limit your variable names to two characters to avoid confusion. Unless you say otherwise, all variables in BASIC are floating point numerical values with a range from about $1x10^{-39}$ to $1.7x10^{38}$.

Now lets enter a simple program that uses variables. First you should erase the old program in memory be entering NEW and a carriage return. You can confirm that its gone with a LIST command. Now enter the following:

```
1000 M1=(A+B)/2
1100 M2=SQR(A*B)
1200 PRINT "ARITHMETIC MEAN IS",M1
1300 PRINT "GEOMETRIC MEAN IS",M2
```

Lets run the program without defining the variables. Both results will be zero. In BASIC, an undefined variable (that is, one that has not been assigned a numerical value), has a value of zero and no error is signalled when it is used in an expression. Now enter the following two statememts:

```
10 A=2
20 B=3
```

and run the program again. The answers should be 2.5 and about 2.44948974. To try other inputs, you simply change statements 10 and 20 and re-run the program which is easier than inserting the numbers directly into statements 1000 and 1100.

PRINT AND INPUT

Before continuing, note that it is easy to label your output in a print statement by enclosing the label in "" and using a comma to separate the label from the variable to be printed. You can even have several variables and labels printed at once by stringing them together such as:

```
1200 PRINT "ARITHMETIC MEAN IS",M1,"GEOMETRIC MEAN IS",M2
```

BASIC prints numbers in their easiest to read form without sacrificing accuracy. Thus small integers are printed as such without a decimal point. Likewise, reasonable valued decimal fractions or mixed numbers are printed in normal notation. Very large or small numbers however are printed in scientific notation such as: 4.29496731E+09 or 2.32830643E-10. You can use the same format for entering scientific notation numbers if desired.

An even easier way to input values into a BASIC program is the INPUT statement. To try this, first delete statement 20 by simply entering 20 and a carriage return (this replaces statement number 20 with a null statement thus erasing it). Now replace statement 10 with:

```
10 INPUT A,B
```

When you run the program, BASIC will print a ? and then wait for you to enter two numbers separated by a comma. Type in:

```
3,4
```

As when entering BASIC commands or statements, you can correct errors with the backspace key. BASIC doesn't actually look at what you typed until you press carriage return. If BASIC does see an error in your input, it will respond with REDO FROM START and another ?. In such a case you simply enter the two numbers again and the program will continue.

For a self explanatory program, you can preceed the input statement with a PRINT statement such as:

```
5 PRINT "ENTER 2 NUMBERS TO TAKE MEAN OF WITH A COMMA BETWEEN THEM - ";
```

Type this in and then run the program again. The semicolon (;) at the end of the above PRINT statement prevents BASIC from doing a carriage return after printing the message and thus leaves the cursor positioned so your input is on the same line as the message. It is possible to combine the previous PRINT and INPUT statements as in the following:

```
5 INPUT "ENTER 2 NUMBERS TO TAKE MEAN OF WITH A COMMA BETWEEN THEM - ";A,B
```

As a final refinement to the program, you can put it into a loop so that it will automatically ask for another pair of numbers without you having to RUN it again. Type in this statement:

```
2000 GOTO 5
```

Now when the program is done and gets to statement 2000, it will immediately jump back to the beginning and "run" itself again. One problem with such an "infinite loop" however is that it will never terminate; the program will just keep asking for numbers. You can halt such a running program however by holding the CTRL key down and typing a C. The program will be interrupted and BASIC will print: BREAK IN 10 or whatever line was executing at the time you typed CTRL/C. A program may also be halted while executing an input statement by just typing a carriage return, i.e., giving no number at all. The "null" response is interpreted as a request to end the program.

FOR - NEXT LOOPS

For controlled loops, that is loops that execute a specific number of times and then exit, BASIC offers the FOR statement. The general form of the FOR statement is as follows:

FOR ⟨variable name⟩ = ⟨starting value⟩ TO ⟨ending value⟩ STEP ⟨step value⟩

where ⟨variable name⟩ is any variable name and ⟨starting value⟩, ⟨ending value⟩ and ⟨step value⟩ can be any number, variable, or expression. For example, the statement:

```
FOR N1=10 TO 90 STEP 3
```

would set up a loop in which the variable N1 would take on values of 10, 13, 16, 19, ..., 85, 88. Note that the loop variable (N1) always starts at the starting value and stops at the highest value that does not exceed the ending value. The following would also be a legal FOR statement:

```
FOR X=3*XS TO 3*XS+200 STEP 200/3
```

In this case, expressions have been used to define the starting, ending, and step values. The expressions are evaluated when the loop starts thus even if the loop itself modifies the value of XS, there will be no effect on the number of times the loop executes.

The STEP keyword and step value are optional. If a step is not specified, then a step value of positive 1 will be assumed. It is also permissable for the final value to be less than the initial value and a negative step to be specified. Note that a FOR loop is always executed at least once even if the final value is less than the initial value and a positive step is specified.

The FOR statement is always placed at the beginning of the statements that will form the loop. The end of the loop is designated with the NEXT statement. The NEXT statement also specifies the loop variable which identifies which FOR loop (in case there are several) is to be terminated. Thus the appropriate NEXT statement to terminate the second FOR example above would be: NEXT X .

Let's try a simple program using a FOR-NEXT loop. First erase the previous program by entering NEW and a carriage return. Now type in the following:

```
10 PRINT "ENTER NUMBER TO TAKE FACTORIAL OF - ";
20 INPUT N
30 A=1
40 FOR I=1 TO N
50 A=A*I
60 NEXT I
70 PRINT N;" FACTORIAL IS ";A
```

Now RUN the program to see what it does. The FOR loop starts with statement 40 and ends with statement 60. The single statement inside the loop (statement 50) performs the actual factorial calculation by repeated multiplication of successively higher numbers. Note the use of semicolons in the PRINT statements to keep things close together on the same line.

FOR-NEXT loops can also be "nested together" or in other words, you can set up a loop within a loop. The best way to see this is by an example. Type in the following statements to update the program above:

```
10 PRINT "ENTER HIGHEST NUMBER TO TAKE FACTORIAL OF - ";
20 INPUT M
25 FOR N=1 TO M
80 NEXT N
```

After entering the program, LIST it completely so you can see it all at once. Now RUN it and type in something like 10. The FOR N=... NEXT N loop is called the outer loop because it encloses the inner FOR I=... NEXT I loop. Thus for each iteration of the outer loop (each value of N), the inner loop executes N times. You can even have "loops within loops within loops" and so forth with no real practical limit. The ability to nest loops like this is a very powerful feature of BASIC because just a few statements can specify a lot of calculation. Note that nested loops must always in fact be nested. An inner loop must always be completely enclosed by an outer loop.

IF STATEMENT

Perhaps the most important feature of any programming language is its conditional statements. BASIC has two of which the IF statement will be described here. The simplest form of the IF statement is as follows:

IF relational expression THEN statement

In operation, the relational expression is evaluated and if it is true, then the program executes the statement following the THEN keyword. If the relational expression is false, then the statement following THEN is ignored and next statement in line is executed. The relational expression itself consists of two numbers, variables, or expressions separated by a relational operator. Relational operators are one of the following:

<	less than	<=	less than or equals
>	greater than	>=	greater than or equals
=	equals	<>	not equal

In cases where the relational operator is made up of two symbols (such as =), the order of the two symbols is not significant. Following are some example IF statements and their meaning:

510 IF A=0 THEN GOTO 200	If A is exactly equal to 0 then jump to statement 200, otherwise continue with the next statement.
120 IF M*3>N THEN GOTO 1276	If the value of M*3 is greater than the value of N then jump to statement 1276, otherwise continue.
58 IF A<.5 OR A>=1 THEN GOTO 76	If A is either less than .5 or is greater than 1 then jump to statement 76, otherwise continue.
500 IF X=3 AND Y=45 THEN GOTO	If X is exactly equal to 3 and Y is exactly equal 45 then jump to statement 600, otherwise continue.
35 IF A>100 THEN A=100	If A is greater than 100 then set A equal to 100, otherwise leave A unchanged.

Note that the ability to put any arbitrary statement after the THEN keyword can prove to be very handy in limit testing and correction. Keep in mind though that the next statement in line is always executed when the conditional statement is not a GOTO as in the first few examples.

MTU BASIC is also capable of handling arrays of variables as well as individual variables. Arrays are given names just like individual variables except that now a single name can refer to an entire group of numbers. For example, let's assume that an array called AR consists of 5 elements as follows:

AR(1)=3.1 AR(2)=5.4 AR(3)=7.6 AR(4)=13.8 AR(5)=0

Note that a specific element of the AR array is designated by a number in parentheses. In a program, you can also use a variable or even an expression to select the array element needed. The value in parentheses is called a subscript because it selects the desired array element much like a subscript in a mathematical expression does. The subscript is always truncated to an integer (rounded down) before being used.

To see how arrays and subscripts are used, erase the previous program with a NEW and then type in and run the following:

```
100 FOR I=0 TO 10
110 A(I)=I*I
120 NEXT I
```

This loop will fill an array called A with a table of squares. Now spot check a couple of elements by typing PRINT A(3) and PRINT A(7) to see if 9 and 49 respectively are stored.

BASIC automatically provides memory space for 11 elements (subscripts 0 - 10) when an array is first used. If more elements are needed, a DIM statement should be used to specify how many are needed. For example, if 50 is the maximum subscript that will be used with an array called A. then the statement: DIM A(50) will reserve enough space for it. If a DIM statement is used, it should be executed in the program before the array itself is used. In fact, it is a good idea to explicitly declare the dimension of all arrays. You should note that an array may have the same name as a variable. BASIC can always distinguish between the two because the array name will always be followed by a subscript while the individual variable will not.

Arrays may also have more than one dimension. A two dimensional array called A2 for example might have 15 rows of 25 columns. The DIM statement for A2 would be DIM A2(15,25) . You would reference an element of A2 with a double subscript such as A2(9,14) or A2(I,J) which would reference the 9th row and 14th column or Ith row and Jth column respectively. A very common use of two-dimensional arrays is storage for X and Y point coordinates in a graphics program as will be demonstrated later. You can even have 3 dimensions and in fact there is no limit at all (except memory space) to the number of allowable dimensions. Memory is used at the rate of 5 bytes per array element however so don't reserve any more array space than necessary.

ADDITIONAL FUNCTIONS

Besides the SQR (square root) function already used, BASIC has a large number of other mathematical functions available. A function is used just like a simple variable and consists of the function name followed by the argument of the function in parentheses such as: A+SQR(B*B+C*C) which takes the square root of B^2+C^2 and then adds it to A. The standard mathematical functions provided are as follows:

SIN	Trigonometric sine, argument in radians
COS	Trigonometric cosine, argument in radians
TAN	Trigonometric tangent, argument in radians
ATN	Trigonometric inverse tangent, result in radians
EXP	Natural exponential
LOG	Natural logarithm
SQR	Square root
ABS	Absolute value
INT	Round down to next lowest integer
SGN	Sign function (=-1 for < 0, =0 for =0, =+1 for >0)
RND	Random number between 0 and 1 if argument is positive (see BASIC manual for action on zero and negative arguments)

In addition, you can take a number to any power by using the form: A^X which gives A to the X power. If X is an exact integer then it would be permissable for A to be negative but if X has a fractional part, A must be positive.

MORE ON EXPRESSIONS

MTU BASIC has tremendous flexibility in forming expressions from constants, variables, and functions. However when an expression contains multiple elements and operators, the order of evaluation becomes important. In general, parentheses can always be used to explicitly define the order of evaluation. In the absence of parentheses, some operators have priority over others according to the following rules:

1. Parentheses have highest priority so subexpressions in parentheses are evaluated first from the inside out.

2. Exponentiation (^) has the next highest priority.

3. Negation has the next highest priority.

4. Multiplication and divison are equal and are next highest.

5. Addition and subtraction are equal and are the lowest in priority.

When there is a string of operations of equal priority, processing is from left to right.

Tokens and keywords are two things BASIC uses when executing BASIC statements. Understanding a little about what they are and how BASIC recognizes and interprets them can be helpful in understanding some seemingly trivial errors that can occur when using MTU BASIC.

Whenever a command or program line is entered, BASIC will first scan the line looking for all commands, functions, and operators. These commands, functions, and operators are collectively referred to as "keywords". When scanning a line, any sequence of characters which matches a recognized keyword is replaced by a special character, called a "token". Each possible has its own associated unique token. Once all keywords are "tokenized", the line is executed or stored in memory.

All characters in a BASIC line are subject to tokenization except when the characters are within quotes or when they are part of a REM statement. (REM statements are "remarks" which are helpful for explaining the program to human readers.)

This tokenization process has two advantages. First, execution is greatly speeded up, and second the program is made more compact in memory. The important point to remember is that before BASIC can perform the operation called for by a keyword, it must be tokenized!

When listing a program, the tokens are automatically converted back to their associated keyword all spelled out. This makes the tokenization process essentially transparent to the user but can also lead to subtle errors. For example, there can be confusion if a character sequence in a BASIC line unintentionally matches a keyword such as the statement:

```
FACTOR = 10
```

which may seem like a valid statement. However, OR is a keyword. This means that BASIC will interpret this statement as if it was:

```
FACT OR = 10
```

which results in a SYNTAX ERROR. An easy way to avoid this problem is to always use just 1 or two characters for variable names. The only keyword which might sneak up on you is "FN" which is used for user defined functions.

To take a look at tokenization in action, we can use the FRE() function. This function returns the number of unused bytes available in memory. Enter NEW followed by PRINT FRE(0). The NEW clears BASIC memory and the number printed indicates how much memory is available. Now enter the program line:

```
10 PRINT
```

then enter PRINT FRE(0) and note that the number is 6 less than the number printed before. Each line of a program will take up 5 bytes plus the number of characters in the tokenized line, excluding the line number. This means that the characters P-R-I-N-T were converted to a 1 byte token. Now enter this program line:

```
20 "PRINT"
```

then PRINT FRE(0) and note that the new line took up 12 bytes, 6 more than line 10. Adding the two quote marks caused the line to take up 6 more bytes, not 2, because the quotes prevented the PRINT keyword from being tokenized.

LINE EDITING IN MTU BASIC

MTU BASIC has the same line editing features as the CODOS operating system does plus a unique EDIT command. While typing in a BASIC command or statement, you can use the backspace key to backup and correct any mistakes you may notice.

One simple fact thats helpful to know is that BASIC sees what you see on the statement line when you press RETURN regardless of where the cursor is. So if you discover that the statement keyword, for example, was mispelled after you had entered the entire statement but before you press return, you can backspace to the error, correct it, and then press return right away without moving back to the end.

You can also insert and delete characters in the middle of the line with the INSERT and DELETE keys (located near the shift keys). The DELETE key merely deletes the character the cursor is covering and moves everything that follows left one position to fill the hole. Pressing the INSERT key puts the keyboard in insert mode. In insert mode, when you enter a character, text under and to the right of the cursor moves right one position and the character entered goes into the resulting hole. While in insert mode, the RUB OUT key "unenters" the last character. Backspace and the cursor keys still perform their function and then cancel the insert mode. Probably the most frequent use of insert will be to go back and insert a forgotten parenthesis or comma in the statement.

Another useful function is the ability to recall a previously entered statement line for editing. To recall the previous line, hold down CTRL and press B (for Backup) and the previously entered line will appear. This is most useful just after BASIC has detected an error on an unnumbered (immediate execution) line like a misspelled keyword. You would just press CTRL/B and the line is redisplayed ready for editing to correct the error instead of having to type it in all over again. As a matter of fact, you can recall additional lines back into the "history" of your session with BASIC by continuing to type CTRL/B. If you have been doing the experiments in the previous sections, go ahead and try it. There is a 256 character limit to the "archive", however, but that is enough to hold 5 to 10 average BASIC lines.

For editing statements that are already stored away in a program, MTU BASIC offers the EDIT command. For example, if you wish to change something in line 520, you would just enter EDIT 520 and a carriage return. BASIC responds by listing line 520 with the cursor on the last character. At this point you can edit the line with backspace, strikeover, insert, delete, and cursor left or right. just as if you had typed it in up to that point. When the line has been edited to your satisfaction, press return to update the line in memory. Only the cursor left and right keys should be used for editing the line; avoid using cursor up or down until you have more experience with BASIC. Note that if you edit the statement number, you will actually make a copy at the new number of the statement when you press return.

SAVING AND LOADING BASIC PROGRAMS

You will probably wish to save on disk any significant program you write in MTU BASIC so that you may easily load it for reuse later. To save a program in its compact memory image format you use the SAVE statement. The general format of the SAVE statement is: SAVE "FILENAME" where FILENAME is a legal CODOS filename enclosed in double quotes. If you have not read the Getting Aquainted With CODOS section, the filename must start with a letter and be between 2 and 12 letters or numbers long (all letters must be upper case). If a file name extension is not specified, BASIC will supply a .B. The filename chosen must not already exist on the disk. If it does, an error will be given.

A BASIC program may be recalled from the disk with the LOAD command. The format of the LOAD command is: LOAD "FILENAME" where FILENAME is the same file name that was used to save the program. Note that LOADing a program automatically erases whatever program, if any, was previously in memory. An error is given if the file name cannot be found on the disk or if the file was not created with BASIC's SAVE command. After the program is loaded from disk, you can edit it or run it. If the program is to be run immediately after loading, you can combine loading and running by using the RUN command followed by a file name enclosed in double quotes. Thus the command: RUN "TESTPROG" would load and immediately run the BASIC program called TESTPROG.

The SAVE, LOAD, and RUN commands deal with BASIC programs represented in a compact internal "tokenized" format. There are also commands for dealing with BASIC programs in their normal text format. To save a BASIC program in text format, the LIST command followed by a file name in double quotes can be used. For example, the command: LIST "PROG2.T" would save the entire program in memory in a file called PROG2.T. You can just save a portion of a program by giving statement numbers as with the normal LIST command. Thus LIST "SUBNORMALIZE.T",1000-1999 would just save statements 1000 through 1999 on the file. Since the files created by LIST are normal text files, it is desirable to use a .T extension to the name as in the examples. Otherwise, BASIC will assume .B which could cause confusion with programs stored in tokenized format.

The ENTER command is used to reload BASIC programs that had been stored with the LIST command. It can also be used to load BASIC programs prepared with the MTU Editor or even programs prepared on a different computer. The general format of ENTER is: ENTER "FILENAME" where FILENAME is the name of the file containing the program in text format. Note that unlike LOAD, ENTER noes not clear out the old program in memory. The effect in fact is just as if you had typed in all of the statements in the file. This can be handy if you wish to maintain a collection of general subroutine files. Then to add a subroutine to your program, you can simply ENTER it. Since LIST and ENTER deal with programs in their full text form rather than the compact and efficient tokenized form, program saving and loading using them is much slower than when using SAVE and LOAD.

You may leave BASIC and re-enter CODOS at any time to look at the directory, delete old program files, etc. by typing BYE and a carriage return. When finished with CODOS, you re-enter BASIC with a GO 0 command.

USING THE INTEGER GRAPHICS LIBRARY

Perhaps the most unique feature of MTU BASIC is the ability to selectively add "command libraries". A command library is a set of specialized command processors that can be added to BASIC when the user program requires their functions. Loading a library has the effect of adding a new set of commands to BASIC's keyword list. These new keywords can then be tokenized and executed like the standard keywords. One advantage of calling in the command processors only when needed is that memory is saved. Another advantage is that more command libraries may be added by MTU or the user at any time without requiring a version of BASIC itself.

Standard MTU BASIC is supplied with 3 different command libraries. In this section you will get aquainted with how libraries are used in general and with the Integer Graphics Library (IGL) in particular.

USING THE LIB STATEMENT

When MTU BASIC is started, only the standard command set is available. To bring in a library of additional command processors, you must use the LIB statement. The general form of the LIB statement is: LIB "LIBRARYNAME" where LIBRARYNAME is the the name of the command library which is actually just a file name on the disk. The Integer Graphics Library is called IGL.

Before actually loading in the IGL library, try entering the following BASIC command:

```
SMOVE 350,100
```

You should be greeted by a SYNTAX ERROR message since SMOVE is not in the standard command set. At this point, BASIC actually interprets SMOVE as a variable name. Next enter NEW to make sure no program is in memory, and enter this program line:

```
10 SMOVE 350,100
```

Now enter RUN and note the same SYNTAX ERROR message. Again SMOVE is not being recognized as a command. To load the IGL library, enter the command:

```
LIB "IGL"
```

You should hear the disk click as the IGL command processors are read in from disk. Now enter the SMOVE 350,100 command again. Note that the command is now accepted without error. This is because the IGL library has added a new set of keywords to BASIC, one of which is SMOVE. Thus SMOVE can now be tokenized, and therefore executed. Enter RUN to execute the previously entered program line. You will still get the SYNTAX ERROR since the line was entered before the IGL library was loaded. The SMOVE in this line wasn't tokenized so it still can't be executed. To correct this line, it must be reentered while the IGL library is loaded.

To summarize the above discussion, for library commands to execute properly, the associated library must be loaded before the commands are entered. This applies not only to programs entered from the keyboard, but also to programs entered from disk using the ENTER command.

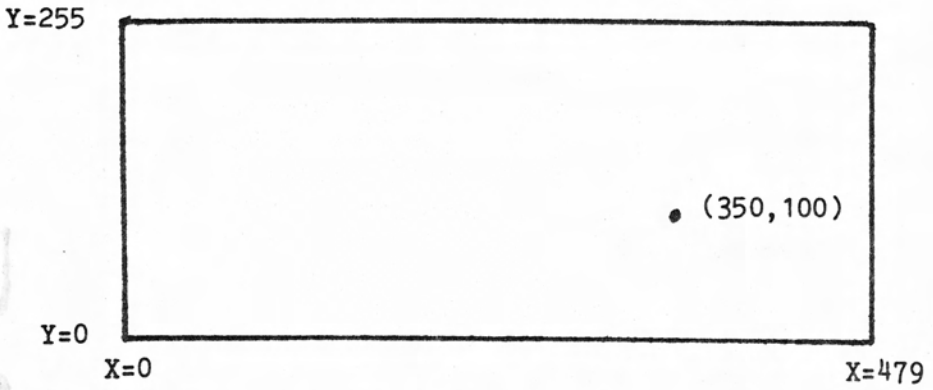
If you accidentally enter a large number of library commands while the library wasn't loaded, just LIST the lines to disk, load in the library, then use the ENTER command to reenter those lines. This is faster than reentering them by hand.

Another problem, opposite from the one above, can occur when LISTing a program. If a tokenized command is found for a library which is not loaded, then BASIC will be unable to convert it back to its original keyword. When BASIC finds such a token, it will look up the name of the library in the SYSLIBNAM.Z disk file. Once found, BASIC will print the name of the library followed by "?NOT LOADED ERROR", and abort the list command. To correct this problem, load in the needed library and reexecute the list command.

For turnkey "load and run" BASIC programs, the LIB command itself can be put in as a program statement. If it is placed at the beginning of the program, then when the program is RUN, it will load in the command processors it needs automatically.

THE MTU-130 GRAPHICS SCREEN

The MTU-130 display screen is actually just a very large matrix of dots, each of which may be on (a tiny spot of light) or off. Each of the 122,880 dots is independent of the others which means that any kind of image can be constructed with the proper software. The dot array consists of 256 rows of dots with 480 dots in each row. The location of any dot can be specified by giving its X and Y coordinates as illustrated below:



X defines the column number and can range between 0 and 479. Y defines the row number and can range between 0 and 255. The example point shown in the drawing has an X coordinate of 200 and a Y coordinate of 100. If a coordinate is out of range, it will either be set to the nearest extreme value or an error message will be given.

The most fundamental operation is clearing the screen which is performed with the SCLEAR command. Go ahead and enter SCLEAR to prove to yourself that it works (be sure that IGL has been re-LIBbed in after the previous experiment). Note that SCLEAR clears the entire screen including the function key legends. Like all IGL commands, SCLEAR can be given a statement number and placed in a program as well as executed immediately.

PLOTTING COMMANDS

The IGL line plotting commands make use of an invisible graphice cursor. Generally they plot from the current graphics cursor position, wherever it might be, to a specified point on the screen. The SMOVE command is used to establish a new cursor position without drawing anything. For example, to place the graphic cursor at X=350 and Y=100 as illustrated above, you would enter SMOVE 350,100 and a carriage return. Go ahead and do this.

The SDRAW command will draw a solid white line from the current cursor position to a specified coordinate. Try entering the following command: SDRAW 170,200. You should see a short diagonal line drawn on the screen starting at X=350,Y=100 and ending at X=170,Y=200. Right after the line was drawn, the whole screen probably scrolled up one line when the READY message was printed. You can avoid this while experimenting with the graphics commands by typing CTRL/L occasionally to clear the screen and return the text cursor to the top of the screen. Go ahead and type CTRL/L to home the text cursor then enter the SMOVE 350,100 and SDRAW 170,200 commands again to draw the line again.

Now enter SDRAW 200,240. Another line starting at the end of the previous line and ending at X=200,Y=240 will be drawn. The rule is that SDRAW leaves the graphic cursor at the end of the previous line drawn. You can use this fact to advantage when you have a number of connected line segments to draw such as in a line graph.

Speaking of graphs, let's work on a simple program for plotting the graph of a mathematical function. Since everyone else always uses the sine wave as an example, let's try a tangent wave instead. Enter the following program:

```
10 SCLEAR
20 FOR X=0 TO 479 STEP 2
30 TH=6.283*X/479
40 FC=TAN(TH)
50 Y=128+25*FC
60 SMOVE X,Y : SDRAW X,Y
70 NEXT X
```

After you're reasonably sure it has been entered correctly, run it. Statement 10, of course, clears the screen. The loop started in statement 20 scans accross the screen width two coordinate units at a time. Statement 30 scales the argument for the tangent function so that it is zero when X is zero and is two times PI (about 6.283) when X is 479. This statement controls how many cycles of the tangent wave will be plotted. The values given plot one cycle. Statement 40 sets FC to the value of the tangent function at the argument calculated in statement 30. By changing this statement (and perhaps some of the scaling), this program can plot any mathematical function. Statement 50 scales the function value so that it is suitable for plotting. Adding the 128 (half the Y range) effectively puts the Y=0 axis in the middle of the screen. Multiplying the function value by 25 lets us see function values between approximately -5 and +5. Line 60 is a double statement which effectively plots a single point at the Current X and Y coordinate values. The SMOVE first moves the graphic cursor to X,Y and the SDRAW draws a "line" of zero length there which results in a dot being displayed. The colon (:) separates the two statements and can be used to have any number of any kind of statements on one line.

Now let's refine the program. Axes are easily plotted by adding the following statement:

```
15 SMOVE 0,0 : SDRAW 0,255 : SMOVE 0,128 : SDRAW 479,128
```

Go ahead and type it in and run the program again. You will probably notice that that the points are not close enough together to get a continuous line effect. This can be corrected by connecting the computed points with lines instead of just plotting them as points. To try this, modify statement 60 as follows:

```
60 SDRAW X,Y
```

and run the program again. This time the points will be connected with lines including the points on either side of the discontinuity at PI/2 and 3*PI/2.

There can be side effects from connecting the points however. Change statement 30 so that it reads:

```
30 TH=6.283*X/479+.75
```

and run the program again. You will get the same shape just shifted a little to the left but you will also get an extraneous line from the end of the X axis to the first point plotted. This is because for the first point, the previous graphic cursor position is at the end of the X axis (from statement 15) rather than a previous point on the graph. One way to correct this problem is to insert the following statement:

```
55 IF X=0 THEN SMOVE X,Y
```

Now when the program is run, an initial move to the first point will be executed before drawing to it. Just for kicks, try plotting a sine curve by changing line 40 to:

```
40 FC=SIN(TH)
```

and running the program again. You may wish to change statement 50 as well to stretch the plot vertically (make the 25 larger) since the sine function restricts itself to a -1 to +1 range.

The SMOVE and SDRAW commands are quite useful but will only generate white lines. The SPEN command in conjunction with PENMODE will allow you to draw other "kinds" of lines. The SPEN command sets the drawing "color" when SPEN is used according to the table below:

PENMODE	LINE TYPE
0	None (same as SMOVE)
1	Draw white (same as SDRAW)
2	Draw black
3	Draw flip (each dot along the line is flipped to the opposite of its current "color")
4	None (same as SMOVE)
5	Draw dashed white
6	Draw dashed black
7	Draw dashed opposite the background color

The SPEN command is used exactly like the SDRAW command except that it draws according to the current pen mode. The draw black mode for example can be used to erase a previously drawn line by drawing it again in black. The draw flip mode is probably the most useful when drawings must change because a line can be drawn and later erased (by flipping the line again) without disturbing any other lines it may cross.

The dashed line modes are also very useful. Before the dashed mode can be used however, the dash pattern must be established. The DASHPAT command is used to do this. The command DASHPAT 240,240 for example will establish a dash pattern of 4 dots on alternating with 4 dots off (the IGL manual gives other dash patterns). To try this out, add or modify the following statements in the program you have been experimenting with then run it again:

```
 5 PENMODE 5
 6 DASHPAT 240,240
60 SPEN X,Y
```

For any kind of finished graph you will want to add text labels in various places. The LABEL command is provided to do this. The text is positioned so that the lower left corner of the first character drawn is at the current graphic cursor position. Try adding the following statement to your program:

```
100 SMOVE 150,240 : LABEL "SINE WAVE PLOT"
```

You can also print variables as labels such as: LABEL A which will print the value of A wherever the graphic cursor is. This can be useful for calibrating axes for example. Be aware however that positive numbers will be preceeded by a blank and that numbers that are not exact may be printed with a lot of digits or in exponential notation.

The commands discussed so far all use absolute coordinates. There is another series that specifies coordinates relative to the graphic cursor position. The command:

```
SRDRAW 50,-30
```

for example will draw a line from the graphic cursor to a point 50 units to the right and 30 units below the graphic cursor. This can be quite useful if you have a data array of relative coordinates specifying some kind of graphic shape. You colud than SMOVE to any point and then draw the shape using relative coordinates without having to alter the shape data table itself. Another use is plotting points since there is no "SPOINT" command in IGL. For example, to plot a point at X,Y, use:

```
SMOVE X,Y : SRDRAW 0,0
```

SRMOVE and SRPEN are the relative counterparts to SMOVE and SPEN. Note that relative coordinates are restricted to a range of -128 to +127.

To make a graphics hardcopy, use the SPRINT command. There are several SPRINT programs provided to perform dot graphics screen prints. The names of the files on disk will have appropriate letters appended to SPRINT to indicate which printer they are used with. You will have to rename the version appropriate to your printer to SPRINT.Z in order for IGL to use it. If you have another printer which is capable of dot-addressable graphics, you may assemble your own custom SPRINT.Z program by modifying one of the other source programs provided on the distribution disk.

GRAPHIC DATA INPUT COMMANDS

The MTU-130 and the IGL library provides two different interactive methods for the user to input graphic coordinate information to a BASIC program. One of these is the GRIN (GRaphic INput) cursor and the other is the light pen. Both are exceptionally easy to program and use, a feature that sets the MTU-130 apart from other computers.

To see how the GRIN cursor works, type in the following command:

```
SGRIN A$,X,Y
```

You should see a large flickering "crosshair" somewhere on the screen. You can move the intersection point horizontally with the left and right cursor keys and vertically with the up and down cursor keys. Holding the shift key down will speed up the movement by a factor of 5. Now move the cursor so that it is approximately in the center of the screen. You can terminate the SGRIN command and "register" its current position by pressing any other key. For this example, press the "W" key to terminate. BASIC should respond with READY. To find out what was stored into the A$, X, and Y variables, enter:

```
PRINT A$,X,Y
```

The W printed is the value of A$ which is the character you used to terminate the SGRIN command. X should be around 240 and Y around 120 depending on how accurately you estimated the center screen position. Incidentally, A$ is called a <u>string variable</u> in MTU BASIC. String variables may hold character strings of variable length just as the normal numerical variables hold numbers of variable sizes.

Now let's try a simple drawing program using the SGRIN command. First enter NEW to erase the previous program and then enter the following:

```
10 SCLEAR
20 SGRIN A$,X,Y
30 IF A$="M" THEN SMOVE X,Y
40 IF A$="D" THEN SDRAW X,Y
50 IF A$="X" THEN GOTO 10
60 IF A$="Q" THEN END
70 GOTO 20
```

To use the program first RUN it which will clear the screen and show the crosshair cursor. After you have positioned the crosshair where you want it, you can press M to move the graphic cursor to that position. If you press D, a line will be drawn from the previous graphic cursor position to the crosshair position. The X key will clear the screen so you can start over and the Q key will terminate the program. Note how easy it is to maneuver the cursor to a precise point such as when closing up a square or triangle. The program itself should be self explanatory. This program illustrates how simple it is to make use of the graphics capabilities of the MTU-130. You can easily add statements to save the data points in an array, add editing functions, etc. to the program.

The light pen offers a different way to enter graphic data into the MTU-130 that has both advantages and disadvantages when compared to the GRIN cursor. The light pen command is SLTPEN and is followed by three numerical variables. The first variable is a "flag" that tells whether the pen saw light when the command was executed. If the pen sees light, the flag is set to 1 and the second and third variables are set equal to the X and Y coordinates respectively of where on the screen light was seen. The light pen is sensitive only to light from the CRT screen; it completely ignores any ambient room light.

Try entering and running the 3 line program below to see how the light pen works (be sure to erase the previous one with a NEW command):

```
10 SLTPEN F,X,Y
20 IF F=0 THEN GOTO 10
30 PRINT X,Y : GOTO 10
```

Now while the program is running, pick up the light pen and point it to anything you see on the screen. When the pen sees light, you should see the coordinates of the pen's position printed out. If the pen refuses to respond to anything, turn up the monitor's brightness a little. If you are using a non-MTU display monitor, make sure the tube's phosphor screen is not a "long persistence anti-flicker" P39 type. You can stop the program by typing a CTRL/C.

To see the real potential of the light pen, key in the program below:

```
10 PENMODE 1 : SFILL 0,479,0,255 : PENMODE 2
20 SLTPEN F,X,Y
30 IF F=0 THEN GOTO 20
40 SPEN X,Y : GOTO 20
```

Statement 10 uses the SFILL command to fill the screen with white so the pen can see light anywhere on the screen. SFILL uses the current pen mode and fills the rectangle bounded by the Xmin,Xmax,Ymin,Ymax arguments given. After filling the screen, the pen mode is set to 2 which is required to plot black against white. Statements 20-40 constantly read the light pen and draw lines between successive coordinates returned by the pen. Running this program literally allows you to draw on the display screen! Use CTRL/C to halt the program. With suitable refinements, this program could be used for direct entry of pictorial data, curves, and other information. Traditional light pen uses such as menu selection are also easy to implement and are virtually instantaneous in response speed.

As one last experiment, try this program:

```
10 PENMODE 1 : SFILL 0,479,0,255 : PENMODE 2
20 SLTPEN F,X,Y
30 IF F=1 THEN GOTO 50
40 TONE : GOTO 20
50 TONE (255-Y),X/3 : GOTO 20
```

and then move the light pen around on the screen while it is running. While it doesn't really turn the MTU-130 into a musical instrument, it does illustrate the tremendous versatility that has been designed into MTU BASIC.

There are many more commands, functions, and techniques available in MTU BASIC, the IGL library, and other libraries. Hopefully this chapter has prepared you to read and understand the detailed descriptions that may be found in the BASIC section of the MTU-130 System Manual.

Chapter 5.

GETTING ACQUAINTED WITH THE EDITOR

The MTU EDITOR is a general-purpose, full-screen text editor for generating and modifying files of ASCII characters. It is well suited for editing programs and data. It has been specially designed to allow you to edit in place a text file larger than memory. You can use the Editor on a file that already exists or you can create a new file while in the Editor. It edits files in the ASCII form, including files LISTed from BASIC. The Editor is not only convenient and expedient but it is also flexible.

The Editor can best be understood through some hands-on experience. First you need to enter the Editor. After the prompt "CODOS> ", type in "EDIT" and the name of a file that does not exist such as "EXPERIMENT1.E":

```
CODOS> EDIT EXPERIMENT1.T
```

and press RETURN. When the screen redisplays, you will be in the Editor. The top three lines are called the Command/Status Area. The left hand portion of the top line is reserved for typing in commands. The two numbers in the right hand portion of the top line tell you where you are in the file. The second and third lines are blank for now but normally display error messages and prompts. The horizontal line running across the screen separates the Command/Status Area from the area reserved for your text file. Because you typed in the name of a file that does not exist, the Text Window, as this area is called, is blank. The tic marks on the horizontal line indicate where there are tabstops set. The cursor is in column 1 line 1 of the Text Window. Depress the CAPS LOCK key on the left side of the keyboard so any letters you type will be upper case. Now try typing in the following lines of a BASIC program so you can do some experimenting. After you type the first line, press RETURN to advance the cursor to the next line.

```
100 DIM A$(15): REM ALLOCATE SPACE FOR STRING MATRIX
110 FOR I=1 TO 15:READ A$(I):NEXT I
115 REM READ IN STRINGS
120 F-0:I=1
125 REM SET EXCHANGE FLAG TO ZERO AND SUBSCRIPT TO ONE
130 IF A$(I) = A$(I+1) THEN 180
```

Look at the keyboard for a moment. The keys set apart on the right of the keyboard are the cursor control keys. The arrows indicate in which direction you will be moving the cursor when you press those keys. They are called Cursor-Up, Cursor-Right, Cursor-Down, and Cursor-Left. The key marked HOME will always move the cursor directly to column 1 line 1 of the text window. Press HOME. Now press the Cursor-Down key three times. The cursor is at the beginning of the line that reads 120 F-0:I=1. Press the Cursor-Right key five times to position the cursor on the minus sign between F and 0. Press the equal sign key. You have changed the minus sign to an equal sign.

The key BACKSPACE in the upper right hand corner of the keyboard will also get your cursor back to a character you want to change while you are still typing the line. RUBOUT not only backs up the cursor but also erases the character and replaces it with a blank. Press the BACKSPACE key twice and strike over the F with an A. Now press RUBOUT followed by F.

Eliminate the equal sign altogether by pressing the DELETE key in the lower left hand corner of the keyboard. DELETE erases the character the cursor is on and closes up the gap. Press the INSERT key in the lower right hand corner of the key board, followed by the equal sign key, to put the equal sign back in. You only have to press the INSERT key once to be able to insert additional characters into the line at that spot. To tell the Editor you are finished inserting characters, press RETURN, BACKSPACE or any of the cursor control characters.

Cursor-Up, Cursor-Right, Cursor-Down, Cursor-Left, DELETE, BACKSPACE, RUBOUT and SPACE (the spacebar) have the ability to perform their respective functions more than once if you hold them down. Try holding each key down for a second. Retype whatever was deleted or rubbed out after you are all through and send the cursor HOME.

Hold down SHIFT and press DELETE. The entire line of text disappears and the remaining text scrolls up. Hold down SHIFT and press INSERT. A new empty line will appear before the line you were on. Retype the first line of your program on this blank line you have just created.

You have probably already noticed the eight boxes at the bottom of the display. The legends in these boxes are the commands you can use to expedite editing your file. The keys marked F1 through F8 at the top of the keyboard correspond to these legends. Press F1. As the legend in the first box indicates, you just moved the cursor out of the text window and into the command area above the horizontal line running across the screen. F1 now says TEXT. Press it and see what happens. You are back in the Text Mode. The legend box for F1 says COMMAND again.

The rest of the commands displayed in the legend boxes make up the " primary menu". Press OTHER. You are now in the secondary menu. Press OTHER again to return to the primary menu. Each of the commands in these menus is described in detail in the MTU Text Editor Manual. What you have seen here is merely a sample of some of the kinds of things you can do with the Editor. These and many other features are fully explained in the MTU Text Editor Manual.

Press QUIT in the primary menu to exit the Editor or, if you prefer, type it in up in the command area and press RETURN. The Editor will write your file to disk. If you had been editing a file that already existed, the Editor would have also updated the file when you QUIT. The Editor returns you to CODOS. If you actually wanted to run the BASIC program you have edited here, you could now execute BASIC and ENTER your file. Naturally, the Editor can be used to create or alter any sort of text file, not just BASIC programs.

REPORTING SOFTWARE BUGS

MTU-130 software has been carefully written and tested to assure reliable operation. Our users have consistently cited software reliability as one of the most desirable features of the MTU-130. However, in any set of programs as extensive as those supplied with your MTU-130 there are generally at least a few obscure "bugs". Unlike competitive ROM-based systems, the MTU-130's system software can be readily upgraded to correct any bugs which are discovered. At MTU, we ARE interested in hearing about any bugs which you may find in the system so that we can make corrections in future releases of the software.

Unfortunately, most of the apparent bugs reported to us turn out to be user oversights or improper usage. In order to avoid your spending unnecessary time preparing a report of a bug you have found please follow the steps shown below:

1. Please do not confuse reporting a bug with making a suggestion for an improvement. A bug is system behavior which is contrary to the published description. We also want to hear your suggestions for changes to improve the system but not on a bug report.

2. Recheck your program or your work.

3. Reread the appropriate section of the manual. Are you sure you haven't overlooked some requirement?

4. Once you are convinced that a bug does indeed exist, write a SIMPLE program or example that demonstrates it. If the bug disappears when you try to reproduce it in a simple program, it is probably not a system bug.

5. Once you are absolutely satisfied that you have discovered a genuine bug, send us a letter and a disk with a file illustrating the problem. If you send us a disk, we will try to send you a written reply within ten days. If a "fix" for the problem is readily available, we will include it on the disk when we return it to you. When sending us a disk and a letter, be sure to specify User Number of your MTU-130 (given by the CPUID program) and your phone number. Also be sure that you specify any additional hardware on your system or any software changes you have made from that on the distribution disk provided with your MTU-130 (even if they are in no way related to the problem being described).

HOW TO RUN THE SHOOT GAME PROGRAM

SHOOT is a simple graphics oriented simulation game that pits your cannon firing skills against your opponent's in a long range artillery war.

The object of the game is to hit your opponent's cannon with a well aimed shot from your cannon. To make life more interesting, there is an intervening mountain which your shot must clear and a fairly stiff wind blowing which may alter the shell's trajectory substantially.

You aim your shot by entering two numbers with a comma between them. The first number is the elevation (firing angle) from 0 (horizontal) to 90 (vertical). You may even exceed 90 degrees (and thus actually fire away from your opponent) if necessary to overcome a strong wind. The second number is the number of bags of gunpowder you wish to use. Obviously more powder will give the shell a greater velocity. Too much powder however will explode your own cannon! A typical charge is around 10 bags of powder.

To start SHOOT, load up MTU BASIC and then enter the BASIC command:

```
RUN "SHOOT"
```

If you are using the Demonstration Disk that come with the MTU-130, you can start SHOOT by pressing the GAMES function key in the main menu and then the SHOOT function key in the games menu.

In either case the program will be loaded, the Integer Graphics library will be loaded, and then it will run. After a several second delay, the terrain, two cannons, and the wind velocity and direction will be drawn along with a ? at the left screen edge. The player for the left cannon should enter his elevation and powder charge separated by commas (fractional values are accepted) and then a carriage return. The cannon then fires and the shell's trajectory is shown. Assuming there was not a direct hit, the player on the right will be allowed to fire his cannon next. This continues until one of the cannons is blown up or CTRL/C is entered for the firing data. The program may be restarted by typing RUN.

There are doubtless many improvements that can be made to this program to enhance its value as a game. Its real purpose however is to illustrate the use of graphics with MTU BASIC. You are encouraged to list the program, study it, and experiment with it.

FONTEDIT

The CODOS I/O Driver software uses a font table to obtain the 5 X 7 dot pattern for each ASCII character from 20 (hex) through 7F (hex). This font table is found in memory from FD50 to FFEF (hex) in bank 1. Changing the contents of this table will change what is displayed on the screen for the corresponding characters. The FONTEDIT program provides a simple means of changing this table, or creating a new one. This will allow you to create your own customized font.

The FONTEDIT program maintains its own font table which is stored in an integer array. (Important note! The font table referred to in the rest of this document refers to the font table maintained by the FONTEDIT program, not the font table currently being used by the operating system.) The dot pattern for any character in this font table may be displayed in a greatly expanded form. This expanded form may then be changed using the light pen or the cursor keys. Once the desired changes are made, the edited dot pattern may be stored back in the font table. After all the changes to the font table have been made, it can be saved to disk. The font table will be written to disk as a CODOS loadable file. This means a CODOS GET command can be used to load the edited font table into memory at FD50 (hex) in bank 1. Your edited font then becomes the current font used by the operating system.

THE MAIN MENU

When the FONTEDIT program is started up, you will be presented with the main menu. The functions available in the main menu are described below.

GET — Read the dot pattern for a character from the font table and display it in expanded form. You will be asked the character to be read.

NEXT — Perform the GET function above for the next character. The next character is the next sequential character from the most recent GET or STORE function.

CHANGE — Enter CHANGE menu. This menu allows you to modify the expanded dot pattern display. See CHANGE MENU below.

STORE — Store the dot pattern of the expanded character display in the font table. You will be asked for the character into which the pattern is to be stored.

LOAD — Load a font table file from disk. You will be prompted to enter the name of a file to load. If you need to specify a drive number with the name, enclose the entire name specification within quotes. Responding with just a carriage return will return you to the MAIN MENU without loading a file. The supplied STDFONT.Z file contains the standard CODOS font table.

SAVE — Save the font table to disk. You will be prompted to enter the name of a file in which to write the font table. If you need to specify a drive number with the name, enclose the entire name specification within quotes. Responding with just a carriage return will return you to the MAIN MENU without saving a file. If the specified file exists, you will be asked if you wish to overwrite the existing font table in this file.

DISPLAY — Display the dot patterns for all of the characters normal size in the upper left corner of the display.

OTHER — Jump to the second part of the MAIN MENU.

The second part of the MAIN MENU has these functions:

CLR FONT Clear the font table.

OPEN 0 Open drive 0.

OPEN 1 Open drive 1.

QUIT Halt the FONTEDIT program and return to BASIC.

TO CODOS Halt the FONTEDIT program and return to CODOS.

EXIT Jump back to first part of MAIN MENU.

THE CHANGE MENU

When you select the CHANGE function, you will be asked whether you want to use the light pen or cursor keys to edit the dot pattern. Using the light pen is easier, but if you wish, you can use the cursor keys. Once you have made your selection the CHANGE MENU is presented which has the following functions:

ON Turn on the selected "dot", or box, in the expanded character display. When using the light pen, a "dot" is selected by pointing the light pen to the corresponding box in the array of boxes on the right side of the display. You should have the light pen pointed to the box when the F1 key is pressed. When using the cursor keys, move the blinking cursor to the "dot" you want turned on, then press the F1 key.

OFF Turn off the selected "dot", or box, in the expanded character display. When using the light pen, a "dot" is selected by pointing the light pen to the corresponding box in the array of boxes on the right side of the display. You should have the light pen pointed to the box when the F3 key is pressed. When using the cursor keys, move the blinking cursor to the "dot" you want turned off, then press the F3 key.

DESCNDR Toggle the Descender flag on or off. Each time the F4 key is pressed the flag will be switched to the other state. Turning the descender flag on will cause the character to be displayed two dots lower than normal. This is naturally intended for creating lower case letters.

J DOT Toggle the J DOT flag on or off. If on, the J Dot flag will cause a dot to appear appropriately for the lower case "J" character (which doesn't quite fit in a 5 X 7 dot matrix). The J Dot flag will not have an effect unless the Descender flag is also on.

CLEAR Clear the expanded character display, i.e. turn all the "dots" off.

EXIT Jump back to the MAIN MENU. If you want your changes to be placed in the font table, you must execute a STORE function in the MAIN MENU. The changes made while in the CHANGE MENU affect only the expanded character display.