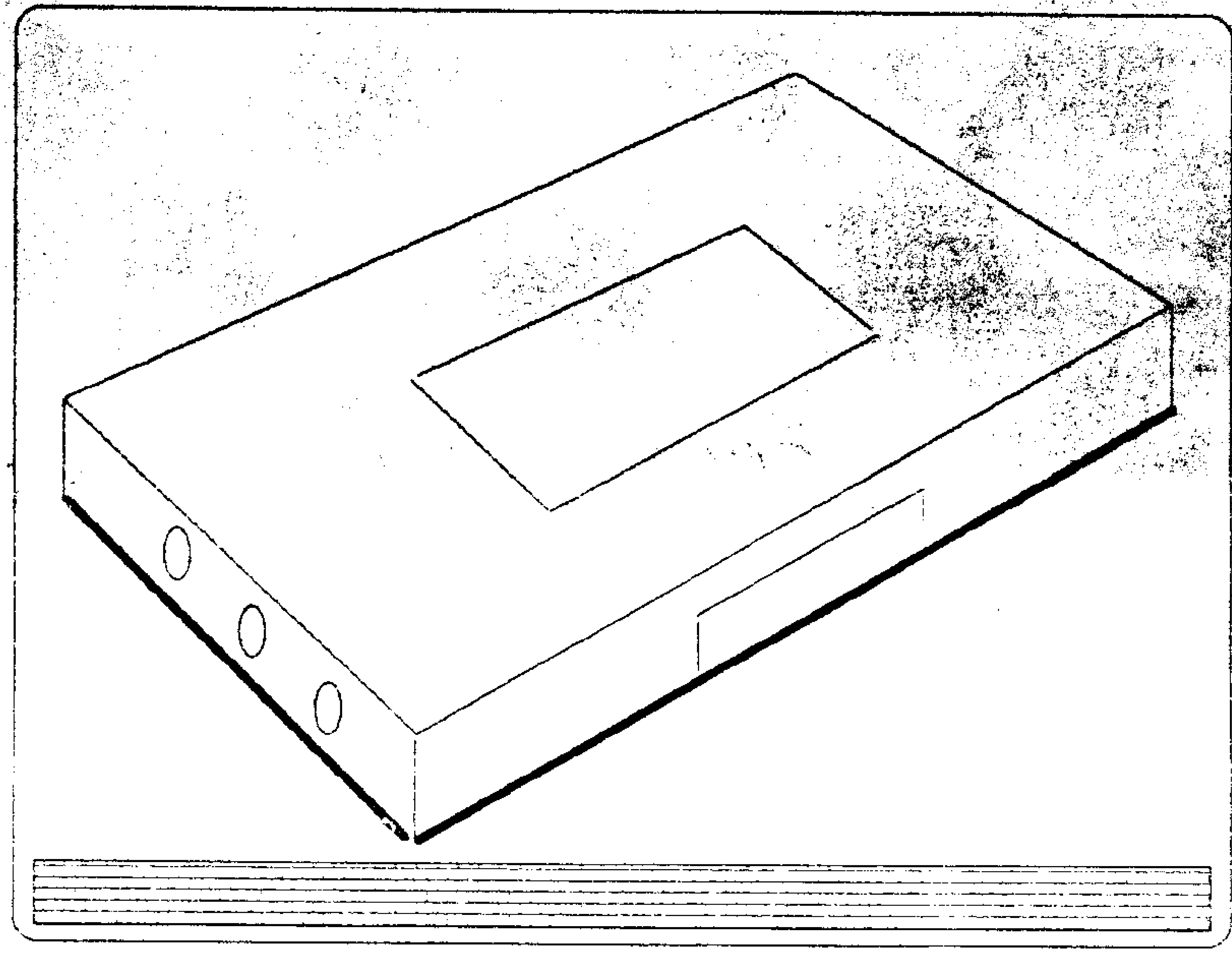
Foundation

128 K Memory Card



This document descibes the Foundation 128K Memory Card and the optional Device Service Routine firmware.

128K Card Documentation

CONTENTS

Introduction

The Foundation 128K card provides 128K bytes of high-speed random-access memory for the Texas Instruments 99/4A computer. This memory is addressed by an assembly language program as four banks of 32K bytes each, with inter-bank switching performed using Communications Register Unit instructions. Alternatively, with the optional Device Service Routine firmware, the upper three banks of memory are available for use by an application program as if they were either a single file, "MEM96", or as if they were three files named "MEM96A", "MEM96B", and "MEM96C". In this mode of operation, the upper banks of memory can be used to greatly expand the data-handling capabilities of the 99/4A while using the lowest bank to store either programs or data.

Specifications

Amount of Memory	131,072 bytes
Organization	4 banks of 32KB each mapped as >2000 to >3FFF and >A000 to >FFFF
Memory Technology	4164 64Kx1 dynamic RAM
Memory Speed	200 nsec. at chip level. full bus speed at bus level
Support Circuits	series 7400 low power Schottky TTL logic and TMS4500A VLSI memory controller
Software	MEM96 file emulation with DSR option

Programming the 128K Card

Using the 128K Card with the DSR Option

Most users of the 128K card will find that its value is greatly increased when it is coupled with the firmware provided by the DSR option. Using the DSR option, the memory appears to the user almost like two different devices. One of them is designed to be used as 32KB of "normal" memory; that is, both programs and data are loaded into it and programs treat it pretty much as they would memory from any manufacturer. The other pseudo-device is called MEM96, and it is the main subject of this section.

MEM96 is provided as a way of circumventing some very basic limitations that are built into the TI 99/4 computer family. The TI 99/4 computers are only capable of addressing 64KB of memory. When TI designed the machines, they carefully laid out what the requirement for memory would be and allocated all of the space that was available. Nevertheless, some people (including you) either need more memory already or can see that they will in the not-too-distant future.

What Foundation has done is to provide "firmware" (more on this later) that allows 128KB of memory to appear to a program as if it is both "normal" memory and a group of files that are available just as if they were files on some other device such as a disk drive. Naturally, since the "files" are really in very high-speed memory, access to them is much faster than it would be to a file on a disk. How much faster? This varies depending on exactly what you are doing, but it ranges from being a few hundred to a few thousand times faster. This means that some jobs that would take an entire afternoon on a disk-based system (for example, sorting a large file of names and addresses), can complete in a few minutes executing in MEM96.

What exactly is MEM96 and how do you go about using it? To start with, MEM96 is a file that you access just as you would any other file. That is, you open it, read or write to it, and close it when you are finished.

128K Card Documentation

The main goal of providing MEM96 was to make additional storage capability available to programs. The model chosen was that of a fixed-length random-access file in update mode. If opened in other modes, MEM96 may or may not return an error condition, but results are unpredictable. Records may be any size from 4 to 255 bytes, and may be either "display" or "internal" format. No notification of an end-of-file condition is given, but if a program attempts to access record numbers that are too large, a DSR error code will be returned. It thus becomes important to calculate the largest record that will fit in a file somewhere toward the beginning of your program, and to check that the record number you specify is in-bounds before each access.

This paragraph is going to tell you how to calculate the maximum record number. It leads up to a simple formula that you can just plug in at the beginning of a program, so you can skip it for now if you want to. In order to calculate the maximum record number, you should think of memory as consisting of 8KB blocks. The MEM96 software and other follow-on products from Foundation use the highest 8KB block of memory for temporary storage, so MEM96 actually has 88KB available for file storage. (You start with 128KB; "normal" memory takes up 32KB, and the MEM96 software takes up 8KB as discussed.) This works out to eleven 8KB blocks. When you open MEM96 as a file, you specify a record size. If that record size doesn't divide evenly into 8192, anything left over is not used for storage. To calculate the maximum record number, just figure out how many records will fit evenly into 8192 bytes and then multiply that number by eleven. That is, somewhere toward the beginning of a program, insert the statement:

```
30 MAXREC=11*INT(8192/RECSIZE)
```

where you have previously defined RECSIZE to be the record size and your OPEN statement looks something like:

```
100 OPEN #1:"MEM96", FIXED RECSIZE, INTERNAL
```

Following this, you should check before each read or write to MEM96 that the record number is less than MAXREC. For example:

```
200 IF RECNUM>MAXREC THEN 1000
210 INPUT #1: I,A$
.
1000 REM ERROR - RECNUM TOO LARGE
```

Everything so far has been talking about using the entire upper three banks of memory as one large pseudo-file called MEM96. Often, it would be more convenient to consider it as a few smaller files so that, for example, you can have different record sizes for different purposes. To accomodate this, there are two different ways that you can treat the upper three banks. One of them is what we have been discussing so far, that is, to treat all three banks as one large file called MEM96. The other approach is to treat them as three smaller files called MEM96A, MEM96B, and MEM96C, respectively. MEM96A and MEM96B are each 32KB, while MEM96C is 24KB. Obviously, if you are using MEM96A,-B, and -C, you should not use MEM96 or vice versa - otherwise you could find yourself accidentally overlaying a record.

There is one other pseudo-file you should know about. It is called MEMINIT, and it is important that you call it at the beginning of each program. It can be called using the following sequence:

```
10 OPEN #1: "MEMINIT", FIXED 10
20 CLOSE #1
```

MEMINIT initializes a "first-time flag" for the MEM96 software. If it is not set, the screen may vanish or strange graphics may appear when you try to run your program. You will be tempted to ignore MEMINIT because nine times out of ten you can get away with it. However, it can be a confusing bug to track down on that tenth time, and the best practice is to just make it a habit which you do automatically.

The easiest way to illustrate the workings of MEM96 is with an example. Here is a sample program that opens MEM96, performs some reads and writes to it, and then closes it:

128K Card Documentation

Background Technical Information The 128K card behaves like "normal" memory in that each of its four banks is mapped into four segments of 8K bytes each. These segments are located at hex addresses >2000 to >3FFF (low segment) and >A000 to >FFFF (high segments). Thus, to a user application program, the existence of the additional three banks is ordinarily invisible. When an application program is ready to switch from one bank to another, it sets the appropriate CRU bits according to the following table:

CRU Address	Bank Number	
>1E02	>1E04	
0	0	0
1	0	1
0	1	2
1	1	3

That is, the binary equivalent of CRU bits 1 and 2 offset from CRU address >1E00 selects the bank number. The following brief program segment illustrates this:

```
LI      R12,>1E00    SELECT CRU BASE ADDRESS
SBO     1            SET CRU BIT >1E02
SBZ     1            CLEAR CRU BIT >1E02
```

The above example would select bank 1. To execute properly, this code segment would need to appear in an address space outside of the 32K of expansion memory. For all practical purposes, this means that the bank-switching code needs to execute in a TI Mini-Memory Module in the 4K of RAM located between >7000 and >7FFF. It is also possible to write a program that loads bank switching instructions into the CPU PAD area from >8300 to >83FF, but this means a complex routine to save and restore the contents of PAD before and after using it.

On power-up and whenever the console issues a reset signal to the expansion box (e.g. whenever a command cartridge is inserted or removed), the 128K memory resets itself to bank 0. Other than that, no power-up initialization routines are initiated by the 128K device service routine. However,it should be noted that when the console is powered down, the noise on the ribbon cable to the expansion box will usually corrupt the contents of some memory locations.

```
100 REM  MEMDEMO
         -------
110 REM  THIS SHORT PROGRAM DEMONSTRATES HOW MEM96 CAN BE USED TO
                 WRITE AND READ RELATIVE RECORDS
120 REM  THE PROGRAM STARTS BY TYPING "OUTPUT ?"
130 REM  IF YOU TYPE "Y" (FOR YES), IT WILL PROMPT YOU TO ENTER
                 RECORDS UNTIL YOU GIVE IT A RECORD NUMBER
140 REM  THAT IS LESS THAN ZERO. IF YOU TYPE "N", IT WILL ASK YOU
                 TO ENTER RECORD NUMBERS AND THEN WILL PRINT
150 REM  THE RECORD STORED AT EACH RECORD NUMBER.

170 OPEN #1:"MEMINIT",FIXED 10
180 CLOSE #1
190 REM  INITIALIZE MEM96 SOFTWARE
200 RECSIZE=64
210 MAXREC=4*INT(8192/RECSIZE)
220 REM  WOULD BE 4*THIS EXPRESSION FOR MEM96B OR 3* IT FOR MEM96C
230 PRINT "OUTPUT";
240 INPUT SWITCH$
250 IF SWITCH$="Y" THEN 280
260 IF SWITCH$="N" THEN 280
270 GOTO 230
280 OPEN #1:"MEM96A",FIXED RECSIZE,INTERNAL,RELATIVE
290 PRINT "RECORD NUMBER"
300 INPUT RECNUM
310 IF RECNUM<1 THEN 430
320 IF RECNUM<MAXREC THEN 350
330 PRINT "RECORD NUMBER TOO LARGE"
340 GOTO 290
350 IF SWITCH$="Y" THEN 390
360 INPUT #1,REC RECNUM:A$
370 PRINT A$
380 GOTO 290
390 PRINT "TEXT ";
400 INPUT A$
410 PRINT #1,REC RECNUM:A$
420 GOTO 290
430 CLOSE #1
440 END
```

A sample run of this program is shown on the next page.

Here is a sample run of the program from the previous page:

```
RUN
OUTPUT? Y
RECORD NUMBER
? 1
TEXT ? THIS SHOULD GO IN RECORD ONE
RECORD NUMBER
? 3
TEXT ? HERE IS DATA FOR RECORD 3
RECORD NUMBER
? 100
TEXT ? ANOTHER RECORD TO BE STORED
RECORD NUMBER
? 1000
RECORD NUMBER TOO LARGE
RECORD NUMBER
? -1

** DONE **

>RUN
OUTPUT? N
RECORD NUMBER
? 1
THIS SHOULD GO IN RECORD ONE
RECORD NUMBER
? 100
ANOTHER RECORD TO BE STORED
RECORD NUMBER
? -1

** DONE **

>
```

FOUNDATION COMPUTING

Disk File Emulator

Table of Contents

Program License Agreement

YOU SHOULD READ CAREFULLY THE TERMS AND CONDITIONS SET FORTH IN THE PROGRAM LICENSE AGREEMENT . POSSESSION OF THIS PRODUCT INDICATES YOUR ACCEPTANCE OF THESE TERMS AND CONDITIONS.

Disk File Emulator

Overview

Organization of Document

Every "how-to" manual has to make some choices as to how information ought to be presented. This one is aimed at people that have had their computer for a while, that have just received the disk-file emulator firmware for the 128K card, and that are trying to figure out how to use it. It should be read in a comfortable chair not too far from your computer, because a lot of the things that it talks about will be clearer if you can just try them out on the machine. The easy things are covered first, and all the details of how you write programs to use the disk emulator are pushed toward the back.

Disk Emulator Functions

Just what does the disk file emulator do? The easiest way to approach this is by relating it to something you are already familiar with, the standard "MEM96" software that is built into every 128K card. MEM96 provided one function for your computer: it allowed you to treat the extra three banks of memory as files of relative records that programs could write to or read from. The disk file emulator provides three new functions: (1) now you can use the upper three banks of memory to save or load programs, (2) you can treat them as sequential files, and (3) you can interact with a built-in program to keep track of memory files.

It does this by providing a new pseudo-device called "DSKX". DSKX is like MEM96 in that it provides three files, each corresponding to one bank of memory. It is more general in that these files can be given any file-name; for example, you could use "DSKX.MYFILE" to refer to memory bank 2. Like MEM96, files in DSKX do not interfere with the use of bank 0 "regular" memory.

The files in DSKX correspond very closely to files on a TI disk, so that application programs like Multiplan or TI-Writer can send their output into memory files. Also, for many application programs including most of TI's Solid State Software, you can save and load data from DSKX. Thus, for example, you can save word processing files into DSKX and reload them more quickly than you could from disk. Also, you can store source text that you have created with the Editor/Assembler into a DSKX file and assemble it from there. Section 5 discusses ways to do this.

Getting Started

Just a brief word about installing or reinstalling the 128K Card . . . do remember to power down the expansion box and wait a full two minutes before inserting any expansion module. You really can damage modules if you hurry too much.

From here on, the assumption will be that you've installed your 128k card and that your computer is available to try out examples. Here's a quick one: turn on your Expansion Box then your 99/4A console and press 1 to go into Basic. Type DELETE "MEMINIT". The screen should then, without complaining at all, display a Basic prompt. If it succeeds, the odds are that your 128k card survived shipping and everything is working. If it didn't, you either typed "MEMINIT" in small letters or you should call Foundation about a possible hardware problem. What you just did was initialize the 128k card. For now, don't worry about what this was supposed to prove; it will be explained later.

Loading and Saving Basic Programs

Loading and Saving from a Disk File

One of the first things that people learn to use their disks for is loading and saving programs. Let's relate DSKX to a disk by first recalling how you would save a program to a disk file. First, go into Console Basic. Here is a sample session that will save and reload a short program:

Disk File Emulator

Chaining Programs in Extended Basic

At this point, you should be in Extended Basic. Type "OLD DSKX.TEST" and then list the file you have in memory. It should be there, intact. The point that is being illustrated is that DSKX is a handy place to put data when you are switching from one application to another . . . in this case, you were switching from Basic to Extended Basic. Another useful case is printing Multiplan output into a file that can be edited by TI-Writer, but we'll get to this later.

Extended Basic offers a statement that makes it possible to split one large program into several smaller ones—the "RUN" statement. Combined with the fast load from DSKX, this is a very powerful capability. Here is how you use it: (again, this assumes that our short test program is still in main memory)

```
>LIST
  10 FOR I=1 TO 3
  20 PRINT I
  30 NEXT I
  40 STOP
>5 PRINT "THIS WAS LOADED FROM DSKX"
>SAVE "DSKX.TEST"
>40 RUN "DSKX.TEST"
>RUN
THIS WAS LOADED FROM DSKX
     1
     2
     3
THIS WAS LOADED FROM DSKX
     1
     2
     3

* READY *
>
```

What you just did was write a program that automatically ran a second program. The first group of messages above was displayed by the program in memory. When it got to line 40, it loaded and ran DSKX.TEST, which printed out the second group of messages. You can use the RUN statement to build up complicated systems of programs that interact with one another. One warning is that Extended Basic takes up time initializing a program when it is run, so for long programs you will not get the near instantaneous response that you just saw.

Warning About Size Constraints

One restriction needs to be mentioned. The three extra banks of memory provided by the 128k card offer 32K, 32K, and 24K of memory, respectively. (The upper 8K of the last bank are used by the disk emulator software). An Extended Basic program can grow to more than 32K of combined stack and program space, so you may come across a program that cannot be stored in DSKX. Please send us a copy if you do; none of the test programs that we put together ran into any problems.

Memory Manager Module

Introduction

Disk File Emulator

The line across the bottom of the screen shows memory management commands that are available. 'Delete" will delete a file. Try it now. Type "D" and when the computer asks you for a file number, type "3". The directory will immediately change to show that file 3 has been deleted. Now try typing "R" for "Rename". Select file 2 and press the <ENTER> key. In response to "New Name?", type some legal file name of less than eleven characters, for example FILE2<ENTER>. The directory entry for file 2 should change to match the new name.

The last command, "Init", will delete all three files and re-initialize DSKX. Try typing "I". The computer will ask you "Really?", and if your reply is a Y for Yes, it will delete all three files just as if you had typed "DELETE MEMINIT" from Basic. Go ahead and delete them; we will have no further need of this test program.

Finally, to exit from MMM, hold down the <FCTN> key and the 9 key at the same time. Some of the plastic strips that come with software packages label this key combination as "ESCAPE", which is what you want to do. Incidentally, if you want to escape in the middle of a command, just type <FCTN> 9 before you hit <ENTER>.

Lastly, notice that when you return to Basic the screen is exactly as you left it before you entered MMM. If you were in the middle of some complicated display, this can be a handy thing.

Convenient Places to Access MMM

It's easy to forget what you had previously stored in DSKX, so MMM is a very convenient tool to have available. You'll be glad to discover that you can access MMM from within programs like TI-Writer and the Editor/Assembler. Any time that a program asks you for a file name, you can usually type "MMM" or "MMM." and take a look at the contents of DSKX. That is, you're not limited to accessing MMM by typing DELETE "MMM" from Basic. Any I/O operation except a CLOSE will transfer control to MMM. In many programs, the most convenient way to get into MMM is by pretending that you want to do a printout to MMM instead of to the RS232 interface.

The main restriction on doing this is that MMM formats its display for a thirty-two character screen. If you run MMM from a program that sets up the screen for forty characters (TI Writer, for example), the MMM display will be hard to read.

Using Sequential Files

Or, some ways that standard programs can use sequential files.

TI Writer

TI Writer is a well-behaved program in that it lets you specify a device name as part of a filename. This means that you can load and save TI-Writer files to DSKX. All you have to do is specify "DSKX.filename" instead of "DSK1.filename" on the command line. For example, there is a document named "FORMATDOC" on the master diskette that TI Writer is distributed with. Load this file into memory by typing "LF" on the TI-Writer command line and specifying "DSK1.FORMATDOC" when you get the "LOAD FILE, enter filename:" prompt. Now save it into DSKX by doing a command of "SF". TI-Writer will prompt you with "SAVE FILE, enter filename:". At this point, all you need to do is type "DSKX.FORMATDOC".

The other way that you can make use use of DSKX is by printing to it. This can be done either from the Text Editor or the Text Formatter. Along these lines, you might try printing to device "MMM" to see the directory. Though the directory is visible, screen formatting is incorrect because TI-Writer uses 40 columns rather than the 32 that MMM requires.

Disk File Emulator

In order to use any file, whether it is on diskette or on the 128K card, you need to do three things:

1) "open" it, that is, tell the computer where the file is located, what kind of file it is, etc.
2) "input" or "print" information to it, and
3) "close" it, that is, tell the computer that you are through with using it.

There are statements in TI Basic and Extended Basic that do each of these things.

OPEN

Before using a data file stored on the 128K card, you need to describe the characteristics of the file to the program. The OPEN statement for the 128K card has the following general form:

OPEN #file-number:"device-name" [,file-organization] [,file- type] [,open-mode] [,record-type]

For example,if you intend to read and write thirty character records to a relative file named DSKX.MYFILE, a file OPEN statement might look like this:

100 OPEN #1:"DSKX.MYFILE", RELATIVE, DISPLAY, UPDATE, FIXED 30
DSKX provides three files that correspond to the upper three banks of memory on the Foundation 128K card. When a file is opened on DSKX, the lowest available file slot is used. For example, starting with a clean slate, if you opened three files they would be placed in banks 1,2, and 3 in that order. If you then deleted the files in banks 1 and 3, the next file created would be placed in bank 1. The only difference between the files that are created in different banks is that banks 1 and 2 provide up to 32Kb of storage while bank 3 provides up to 24Kb.

file-number - The file-number (1 through 255) is assigned to a file by the OPEN statement. It is a number that Basic uses to keep track of open files, and it will be used in all later statements that refer to this file, e.g. INPUT, PRINT, and CLOSE statements. You can assign any file number that is convenient for you. Most programs number the first file as 1, the second as 2, etc.

device-name - Most of the cards that plug into the TI Peripheral Expansion Box have one or more "device names" that software uses to refer to them. For example, if you have a disk, you know that you refer to your first disk drive as "DSK1". Similarly, you use the device name "DSKX" to refer to the disk-file emulator psuedo-device. If you are using DSKK in a program that also uses the "MEM96" pseudo-device provided with all 128K cards, you should be aware that these files will overlay each other. That is, DSKX will place a file in bank 1, 2, or 3 of the 128K card. MEM96A directly corresponds to bank 1, so any changes made to MEM96A will also change the contents of the first DSKX file. MEM96A,-B, AND -C can coexist with DSKX as long as this overlap is taken into account. It is also possible to refer to device MEM96 without a suffix to create one large file that spans all three memory banks. This large file is restricted to being a file of fixed length relative records, and it is easy to get into trouble by using both MEM96 and DSKX.xxx in the same program.

file-organization - RELATIVE: Records in a relative file can be read in any order. This is in contrast to a sequential file, where records can only be read or written one after the other. This has two implications. First, when we get to the "INPUT" and "PRINT" statements, you will see that for relative files you need to tell the computer which record it is that you want to access. Second, all the records in a relative file need to be the same size. You must decide what size to use in advance, and tell the machine what your decision is. (See FIXED, below.) Relative files on the 128k card are pre-extended to their full length, since an entire bank of memory is allocated for each file. SEQUENTIAL: For sequential files, the system software keeps track of where you are in the file so that you do not need to specify a record number on reads or writes. Sequential files are read back in the order that they were written. Files on DSKX or MEM96A,-B, or -C may be opened for sequential access. As noted above, pseudo-device MEM96 (with no suffix) can only be opened for RELATIVE access.

Disk File Emulator

Taking things one step at a time, first consider the INPUT statement. The INPUT statement has the following format:

INPUT #file-number [,REC record-number]: variable-list for a relative file, or

INPUT #file-number: variable-list for a sequential file.

You always need to specify the file-number and variable-list, and may optionally specify a record-number for relative files.

> file-number - The file-number is the number that was assigned to this file in the OPEN statement, and the discussion from the OPEN statement applies here as well.
>
> record-number - With relative files, you may tell the computer which record you want to retrieve. The record number varies from 0 up to the largest record that will fit into 32Kb or 24Kb. If record-number is left blank, the next sequential record will be read.
>
> variable-list - The INPUT statement reads one record from a file in DSKX into the variables that you specify as "variable-list". These should correspond directly to the variable-list that you used to write the record. That is , if you wrote out a number followed by a character string, variable-list should consist of a numeric variable followed by a character string variable.

Suppose that you opened DSKX.F1 with the following statement:

10 OPEN #1: "DSKX.F1", RELATIVE, INTERNAL, UPDATE, FIXED 64

and then stored a list of customer names and telephone numbers, one name to each record. Then later in the same program or in a different program, you could input the name and phone number stored in the tenth record with the following line: 100 INPUT #1, REC 10: A$

PRINT

The PRINT statement is used to write data into DSKX files. It has the following format:

> PRINT #file-number, REC record-number: print-list for relative files, and
>
> PRINT #file-number: print-list for sequential files.

You always need to specify the file-number and print-list, and may optionally specify a record-number for relative files.

> file-number - The file-number is the number that was assigned to this file in the OPEN statement, and the discussion from the OPEN statement applies here as well.
>
> record-number - For relative files, you may tell the computer which record you want to store. The record number varies from 0 up to the largest record that will fit into the current file. If record-number is left blank, the next sequential record will be written.
>
> print-list - The print-list is the list of variables that you want to place into DSKX.xxx . It consists of a list of numeric or string expressions with items in the list sepa- rated by commas or semicolons.

For example:

PRINT #1, REC 110: .06*COST, "Sales Tax"

Disk File Emulator

DSKX	Device providing three named files, each referred to as DSKX.filename. Files on DSKX overlay MEM96A, -B, and -C, and have the same capabilities.
MMM	Memory Management Module. All I/O operations to pseudo-device MMM start up a program that can be used to interactively manage the contents of DSKX.
MEMINIT	All I/O operations to MEMINIT delete all files from DSKX and initialize internal variables to a known starting point.

Error Messages

All error messages generated by an application program accessing DSKX come from bits being set in the error field of the DSR flag/status byte. Basic and several other programs refer to these as "I/O Errors". The following errors may be reported:

2	Bad Open Attribute. One or more OPEN options are illegal or do not match the stored file characteristics.
3	Illegal Operation. Input/Output command not valid. Also used as a catchall for errors that do not fall into some other category,
4	Out of Space. No space left on DSKX or out of directory entries.
5	End of File. Attempting to read past the end of a file.
6	Device error. Internal error in DSKX or directory corrupted by file operations.
7	File error. Requested file not found.

Program License Agreement

Foundation Computing, Licensor, provides this program, referred to hereunder as disk-file emulator software or Licensed Program, and licenses its use. You assume responsibility for the selection of the prograr to achieve your intended results, and for the installation, use, and results obtained from the program.

1. You may use the Licensed Program on a single machine at a time.

2. You recognize the proprietary nature of the Licensed Program and materials and agree to preserve protect Foundation Computing's interest herein.

3. *Title* - Title and Ownership to the Licensed Program resides in Foundation Computing.

4. Term - The License is effective until terminated. You may terminate the license at any time t the Licensed Program and Materials together with all copies. It will also terminate if you fail t with any term or condition of this agreement. You agree upon such termination to destroy th Program and Materials together with all copies in whatever form. Within thirty days after te Licensee shall certify in writing that through its best efforts and to the best of its knowlec part or in whole, of the terminated and/or discontinued Licensed Program and Material re been destroyed.

5. The disk-file emulator software is being made available "as is", without any warran pressed or implied, including, but not limited to the implied warranties of merchanta particular purpose. The entire risk as to the quality and performance of this softwa the program prove defective, you assume the entire cost of all servicing, repair, or

6. Foundation does not warrant that the disk-file emulator software will meet your r operation will be uninterrupted or error free.