6809 FLEX™ Adaptation Guide

Technical Systems Consultants, Inc.

6809 FLEX™ Adaptation Guide

™FLEX is a trademark of Technical Systems Consultants, Inc.

DISCLAIMER

PLEASE READ BEFORE OPENING THE DISKETTE ENVELOPE!

This version of FLEX is not for beginners. It is assumed that the user has a good knowledge of assembly programming and of his hardware. Technical Systems Consultants, Inc, cannot and will not be held responsible for the adaptation of FLEX nor for the operation of the resulting product. FLEX has been proven in the field for over two years and the adaptation procedure has been tested on numerous systems. For these reasons:

NO TECHNICAL ASSISTANCE FOR THE ADAPTATION OF FLEX WILL BE PROVIDED BY TECHNICAL SYSTEMS CONSULTANTS, INC!

Knowing this, read the FLEX Adaptation Guide thoroughly and make a decision as to whether or not you are capable of performing the adaptation without assistance. If not, Technical Systems Consultants, Inc, will refund your money (less shipping and damage charges) if you return the manual and diskettes IN THEIR UNOPENED ENVELOPE. Once the diskette envelope has been opened, no returns will be accepted!

TABLE of CONTENTS

TABLE of CONTENTS (Continued)

1.0 INTRODUCTION

1.1 Important Documents

There are two very important documents which ABSOLUTELY MUST be read before continuing. The first is a yellow disclaimer document and the second is a green copyright information sheet. They should be the first two sheets of this manual. These two documents are perhaps the most important reading in the entire set of FLEX documentation and it is imperative that the user read and fully understand them before attempting any adaptation of FLEX.

1.2 What You Received

The general version of FLEX should include the following items:

1) FLEX Adaptation Guide
2) FLEX User's Guide
3) FLEX Advanced Programmer's Guide
4) Text Editing System Manual
5) Assembler Manual
6) Two diskettes sealed in an envelope
7) Yellow Disclaimer Sheet
8) Green Copyright Information Sheet
9) Loose-leaf binder

If you are missing any of these items, contact our order department immediately.

1.3 System Requirements

In order to perform the adaptations and to run FLEX, there are certain hardware and software or firmware requirements. Specifically they are:

1) Computer system with 8K of RAM at $C000 and at least 12K of RAM beginning at location $0000.
2) A system console or terminal such as a CRT terminal or printer terminal.
3) A single 8 or 5 1/4 inch disk drive with controller capable of running soft-sectored format with 256 byte sectors.
4) A monitor ROM or some program affording the ability to begin execution at any desired point and to enter code into the system. This coding may be done by hand, but some sort of storage method such as cassette or paper tape would be helpful. Additionally, since the user is required to write several routines, an editor/assembler package will make the adaptation much easier.

1.4 How to use the Adaptation Guide

This manual contains all of the necessary instructions for the adaptation of FLEX to any system meeting the requirements listed above. This adaptation is not a simple step, however, and you may save some headaches by beginning the process in the correct order as explained shortly. Before attempting to install FLEX, the manuals should be read and understood. A good order for reading the manuals is to read section 2 of this Adaptation Guide titled 'The FLEX Disk Operating System', then read the FLEX User's Guide (not necessarily reading all the command descriptions therein), and then read the remainder of this Adaptation Guide. After reading all this material, be sure to re-read the yellow disclaimer sheet and decide whether you are capable of performing the adaptations.

One suggestion that will be made often in this manual is to keep things simple. Since you are starting from the ground up, it will be best to keep all routines simple at first. Once things are running in the simplest, lowest level form, it will be much easier, using the now available FLEX facilities, to improve the routines and add new devices.

2.0 The FLEX DISK OPERATING SYSTEM

2.1 Disk Operating System Concepts

For those users who are new to disk operating systems, it might be appropriate to briefly discuss some basic concepts. There are two major reasons to have an operating system. First is that it relieves the programmer from the task of writing the low-level I/O and file management routines each time a piece of software is written. That work has all been done by the authors of the operating system allowing the user to concentrate on his application software. The second major reason is that it removes all hardware interfacing from the application program. This, of course, makes application programs shorter and easier to write, and has the added advantage of making the application program transportable to any computer system running the same operating system. The advantages of software transportability should be immediately obvious.

The FLEX Disk Operating System was originally designed to support a single-user system with floppy disks. As we shall see however, it is not restricted to floppy disks only. FLEX contains routines to handle all the "low-level" tasks associated with maintaining data on disks. Rather than having to write programs which must keep track of what data is where on the disk, worry about how much space is available, control the selection of drives, seek to tracks, load the head, etc., the programmer can let FLEX take care of these duties and merely keep track of his data by named files. A "file" is simply a collection of data which is stored on the disk under a unique "filename". It can contain anything from a source listing to a collection of data from a BASIC program to the text for a letter. FLEX maintains a directory on track 0 (the outermost track) which contains the name and starting address (track and sector number) of each file stored on the disk. The user program can call on FLEX routines to create such files, write data to them, read data from them, delete them, load them into memory, rename them, etc. FLEX also has several user-accessible "convenience" routines which have nothing to do with the disk, but allow the user to do things like print a string, get a decimal number from the input line, classify a character, etc. In general, FLEX is a very powerful tool which saves application programs (and programmers) from doing a lot of housekeeping chores.

2.2 A Brief Overview of FLEX Adaptation

To make things more clear as you progress through the adaptation procedure, let's go through a brief summary of the steps involved. The whole idea of the adaptation process is to perform the necessary steps to interface FLEX to your particular hardware. The main body or core of FLEX does not care what kind of hardware it is running on. It communicates with the actual hardware through two packages of routines which must be user written and which are unique for various hardware configurations. The core of FLEX doesn't change - only these two hardware interface packages. These packages are a set of low-level disk driver routines and a set of console or terminal I/O routines. Throughout the manual we will refer to these packages as the DISK DRIVERS and the CONSOLE DRIVERS respectively. As an example, when FLEX wants to read a sector of information from the disk, the core of FLEX doesn't care what kind of disk it is or where it is located. The core of FLEX simply asks the disk driver package to read sector number 4 on track number 18 and expects it to do whatever it must to read that sector. Thus the heart of the adaptation process is writing the routines for the Console Driver and Disk Driver packages.

(1) The first step is to write "Console I/O Driver" and "Disk Driver" routines for interfacing to the system console or termina l and to the disk controller. The development of these routines may be carried out in a number of ways. If the user has access to another 6809 development system with editor and assembler, he should by all means take advantage of that power. Alternatively, it may be necessary to write the routines on the system being adapted. This implies that either some sort of tape editor and assembler must be used or the routines must be hand-assembled into object code. In either case, it is convenient to have a mass storage device on-line to save and load the drivers during development.

(2) Once the drivers are written, they must be fully tested. A program is provided to aid in testing the Disk Drivers.

(3) After the drivers have been proven functional, a short program is supplied which will allow FLEX to be loaded in from disk. The FLEX on disk has no drivers, but when loaded into memory will make use of the resident, user supplied drivers. Once this FLEX is in memory and running, any of the features of FLEX can be utilized. For example the disk editor and assembler can be used to develop the remaining software required for a complete system.

(4) The user will now save his drivers on disk and append them onto the core of FLEX to produce a complete version of FLEX on the disk.

(5) In order to load the full version of FLEX, a couple of bootstrap loader routines are required. Once these are written and tested, the FLEX system is basically complete and may be easily booted up at will.

(6) There is one further routine that must be user supplied which communicates directly with the disk hardware. That is the "NEWDISK" routine which initializes a blank disk to the format required by FLEX.

When the NEWDISK routine is functional, the user has a complete, fully interfaced version of FLEX! At this point the user may go back and upgrade the initial driver packages to include advanced features such as double-sided double-density disks, printer spooling, hard disks, etc.

Appendices E and F have listings of skeletal bootstrap loader and NEWDISK routines. The source listings of these routines are also on the supplied FLEX disks. Once FLEX is running, the user may wish to make use of these source files as a starting point for his own loader and NEWDISK routines.

2.3 FLEX Disk Format

There is a defined format for FLEX disks which is essentially IBM floppy disk compatible, but uses 256 bytes per sector. Track number 0 (the outermost track) is reserved for system information and directory. The remainder is available for user files. Each file may be thought of as a chain of sectors which are linked together. This linking is accomplished by placing the track and sector address of the next sector in the chain into the first two bytes of a sector's data. The third and fourth bytes of each sector are reserved for a value used in random file accessing techniques. Thus each data sector on the disk is actually only capable of holding 252 bytes of user data. The last sector in a file chain has a forward link (track and sector address) of zero which marks it as the last sector. All the sectors on the disk which are not part of a file are linked together in the same fashion as a file, but are collectively called the "free-chain" and are not treated as a normal file. The directory, which starts with sector number 5 on track 0, is also just a chain of sectors. This chain initially contains all the sectors from number 5 up on track 0, but can grow out onto other tracks if necessary. Track 0 sector 3 is called the "System Information Record" and maintains certain data about the disk such as where the free-chain is located, the number of sectors per track, the disk name, etc. Sectors 1 and 2 on track 0 are reserved for a bootstrap loader. Further details about disk formats for double-sided and double-density disks may be found in Appendix B.

3.0 The CONSOLE I/O DRIVER PACKAGE

In order to operate FLEX, it is necessary to have a system console or terminal connected to the computer. This unit can be a CRT terminal, printing terminal, or most any keyboard/display device. Since this device can differ from installation to installation, it is necessary that the user adapt his particular console to FLEX. This adaptation is done through the Console I/O Driver package or simply the Console Drivers. Anytime FLEX must perform input or output to the system console, it does so by using the routines provided in this package.

As we shall see later, FLEX has the ability to perform printer spooling. Printer spooling requires the use of interrupts and a hardware interval timer. This timer can vary from installation to installation as can the interrupt routine handling procedure. Thus the interrupt handling and timer control routines must be user supplied. These routines are also included in what is called the Console I/O Driver package even though they really are not associated with the console. In this section, we will merely point out where these interrupt routines are located. Full descriptions will be given in a later section. It is not necessary to have them in order to bring up FLEX and in fact many users will not be able or will not desire to implement the printer spooling feature.

3.1 Console Driver Routine Descriptions

A small portion of the 8K space where FLEX resides has been set aside for the Console Drivers. This area begins at $D370 and runs through $D3E4. If the user's driver routines do not fit in this space, the overflow will have to be placed somewhere outside the 8K FLEX area. To inform FLEX where each routine begins, there is a table of addresses located between $D3E5 and $D3FC. This table has 12 two-byte entries, each entry being the address of a particular routine in the Console I/O Driver package. It should look something like this:

```
* CONSOLE I/O DRIVER VECTOR TABLE

        ORG   $D3E5     TABLE STARTS AT $D3E5
INCHNE  FDB   XXXXX     INPUT CHARACTER W/O ECHO
IHNDLR  FDB   XXXXX     IRQ INTERRUPT HANDLER
SWIVEC  FDB   XXXXX     SWI3 VECTOR LOCATION
IRQVEC  FDB   XXXXX     IRQ VECTOR LOCATION
TMOFF   FDB   XXXXX     TIMER OFF ROUTINE
TMON    FDB   XXXXX     TIMER ON ROUTINE
TMINT   FDB   XXXXX     TIMER INITIALIZATION
MONITR  FDB   XXXXX     MONITOR ENTRY ADDRESS
TINIT   FDB   XXXXX     TERMINAL INITIALIZATION
STAT    FDB   XXXXX     CHECK TERMINAL STATUS
OUTCH   FDB   XXXXX     OUTPUT CHARACTER
INCH    FDB   XXXXX     INPUT CHARACTER W/ ECHO
```

The 'XXXXX's represent the address of the particular routine listed.

The individual routines associated with actual console I/O are described here. Those associated with the timer and interrupts are deferred to a later section. They will simply be disabled for now.

INCH Address at $D3FB
This routine should get one ASCII input character from the terminal and return it in the 'A' accumulator with the parity bit (the highest order bit) cleared. If no character has been typed when the routine is started, it must wait for the character. The character should also be echoed to the output device. Only 'A' and the condition codes may be modified.

INCHNE Address at $D3E5
This routine inputs a single character exactly like the INCH routine described above with the one exception that it does NOT echo the input character to the output device. As with INCH, only 'A' and the condition codes may be modified.

OUTCH Address at $D3F9
This routine should output the character found in the 'A' accumulator to the output' device. If the output device requires the parity bit to be cleared, that can be done here. No registers should be modified except condition codes.

STAT Address at $D3F7
This routine checks the status of the input device. That is to say, it checks to see if a character has been typed on the keyboard. If so, a Not-Equal condition should be returned (a subsequent BNE instruction would cause a branch). If no character has been typed, an Equal to zero condition should be returned. No registers may be modified except condition codes.

TINIT Address at $D3F5
This routine performs any necessary initialization for terminal I/O to take place. All registers may be destroyed except for the stack pointer.

MONITR Address at $D3F3
This is the address to which execution will transfer when FLEX is exited via the MON command. It is generally the reentry point of the system's monitor ROM. If no monitor is present, this address could be set to FLEX's warm start ($CD03) which effectively nullifies this command.

The remaining routines are all associated with interrupt handling and timer control for printer spooling. For now these routines should simply be disabled. The three timer control routine vectors (TMINT, TMON, TMOFF) should point to an RTS instruction. The interrupt handler routine vector (IHNDLR) should point to an RTI. The two interrupt vector addresses (SWIVEC and IRQVEC) should point to some area in ROM or some unused address space such that when FLEX tries to store values into those points, nothing will happen. An example of these routines may be found in Appendix G.

3.2 Implementing the Console I/O Driver Routines

At this point, the user should develop the driver routines described above. The code produced should be entered into the memory spaces named.

If using a terminal which is interfaced through an ACIA (which is the preferred type), the code can be identical to that given in the sample Console Drivers found in Appendix G. The only change that may be required would be the address of the ACIA defined in the EQU statement near the beginning.

Note that it may be possible to utilize I/O routines already contained in your system's monitor ROM. If those routines fully meet the specifications given above, you could simply place the address of each applicable ROM routine into the vector table.

Once the routines have been entered, test them fully to ensure that they are functioning properly.

4.0 The DISK DRIVER PACKAGE

All communication between FLEX and the disk hardware controller(s) is done through a set of 10 routines which comprise the Disk Driver Package. The main body or core of FLEX is totally isolated from the disk controller except via these driver routines. In other words, FLEX does not care what the disk controller or drives look like. It simply calls on these routines and expects them to do all interfacing with the disk hardware. Since the disk hardware can vary from installation to installation, the user must supply these disk driver routines for his particular system. They control the very basic, low-level disk operations associated with reading and writing physical disk sectors. All file handling and character-at-a-time I/O which FLEX performs is built upon these simple driver routines.

4.1 The Disk Driver Routines

There is memory set aside for the drivers from DE00 to DFFF hex. If necessary, the routines can overflow into other portions of memory such as the top of the user RAM area or on top of the printer spooling section of FLEX if that function will not be used. There are hints later in the manual for where and how to overflow the allotted driver routine space. The individual routines can be placed anywhere, but in order for FLEX to know where they are, a jump table must be defined in the area from $DE00 to $DE1D. It appears as follows.

```
*
* DISK DRIVER ROUTINE JUMP TABLE
*         DE00                    ORG  $DE00
DE00 7E XXXX   READ    JMP  XXXXX   Read a single sector
DE03 7E XXXX   WRITE   JMP  XXXXX   Write a single sector
DE06 7E XXXX   VERIFY  JMP  XXXXX   Verify last sector written
DE09 7E XXXX   RESTORE JMP  XXXXX   Restore head to track #0
DE0C 7E XXXX   DRIVE   JMP  XXXXX   Select the specified drive
DE0F 7E XXXX   CHKRDY  JMP  XXXXX   Check for drive ready
DE12 7E XXXX   QUICK   JMP  XXXXX   Quick check for drive ready
DE15 7E XXXX   INIT    JMP  XXXXX   Driver initialize (cold start)
DE18 7E XXXX   WARM    JMP  XXXXX   Driver initialize (warm start)
DE1B 7E XXXX   SEEK    JMP  XXXXX   Seek to specified track
```

A full description of each of the above mentioned routines follows. Each lists the necessary entry parameters and what exit conditions must exist. Note that "(Z)" represents the Zero condition code bit and "(C)" represents the Carry condition code bit. All other letters in parentheses represent CPU registers. In most cases the B register is reserved for "Error Conditions" upon return. If there is no error, the B register may be.destroyed. The "Error Condition" referred to is the status returned by a Western Digital 1771 or 1791 floppy disk controller chip. Those statuses are briefly described here. An error is indicated by a "1" in the indicated bit position.

BIT	READ	WRITE	OTHER
7	not ready	not ready	not ready
6	0	write protect	write protect
5	0	0	0
4	not found	not found	seek error
3	CRC error	CRC error	CRC error
2	lost data	lost data	0
1	0	0	0
0	0	0	0

If the Western Digital chip is not used, these statuses must be simulated by the user s routines.

4.2 Disk Driver Routine Specifications

Each description lists any necessary entry parameters and the proper state of certain registers on exit. Unless stated otherwise, the 'Y', 'U', and 'S' registers must NOT be altered by any of the routines.

```
READ     This routine reads the specified sector into memory at the
         specified address.  This routine should perform a seek
         operation if necessary.  A sector is 256 bytes in length.
         ENTRY - (X) = Address in memory where sector is to be placed.
                 (A) = Track Number
                 (B) = Sector Number
         EXIT  - (X) May be destroyed
                 (A) May be destroyed
                 (B) = Error condition
                 (Z) = 1 if no error
                     = 0 if an error

WRITE    This routine writes the information from the specifed memory
         buffer area to the disk sector specified.  This routine should
         perform a seek operation if necessary.  A sector is 256 bytes
         in length.
         ENTRY - (X) = Address of 256 memory buffer containing data
                       to be written to disk
                 (A) = Track Number
                 (B) = Sector Number
         EXIT  - (X) May be destroyed
                 (A) May be destroyed
                 (B) = Error condition
                 (Z) = 1 if no error
                     = 0 if an error

VERIFY   The sector just written to the disk is to be verified to
         determine if there are CRC errors.  No seek is required as
         this routine will only be called immediately after a write
         single sector operation.
         ENTRY - No entry parameters
         EXIT  - (X) May be destroyed
                 (A) May be destroyed
                 (B) = Error condition
```

```
                (Z) = 1 if no error
                    = 0 if an error

RESTORE   A restore operation (also known as a "seek to track 00") is to
          be performed on the specified drive.  The drive  is  specified
          in the FCB pointed to by the contents of the X register.  Note
          that the drive number is the  4th  byte  of  the  FCB.   This
          routine  should  select the drive before executing the restore
          operation.
          ENTRY - (X) = FCB address (3,X contains drive number)
          EXIT  - (X) May be destroyed
                  (A) May be destroyed
                  (B) = Error condition
                  (Z) = 1 if no error
                      = 0 if an error

DRIVE     The specified drive is to be selected.  The drive is specified
          in the FCB pointed to by the contents of the X register.  Note
          that the drive number is the 4th byte of the FCB.
          ENTRY - (X) = FCB address (3,X contains drive number)
          EXIT  - (X) May be destroyed
                  (A) May be destroyed
                  (B) = $0F if non-existent drive
                      = Error condition otherwise
                  (Z) =1 if no error
                      =0 if an error
                  (C) =0 if no error
                      =1 if an error

CHKRDY    Check  for a drive ready condition.  The drive number is found
          in the specified FCB (at 3,X).  If the user's controller turns
          the  drive  motors  off  after  some  time delay, this routine
          should first check for a drive ready condition and  if  it  is
          not  ready, should delay long enough for the motors to come up
          to speed, then check again.  This delay should be done ONLY if
          not  ready  on  the  first  try  and ONLY if necessary for the
          particular drives and  controller!  If  the  hardware  always
          leaves  the  drive  motors  on,  this routine should perform a
          single check  for  drive  ready  and  immediately  return  the
          resulting  status.   Systems  which do not have the ability to
          check for a drive ready condition should simply always  return
        a ready status if the drive number is valid.
          ENTRY - (X) = FCB address (3,X contains drive number)
          EXIT  - (X) May be destroyed
                  (A) May be destroyed
                  (B) = Error condition
                  (Z) = 1 if drive ready
                      = 0 if not ready
                  (C) = 0 if drive ready
                      = 1 if not ready
```

QUICK   This routine performs a "quick" drive ready check. Its function is exactly like the CHKRDY routine above except that no delay should be done. If the drive does not give a ready condition on the first check, a not ready condition is immediately returned. Entry and exit are as above.

INIT   This routine performs any necessary initialization of the drivers during cold start (at boot time). Actually, any operation which must be done when the system is first booted can be done here.
ENTRY - No parameters
EXIT - A, B, X, Y, and U may be destroyed

WARM   Performs any necessary functions during FLEX warmstart. FLEX calls this routine each time it goes thru the warm start procedure (after every command). As an example, some controllers use PIA's for communication with the processor. If FLEX is exited with a CPU reset, these PIA's may also be reset such that the controller would not function properly upon a jump to the FLEX warm start entry point. This routine could re-initialize the PIA when the warm start was executed.
ENTRY - No parameters
EXIT - A, B, X, Y, and U may be destroyed

SEEK   Seeks to the track specified in the 'A' accumulator. In double-sided systems, this routine should also select the correct side depending on the sector number supplied in 'B'.

```
ENTRY - (A) = Track Number
        (B) = Sector Number
EXIT -  (X) May be destroyed (See text)
        (A) May be destroyed (See text)
        (B) = Error condition
        (Z) = 1 if no error
            = 0 if an error
```

4.3 Developing the Disk Driver Routines

It should be reiterated that the best approach to use in writing these disk driver routines is one of simplicity in the beginning. The first set of drivers written should be for a single-sided, single-density floppy disk. Once these drivers are fully functional and FLEX is up-and-running, it will be much easier to upgrade them to double-sided or double-density and to add hard disks or whatever.

The READ and WRITE single sector routines are the heart of the Disk Driver Package. As mentioned, they must perform a seek operation to the proper track. It will probably be easiest and most efficient to call on the SEEK routine described above to perform this operation. If this is the case, it is important that the user ensure that the exit conditions of the SEEK routine are compatible with the READ and WRITE routines. For example, it may be desirable for the SEEK routine to preserve the X register so that READ and WRITE can assume the memory address for the sector remains intact across a seek call.

The READ and WRITE routines need not be concerned with retries when errors are encountered. FLEX takes care of this operation automatically.

CHKRDY and QUICK are used by FLEX to determine if a disk is ready to carry out some operation. If not, FLEX will report a "drive not ready" error. Some systems (many minifloppy systems) do not provide the ability to check for a drive being ready. If this is the case, the best solution is to simply be sure the drive specified is a valid number and if so, immediately signal the drive as ready. Thus if a drive is not actually ready when accessed, it will most likely "hang up" waiting for a disk to be inserted and the door closed.

In multi-drive systems, it is important that the drivers keep tabs on which track each drive is left on. This is at least true in the case of the Western Digital controller chips. On these chips, there is only one track register and that is for the currently selected drive. If the user selects another drive and seeks to some track on it, when he comes back to the first drive he will not know which track he is on. To overcome this, it will probably be necessary to keep a list of what track each drive was last on. Whenever the current drive is changed, the current track for that drive should be saved and the track which the new drive was last on should be picked up and put in the controller's actual track register.

The SEEK routine itself should not attempt any reading. Specifically, it should not attempt to read the sector ID field to determine if it is actually at the correct track. It simply seeks until it is positioned over what it thinks is the correct track. If something is wrong and it is not really on the correct track, the read or write routine will find out about it and report such an error. Now if this is the case (the drivers have lost track of what track they are actually on), all should eventually be corrected by FLEX. When FLEX gets a read or write error (which may be due to being on the wrong track), it retries several times on the same track. If none of these tries are successful, FLEX performs a restore operation and then re-seeks to the specified track. After re-seeking, FLEX attempts several more reads or writes and if still unsuccessful, the whole procedure of restoring and re-seeking is repeated. A total of three such re-seeks and associated retries are attempted before FLEX finally gives up and reports a read or write error. It is the restoring and re-seeking that will get the drivers back on the right track number if they were lost. When a restore operation is performed, the controller knows exactly which track it is on (track 0) and can start anew with this correct track number.

If there is enough room, the user may wish to put a check in the SEEK routine to assure that an illegal track number is not specified. In such a case, SEEK would have to know what the highest track number should be and if a supplied track number is greater, an error should be returned. This error would be a record not found type error.

The RESTORE routine is the only one which must perform a drive select before carrying out its function (except of course for DRIVE whose function is to select a drive). All other routines can assume that the drive has been selected before they were called.

Once the disk driver routines have been written, they should be entered into memory in the space provided. Also, be sure the jump table is entered into memory as shown. You should now have a set of Console I/O Drivers and Disk Drivers in memory. At this point you are ready to test the routines.

4.4 Overflowing the Disk Driver Area

If the user is unable to fit his disk driver routines in the space allotted ($DE00 to $DFFF except for the jump table), it is possible to overflow the routines into other areas. As long as the jump table points to the beginning of each routine, they can be placed anywhere in memory. Obviously, it would be best if the routines can be fit in the reserved space. If not, they could overflow into one of three places: the upper end of user memory, the printer spooler area (if printer spooling is not implemented), or additional RAM memory placed above FLEX's $DFFF upper limit. If the third case is possible. there is absolutely no problem as that memory would not be used by FLEX or any of its support software. Using the printer spooler area is a good solution if the printer spooling feature will not be implemented, but there is one complication. FLEX has assembled code in the printer spooler area and when FLEX is loaded, this code is loaded. Thus if the user has placed driver routine code in this area, loading FLEX will overwrite that code. In later versions of the drivers, this is no real problem since the drivers will be appended onto the end of the FLEX file. This means that the drivers would be loaded over the top of any FLEX code if assembled to the same addresses. For more information on using the printer spooler area, see Section 12.

At this stage of the development of FLEX, the best place to overflow the drivers (assuming there is no RAM above FLEX) is at the top of user memory. For example, if you have 32K of user memory (besides the 8K for FLEX), you might reserve 256 bytes from 7F00 to 7FFF for drivers. Since your initial drivers are stored in memory, this would put the overflow out of the way such that no code in FLEX will load over your drivers. One caution about this technique - it requires that a different MEMEND value be set. For our example, the new MEMEND should be $7EFF. For more information on changing the MEMEND value. consult the FLEX Advanced Programmer's Guide or see Section 12.

These same overflow techniques can also be applied to the Console I/O Driver Package if necessary.

5.0 TESTING THE DISK DRIVER ROUTINES

Once the disk driver and console I/O driver routines have all been written and entered into the computer, we are ready to test the driver routines. Before doing so, however, it would be wise to save the code for all the routines onto some mass storage device such as cassette or paper tape if available. This will allow you to quickly reload the routines should something go wrong which wipes out memory. The user should attempt to test these driver routines as fully as possible. Some patience and thoroughness in this step could save a lot of frustration and delay later.

5.1 Preparing a Disk

At this point we are finally ready to use one of the supplied disks. If you have read the manual and the yellow disclaimer and feel confident that you can handle the FLEX adaptation procedure, open the envelope containing the two disks. The two disks are identical in terms of the data which has been stored on them. Each contains all the standard FLEX utility commands and, of course, the core of FLEX itself. Hopefully, you will only need one of the disks - the second is provided only as a backup should the first be destroyed. The intent is that only one of the two disks be used for all testing and development unless it is hopelessly destroyed. Note that Section 13 describes how you can purchase additional General FLEX disks should you destroy both of the supplied ones.

Select one of the two disks and be certain that it is write-protected. The first several steps of testing will not require writing anything to the disk and keeping it write-protected will prevent your routines from writing when they should not. 8" and 5 1/4" floppies are write-protected in different ways. The 8 inch floppies are write-protected when a cutout notch on the leading edge of the disk (as it is inserted into a drive) is left exposed. If the cutout is covered with a piece of opaque tape, the disk is "write-enabled" or NOT write-protected. 5 1/4 inch floppies are just the opposite of the 8 inch. 5 1/4 inch disks are write-protected when a cutout notch on the side of the disk is covered with opaque tape, and they are write-enabled if the cutout is left exposed. Be sure the disk you are using is write-protected. The disk is now ready for use in the ensuing test procedure.

5.2 Tests Without Using a Supplied Disk

Throughout this section, we will refer to the supplied FLEX disk as the "FLEX Disk". You should obtain a blank or non-FLEX disk for use in the testing and we will refer to it as the "Scratch Disk". Some of the driver routines can be tested without inserting the FLEX Disk or by using a Scratch Disk. In particular they are DRIVE, RESTORE, CHKRDY, QUICK, SEEK, and probably INIT and WARM. Now let's go through the routines one at a time.

INIT and WARM These routines are not specifically defined for the general case. Their function depends entirely on what is required by the particular controller and disks in use. Since the user defined and developed these routines, it is assumed the user will be able to determine how they might best be tested. Indeed, these routines may not even be required for your particular installation.

DRIVE The Drive Select routine can probably be tested with no disk installed whatsoever. To be sure, however, it is suggested that a scratch disk be installed during the test. This routine is easy to test if the disk drives in use have LED's or lights which indicate the drive is selected. If this is the case, simply write a little routine which calls the DRIVE routine with the proper entry parameters (see section 4.2) and then returns to your monitor. If the routine functions properly, the light should come on on the selected drive. Switch back and forth from one drive to the other (if you have more than one drive) to ensure you can select any connected drive. If your drives do not have a drive selected indicator, this routine will be much more difficult to test. You might just try calling it and being sure it returns properly. If so, assume it is working. If it is not, you will find that out as we proceed.

RESTORE The Restore routine is a relatively easy routine to test. It should be tested with a scratch disk installed in the drive and the door closed. Before a restore operation can be performed on a drive, the desired drive must be selected by the DRIVE routine. Thus to test RESTORE, write a short routine which first calls DRIVE to select the desired drive and then calls RESTORE to restore the head to track zero. The proper entry parameters must be setup for these calls as outlined in section 4.2. If the RESTORE routine is functioning properly, you should see the disk drive head move to the outside edge of the disk (assuming you have removed the cover on your disk system, of course). If the head is already at track zero before testing the command, or to retry the RESTORE command after one restore, it is possible to physically move the head out from track zero. To do this, remove the disk, turn off the power to the disk drive, remove the cover so that the head assembly is exposed, and gently push the head assembly away from track zero (toward the hub) with your fingers. The head itself is delicate, so be sure you are pushing on some solid part of the head assembly (not the head itself) and do not force it if it resists. Once the head is away from track zero, power the drive back up and test the RESTORE routine.

CHKRDY and QUICK These routines simply return a status - either "ready" or "not ready". They are quite simple to test. To test the drive "not ready" case, open the door on the drive under test. To test the drive "ready" case, insert a scratch disk and close the door. Note that a drive select must be done before checking the status.

SEEK The SEEK routine must be tested with a disk installed. The user should be able to get positive feedback as to whether or not the routine is functioning properly by watching the movement of the disk drive head. Before testing seek, it may be necessary to perform a RESTORE operation. This is to ensure that the controller is not lost as to which track it is on. For example, if the controller track register says it is on track #6 but the head is actually positioned on track #32, there could be problems if a seek to track #73 was attempted. By performing a restore operation, the controller will be able to get back on track (pun intended) such that the track register says #0 and the head is actually on track #0. Once a single restore has been performed, the controller and drivers should be able to keep up-to-date as to which track they're on without subsequent restores. So to test the SEEK routine, first perform a restore operation, then write a routine to select the desired drive and then call the SEEK routine with the proper entry parameters to seek to some random track on the disk. Test this routine fully to see that-it seeks properly in both directions and visually seems to go to the correct track position.

5.3 Testing the READ Routine

Now we've come to the real thing! Testing the READ routine is perhaps the most important step in adapting FLEX to your hardware. As mentioned before, the READ and WRITE single sector routines are the heart of the whole Disk Driver Package. Your WRITE routine is probably very similar to your READ routine, so most of the testing you do here will probably also apply to the WRITE routine without having to actually perform dangerous disk writes. The READ routine does rely on some other routines like SEEK, so be certain that they are functioning properly before testing READ.

For the first time, you will be using a FLEX Disk. As stated earlier, be certain it is write-protected and that you only use one of the two supplied disks if possible.

If desired, the READ routine can be tested by writing a short routine to select the drive and then call the READ routine with the desired entry paramters. As a convenience for testing, however, we have provided the listing for a short single sector test utility appropriately called "TEST". This assembled source listing is found in Appendix C. Using your system's monitor ROM or whatever means you have, enter the code listed for this program. TEST assumes that all the Disk Driver and Console I/O routines are also installed in memory. Once this code is entered, begin execution of TEST by jumping to location $0100. You should see a carriage return and line feed output to the console, followed by this prompt:

```
F?
```

This is a prompt for the "Function" desired. The function may be a READ single sector, a WRITE single sector, or a return to the system monitor. To perform a READ, type an "R" (upper case); to perform a WRITE, type a "W" (upper case); to return to the monitor, type any other character.

!!! FOR THE TIME BEING, DO NOT ATTEMPT A WRITE COMMAND (W) !!!

Enter an "R" to do a READ command and TEST should respond with:

```
D?
```

This is a prompt for the desired drive number (a single digit from 0 to 3). After entering a drive number you should be prompted with:

```
T?
```

Thi is a prompt for a two-digit, hexadecimal track number. You can select any track you like, but be sure it is not a higher number than the number of tracks on the disk. Next you will receive the prompt:

```
S?
```

which is a prompt for a two-digit, hexadecimal sector number. Any sector number may be given since an error should be returned if the drivers can't find the desired sector.

The sector number prompt is the last one, and once entered, the selected function should be carried out. Under a READ command, if there was no error, the data from the sector will be. displayed on the console in hexadecimal. There will be 16 rows of 16 bytes each. This display can be examined to see if the data was read correctly. If an error occurs in the READ operation, instead of displaying data TEST will print:

```
E=XX
```

This signifys an Error occurred and the "XX" represents the hexadecimal value in the 'B' accumulator (the error condition) on return. In either case, TEST will immediately start all over again with the function prompt.

With a FLEX Disk inserted, begin by reading sector #01 on track #00. This is where a bootstrap loader program will reside in the final system, but for testing purposes this sector has been setup with a special data pattern. The first byte in the sector is $00, the second is $01, the third is $02, and so on to the last byte which should be $FF. Once you are able to read this sector, try other random sectors on the disk. You can be certain you have read the correct sector in most cases by looking at the first two bytes of the data. In most sectors these two bytes point to the next sector in the chain of sectors (see section 2.3). Thus if not the last sector on a track, the first byte should be the track number and the second byte should be the sector number plus one. The last sector on the track will have the first byte equal to the track number plus one and the second byte equal to $01. The only exception to this is any sector which is at the end of a file's chain of sectors, at the end of the directory (the last sector on track #0 on the FLEX Disk), the System Information Record (track #0 sector #3), or at the end of the free chain (the last sector on a FLEX Disk). These sectors have zeroes in both bytes one and two. On the FLEX Disk, any sector which does not have data stored in it (a free sector) should have all zeroes past bytes one and two.

Test the READ routine thoroughly! Be sure you test the limiting cases such as the first and last sectors on several tracks, especially on track #0 and on the last track on the disk. Do not continue with the FLEX adaptation until you have firmly convinced yourself that the READ routine and all of the other supporting routines tested are functioning perfectly!

5.4 Testing the WRITE Routine

Now we come to the most dangerous part of the FLEX adaptation process the WRITE routine. If this routine runs wild, portions of data on a FLEX Disk could be destroyed. For this reason, it is suggested that you thoroughly examine your WRITE routine code to make certain there are no visible bugs before running it. Where possible, make sure it does the same things as the now functioning READ routine (such as seeking and possibly setting up the controller chip or DMA device). If the WRITE routine does fail and that failure causes indiscriminate writing to the disk, chances are that only one track will be destroyed. Thus before switching to the supplied backup FLEX Disk, continue testing the WRITE routine on the damaged disk by attempting to write to different tracks. As with the READ routine, the user can develop his own testing procedure for the WRITE routine or the supplied TEST program can be entered and used if desired. If the TEST program is used, it differs from the READ command testing as follows. To perform a WRITE operation the "F?" prompt should be answered with an upper case 'W'. The subsequent Drive, Track, and Sector prompts are then answered as before. The data buffer which should be written to the disk is assumed by TEST to be at $1000. Before entering TEST to do the WRITE command, the user can go to the 256 bytes found at $1000 and setup whatever data he would like written to the disk sector. Another method of setting up this data buffer is by doing a READ command in TEST. The data read from the specified disk sector is placed into memory at $1000. Thus, after a read operation, the data is all setup for writing back to the disk. In order that you do not mess up the data which is stored on the disk, the best method of testing would be to read some sector with the 'R' function and then immediately write it back out without changes via the 'W' command.

When the sector number has been given to TEST, it immediately attempts to write the data to the disk. If the write procedure functions properly and there are no errors, TEST will print an "OK" on the screen and start all over by prompting for another command. If errors occur during the write, the same error messages described under the READ command are given.

For the initial testing of the WRITE command place a scratch disk in a drive and attempt a write of any data to it. Since your scratch disk is not likely to be formatted in FLEX's 256 byte format, an error should result from the attempted write. The point here is to see that the WRITE routine does peform the seek, load the head, and try to write data. If the routine is going to blow up it is best that it happen on a scratch disk and not one of the FLEX disks. Ensure that the routine properly returns with a valid error code.

Before attempting a write to the FLEX disk, it is important to note that there is data stored on the disk (FLEX itself as well as several utility commands) and that almost all the sectors are linked together by the first two bytes of each sector. Thus when writing to this disk it is important that you do not write over the data which is presently stored in a sector or over the link bytes if the sector is empty. This can be avoided as follows. There are three sectors on track zero which are unused on the FLEX Disk. Sectors number one and two are reserved for a bootstrap loader program and sector number four is reserved on all FLEX disks for future expansion. These three sectors are not linked to any other (or don't need to be); thus any desired data can be written to these sectors. For example, you might read sector #1 on track #0 which was setup with a special data pattern and attempt to write this data to sector #4 on track #0. Be sure you do not alter any other sectors on track zero.

All other sectors on the disk are part of a chain of sectors and their first two bytes are a link address to the next sector in the chain. If data is written to any of these sectors, it is imperative that the first two bytes remain unchanged! You will always be safe to read a sector and write it back out without changes (safe, that is, if your write routine functions properly). If you wish to change some of the data to make sure you actually are writing the sector, do so on a sector which is empty. The FLEX Disk is not full, only the first several tracks have files stored on them. If you write to sectors which are on the last few tracks, you will most likely be writing into free sectors. Initially, all the free sectors will be filled with zeroes (except, of course, for the first two link bytes). It will not hurt for you to change any of the zero bytes in a free sector and they may be left non-zero after testing.

Now you are ready to attempt writing to a supplied FLEX disk. Remove the write-protection from the disk (cover the cutout on an 8 inch disk; uncover the cutout on a 5 1/4 inch disk) and insert it in a drive. Perform several write commands as outlined above. After writing a sector, the data should always be read back to be certain that it was actually written as desired. Firmly convince yourself that your WRITE single sector routine is functioning exactly as it should.

5.5 Testing the VERIFY Routine

The VERIFY routine is a difficult one to test. VERIFY is only called by FLEX directly after performing a WRITE single sector operation. If the write operation functioned properly and didn't report an error, then chances are the VERIFY routine will not find an error in the data. It is used as a security measure to guarantee that all data is valid. Since VERIFY won't likely find an error, it is difficult to test to see if it really would report an error. It is recommended that you basically assume VERIFY to be OK and skip thorough testing of it. Do try calling it directly after doing a single sector WRITE operation to see that it returns properly and reports no error. If it does that, simply assume it to be functional. The VERIFY routine will probably be very similar to the READ routine anyway, with the exception of what is done with the data. READ places the 256 bytes into memory; VERIFY tests to be sure they can be read and simply discards them if so. If your READ and VERIFY routines are similar, this is more justification to assume the VERIFY routine is good.

6.0 BRINGUP UP THE INITIAL VERSION OF FLEX

At this point, all the driver routines for the Console I/O Driver package and the Disk Driver package should have been written, fully debugged, and should be resident in memory. If possible, these routines should be saved onto some mass storage device such as cassette or paper tape for quick reloading should problems arise. We are now ready to load up FLEX and, using these driver routines, test the operation of the entire operating system.

6.1 Loading FLEX with QLOAD

A short program has been supplied to load the core of FLEX from the disk into its place in memory. The program is called 'QLOAD' for Quick Loader and is listed in Appendix D. The code for QLOAD should be entered into memory at $C100 as given in the assembled listing. QLOAD is really a complete FLEX file loader that directly calls upon the routines in the Disk Driver Package. It differs from loaders that we will use later in that it assumes that the file it is to load is stored on the disk beginning with sector #1 on track #1. On the supplied FLEX disks, the file which begins there is called "FLEX.COR". This file is the main body or "core" of FLEX as the filename extension implies. It contains everything FLEX needs to run in a system except for the Disk Drivers and the Console I/O Drivers. Since we already have these drivers in memory, we need only load FLEX.COR by using QLOAD in order to run our first version of FLEX.

Once the code for QLOAD has been entered, write-protect a FLEX Disk, insert it into drive #0, and jump to location $C100 which is the starting address of QLOAD. If all works well, QLOAD should read the file from the disk and jump to your system monitor. The FLEX.COR file is over twenty sectors in length, so it will probably take a couple of seconds to read. If QLOAD does not perform as described, reload your drivers, carefully check the QLOAD program code in memory, and try again. If it still fails, there may be something wrong in your drivers.

If the load does take place, and QLOAD returns control to your system monitor, you are ready to begin execution of FLEX. This is done by jumping to $CD00. At $CD00 there is a short initialization routine which sets up several pointers for FLEX, checks to see how much memory is in the system, and then prompts for the date. After the date has been entered, the disk in drive #0 is scanned for a file called "STARTUP.TXT" as explained in the FLEX User's Guide. There is no startup file on the supplied disks, so the initialization routine will finally jump to FLEX's warm start address and you will receive the three plus-sign prompt. If FLEX does not come up for you, you either did not actually get a complete load of FLEX or there still may be errors in your drivers. In either case, you would have to go back and try again.

6.2 Testing FLEX with Read-Only Commands

Assuming FLEX loaded OK and you received the three plus-sign prompt, you are now ready to use FLEX. The first tests should only involve operations which perform reads from the disk. Do not attempt any writing until you are convinced the reads are functioning. You can be sure you are only reading by leaving the disk write-protected. That way if you do inadvertently attempt a write, the disk will be protected.

The best method of testing the read operations of FLEX is to simply sit down and begin executing commands which perform reads. Some of these commands are CAT, ASN, DATE, LIST, TTYSET, and VERSION. For proper syntax and use of these commands, read the FLEX User's Guide. To use the LIST command you might try the following:

```
+++LIST 0.ERRORS.SYS
```

This should list the system error file which contains all of FLEX's error messages.

6.3 Testing FLEX with Write Commands

Now you are ready to use FLEX to write information on the disk. Remove the write-protection from a supplied FLEX disk and insert it into drive #0. A convenient method of writing some information into a sector is to create a short text file using the BUILD command. Read over the description of that command and when understood, type the following command to FLEX:

```
+++BUILD JUNK
```

FLEX should perform some disk activity associated with loading the BUILD command and preparing a file called 'JUNK.TXT' and then print BUILD's prompt which is an equals sign ('='). When that prompt is received, type a short line of text as follows:

```
=THIS IS A FILE CALLED JUNK.
```

When a carriage return is hit after typing the period, FLEX should load the head and perform some disk activity. This is actually where FLEX is opening the file called JUNK. If all goes well, you should receive another equals sign prompt almost immediately. Type three more lines in like this:

```
=THIS IS THE SECOND LINE.
=THIS IS THE THIRD AND FINAL LINE.
=#
```

When the last carriage return is hit (after the pound sign), FLEX will attempt to write the three lines of data to the file and proceed to close it. If everything works, you should see FLEX's prompt ('+++') after a second or two. Do a CAT command on the disk to see if the file 'JUNK.TXT' was placed in the directory. Now view the contents of that file by executing a list command like this:

```
+++LIST JUNK
```

You should see the three lines typed into JUNK displayed on the console. If this test of BUILD all went as described, you are well on your way to finishing the FLEX adaptation! If things did not work as described, you will have to go back and look for bugs in your routines. Your FLEX disk may be destroyed and it may be necessary to break out the second FLEX disk supplied.

6.4 Using this Version of FLEX

Assuming that all the functions of FLEX have been tested to the best of your ability and that no problems have arisen, you may now wish to use this version of FLEX in the remainder of the adaptation process. The utilities included with FLEX include a disk editor and assembler. These will save you much time if you have been assembling code by hand.

7.0 PREPARING A BOOTABLE VERSION OF FLEX

The only version of FLEX itself on the supplied disks is the file, FLEX.COR. This file is the core of FLEX and does not contain any disk or console drivers. The final version of FLEX on a disk which may be "bootstrap loaded" must also contain the disk and console driver routines. In this section we will create a new file on the disk called "FLEX.SYS" which contains the core of FLEX and all the driver routines. Of course in order to do this, the FLEX setup in memory in section 6 must be running properly. All we need do is save the two driver packages on disk as two files and then append them onto the FLEX.COR file. These steps can all be accomplished with simple FLEX commands.

The first step is to save the code for your Disk Driver routines as a file called 'DISK.BIN'. This is done with the following FLEX command:

```
+++SAVE DISK,<SSSS>,<EEEE>
```

Where <SSSS> and <EEEE> represent the Starting and Ending addresses of your Disk Drivers code. After executing the command you might double check that the file was really saved by doing a CAT command and making sure there is a file called 'DISK.BIN'.

Next, save your Console I/O Driver routines in a file called 'CONSOLE.BIN' with the following command:

```
+++SAVE CONSOLE,<SSSS>,<EEEE>,CD00
```

where <SSSS> and <EEEE> represent the Starting and Ending addresses of your Console I/O Drivers code. The 'CD00' is a "transfer address" for the file. A transfer address is an address saved with a binary file to tell it where to begin execution. The final version of FLEX is just a standard binary file on the disk and as such must have a transfer address so the bootstrap loader will know where to begin execution once FLEX has been loaded. Since we are going to append the CONSOLE file (and DISK file) onto the core of FLEX, this transfer address will eventually get into the final, bootable version of FLEX. Perform a CAT @ommand to be sure that the CONSOLE.BIN file now exists on the disk.

The APPEND command in FLEX allows two or more files to be appended together to create a new file. We can use it to prepare our final, bootable version of FLEX with the following command:

```
+++APPEND FLEX.COR,DISK.BIN,CONSOLE.BIN,FLEX.SYS
```

If all goes well, you should now have a file called 'FLEX.SYS' on the disk. It is a complete version of FLEX which you will be able to boot up after completing the next section.

8.0 BOOTSTRAP LOADING OF FLEX

At this point, the user should have a fully functional version of the FLEX Disk Operating System stored on disk. Now you are faced with the problem of loading that operating system into memory and beginning execution of it. Generally, loading FLEX will be the first thing done after powering the computer on, but short of loading all the Disk and Console driver routines along with the QLOAD we have no way of performing this load. That is where a "bootstrap loader" is needed. In this section the user will be instructed to write a bootstrap loader for his system.

8.1 The Concept of Bootstrap Loading

The problem we face is obvious. When the computer is first powered on, FLEX is not resident and there is no way of loading it. The solution is to write a short program whose only purpose is to load FLEX and begin execution of it. This type of program is referred to as a "bootstrap loader" since the system is essentially "pulling itself up by its bootstraps". Once this bootstrap loader has been developed, it can be used to load FLEX. However, we still have the same problem - how do we get the bootstrap loader into the computer after powering on? Fortunately, this problem is not as great since the bootstrap program is much smaller than FLEX. There are three obvious solutions.

1) The bootstrap program could be hand-entered each time the system was powered on.
2) The bootstrap program could be loaded from cassette or paper tape each time the system was powered on.
3) The bootstrap program could be entirely stored in ROM.

The first two are obviously very undesirable. The third is feasible, but a typical bootstrap program will be close to 256 bytes and this might be considered a waste of ROM space.

There is another solution which is not quite so obvious, but which is perhaps the best and most used solution. That is to use a two-stage booting process. The idea is to put the bootstrap loader which we have been discussing on the disk and then write another dumb, very short bootstrap program to read in the intelligent FLEX bootstrap loader. This dumb bootstrap program should be very small since it will only have to read in one sector which is defined to contain the intelligent FLEX bootstrap loader (assuming that loader fits in 256 bytes or one sector). On a FLEX disk, this defined boot program sector is sector #1 on track #0. If absolutely necessary, the boot can overflow onto sector #2 which has also been reserved. Since the dumb bootstrap program is so short it is now feasible to place it in ROM.

Before going any further. let's review some nomenclature. Throughout the manual when "booting FLEX", "booting up", or simply "booting" is mentioned, it refers to the entire procedure of loading FLEX which involves the two stages of bootstrap loading. To avoid confusion in the remainder of this section, we must come up with a way to differentiate between the two bootstrap programs or operations. When we refer to the intelligent bootstrap program which resides on disk and which loads FLEX, we will use the term "FLEX loader" or simply "loader". The dumb bootstrap program which resides in ROM we shall refer to as the "ROM boot".

8.2 Writing a "ROM Boot" Program

The ROM boot program can be written and debugged before writing the FLEX loader. Assuming the FLEX loader will fit in one sector (256 bytes or less), our ROM boot will only have to read sector #1 from track #0 into memory and then jump to the beginning of the loader. One thing that makes this ROM boot short and simple is that no seeking operation need be done. Since the only sector to be read is on track #0, a restore operation can be performed to get there. Thus the basic steps to be performed by the ROM boot program are:

1) Select drive #0

2) Do a restore to track #0 operation

3) Read sector #1 into memory at $C100

4) Jump to $C100

As can be seen, the FLEX loader which we are reading is assumed to be assembled for operation at $C100. That loader will assume that the ROM boot has already selected drive #0, so don't de-select the drive before jumping to $C100.

At this point the user should develop his ROM boot program. Note that the FLEX editor and assembler can be used for this work. An example of a ROM boot program may be seen in Appendix G. The ROM boot program can be located anywhere outside the 8K reserved for FLEX. It may be advantageous to initially assemble the boot somewhere in low memory (like $0100) for testing purposes and when debugged, reassemble it to some high address for burning into ROM. For testing purposes, it is suggested that step 4 in the instructions above should be changed to a jump to your monitor. Thus you could execute the ROM boot which when finished would return to your monitor. This would allow you to use your system monitor to examine the 256 bytes at $C100 to be sure you are actually reading the correct data in from the disk. In any event the data you read will not yet be a valid FLEX loader program and you will therefore not want to attempt to execute it.

When you are convinced that the ROM boot is functioning properly, save the code on tape or on disk using the SAVE command. It should not be burned into ROM until actually tested with the FLEX loader on disk. We will test this ROM boot further after the FLEX loader has been written.

8.3 Writing a "FLEX Loader" Program

The sole purpose of the FLEX Loader is to load FLEX from the disk and begin its execution. This is actually a simple file loader since FLEX resides on the disk just like any other file. The only major difference in this FLEX loader and the standard file load routine used within FLEX is that no filename is specified. Instead, it is assumed that the FLEX loader already knows where FLEX resides on the disk when called. Specifically, the FLEX loader (which resides at $C100) assumes that the track and sector location of FLEX is at $C105 and $C106 respectively. Since FLEX can reside anywhere on the disk, we need a way to tell the FLEX loader just exactly where FLEX is on the particular disk in use. That is the function of the LINK command found in FLEX. It looks up FLEX in the directory to find the starting track and sector and writes this information into the sixth and seventh bytes of track #0 sector #1. When the FLEX loader is read in from that sector, those two bytes will be placed at $C105 and $C106 and the loader thus knows exactly where to go to get FLEX.

Now that you know how the FLEX loader works, it is time to write one. Actually, most of the writing has already been done for you. The skeletal FLEX Loader program listed in Appendix E has the entire loader with the exception of a single sector read routine. The loader resides at $C100. The user need only replace the READ routine found in that listing with one of his own writing. This single sector read routine should be almost exactly like the one developed for the Disk Driver Package. It is called with the track and sector numbers in 'A' and 'B' and the address of where to read the data into memory in 'X'. A NOT-EQUAL status should be returned if an error occurred. Note that no error code need be returned in the 'B' register. If there is an error, the FLEX loader will just start all over with the loading process. If there was no error, the routine should return an EQUAL status. Note that the read routine is responsible for any necessary track seeking. There are around 128 bytes of space for this read sector routine. If at all possible the user should fit the read sector routine within this space so that the entire FLEX loader will fit in one sector. If this is not possible see section 8.4.

Once the user has developed his-FLEX loader routine and has the code residing at $C100, it can be put onto the disk on track #0 sector #1 by use of the PUTLDR command found on the FLEX Disk. The syntax for the command is quite simply:

```
+++PUTLDR
```

It assumes that there is a 256 byte (or less) loader program resident in memory at $C100. PUTLDR simply writes this data out to sector #1 of track #0. As described earlier, we must now tell the FLEX loader where FLEX resides. This is done with the LINK command as follows:

```
+++LINK FLEX
```

This assumes your final version of FLEX (which includes all the drivers) has been called FLEX.SYS. The LINK command will look up FLEX.SYS in the directory, find its starting address, and write the starting track and sector number into the sixth and seventh bytes of the FLEX loader in track #0 sector #1.

Your FLEX disk is now ready for booting or at least for testing prior to booting. Reload the ROM boot you prepared earlier and execute it with the FLEX disk in drive #0. It should pull the FLEX loader into memory at $C100 and jump to it. The FLEX loader should then in turn load and execute FLEX. If this process does not take place, you probably have an error in your FLEX loader and will have to redo your code.

Once you have the boot operation working properly such that you can bring FLEX up having only the ROM boot program in memory, you should reassemble the ROM boot to a convenient location and burn it into PROM. When this is done, you will have a complete, bootable version of FLEX ready for normal use!

8.4 Hints on a Two Sector FLEX Loader

If you were able to fit your FLEX loader program into 2@6 bytes or one sector, you can skip this section completely. If not, you should attempt to develop a FLEX loader that will fit in 512 bytes or 2 sectors. If you can do this, the loader can be stored on track #0 sectors #1 and 2. Sector #2 on track #0 has been reserved for just this purpose. You will have to write your own routine to write the loader to these two sectors however, since the supplied PUTLDR command only writes 256 bytes. The other problem is that the ROM boot must now be able to read both sectors from the disk. This can certainly be done, it just means that your ROM boot will take up more space. If the ROM boot ends up being very large, you may decide it is just as easy to put the entire FLEX loader in ROM and execute it directly without having to load it from disk with a ROM boot.

9.0 THE NEWDISK ROUTINE

FLEX has its own defined format for diskettes. All disks must be prepared with this format before they can be used by FLEX. One distinguishing characteristic of the FLEX format is that FLEX uses 256 byte sectors. This fact along with the necessity of setting up special information on FLEX disks requires that all disks be formatted or initialized with the FLEX format before use. This initialization procedure is done with the "NEWDISK" command. Since the NEWDISK command deals directly with the disk controller to write entire tracks of data, it must be user supplied. If the disk controller in use is either a Western Digital 1771 or 1791 based floppy disk controller, the supplied skeletal NEWDISK routine in Appendix F can be used with only minor modifications. If not, the skeletal NEWDISK may be used as a guide, but the user's NEWDISK routine will have to essentially be written from the ground up. The NEWDISK routine is not a simple one and may take considerable effort to develop. It is, however, essential to the use of FLEX.

9.1 The General NEWDISK Procedure

Let us begin by discussing the actual functions of a NEWDISK routine. They are six in number:

1) Formatting a blank disk with 256 byte sectors linked together by the first two bytes of data in each.
2) Testing all the sectors written and removing any bad sectors by altering their links such that they are removed from the free chain.
3) Establishing the end of the free chain by writing a forward link of 0.
4) Initializing the directory on track #0.
5) Setting up the required information in the System Information Record (sector #3 on track #0).
6) Storing the FLEX boot loader program on track #0 sector #1.

Now let's discuss each step in more detail.

9.1.1 Formatting the disk with 256 byte sectors.

This step is the most difficult part of the NEWDISK process. Each track must be written so that there are a certain number of 256 byte sectors on each track. With most controllers it is necessary for such a routine to do all the track setup including gaps, sector ID fields, data fields, and CRC values. The actual data in each sector is really not critical. IBM puts a hex E5 in each byte, Technical Systems Consultants generally puts zeroes in each byte. This step of the NEWDISK routine is also where all the sector linking takes place. As discussed previously, all the sectors are linked together by addresses stored in the first two bytes of the data field of each sector. The first byte is the track on which the next sector in the chain is found, and the second byte is the sector number of the next sector on that track. For example, the first two data bytes of sector #1 on track #1 should be $01 and $02 which says the next sector in the chain is on track number $01 and sector number $02. If a disk has 15 ($0F) sectors on each track, the last sector on track #1 (sector #15) should have $02 and $01 as its first two data bytes. This means the next sector in the chain is on track number $02 and sector number $01. When this step is complete, you should have a disk with one long chain of linked sectors beginning with sector #1 on track #0 and ending with the last sector on the last track. It may be desirable to implement "sector interleaving" in this formatting step. See section 9.4 for a description of this technique.

9.1.2 Testing and removing bad sectors.

This step is intended to verify that all the sectors written in the first step can be properly read. This simply requires attempting to read every sector on the disk and checking for errors. If there are no errors, this step is complete. If there are bad sectors found on track #0 and the sector number is #5 or less, a fatal error should be reported and the NEWDISK routine aborted. If bad sectors are found elsewhere, they should be linked out of the chain of sectors. This means the forward link in the sector preceding the bad one should be changed so that it points to the next sector after the bad one. This is not a trivial task if the bad sector is the last one on a track or if there are two bad sectors in a row. Before starting this check for bad sectors, you should have a count of the number of data sectors on the disk. Data sectors are all sectors except those on track #0. As bad data sectors are found and effectively removed by the re-linking process, this count of total data sectors should be decremented. In the end, this count will be placed in the System Information Record so that FLEX can know when a disk is full.

9.1.3 Establishing the end of the free chain.

The end of the free chain of data sectors is easily established by changing the forward link (first two data bytes) of the last good sector on the disk to zeroes. The single sector read and write routines from FLEX can be used for this purpose.

9.1.4 Initializing the directory.

The directory starts with sector #5 on track #0 and initially ends with the last sector on track #0. This step should establish the end of the chain of directory sectors by changing the forward link of the last good sector on track #0 to zeroes. The 252 data bytes in all directory sectors must also be zeroes. The single sector read and write routines from FLEX can be used for these purposes.

9.1.5 Setting up the System Information Record (SIR).

The SIR contains specific information about the disk which should be setup by this step. Each item of information stored in the SIR has a defined offset or location within the sector. The following table gives the beginning and ending offset of each piece of information in decimal. Note that the first byte of the SIR is an offset of 0.

Begin	End	Information
0	1	Two bytes of zeroes (Clears forward link)
16	26	Volume name in ASCII
27	28	Volume number in binary
29	30	Address of first data sector (Track-Sector)
31	32	Address of last data sector (Track-Sector)
33	34	Total number of data sectors in binary
35	37	Current date (Month-Day-Year) in binary
38	38	Highest track number on disk in binary
39	39	Highest sector number on a track in binary

The volume name and number are arbitrary as supplied by the user. If they weren't bad, the first and last data sectors will be sector #1 on track #1 and the last sector on the last track. The total number of available data sectors does not include any sectors from track #0. The highest track number is the actual number of the last track. For example, there are 77 tracks on a standard eight inch disk but since the first one is numbered as #0, the highest track number would be #76 or hex 4C.

9.1.6 Storing the FLEX boot loader on the disk.

So that any disk can be used for booting purposes, we must have the FLEX loader program stored on track #0 sector #1. The NEWDISK routine is a logical place to do this, although this step may be omitted if the disk will not be used for booting. A convenient way to store the loader on disk is to let NEWDISK assume that the loader is in memory at $C100. Thus NEWDISK need only write a single sector of data to sector #1 on track #0 beginning at $C100. The actual FLEX loader program can then be simply appended onto the NEWDISK program so that whenever NEWDISK is loaded, the FLEX loader code is also loaded. Of course, if your FLEX loader is larger than 256 bytes, you would have to save two sectors on the disk.

9.2 A Western Digital NEWDISK Example

If your disk controller hardware utilizes either a Western Digital 1771 or 1791 floppy disk controller chip, you should be able to use the skeletal NEWDISK supplied in Appendix F and on the supplied FLEX disks. The only part of this skeletal NEWDISK which must be added is the Write Track routine near the end. A full specification of the write track routine is give.n in the listing comments.

This NEWDISK will write 256 bytes of data found at $C100 onto the disk after it is formatted. It is assuming that a FLEX loader program is resident in that memory area when NEWDISK is executed. For testing purposes, it is not necessary that any meaningful data be at location $C100. NEWDISK will still write the data to disk, but since you are only in a testing stage and will not be attempting to boot from the new disk, it makes no difference what is on track #0 sector #1. When you finally have NEWDISK working, you can add the FLEX loader routine to be saved on disk. Assuming you have the FLEX loader code in a binary file on disk, the easiest way to put it and NEWDISK together is with the APPEND command. Thus when this appended version of NEWDISK is loaded, the FLEX loader will'also be loaded into the $C100 area. The command to do this appending should look something like this:

```
+++APPEND NEWDISK.BIN,LOADER.BIN,NEWDISK.CMD
```

where the version of NEWDISK you have been working on is assumed to be called NEWDISK.BIN and the FLEX loader file is called LOADER.BIN. The resulting file is a completed NEWDISK ready for use and is called NEWDISK.CMD.

9.3 Hints on a Non-Western Digital NEWDISK

If the user does not have a Western Digital based disk controller, he will essentially have to write his NEWDISK from the ground up using the description given in section 9.1. It may be helpful to use the Western Digital NEWDISK found in Appendix F as a guide. There is a large section of that sample which can be used in a non-Western Digital NEWDISK.

There are two major sections to the skeletal NEWDISK. The first actually does the disk formatting as described in section 9.1.1. It calls on the Write Track routine documented in the NEWDISK listing. This section can probably not be used at all in a non-Western Digital NEWDISK. The second section performs steps 2 through 6 as described in section 9.1. It can probably be used as is in any NEWDISK the user may write. The only changes will probably be the locations from where the values written into the SIR are picked up.

9.4 Sector Interleaving

Sector interleaving is a technique which can be applied to floppy disks to maximize the speed with which sequential disk data can be read. For the most part, files are stored in contiguous groups of sectors on a disk. For example, a file may occupy six sectors on a single track with numbers 3 through 8. If this file was read by FLEX, sector 3 would be read first, followed by sector 4, then sector 5, etc. If these sectors are physically sequential on the disk, we would see a phenomenon often referred to as "missing revolutions". This is a consequence of FLEX not being able to read all the sectors in one revolution of the disk. It takes a certain amount of time for the data to be handled by FLEX and the address of the next sector to be readied. In this time, the next physical sector or sectors after the one just read will have already passed the read head. In fact, our hypothetical 6 sector file would require 6 revolutions of the disk to read. Now with a disk spinning at 360 RPM this may not sound like much, but it does add up and is very noticeable.

A simple solution to this problem is sector interleaving. This refers to the technique of placing the sectors on a track in an order which is not physically contiguous. In other words, while the first physical sector on the track may be numbered as #1, the second physical sector would not be #2. Sector number 2 (the second "logical" sector) will be placed a few physical sectors away from the first logical sector so that FLEX has time to do its processing before that sector comes under the read head. Thus logical sector number 2 may be put in physical sector number 6. The logical sectors are thus "interleaved".

The distance (number of physical sectors) between logical sectors for maximum performance is dependant on several factors. These factors include how fast the disk is rotating, how many sectors are on a track, and most importantly whether the user wishes to optimize the system for reading or writing and whether for binary or text files since it takes different times for FLEX to process the data. The distance or interleaving amount used is best found by experimentation. Technical Systems Consultants usually formats disks with interleaving optimized for reading text files. As an example, the following are interleaving schemes used by Technical Systems Consultants for single-sided, single-density 8 and 5 1/4 inch disks.

<u>Eight inch disk</u>

physical sector #	logical sector #
1	1
2	6
3	11
4	3
5	8
6	13
7	5
8	10
9	15
10	2
11	7
12	12
13	4
14	9
15	14

<u>Five inch disk</u>

physial sector #	logical sector #
1	1
2	3
3	5
4	7
5	9
6	2
7	4
8	6
9	8
10	10

The user may want to experiment with different interleaving configurations to determine the best setup for his needs.

10.0 PRINTER SPOOLING and INTERRUPT HANDLING

Printer spooling is a term which refers to the process of sending a disk file to the printer for output while other use is being made of the system. In effect, this is a dedicated multi-tasking operation. There are two dedicated tasks: the normal operation of FLEX and the spooling of a disk file out to a printer. Normally only the first of these two tasks is being executed, that being the normal running of FLEX. However, when a PRINT command is executed under FLEX, the second task is started and both tasks appear to be running at the same time. In actuality there must be a hardware interval timer in the system capable of producing interrupts. The PRINT command starts the printer spooling process and turns this timer on. Basically what happens from there is that each time an interrupt comes through, FLEX switches to the other task so that both appear to be occurring simultaneously. This section covers the implementation of this printer spooling feature and the interrupt handling required.

10.1 Hardware Requirements

As mentioned, the system must have a hardware interval timer capable of producing interrupts in order to implement printer spooling. The interrupts produced must be IRQ type interrupts. This timer must be able to be turned on or off by the system under software control (either producing interrupts or not). The routines for controlling this timer must be user supplied and are discussed in section 10.3.
The time interval between interrupts can vary considerably, but a recommended value is 10 milliseconds. If the printer in use is a buffered parallel type printer, this interval can be higher but should not go over 100 milliseconds.

10.2 Firmware Requirements

If printer spooling is to be implemented, FLEX must obviously have control of the interrupts. Both the IRQ and the SWI3 interrupts are used, the IRQ's coming from the hardware timer and the SWI3's coming from FLEX software and drivers. FLEX requires that there be a specific location in RAM memory for each interrupt into which the address of an interrupt handling routine can be stored. These locations could be the actual interrupt vectors for the CPU, but generally the system's monitor ROM has defined locations in lower RAM where the interrupt handling routine vectors can be stored.

10.3 Additional Console I/O Drivers for Printer Spooling

In order to implement the printer spooling feature, it is necessary to complete the remaining routines in the Console I/O Driver Package. These are the routines associated with controlling the timer and handling the interrupts. There is an entry for the address of each of these routines in the Console I/O Driver package's vector table as seen in Section 3.

TMINT Address at $D3F1
This routine performs any necessary initialization for the interrupt timer used by the printer tpooling process. Any registers may be modified.

TMON Address at $D3EF
This routines "turns the timer on" or in other words starts the interval IRQ interrupts. Any registers may be modified.

TMOFF Address at $D3ED
This routine "turns the timer off" or in other words stops the interval IRQ interrupts. Any registers may be modified.

IRQVEC Address at $D3EB
The IRQ vector is the address of a two byte location in RAM where FLEX can stuff the address of its IRQ interrupt handler routine. In other words, when an IRQ interrupt occurs control should be transferred to the address stored at the location specified by the IRQ vector. This IRQ vector location (address) should be placed in the Console I/O Driver vector table.

SWIVEC Address at $D3E9
The SWI3 vector is the address of a two byte location in RAM where FLEX can stuff the address of its SWI3 interrupt handler routine. In other words, when an SWI3 interrupt occurs control should be transferred to the address stored at the location specified by the SW13 vector. This SWI3 vector location (address) should be placed in the Console I/O Driver vector table.

IHNDLR Address at $D3E7
The Interrupt Handler routine is the one which will be executed when an IRQ interrupt occurs. If using printer spooling, the routine should first clear the interrupt condition and then jump to the 'change process' routine of the printer spooler at $C700. If not using printer spooling, this routine can be setup to do whatever the user desires. If it is desirable to do both printer spooling and have IRQ's from another device (besides,the spooler timer), this routine would have to determine which device had caused the interrupt and handle it accordingly.

10.4 Disk Driver Changes for Printer Spooling

There is one set of changes which should be added to your disk driver routines if printer spooling is implemented. As described earlier, when printer spooling it taking place, FLEX is essentially a two task system. Now for the best possible performance and to ensure that FLEX does not miss characters typed on the console while it is busy printing, the printer task should have less priority than the task which is the running of FLEX. One way to give the printer task less priority is to never wait for disk operations to take place while executing the printer task. For example, if we are currently running the printer task (the FLEX task is inactive) and it is necessary to read a sector of data from the file to be printed, we should not wait for the sector read operation to take place. Instead we should initiate the sector read and then immediately switch back to the FLEX task. This switch to the other task is performed with a software interrupt (SWI3). The drivers can tell if they are running the printer task by checking a byte called PRCNT at $CC34. If non-zero, the printer task is the one currently executing. Thus, the code which must be added to the drivers should look something like this:

```
         TST  PRCNT      EXECUTING PRINTER TASK?
         BEQ  CONTIN     SKIP IF NOT PRINTING
         SWI3            IF PRINTING, SWITCH TASKS
  CONTIN ...             CONTINUE WITH OPERATION
```

This test should be placed just before each point in your drivers which could possibly take a long time to execute. The following points are likely candidates for this test:

1) A sector read operation
2) A sector write'operation
3) A seek operation
4) The delay in CHKRDY (if there is one)
5) Any waiting or delaying in the drivers

See the sample set of drivers in Appendix G for examples of the implementation of this task switching.

11.0 ADVANCED DISK ADAPTATIONS

Now that the user has a fully functional version of FLEX implemented for a single-sided, single-density, soft-sectored floppy disk system, he may wish to upgrade the system to include features such as double-sided disks, double-density disks, hard disks, mixtures of disk types, etc. This section is intended to give suggestions for implementing some of these features.

11.1 Double-Sided Disks

FLEX should treat the double-sided disk just like a single-sided one with twice as many sectors on each track. Thus a double-sided standard eight inch disk will still have 77 total tracks. Instead of 15 sectors per track, however, there will now be 30. All that must happen is that the drivers must check to see which sector number they are preparing to read or write. If less than or equal to the number of sectors per track on a single-sided disk, the drivers should select side #0. If greater than the number of sectors per track on a single-sided disk, the drivers should select side #1. Side #0 is actually the bottom side of a disk or the side opposite the label. This selection of side should be done in the seek routine.

As an example, let's examine a portion of a seek routine for some hypothetical system which is to be setup for double-sided eight inch floppies. The code might look something like this:

```
SEEK    STB  SECTOR    SAVE SECTOR NUMBER
        CLR  SIDE      ASSUME SIDE #0
        CMPB #15       WHICH SIDE IS SECTOR ON?
        BLS  SEEK1     SKIP IF ON SIDE #0
        LDB  #$FF      ELSE, SELECT SIDE #1
        STB  SIDE
SEEK1   ...            CONTINUE WITH SEEK OPERATION
```

Of course the value of 15 would change depending on the actual disk format desired. For example, Technical Systems Consultants formats single-density, single-sided minifloppy disks with 10 sectors per track. The actual side select mechanism for your controller may also be entirely different than the example shows.

11.2 Double-Density Disks

Double-density disks are usually not really different from single-density disks with the exception of the fact that there are more sectors per track. Technical Systems Consultants has altered this concept slightly. In our specifications, a "double-density disk" actually has track #0 written in single-density while all other tracks are written in double-density. This means a slight loss in the number of sectors which could be put on the disk, but the advantage is that a disk system can now accept either single or double density disks interchangeably without requiring the operator to specify what type of disks are in use. This technique does require software control of the density selection, but most double density controllers permit this.

Anytime the drivers are accessing a sector on track #0, they automatically select single density. This permits the ROM boot program to be much simpler. On all other tracks the drivers make one attempt to read or write a sector. If there is an error, the drivers should switch to the other density and return. Since FLEX makes several attempts to read or write a sector when errors are returned, if the error was due to attempting to read under the wrong density, this will be taken care of on the next retry. Best results will be achieved if the drivers keep track of what density they think each drive is. This will result in correct reading and writing most of the time. If, at some point, the operator changes a disk to one of the opposite density, the first access of that disk will cause an error (which should be transparent to the user since FLEX will retry) but on future accesses the right density should be known and used such that there are no more errors.

Let's examine another hypothetical disk system case and see how all this fits together. Somewhere in the drivers will be a set of four bytes which indicate the density which the drivers assume each drive to be. If a byte is zero, the drivers will attempt a double-density access; if non-zero, a single-density access will be attempted. These bytes might be setup as follows:

```
DNSITY  FCB  0,0,0,0    INITIALIZED TO DOUBLE-DENSITY
```

Now at the end of our read and,write routines we must check for an error. If there was no error vie can immediately exit. If there was an error, we should switch to the opposite density by indicating this switch in the bytes setup above. The code for this portion of one of these routines might look something like this:

```
READ    ...              MAIN BODY OF READ ROUTINE
        ...              ERROR CONDITION LEFT IN B
        ...
READ6   BITB  #$10       SECTOR NOT FOUND ERROR?
        BEQ   READ8      SKIP IF OTHER ERROR
        PSHS  B          SAVE ERROR CONDITION
        LDX   #DNSITY    POINT TO DENSITY TABLE
        LDB   CURDRV     GET CURRENT DRIVE NO.
        COM   B,X        SWITCH TO OPPOSITE DENSITY
        PULS  B          RESTORE ERROR CONDITION
```

```
READ8  BITB  #$FC      SHOW ANY ERRORS IN CC
       RTS
```

As can be seen, if the sector could not be found (the only error using the wrong density should give), the correct density flag byte for the current drive is switched to the opposite density. This read routine need not attempt to re-read the sector with this new density since FLEX will do so when it performs a retry.

There is yet another consideration for the double density disk which is also a double-sided disk. The maximum number of sectors per track on one side is different for double-density than single-density. This must be considered when the seek routine makes its decision as to which side to select. For a double-sided, double-density eight inch disk system, the portion of the seek routine given above might look like the following:

```
SEEK   STB   SECTOR    SAVE SECTOR NUMBER
       CLR   SIDE      ASSUME SIDE #0
       PSHS  B,X       SAVE REGISTERS
       LDX   #DNSITY   POINT TO DENSITY TABLE
       LDB   CURDRV    GET CURRENT DRIVE NO.
       LDB   B,X       GET THE DENSITY FLAG
       COMB            00 - SINGLE, FF - DOUBLE
       STB   DENSITY   SET CONTROLLER DENSITY
       PULS  B,X       RESTORE REGISTERS
       BEQ   SINGLE    SKIP IF SINGLE DENSITY
DOUBLE CMPB  #26       WHICH SIDE IS SECTOR ON?
       BLS   SEEK1     SKIP IF ON SIDE #0
       BRA   SIDEI     ELSE, SELECT SIDE #1
SINGLE CMPB  #15       WHICH SIDE IS SECTOR ON?
       BLS   SEEK1     SKIP IF ON SIDE #0
SIDE1  LDB   #$FF      ELSE, SELECT SIDE #1
       STB   SIDE
SEEK1  ...             CONTINUE WITH SEEK OPERATION
```

First we have determined what density the drivers remember the disk as being. The controller is then set to that density. In this example, we assume that storing a $00 in DENSITY selects single density and storing an $FF selects double density. Having done this we check which side the desired sector should be found on. Note that there are two separate checks: one for a single-sided disk and one for a double-sided disk. The correct check is chosen depending on the density in use. In this example, the numbers used for the maximum number of sectors per track on one side are 15 for single-density and 26 for double-density. These are the standard values used by Technical Systems Consultants for eight inch disks.

11.3 Other Disk Configurations

There is nothing restricting the FLEX Disk Operating System to operation on floppy disks only. It is recommended that there be at least one soft-sectored floppy disk drive on a system for software distribution purposes, but there is nothing to keep FLEX from running on a hard-sectored floppy, on a Winchester technology hard disk, or on most any type of disk drive. FLEX can also support a mixture of up to four drives. FLEX has, in fact, been operating for some time on systems using all these configurations. Two areas which must be altered for such operations are the disk driver routines and the NEWDISK routine.

Particular attention must be paid to the amount of storage available on a hard disk. Since a sector address in FLEX consists of an 8-bit track number and an 8-bit sector number, a maximum of 65,535 sectors can be addressed by FLEX. With 256 bytes per sectors, this means one FLEX drive can hold a maximum of 16 megabytes of formatted data. Larger hard disks could be used, but it would require splitting the single hard disk drive into two logical FLEX drives.

Connecting mixtures of drive types onto one system is relatively simple. The driver routines must be written such that they check which drive is specified before performing an operation. Then the appropriate routines for the type of drive associated with that drive number should be called. Thus there must essentially be a set different set of routines for each type drive. For example, suppose we have two eight inch floppys connected as drive numbers 0 and 1, and have a Winchester technolog y hard disk connected as drive number 2. The beginning of the single sector read driver routine might look somet hing like this:

```
READ   PSHS  A          SAVE THE TRACK NUMBER
       LDA   CURDRV     CHECK CURRENT DRIVE
       CMPA  #2         IS IT THE HARD DISK?
       PULS  A          RESTORE TRACK NUMBER
       BEQ   HDREAD     DO HARD DISK READ
       BRA   FLREAD     ELSE, DO FLOPPY READ
```

This does, of course, usurp more memory, but one could conceivably setup a system with one soft-sectored 8 inch floppy, one soft-sectored 5 inch floppy, one Winchester hard disk, and one hard-sectored 8 inch floppy. It would also be conceivable to have four different types of hard disks on a system, each with a different controller.

11.4 NEWDISK Routines

One requirement for each type of disk integrated into a system is the NEWDISK routine. As you have seen, the NEWDISK routine must be peculiar to each type disk drive. A Winchester hard disk, for example, will require its own NEWDISK or formatting program capable of formatting the disk into 256 byte sectors which are addressable through the FLEX drivers. A system with mixed drive types must either have a different NEWDISK command for each, or a single NEWDISK that is intelligent enough to determine the drive type and format the disk accordingly.

12.0 ADDITIONAL CUSTOMIZATION

There are a few features which can be further customized in FLEX that have not been discussed thus far. This section is devoted to these features.

12.1 Setting a Default MEMEND

During FLEX's initialization procedure (done only upon booting) the amount of memory in the system is checked and the last valid memory address saved in MEMEND at $CC2B. By default, the upper limit of this memory check routine is $BFFF so that MEMEND will be below FLEX. It is possible to change this upper limit such that a section of memory just below FLEX is saved for some user required routines or to avoid some peripheral device which may be addressed in that region. This is done by simply overlaying the value stored at $CC2B (should be a $BFFF) with the upper memory limit you desire. This overlaying must be done before the initialization is performed. The easiest way to do this is to simply append the code to overlay this address onto the end of the core of FLEX when preparing a bootable version of FLEX. Thus even though the value $BFFF will be loaded when the core part of FLEX is brought into memory, when the sections of code which the user appended are brought in, the user's upper limit will replace the $BFFF. A convenient method to append a new MEMEND limit is to place the code in the Console I/O Driver Package. For example, if we wanted to limit MEMEND to $7FFF, the following code could be placed at the end of the Console Driver package:

```
        ORG   $CC2B     ORIGIN AT MEMEND LOCATION
        FDB   $7FFF     CODE TO STORE $7FFF AT MEMEND
```

That's all there is to it!

12.2 Altering the FLEX Date Prompt

Upon booting FLEX, the first thing the user sees after a FLEX banner message is a prompt for the current date. This date is stored in the appropriate locations in FLEX as detailed in the Advanced Programmer's Guide. It may be desirable in certain applications to do away with this date prompt or to obtain the date by some other means (such as reading a time of day clock). This version of FLEX provides this ability. There is a subroutine in the FLEX initialization code which displays the prompt, obtains the response, and stores it in FLEX. A call to this subroutine (JSR instruction) is located at $CA02. The user can overlay this call in much the same way that MEMEND was overlayed in the previous section. If some alternate method of obtaining the date is desired, the subroutine call can be overlayed with a call (JSR) to a user supplied subroutine. If the date prompt is to be eliminated, one may simply place a return instruction (RTS) at $CA02. As an example, if we wished to disable the date prompt we might place the following code at the end of the Console I/O Driver package:

```
        ORG   $CA02     CALL IS AT $CA02
        RTS             IMMEDIATELY RETURN
```

Note that if the date prompt is disabled, the system will have garbage in the date locations and any use of the date by FLEX will reflect this.

12.3 Replacing Printer Spooler Code

There is an area of FLEX from $C700 through $C83F which has been defined as the printer spooler code area. If the user does not intend to implement printer spooling in his system, some of this space may be used for other purposes. In particular, the area from $C71C through $C83F may be used. For example, the user may overflow his disk or console driver routines into this area or may overflow his printer driver routines here. If this space is to be used, however, there are two changes which must be made. First is to disable the routines which are presently stored in this area by altering the jump table. This jump table is at the beginning of the printer spooler area and has 6 entries (3 bytes per entry). Each routine to which this jump table points is terminated with a return (RTS). Thus, it is possible for us to "disable" all six routines by replacing the jumps in the jump table with returns. This is basically protection to ensure nothing will attempt to use the jump table.

The second change to be made is to force the queue count (number of files in the print queue) to zero. This is done by setting the byte at $C71B to zero.

The overlay code to disable the printer spooler section code might look something like this:

```
        ORG  $C700         JUMP TABLE STARTS AT $C700
PRSPLI  FCB  $39,$39,$39   REPLACE THE FIRST BYTE
PRSPL2  FCB  $39,$39,$39   OF EACH ENTRY WITH AN
PRSPL3  FCB  $39,$39,$39   RTS ($39) AND THE SECOND
PRSPL4  FCB  $39,$39,$39   TWO BYTES WITH ANYTHING
PRSPL5  FCB  $39,$39,$39
PRSPL6  FCB  $39,$39,$39
        ORG  $C71B         QUEUE COUNT IS AT $C71B
QCNT    FCB  0             FORCE QUEUE COUNT TO ZERO
```

Now the entire area from $C71C through $C83F can be used for any desired purpose. Note that overlaying the printer spooler jump table is done just as described for the overlay in section 12.1. It is NOT possible to place this overlay code into memory before loading FLEX as in that case the printer spooler code would overlay this code.

12.4 Mapping Filenames to Upper Case

There is a mechanism built into this version of FLEX which automatically maps all filenames and extensions Which go through FLEX's GETFIL routine into upper case. This mapping is often quite useful in that a file is referenced by name only and that name can be specified in either upper or lower case. When the GETFIL routine (see the FLEX Advanced Programmer's Guide for a description of this routine) is used to build a filename in an FCB, it checks a byte called MAPUP at location $CC49. If this byte is set to $60 (which it is by default), the name will be mapped, to upper case letters when placed in the FCB. In this manner, a file can be specified in either upper or lower case but will always be converted to upper and placed in the directory in upper case. If desired, this mapping can be turned off such that no mapping occurs and upper case names will be different than lower case names. This is done by merely changing the value stored in MAPUP at $CC49 to $FF. This change can be done at bootup.time by overlaying MAPUP in the same manner described in section 12.1.

13.0 MISCELLANEOUS SUGGESTIONS

The following suggestions are not specifically related to the adaptation of FLEX, but might be of use once FLEX is running.

13.1 Replacement Master FLEX Disks

Do not despair if you accidentally destroy both of the master FLEX disks supplied in this package. Replacement disks can be obtained from Technical Systems Consultants by sending proof of purchase of this package along with $15.00 for each disk ordered. Be sure to specify whether you require 8 or 5 1/4 inch disks and which version of FLEX you have (6800 or 6809). Please do not return the originals for recopying; we will only sell new master FLEX disks.

13.2 Initialized Disks Available

As a service to those who, for any reason, are unable to format their own diskettes, Technical Systems Consultants is selling boxes of 10 brand new disks which have been freshly initialized in the standard FLEX format. These are available in either 8 or 5 1/4 inch single-sided, single-density, soft-sectored formats and must be purchased by the box (10 per box). Prices are as follows:

Box of 8" disks $75.00
Box of 5 1/4" disks $75.00

This price is postage paid anywhere in the continental U.S.

13.3 The FLEX Newsletter

Technical Systems Consultants Inc. publishes a FLEX Newsletter which is full of 6800 and 6809 related FLEX articles. This newsletter is published on an irregular basis of about four per year and contains bug reports, suggestions and tips for using FLEX and related support software, news of new FLEX software packages, user comments, and occasionally includes a free FLEX utility listing. The newsletter costs $4.00 ($8.00 outside U.S. and Canada) for four issues. This is the best way to keep informed of what's happening in the world of FLEX.

13.4 Single Drive Copy Program

For practical use, it is recommended that FLEX (or any disk operating system) be run on at least a two drive system. This allows a user to easily back up his files and to easily create new disks for distribution. There is nothing, however, to keep FLEX from being used on a single drive system. In order to do so one will need a "single drive copy" program which allows files to be copied from one disk to another with only one drive on the system. This involves alternatively inserting two disks into the drive until the entire file, which may not fit in memory, has been copied. The user can certainly develop his own single drive copy routine or can purchase one from Technical Systems Consultants for $15.00. This includes a two page manual and object code disk. Be sure to specify 8 or 5 1/4 inch disk, 6800 or 6809, and include 3% for postage and handling (10% outside U.S. and Canada).

13.5 Give Us Some Feedback

Technical Systems Consultants Inc. is always interested in how and where its software packages are being installed. When you get FLEX up and running, drop us a line and let us know about your hardware configuration. If you would like to share the work you have done in adapting FLEX to your hardware, let us know... there is probably someone else with similar hardware who could benefit from your efforts.

APPENDIX A

6809 FLEX Memory Map

```
C000  -------
         I     System Stack
C080  -------
         I     Input Buffer
C100  -------
         I
         I
         I     Utility Area
         I
         I
C700  -------
         I
         I     Printer Spooler
         I
C840  -------
         I
         I     System/User FCB
         I
C980  -------
         I
         I     System I/O FCB's
         I     (FLEX Initialize at CA00)
         I
CC00  -------
         I     System Variables
CCC0  -------
         I     Printer Drivers
CCF8  -------
         I     System Variables
CD00  -------
         I
         I
         I     Disk Operating System
         I
         I
D370  -------
         I     Console I/O Drivers
D400  -------
         I
         I
         I     File Management-System
         I
         I
         I
DE00  -------
         I
         I     Disk Drivers
         I
E000  -------
```

APPENDIX B
Disk Formats

Almost any conceivable format of floppy disk can be supported by the FLEX Disk Operating System. Technical Systems Consultants Inc. has, however, defined two formats which should be a standard for all FLEX disks to be distributed from installation to installation. Several other formats have also been defined but are not necessarily fixed. All single-density formats are essentially compatible with the 256 byte per sector IBM format. With the exception of track #0 which is in single-density, the defined double-density formats are also essentially compatible with the 256 byte per sector IBM format.

B.1 Defined Distribution Formats

Technical Systems Consultants has defined one 8 inch and one 5 1/4 inch floppy disk format which should be a standard for any disk distributed from one system to another. This standard allows the exchange of software between a ny two FLEX systems with the same size disks. These formats are as follows:

1) 8" SINGLE-SIDED, SINGLE-DENSITY, SOFT-SECTORED DISK
   This disk should be comprised of 77 tracks (numbered 0 thru 76) with 15 sectors per track (numbered 1 thru 15).

2) 5 1/4" SINGLE-SIDED, SINGLE-DENSITY, SOFT-SECTORED DISK
   This disk should be comprised of 35 tracks (numbered 0 thru 34) with 10 sectors per track (numbered 1 thru 10).

B.2 Other Defined Formats

Technical Systems Consultants has defined several other disk formats as described below. These formats are in use in many installations, but there is nothing to restrict the user to them. They are simply offered as guidelines for writing NEWDISK routines. In the following table, SS and DS refer to Single and Double Sided respectively, and SD and DP refer to Single and Double Density respectively.

		Sectors per Track Other than #0		Sectors per Track On Track #0	
Disk Type	# of Tracks	One Side	Total	One Side	Total
8" DS,SD	77	15	30	15	30
8" DS,DD	77	26	52	15	30
8" SS,DD	77	26	26	15	15
5 1/4" SS,SD	40	10	10	10	10
5 1/4" DS,SD	35 or 40	10	20	10	20

NOTES:

1) On double-density disks, track #0 is formatted in single-density to facilitate automatic density selection.
2) Side #0 is the bottom of the disk (opposite the label).
3) Sector size is 256 bytes.
4) Track numbers always begin with #0 and sector numbers always begin with #1.

APPENDIX C
Single Sector READ/WRITE Test Utility

```
                  * TEST UTILITY
                  *
                  *       COPYRIGHT © 1980 by
                  * Technical Systems Consultants, Inc.
                  *        111 Providence Road
                  * Chapel Hill, North Carolina 27514
                  * TESTS SINGLE SECTOR READ AND WRITE ROUTINES.
                  * PROGRAM PROMPTS USER FOR FUNCTION (F?) TO WHICH THE
                  * USER CAN RESPOND 'R' (READ) OR #W@ (WRITE). THEN IT
                  * PROMPTS FOR SINGLE DIGIT DRIVE NUMBER (D?), TWO DIGIT
                  * HEX TRACK NUMBER (T?) AND TWO DIGIT HEX SECTOR
                  * NUMBER (S?). AFTER PERFORMING THE FUNCTION, TEST
                  * REPEATS THE PROMPTING FOR ANOTHER FUNCTION.
                  *
                  * ASSUMES THE CONSOLE I/O PACKAGE DRIVERS ARE RESIDENT.
                  * BEGIN EXECUTION BY JUMPING TO $0100.
                  *

                  * EQUATES

             D3FB INCH    EQU    $D3FB
             D3F9 OUTCH   EQU    $D3F9
             D3F5 TINIT   EQU    $D3F5
             D3F3 MONITR  EQU    $D3F3
             C07F STACK   EQU    $C07F
             C840 FCB     EQU    $C840
             1000 BUFFER  EQU    $1000
             DE00 READ    EQU    $DE00
             DE03 WRITE   EQU    $DE03
             DE0C DRIVE   EQU    $DE0C

                  * TEMPORARY STORAGE

0020                      ORG    $0020
0020              COMMND  RMB    1
0021              TRACK   RMB    1
0022              SECTOR  RMB    1

                  * START OF PROGRAM

0100                      ORG    $0100

0100 10CE C07F    TEST    LDS    #STACK      SETUP STACK
0104 AD   9F D3F5         JSR    [TINIT]     INITIALIZE TERMINAL
```

```
                  * GET COMMAND

0108 10CE C07F    TEST1   LDS    #STACK     RESET STACK
010C BD   58              BSR    PCRLF
010E 86   46              LDA    #'F        PROMPT FOR FUNCTION
0110 BD   4A              BSR    PROMPT
0112 BD   5F              BSR    INPUT      GET RESPONSE
0114 81   52              CMPA   #'R        READ COMMAND?
0116 27   08              BEQ    TEST2
0118 81   57              CMPA   #'W        WRITE COMMAND?
011A 27   04              BEQ    TEST2
011C 6E   9F D3F3         JMP    [MONITR]   EXIT THE PROGRAM
0120 97   20      TEST2   STA    COMMND     SAVE COMMAND
0122 86   44              LDA    #'D        PROMPT FOR DRIVE
0124 BD   36              BSR    PROMPT
0126 BD   01C9            JSR    INHEX      GET RESPONSE
0129 81   04              CMPA   #4         ENSURE 0 TO 3
012B 24   DB              BHS    TEST1
012D B7   C843            STA    FCB+3      SAVE IT
0130 86   54              LDA    #'T        PROMPT FOR TRACK
0132 BD   2E              BSR    HPRMPT     GET HEX PROMPT
0134 97   21              STA    TRACK
0136 86   53              LDA    #'S        PROMPT FOR SECTOR
0138 8D   28              BSR    HPRMPT     GET HEX RESPONSE
013A 97   22              STA    SECTOR     SAVE IT
013C BD   28              BSR    PCRLF      DO LINE FEED

                  * GOT COMMAND, NOW DO IT

013E 96   20              LDA    COMMND     GET COMMAND
0140 81   57              CMPA   #'W        A WRITE COMMAND?
0142 26   52              BNE    DOREAD     IF NOT, ITS A READ
0144 8D   37              BSR    SELECT     SELECT DRIVE
0146 BE   1000            LDX    #BUFFER    POINT TO BUFFER
0149 DC   21              LDD    TRACK      POINT TO TRACK & SECTOR
014B BD   DE03            JSR    WRITE      WRITE THE DATA
014E 26   35              BNE    ERROR
0150 BD   14              BSR    PCRLF
0152 86   4F              LDA    #'O        PRINT  OK
0154 8D   23              BSR    OUTPUT
0156 86   4B              LDA    #'K
0158 BD   1F              BSR    OUTPUT
015A 20   AC              BRA    TEST1      DO AGAIN

                  * PROMPT ROUTINES

015C BD   08      PROMPT  BSR    PCRLF      DO LINE FEED
015E 8D   19              BSR    OUTPUT     OUTPUT PROMPT LETTER
0160 20   15              BRA    QUEST      PRINT QUESTION MARK
0162 BD   F8      HPRMPT  BSR    PROMPT     DO PROMPT
0164 20   56              BRA    INBYTE     GET HEX BYTE
```

```
                        * CARRIAGE RETURN LINE FEED ROUTINE

0166 34    02           PCRLF    PSHS    A            SAVE A
0168 86    0D                    LDA     #$0D         RETURN
016A 8D    0D                    BSR     OUTPUT
016C 86    0A                    LDA     #$0A         LINE FEED
016E 8D    09                    BSR     OUTPUT
0170 35    02                    PULS    A            RESTORE A
0172 39                 RET      RTS

                        * I/O ROUTINES

0173 6E    9F D3FB      INPUT    JMP     [INCH]
0177 86    3F           QUEST    LDA     #'?
0179 6E    9F D3F9      OUTPUT   JMP     [OUTCH]

                        * DRIVE SELECT ROUTINE

017D 8E    C840         SELECT   LDX     #FCB
0180 BD    DE0C                  JSR     DRIVE
0183 27    ED                    BEQ     RET          RETURN IF NO ERROR

                        * DRIVER ERROR

0185 8D    0F           ERROR    BSR     PCRLF
0187 86    45                    LDA     #'E
0189 8D    EE                    BSR     OUTPUT
018B 86    3D                    LDA     #'=
018D 8D    EA                    BSR     OUTPUT
018F 1F    98                    TFR     B,A          GET ERROR CODE
0191 8D    4D                    BSR     OUTHEX
0193 16    FF72                  LBRA    TEST1        START OVER

                        * DO SINGLE SECTOR READ

0196 8D    E5           DOREAD   BSR     SELECT       SELECT DRIVE
0198 8E    1000                  LDX     #BUFFER      POINT TO BUFFER
019B DC    21                    LDD     TRACK        POINT TO TRACK & SECTOR
019D BD    DE00                  JSR     READ         READ THE DATA
01A0 26    E3                    BNE     ERROR

                        * DUMP DATA TO CONSOLE

01A2 8E    1000                  LDX     #BUFFER
01A5 86    10                    LDA     #16          NO OF LINES
01A7 34    02           DUMPI    PSHS    A            SAVE NO OF LINES
01A9 8D    BB                    BSR     PCRLF
01AB C6    10                    LDB     #16          NO OF BYTES
01AD A6    80           DUMP2    LDA     0,X+         GET A BYTE
01AF 8D    2F                    BSR     OUTHEX       OUTPUT IT
01B1 5A                          DECB                 DONE WITH LINE?
01B2 26    F9                    BNE     DUMP2
01B4 35    02                    PULS    A            GET NO OF LINES
01B6 4A                          DECA                 DONE WITH DUMP
```

```
01B7 26   EE                BNE    DUMP1      LOOP IF NOT
01B9 16   FF4C              LBRA   TEST1      GET NEXT COMMAND

                  * INPUT HEX BYTE ROUTINE
01BC 8D   0B      INBYTE    BSR    INHEX
01BE 48                     ASLA
01BF 48                     ASLA
01C0 48                     ASLA
01C1 48                     ASLA
01C2 34   02                PSHS   A
01C4 8D   03                BSR    INHEX
01C6 AB   E0                ADDA   0,S+
01C8 39           RETN      RTS

01C9 8D   A8      INHEX     BSR    INPUT
01CB 80   47                SUBA   #$47
01CD 2A   0C                BPL    INERR
01CF 8B   06                ADDA   #6
01D1 2A   04                BPL    INH2
01D3 8B   07                ADDA   #7
01D5 2A   04                BPL    INERR
01D7 8B   0A      INH2      ADDA   #10
01D9 2A   ED                BPL    RETN
01DB 8D   9A      INERR     BSR    QUEST      PRINT A QUESTION MARK
01DD 16   FF28              LBRA   TEST1      GO START OVER

                  * OUTPUT HEX BYTE (FOLLOWED BY SPACE)

01E0 34   02      OUTHEX    PSHS   A
01E2 44                     LSRA
01E3 44                     LSRA
01E4 44                     LSRA
01E5 44                     LSRA
01E6 8D   08                BSR    OUTHR
01E8 35   02                PULS   A
01EA 8D   04                BSR    OUTHR
01EC 86   20                LDA    #$20
01EE 20   89                BRA    OUTPUT
01F0 84   0F      OUTHR     ANDA   #$0F
01F2 8B   90                ADDA   #$90
01F4 19                     DAA
01F5 89   40                ADCA   #$40
01F7 19                     DAA
01F8 16   FF7E              LBRA   OUTPUT


                            END
```

APPENDIX D
Quick FLEX Loader Utility

```
                   * QLOAD - QUICK LOADER
                   *
                   * COPYRIGHT (C) 1980 BY
                   * TECHNICAL SYSTEMS CONSULTANTS, INC.
                   * 111 PROVIDENCE RD. CHAPEL HILL, NC 27514

                   * LOADS FLEX FROM DISK ASSUMING THAT THE DISK I/O
                   * ROUTINES ARE ALREADY IN MEMORY.  ASSUMES FLEX
                   * BEGINS ON TRACK #1 SECTOR #1.  RETURNS TO
                   * MONITOR ON COMPLETION. BEGIN EXECUTION BY
                   * JUMPING TO LOCATION $C100
                   *

                   * EQUATES

             C07F  STACK   EQU    $C07F
             D3F3  MONITR  EQU    $D3F3
             DE00  READ    EQU    $DE00
             DE09  RESTORE EQU    $DE09
             DE0C  DRIVE   EQU    $DE0C
             C300  SCTBUF  EQU    $C300      DATA SECTOR BUFFER


                   * START OF UTILITY

C100                       ORG    $C100

C100 20    08      QLOAD   BRA    LOAD0

C102 00 00 00              FCB    0,0,0
C105 01            TRK     FCB    1          FILE START TRACK
C106 01            SCT     FCB    1          FILE START SECTOR
C107 00            DNS     FCB    0          DENSITY FLAG
C108 0000          LADR    FDB    0          LOAD ADDRESS

C10A 10CE C07F     LOAD0   LDS    #STACK     SETUP STACK
C10E BE   C300             LDX    #SCTBUF    POINT TO FCB
C111 6F   03               CLR    3,X        SET FOR DRIVE 0
C113 BD   DE0C             JSR    DRIVE      SELECT DRIVE 0
C116 8E   C300             LDX    #SCTBUF
C119 BD   DE09             JSR    RESTORE    NOW RESTORE TO TRACK 0
C11C FC   C105             LDD    TRK        SETUP STARTING TRK & SCT
C11F FD   C300             STD    SCTBUF
C122 108E C400             LDY    #SCTBUF+256

                   * PERFORM ACTUAL FILE LOAD

C126 BD   2F       LOAD1   BSR    GETCH      GET A CHARACTER
C128 81   02               CMPA   #$02       DATA RECORD HEADER?
```

```
C12A 27   0A                BEQ    LOAD2      SKIP IF SO
C12C 81   16                CMPA   #$16       XFR ADDRESS HEADER?
C12E 26   F6                BNE    LOAD1      LOOP IF NEITHER
C130 8D   25                BSR    GETCH      GET TRANSFER ADDRESS
C132 8D   23                BSR    GETCH      DISCARD IT
C134 20   F0                BRA    LOAD1      CONTINUE LOAD
C136 8D   1F       LOAD2    BSR    GETCH      GET LOAD ADDRESS
C138 B7   C108              STA    LADR
C13B 8D   1A                BSR    GETCH
C13D B7   C109              STA    LADR+1
C140 8D   15                BSR    GETCH      GET BYTE COUNT
C142 1F   894D              TAB               PUT IN B
C145 27   DF                BEQ    LOAD1      LOOP IF COUNT=0
C147 BE   C108              LDX    LADR       GET LOAD ADDRESS IN X
C14A 34   14       LOAD3    PSHS   B,X
C14C 8D   09                BSR    GETCH      GET A DATA CHARACTER
C14E 35   14                PULS   B,X
C150 A7   80                STA    0,X+       PUT CHARACTER
C152 5A                     DECB              END OF DATA IN RECORD?
C153 26   F5                BNE    LOAD3      LOOP IF NOT
C155 20   CF                BRA    LOAD1      GET ANOTHER RECORD

                   * GET CHARACTER ROUTINE - READS A SECTOR IF NECESSARY

C157 108C C400     GETCH    CMPY   #SCTBUF+256 OUT OF DATA?
C15B 26   10                BNE    GETCH4     GO READ CHARACTER IF NOT
C15D 8E   C300     GETCH2   LDX    #SCTBUF    POINT TO BUFFER
C160 EC   84                LDD    0,X        GET FORWARD LINK
C162 27   0C                BEQ    GO         IF ZERO, FILE IS LOADED
C164 BD   DE00              JSR    READ       READ NEXT SECTOR
C167 26   97                BNE    QLOAD      START OVER IF ERROR
C169 108E C304              LDY    #SCTBUF+4  POINT PAST LINK
C16D A6   A0       GETCH4   LDA    0,Y+       ELSE, GET A CHARACTER
C16F 39                     RTS

                   * FILE IS LOADED, RETURN TO MONITOR

C170 6E   9F D3F3  GO       JMPI   [MONITR]    JUMP TO MONITOR


                            END
```

APPENDIX E
Skeletal FLEX Loader Utility

```
                 * LOADER - FLEX LOADER ROUTINE
                 *
                 * COPYRIGHT (C) 1980 BY
                 * TECHNICAL SYSTEMS CONSULTANTS, INC.
                 * 111 PROVIDENCE RD, CHAPEL HILL, NC 27514
                 *
                 * LOADS FLEX FROM DISK.  ASSUMES DRIVE IS ALREADY
                 * SELECTED AND A RESTORE HAS BEEN PERFORMED BY THE
                 * ROM BOOT AND THAT STARTING TRACK AND SECTOR OF
                 * FLEX ARE AT $C105 AND $C106.  BEGIN EXECUTION
                 * BY JUMPING TO LOCATION $C100.  JUMPS TO FLEX
                 * STARTUP WHEN COMPLETE.
                 *

                 * EQUATES

            C07F STACK   EQU    $C07F
            C300 SCTBUF EQU     $C300     DATA SECTOR BUFFER


                 * START OF UTILITY

C100                     ORG    $C100

C100 20   0A     LOAD    BRA    LOAD0

C102 00 00 00            FCB    0,0,0
C105 00          TRK     FCB    0         FILE START TRACK
C106 00          SCT     FCB    0         FILE START SECTOR
C107 00          DNS     FCB    0         DENSITY FLAG
C108 C100        TADR    FDB    $C100     TRANSFER ADDRESS
C10A 0000        LADR    FDB    0         LOAD ADDRESS
C10C 10CE C07F   LOAD0   LDS    #STACK    SETUP STACK
C110 FC   C105           LDD    TRK       SETUP STARTING TRK & SCT
C113 FD   C300           STD    SCTBUF
C116 108E C400           LDY    #SCTBUF+256

                 * PERFORM ACTUAL FILE LOAD

C11A 8D   35     LOAD1   BSR    GETCH     GET A CHARACTER
C11C 81   02             CMPA   #$02      DATA RECORD HEADER?
C11E 27   10             BEQ    LOAD2     SKIP IF SO
C120 81   16             CMPA   #$16      XFR ADDRESS HEADER?
C122 26   F6             BNE    LOAD1     LOOP IF NEITHER
C124 8D   2B             BSR    GETCH     GET TRANSFER ADDRESS
C126 B7   C108           STA    TADR
C129 8D   26             BSR    GETCH
C12B B7   C109           STA    TADR+1
```

```
C12E 20   EA                BRA    LOAD1     CONTINUE LOAD
C130 8D   1F       LOAD2    BSR    GETCH     GET LOAD ADDRESS
C132 B7   C10A              STA    LADR
C135 8D   1A                BSR    GETCH
C137 B7   C10B              STA    LADR+1
C13A 8D   15                BSR    GETCH     GET BYTE COUNT
C13C 1F   894D              TAB              PUT IN B
C13F 27   D9                BEQ    LOAD1     LOOP IF COUNT=0
C141 BE   C10A              LDX    LADR      GET LOAD ADDRESS
C144 34   14       LOAD3    PSHS   B,X
C146 8D   09                BSR    GETCH     GET A DATA CHARACTER
C148 35   14                PULS   B,X
C14A A7   80                STA    0,X+      PUT CHARACTER
C14C 5A                     DECB             END OF DATA IN RECORD?
C14D 26   F5                BNE    LOAD3     LOOP IF NOT
C14F 20   C9                BRA    LOAD1     GET ANOTHER RECORD

                   * GET CHARACTER ROUTINE - READS A SECTOR IF NECESSARY

C151 108C C400     GETCH    CMPY   #SCTBUF+256 OUT OF DATA?
C155 26   0F                BNE    GETCH4    GO READ CHARACTER IF NOT
C157 8E   C300     GETCH2   LDX    #SCTBUF   POINT TO BUFFER
C15A EC   84                LDD    0,X       GET FORWARD LINK
C15C 27   0B                BEQ    GO        IF ZERO, FILE IS LOADED
C15E 8D   0D                BSR    READ      READ NEXT SECTOR
C160 26   9E                BNE    LOAD      START OVER IF ERROR
C162 108E C304              LDY    #SCTBUF+4 POINT PAST LINK
C166 A6   A0       GETCH4   LDA    0,Y+      ELSE, GET A CHARACTER
C168 39                     RTS

                   * FILE IS LOADED, JUMP TO IT

C169 6E   9F C108  GO       JMP    [TADR]    JUMP TO TRANSFER ADDRESS


                   * READ SINGLE SECTOR
                   *
                   * THIS ROUTINE MUST READ THE SECTOR WHOSE TRACK
                   * AND SECTOR ADDRESS ARE IN A ANB B ON ENTRY.
                   * THE DATA FROM THE SECTOR IS TO BE PLACED AT
                   * THE ADDRESS CONTAINED IN X ON ENTRY.
                   * IF ERRORS, A NOT-EQUAL CONDITION SHOULD BE
                   * RETURNED.  THIS ROUTINE WILL HAVE TO DO SEEKS.
                   * A,B,X, AND U MAY BE DESTROYED BY THIS ROUTINE,
                   * BUT Y MUST-BE PRESERVED.

C16D C6   FF       READ     LDB    #$FF      MUST BE USER SUPPLIED!
C16F 39                     RTS              THIS CODE DISABLES READ!


                            END
```

APPENDIX F
Skeletal NEWDISK Routine

```
          * NEWDISK
          *
          * COPYRIGHT (C) 1980 BY
          * TECHNICAL SYSTEMS CONSULTANTS, INC.
          * 111 PROVIDENCE RD, CHAPEL HILL, NC.27514
          *
          * DISK FORMATTING PROGRAM FOR 6809 FLEX.
          * GENERAL VERSION DESIGNED FOR WD 1771/1791.
          * THE NEWDISK PROGRAM INITIALIZES A NEW DISKETTE AND
          * THEN PROCEEDS TO VERIFY ALL SECTORS AND INITIALIZE
          * TABLES.  THIS VERSION IS SETUP FOR AN 8 INCH DISK
          * SYSTEM WITH HINTS AT CERTAIN POINTS FOR ALTERING
          * FOR A SINGLE-DENSITY 5 INCH DISK SYSTEM.  THIS
          * VERSION IS NOT INTENDED FOR 5 INCH DOUBLE-DENSITY.

          **************************************************
          * DISK SIZE PARAMETERS
          * **** **** **********
          * THE FOLLOWING CONSTANTS SETUP THE SIZE OF THE
          * DISK TO BE FORMATTED.  THE VALUES SHOWN ARE FOR
          * 8 INCH DISKS.  FOR 5 INCH DISKS, USE APPROPRIATE
          * VALUES. (IE. 35 TRACKS AND 10 SECTORS PER SIDE)
          **************************************************

    004D  MAXTRK  EQU    77        NUMBER OF TRACKS
          * SINGLE DENSITY:
    000F  SMAXS0  EQU    15        SD MAX SIDE 0 SECTORS
    001E  SMAX51  EQU    30        SD MAX SIDE 1 SECTORS
          * DOUBLE DENSITY:
    001A  DMAXS0  EQU    26        DD MAX SIDE 0 SECTORS
    0034  DMAXS1  EQU    52        DD MAX SIDE 1 SECTORS

          **************************************************
          * MORE DISK SIZE DEPENDENT PARAMETERS
          * **** **** **** ********* **********
          * THE FOLLOWING VALUES ARE ALSO DEPENDENT ON THE
          * SIZE OF DISK BEING FORMATTED.  EACH VALUE SHOWN
          * IS FOR 8 INCH WITH PROPER 5 INCH VALUES IN
          * PARENTHESES.
          **************************************************

          * SIZE OF SINGLE DENSITY WORK BUFFER FOR ONE TRACK
    13EC  TKSZ    EQU    5100      (USE 3050 FOR 5 INCH)
          * TRACK START VALUE
    0028  TST     EQU    40        (USE 0 FOR 5 INCH
)          * SECTOR START VALUE
    0049  SST     EQU    73        (USE 7 FOR 5 INCH)
          * SECTOR GAP VALUE
    001B  GAP     EQU    27        (USE 14 FOR 5 INCH)

          **************************************************
```

```
              * WORK SPACE WHERE ONE TRACK OF DATA IS SETUP

        0800  WORK    EQU    $0800     WORK SPACE
        1BEC  SWKEND  EQU    TKSZ+WORK SINGLE DENSITY
        2FD8  DWKEND  EQU    TKSZ*2+WORK DOUBLE DENSITY


              * GENERAL EQUATES

        0101  FIRST   EQU    $0101     FIRST USER SECTOR
        001E  FCS     EQU    30        FCB CURRENT SECTOR
        0040  FSB     EQU    64        SECTOR BUFFER
        0010  IRS     EQU    16        INFO RECORD START
        005D  AVLP    EQU    FSB+IRS+13
        0005  DIRSEC  EQU    5         FIRST DIR. SECTOR
        0009  RDSS    EQU    9         READ SS FMS CODE
        000A  WTSS    EQU    10        WRITE SS FMS CODE
        CCOE  DATE    EQU    $CCOE     DOS DATE

              * FLEX ROUTINES EQUATES

        CD1E  PSTRNG  EQU    $CD1E
        CD18  PUTCHR  EQU    $CD18
        CD39  OUTDEC  EQU    $CD39
        CD42  GETHEX  EQU    $CD42
        CD15  GETCHR  EQU    $CD15
        CD24  PCRLF   EQU    $CD24
        CD1B  INBUF   EQU    $CD1B
        CD2D  GETFIL  EQU    $CD2D
        CD48  INDEC   EQU    $CD48
        D406  FMS     EQU    $D406
        D403  FMSCLS  EQU    $D403
        CD3C  OUT2HS  EQU    $CD3C
        CD03  WARMS   EQU    $CD03

              * DISK DRIVER ROUTINES

        DE03  DWRITE  EQU    $DE03     WRITE A SINGLE SECTOR
        DE09  REST    EQU    $DE09     RESTORE HEAD
        DE1B  DSEEK   EQU    $DE1B     SEEK TO TRACK


              * TEMPORARY STORAGE

0020                  ORG    $0020

0020          TRACK   RMB    1
0021          SECTOR  RMB    1
0022          BADCNT  RMB    1        BAD SECTOR COUNT
0023          DRN     RMB    1
0024          SIDE    RMB    1
0025          DBSDF   RMB    1
0026          DENSE   RMB    1
```

```
0027                DNSITY  RMB    1
0028                INDEX   RMB    2
002A                SECCNT  RMB    2         SECTOR COUNTER
002C                FSTAVL  RMB    2         FIRST AVAILABLE
002E                LSTAVL  RMB    2         LAST AVAILABLE
0030                MAXS0   RMB    1         MAX SIDE 0 SECTOR
0031                MAXS1   RMB    1         MAX SIDE 1 SECTOR
0032                MAX     RMB    1         MAX SECTOR
0033                FKFCB   RMB    4
0037                VOLNAM  RMB    11
0042                VOLNUM  RMB    2
```

```
0100                       ORG    $0100

                   *******************************************
                   * MAIN PROGRAM STARTS HERE
                   *******************************************

0100 20   0C       NEWDISK BRA    FORM1

0102 02            VN      FCB    2          VERSION NUMBER

0103 BD   CD1E     OUTIN   JSR    PSTRNG     OUTPUT STRING
0106 BD   CD15     OUTIN2  JSR    GETCHR     GET RESPONSE
0109 84   5F               ANDA   #$5F       MAKE IT UPPER CASE
0108 81   59               CMPA   #'Y        SEE IF "YES"
010D 39                    RTS


010E 86   0F       FORM1   LDA    #SMAXS0    INITIALIZE SECTOR MAX
0110 97   30               STA    MAXS0
0112 97   32               STA    MAX
0114 86   1E               LDA    #SMAXS1
0116 97   31               STA    MAXS1
0118 BD   CD42             JSR    GETHEX     GET DRIVE NUMBER
011B 1025 0080             LBCS   EXIT
011F 1F   10               TFR    X,D
0121 1083 0003             CMPD   #3         ENSURE 0 TO 3
0125 22   78               BHI    EXIT
0127 8E   0800             LDX    #WORK
012A E7   03               STB    3,X
012C D7   23               STB    DRN
012E 8E   04A8             LDX    #SURES     ASK IF HE'S SURE
0131 8D   D0               BSR    OUTIN      PRINT & GET RESPONSE
0133 26   6A               BNE    EXIT       EXIT IF "NO"
0135 8E   04CA             LDX    #SCRDS     CHECK SCRATCH DRIVE NO.
0138 BD   CD1E             JSR    PSTRNG     OUTPUT IT
013B 8E   0802             LDX    #WORK+2
013E 6F   84               CLR    0,X
0140 5F                    CLRB
0141 BD   CD39             JSR    OUTDEC
0144 86   3F               LDA    #'?        PRINT QUESTION MARK
0146 BD   CD18             JSR    PUTCHR
0149 86   20               LDA    #$20
014B BD   CD18             JSR    PUTCHR
014E 8D   B6               BSR    OUTIN2     GET RESPONSE
0150 26   4D               BNE    EXIT       EXIT IF "NO"
0152 0F   25               CLR    DBSDF      CLEAR FLAG
                   *** PLACE A "BRA FORM25" HERE IF CONTROLLER
                   *** IS ONLY SINGLE SIDED.
0154 8E   0538             LDX    #DBST      ASK IF DOUBLE SIDED
0157 BD   AA               BSR    OUTIN      PRINT & GET RESPONSE
0159 26   06               BNE    FORM25     SKIP IF "NO"
015B 0C   25               INC    DBSDF      SET FLAG
015D 86   1E               LDA    #SMAXS1    SET MAX SECTOR
015F 97   32               STA    MAX
```

```
0161 0F   26        FORM25  CLR    DENSE     INITIALIZE SINGLE DENSITY
0163 0F   27                CLR    DNSITY
                    *** PLACE A "BRA FORM26" HERE IF CONTROLLER
                    *** IS ONLY SINGLE DENSITY.
0165 8E   054C              LDX    #DDSTR    ASK IF DOUBLE DENSITY
0168 8D   99                BSR    OUTIN     PRINT & GET RESPONSE
016A 26   02                BNE    FORM26    SKIP IF "NO"
016C 0C   26                INC    DENSE     SET FLAG IF SO
016E 8E   0562      FORM26  LDX    #NMSTR    ASK FOR VOLUME NAME
0171 BD   CD1E              JSR    PSTRNG    PRINT IT
0174 BD   CD1B              JSR    INBUF     GET LINE
0177 8E   0033              LDX    #FKFCB    POINT TO FAKE
017A BD   CD2D              JSR    GETFIL
017D 8E   0570      FORM27  LDX    #NUMSTR   OUTPUT STRING
0180 BD   CD1E              JSR    PSTRNG
0183 BD   CD1B              JSR    INBUF     GET LINE
0186 BD   CD48              JSR    INDEC     GET NUMBER
0189 25   F2                BCS    FORM27    ERROR?
018B 9F   42                STX    VOLNUM    SAVE NUMBER
018D BD   CD24              JSR    PCRLF     PRINT CR & LF
0190 8E   0800              LDX    #WORK
0193 BD   DE09              JSR    REST      RESTORE DISK
0196 27   15                BEQ    FORMAT    SKIP IF NO ERROR
0198 8E   04B7              LDX    #WPST
019B C5   40                BITB   #$40      WRITE PROTECT ERROR?
019D 26   03                BNE    EXIT2     SKIP IF SO

                    * EXIT ROUTINES

019F 8E   04F1      EXIT    LDX    #ABORTS   REPORT ABORTING
01A2 BD   CD1E      EXIT2   JSR    PSTRNG    OUTPUT IT
01A5 BD   D403      EXIT3   JSR    FMSCLS
01A8 1C   EF                CLI
01AA 7E   CD03              JMP    WARMS     RETURN
```

```
***********************************************************
*
* ACTUAL FORMAT ROUTINE
*
* THIS CODE PERFORMS THE ACTUAL DISK FORMATTING BY PUTTING
* ON ALL GAPS, HEADER INFORMATION, DATA AREAS, SECTOR LINKING,
* ETC.  THIS SECTION DOES NOT WORRY ABOUT SETTING UP THE
* SYSTEM INFORMATION RECORD, BOOT SECTOR, OR DIRECTORY.
* IT ALSO DOES NOT NEED BE CONCERNED WITH TESTING THE DISK FOR
* ERRORS AND THE REMOVAL OF DEFECTIVE SECTORS ASSOCIATED WITH
* SUCH TESTING.  THESE OPERATIONS ARE CARRIED OUT BY THE
* REMAINDER OF THE CODE IN "NEWDISK".
* IF USING A WD1771 OR WD1791 CONTROLLER CHIP, THIS CODE SHOULD
* NOT NEED CHANGING (SO LONG AS THE WRITE TRACK ROUTINE AS
* FOUND LATER IS PROVIDED).  IF USING A DIFFERENT TYPE OF
* CONTROLLER, THIS CODE MUST BE REPLACED AND THE WRITE TRACK
* ROUTINE (FOUND LATER) MAY BE REMOVED AS IT WILL HAVE TO BE
* A PART OF THE CODE THAT REPLACES THIS FORMATTING CODE.
* WHEN THIS ROUTINE IS COMPLETED, IT SHOULD JUMP TO 'SETUP'.
*
***********************************************************

                   * MAIN FORMATTING LOOP

01AD 1A   10       FORMAT  SEI
01AF 0F   20               CLR    TRACK
01B1 0F   24       FORM3   CLR    SIDE      SET SIDE 0
01B3 0F   21               CLR    SECTOR
01B5 8D   40               BSR    TRKHD     SETUP TRACK HEADER
01B7 8E   0849     FORM32  LDX    #WORK+SST POINT TO SECTOR START
01BA 0D   27               TST    DNSITY    DOUBLE DENSITY?
01BC 27   03               BEQ    FORM4     SKIP IF NOT
01BE 8E   0892             LDX    #SST*2+WORK DD SECTOR START
01C1 BD   6E       FORM4   BSR    DOSEC     PROCESS RAM WITH INFO
01C3 0C   21               INC    SECTOR    ADVANCE TO NEXT
01C5 96   21               LDA    SECTOR    CHECK VALUE
01C7 0D   24               TST    SIDE      CHECK SIDE
01C9 26   04               BNE    FORM45
01CB 91   30               CMPA   MAXS0
01CD 20   02               BRA    FORM46
01CF 91   31       FORM45  CMPA   MAXS1
01D1 26   EE       FORM46  BNE    FORM4     REPEAT?
01D3 96   20       FORM47  LDA    TRACK     GET TRACK NUMBER
01D5 D6   24               LDB    SIDE      FAKE SECTOR FOR PROPER SIDE
01D7 BD   DE1B             JSR    DSEEK     SEEK TRACK AND SIDE
01DA BD   0580             JSR    WRTTRK    WRITE TRACK
01DD 0D   25       FORM5   TST    DBSDF     ONE SIDE?
01DF 27   08               BEQ    FORM6
01E1 00   24               TST    SIDE
01E3 26   04               BNE    FORM6
01E5 03   24               COM    SIDE      SET SIDE 1
01E7 20   CE               BRA    FORM32
01E9 0C   20       FORM6   INC    TRACK     BUMP TRACK
01EB BD   02FF             JSR    SWITCH    SWITCH TO DD IF NCSSRY
```

```
01EE 96   20        FORM7   LDA    TRACK      CHECK VALUE
01F0 81   4D                CMPA   #MAXTRK
01F2 26   BD                BNE    FORM3
01F4 16   00BE              LBRA   SETUP      DONE...GO FINISH UP

                    * SETUP TRACK HEADER INFORMATION

01F7 BE   0800      TRKHD   LDX    #WORK      POINT TO BUFFER
01FA 0D   27                TST    DNSITY     DOUBLE DENSITY?
01FC 26   11                BNE    TRHDD      SKIP IF SO
01FE C6   FF                LDB    #$FF
0200 E7   80        TRHDS1  STB    0,X+       INITIALIZE TO $FF
0202 8C   1BEC              CMPX   #SWKEND
0205 26   F9                BNE    TRHDS1
0207 8E   0828              LDX    #WORK+TST
020A 4F                     CLRA              SET IN ZEROS
020B C6   06                LDB    #6
020D 20   15                BRA    TRHDD2
020F C6   4E        TRHDD   LDB    #$4E
0211 E7   80        TRHDD1  STB    0,X+       INITIALIZE TO $4E
0213 8C   2FD8              CMPX   #DWKEND
0216 26   F9                BNE    TRHDDI
0218 8E   0850              LDX    #TST*2+WORK
021B 4F                     CLRA              SET IN ZEROS
021C C6   0C                LDB    #12
021E BD   0B                BSR    SET
0220 86   F6                LDA    #$F6       SET IN $F6'S
0222 C6   03                LDB    #3
0224 8D   05        TRHDD2  BSR    SET
0226 86   FC                LDA    #$FC       SET INDEX MARK
0228 A7   84                STA    0,X
022A 39                     RTS

                    * SET (B) BYTES OF MEMORY TO (A) STARTING AT (X)
022B A7   80        SET     STA    0,X+
022D 5A                     DECB
022E 26   FB                BNE    SET
0230 39                     RTS

                    * PROCESS SECTOR IN RAM

0231 4F             DOSEC   CLRA
0232 0D   27                TST    DNSITY     DOUBLE DENSITY?
0234 26   04                BNE    DOSEC1     SKIP IF SO
0236 C6   06                LDB    #6         CLEAR 6 BYTES
0238 20   08                BRA    DOSEC2
023A C6   0C        DOSEC1  LDB    #12        CLEAR 12 BYTES
023C 8D   ED                BSR    SET
023E 86   F5                LDA    #$F5       SET IN 3 $F5'S
0240 C6   03                LDB    #3
0242 8D   E7        DOSEC2  BSR    SET
0244 86   FE                LDA    #$FE       ID ADDRESS MARK
0246 A7   80                STA    0,X+
0248 96   20                LDA    TRACK      GET TRACK NO.
```

```
024A A7   80               STA   0,X+
024C D6   27               LDB   DNSITY    DOUBLE DENSITY?
024E 27   04               BEQ   DOSEC3    SKIP IF NOT
0250 D6   24               LDB   SIDE      GET SIDE INDICATOR
0252 C4   01               ANDB  #$01      MAKE IT 0 OR 1
0254 E7   80       DOSEC3  STB   0,X+
0256 D6   21               LDB   SECTOR    GET SECTOR NO.
0258 108E 0456             LDY   #SSCMAP   POINT TO CORRECT MAP
025C 0D   27               TST   DNSITY
025E 27   04               BEQ   DOSEC4
0260 108E 0474             LDY   #DSCMAP
0264 E6   A5       DOSEC4  LDB   B,Y       GET ACTUAL SECTOR
0266 E7   80               STB   0,X+
0268 D1   32               CMPB  MAX       END OF TRACK?
026A 26   09       DOSEC6  BNE   DOSEC7    SKIP IF NOT
026C 4C                    INCA            BUMP TRACK NO.
026D 5F                    CLRB            RESET SECTOR NO.
026E 81   4D               CMPA  #MAXTRK   END OF DISK?
0270 26   03               BNE   DOSEC7    SKIP IF NOT
0272 4F                    CLRA            SET ZERO FORWARK LINK
0273 C6   FF               LDB   #-1
0275 5C            DOSEC7  INCB            BUMP SECTOR NO.
0276 34   06               PSHS  D         SAVE FORWARD LINK
0278 86   01               LDA   #1        SECTOR LENGTH = 256
027A A7   80               STA   0,X+
027C 86   F7               LDA   #$F7      SET CRC CODE
027E A7   80               STA   0,X+
0280 0D   27               TST   DNSITY    DOUBLE DENSITY?
0282 26   07               BNE   DOSEC8    SKIP IF SO
0284 30   0B               LEAX  11,X      LEAVE $FF'S
0286 4F                    CLRA            PUT IN 6 ZEROS
0287 C6   06               LDB   #6
0289 20   0C               BRA   DOSEC9
028B 30   88 16    DOSEC8  LEAX  22,X      LEAVE $4E'S
028E 4F                    CLRA            PUT IN 12 ZEROS
028F C6   0C               LDB   #12
0291 8D   98               BSR   SET
0293 86   F5               LDA   #$F5      PUT IN 3 $F5'S
0295 C6   03               LDB   #3
0297 8D   92       DOSEC9  BSR   SET
0299 86   FB               LDA   #$FB      DATA ADDRESS MARK
029B A7   80               STA   0,X+
029D 35   06               PULS  D         RESTORE FORWARD LINK
029F ED   81               STD   0,X++     PUT IN SECTOR BUFFER
02A1 4F                    CLRA            CLEAR SECTOR BUFFER
02A2 C6   FE               LDB   #254
02A4 8D   85               BSR   SET
02A6 86   F7               LDA   #$F7      SET CRC CODE
02A8 A7   80               STA   0,X+
02AA 30   88 1B            LEAX  GAP,X     LEAVE GAP
02AD 0D   27               TST   DNSITY    DOUBLE DENSITY?
02AF 27   03               BEQ   DOSECA    SKIP IF NOT
02B1 30   88 1B            LEAX  GAP,X     DD NEEDS MORE GAP
02B4 39            DOSECA  RTS
```

```
**************************************************************
*
* DISK TESTING AND TABLE SETUP
*
* THE FOLLOWING CODE TESTS EVERY SECTOR AND REMOVES ANY
* DEFECTIVE SECTORS FROM THE FREE CHAIN.  NEXT THE SYSTEM
* INFORMATION RECORD IS SETUP, THE DIRECTORY IS INITIALIZED,
* AND THE BOOT IS SAVED ON TRACK ZERO.  ALL THIS CODE SHOULD
* WORK AS IS FOR ANY FLOPPY DISK SYSTEM.  ONE CHANGE THAT
* MIGHT BE REQUIRED WOULD BE IN THE SAVING OF THE BOOTSTRAP
* LOADER.  SPECIAL BOOT LOADERS MIGHT REQUIRE CHANGES IN THE
* WAY THE BOOT SAVE IS PERFORMED. FOR EXAMPLE, IT MAY BE
* NECESSARY TO SAVE TWO SECTORS IF THE BOOT LOADER DOES NOT
* FIT IN ONE.  ALSO IT MAY BE NECESSARY, BY SOME MEANS, TO
* INFORM THE BOOT LOADER WHETHER THE DISK IS SINGLE OR
* DOUBLE DENSITY SO THAT IT MAY SELECT THE PROPER DENSITY
* FOR LOADING FLEX.
*
* **************************************************************


                   * READ ALL SECTORS FOR ERRORS

02B5 D6   32       SETUP   LDB   MAX       GET MAX SECTORS
02B7 86   4C               LDA   #MAXTRK-1
02B9 DD   2E               STD   LSTAVL    SET LAST AVAIL.
02BB 3D                    MUL             FIND TOTAL SECTORS
02BC DD   2A               STD   SECCNT    SAVE IT
02BE 8E   0101             LDX   #FIRST    SET FIRST AVAIL
02C1 9F   2C               STX   FSTAVL
02C3 96   23               LDA   DRN
02C5 B7   0803             STA   WORK+3
02C8 4F                    CLRA            CLEAR COUNTER
02C9 97   22               STA   BADCNT
02CB 97   20               STA   TRACK     SET TRACK
02CD 97   27               STA   DNSITY    SNGL DNST FOR TRK 0
02CF 4C                    INCA
02D0 97   21               STA   SECTOR    SET SECTOR
02D2 86   0F               LDA   #SMAXS0   RESET MAXIMUM
02D4 97   30               STA   MAXS0     SECTOR COUNTS
02D6 86   1E               LDA   #SMAXS1
02D8 97   31               STA   MAXS1
02DA 0D   25               TST   DBSDF     DOUBLE SIDED?
02DC 26   02               BNE   SETUP1    SKIP IF SO
02DE 86   0F               LDA   #SMAXS0
02E0 97   32       SETUP1  STA   MAX       SET MAXIMUM SECTORS
02E2 8D   10       SETUP2  BSR   CHKSEC    GO CHECK SECTOR
02E4 26   3C               BNE   REMSEC    ERROR?
02E6 0F   22               CLR   BADCNT    CLEAR COUNTER
02E8 DC   20       SETUP4  LDD   TRACK     GET TRACK & SECTOR
02EA 8D   2A               BSR   FIXSEC    GET TO NEXT ADR
02EC 1027 00B3             LBEQ  DOTRK     SKIP IF FINISHED
02F0 DD   20               STD   TRACK     SET TRACK & SECTOR
02F2 20   EE               BRA   SETUP2    REPEAT
```

```
                  * CHECK IF SECTOR GOOD

02F4 8E   0800    CHKSEC  LDX    #WORK     POINT TO FCB
02F7 DC   20              LDD    TRACK     GET TRACK & SECTOR
02F9 ED   88 1E           STD    FCS,X     SET CURRENT TRK & SCT
02FC 7E   038C            JMP    READSS    GO DO READ

                  * SWITCH TO DOUBLE DENSITY IF NECESSARY

02FF D6   26      SWITCH  LDB    DENSE     DOUBLE DENSITY DISK?
0301 27   12              BEQ    SWTCH2    SKIP IF NOT
0303 D7   27              STB    DNSITY    SET FLAG
0305 C6   1A              LDB    KDMAXS0   RESET SECTOR COUNTS
0307 D7   30              STB    MAXS0
0309 C6   34              LDB    #DMAXS1
030B D7   31              STB    MAXS1
030D 0D   25              TST    DBSDF     DOUBLE SIDED?
030F 26   02              BNE    SWTCH1    SKIP IF SO
0311 C6   1A              LDB    #DMAXS0
0313 D7   32      SWTCH1  STB    MAX       SET MAX SECTOR
0315 39           SWTCH2  RTS

                  * SET TRK & SEC TO NEXT

0316 D1   32      FIXSEC  CMPB   MAX       END OF TRACK?
0318 26   04              BNE    FIXSE4    SKIP IF NOT
031A 4C                   INCA             BUMP TRACK
031B 8D   E2              BSR    SWITCH    SWITCH TO DD IF NCSSRY
031D 5F                   CLRB             RESET SECTOR NO.
031E 5C           FIXSE4  INCB             BUMP SECTOR NO.
031F 81   4D              CMPA   #MAXTRK   END OF DISK?
0321 39                   RTS

                  * REMOVE BAD SECTOR FROM FREE SECTOR CHAIN

0322 0C   22      REMSEC  INC    BADCNT    UPDATE COUNTER
0324 27   0A              BEQ    REMSE1    COUNT OVERFLOW?
0326 96   20              LDA    TRACK     GET TRACK
0328 26   0C              BNE    REMSE2    TRACK 0?
032A D6   21              LDB    SECTOR    GET SECTOR
032C C1   05              CMPB   #DIRSEC   PAST DIRECTORY?
032E 22   06              BHI    REMSE2
0330 8E   04E1    REMSE1  LDX    #FATERS   REPORT FATAL ERROR
0333 7E   01A2            JMP    EXIT2     REPORT IT
0336 8E   0800    REMSE2  LDX    #WORK     POINT TO FCB
0339 DC   2C              LDD    FSTAVL    GET 1ST TRACK & SECTOR
033B 1093 20              CMPD   TRACK     CHECK TRACK & SECTOR
033E 26   06              BNE    REMSE3
0340 8D   D4              BSR    FIXSEC    SET TO NEXT
0342 DD   2C              STD    FSTAVL    SET NEW ADR
0344 20   27              BRA    REMSE8    GO DO NEXT
0346 DC   20      REMSE3  LDD    TRACK     GET TRACK & SECTOR
0348 D0   22              SUBB   BADCNT
```

```
034A 27   02                BEQ    REMS35    UNDERFLOW?
034C 2A   03                BPL    REMSE4
034E 4A           REMS35    DECA             DEC TRACK
034F D6   32                LDB    MAX       RESET SECTOR
0351 ED   88 1E   REMSE4    STD    FCS,X     SET CURRENT ADR
0354 8D   36                BSR    READSS    GO DO READ
0356 26   D8                BNE    REMSE1    ERROR?
0358 EC   88 40             LDD    FSB,X     GET LINK ADR
035B 8D   B9                BSR    FIXSEC    POINT TO NEXT
035D 26   07                BNE    REMSE6    OVERFLOW?
035F EC   88 1E             LDD    FCS,X     GET CURRENT ADR
0362 DD   2E                STD    LSTAVL    SET NEW LAST AVAIL
0364 4F                     CLRA             SET END LINK
0365 5F                     CLRB
0366 ED   88 40   REMSE6    STD    FSB,X     SET NEW LINK
0369 8D   2B                BSR    WRITSS    GO DO WRITE
036B 26   C3                BNE    REMSE1    ERROR?
036D 9E   2A      REMSE8    LDX    SECCNT    GET SEC COUNT
036F 30   1F                LEAX   -1,X      DEC IT ONCE
0371 9F   2A                STX    SECCNT    SAVE NEW COUNT
0373 8E   0504              LDX    #BADSS    REPORT BAD SECTOR
0376 BD   CD1E              JSR    PSTRNG    OUTPUT IT
0379 BE   0020              LDX    #TRACK    POINT TO ADDRESS
037C BD   CD3C              JSR    OUT2HS    OUTPUT IT
037F 86   20                LDA    #$20
0381 BD   CD18              JSR    PUTCHR
0384 30   01                LEAX   1,X       BUMP TO NEXT
0386 BD   CD3C              JSR    OUT2HS
0389 7E   02E8              JMP    SETUP4    CONTINUE

                  * READ A SECTOR

038C 8E   0800    READSS    LDX    #WORK     POINT TO FCB
038F 86   09                LDA    #RDSS     SET UP COMMAND
0391 A7   84                STA    0,X
0393 7E   D406              JMP    FMS       GO DO IT

                  * WRITE A SECTOR

0396 BE   0800    WRITSS    LDX    #WORK     POINT TO FCB
0399 86   0A                LDA    #WTSS     SETUP COMMAND
039B A7   84                STA    0,X
039D BD   D406              JSR    FMS       GO DO IT
03A0 27   EA                BEQ    REABSS    ERRORS?
03A2 39                     RTS              ERROR RETURN

                  * SETUP SYSTEM INFORMATION RECORD

03A3 0F   27      DOTRK     CLR    DNSITY    BACK TO SINGLE DENSITY
03A5 8E   0800              LDX    #WORK     POINT TO SPACE
03A8 6F   88 1E             CLR    FCS,X     SET TO DIS
03AB 86   03                LDA    #3        SECTOR 3
03AD A7   88 1F             STA    FCS+1,X
03B0 80   DA                BSR    READSS    READ IN SIR SECTOR
```

```
03B2 26   48                BNE    DOTRK4    ERROR?
03B4 8E   0800              LDX    #WORK     FIX POINTER
03B7 6F   88 40             CLR    FSB,X     CLEAR FORWARD LINK
03BA 6F   88 41             CLR    FSB+1,X
03BD DC   2C                LDD    FSTAVL    ADDR. OF 1ST FREE SCTR.
03BF ED   88 5D             STD    AVLP,X    SET IN SIR
03C2 DC   2E                LDD    LSTAVL    ADDR. OF LAST FREE SCTR.
03C4 ED   88 5F             STD    AVLP+2,X  PUT IN SIR
03C7 DC   2A                LDD    SECCNT    GET TOTAL SECTOR COUNT
03C9 ED   88 61             STD    AVLP+4,X  PUT IN SIR
03CC 86   4C                LDA    #MAXTRK-1 GET MAX TRACK NO.
03CE D6   30                LDB    MAXS0     SET MAX SECTORS/TRACK
03D0 0D   25                TST    DBSDF     DOUBLE SIDE?
03D2 27   02                BEQ    DOTRK2
03D4 D6   31                LDB    MAXS1.    CHANGE FOR DOUBLE SIDED
03D6 ED   88 66     DOTRK2  STD    AVLP+9,X  SAVE MAX TRACK & SECTOR
03D9 FC   CC0E              LDD    DATE      SET DATE INTO SIR
03DC ED   88 63             STD    AVLP+6,X
03DF B6   CC10              LDA    DATE+2
03E2 A7   88 65             STA    AVLP+8,X
03E5 C6   0D                LDB    #13
03E7 108E 0037              LDY    #VOLNAM   POINT TO VOLUME NAME
03EB 8E   0800              LDX    #WORK
03EE A6   A0        DOTR33  LDA    0,Y+      COPY NAME TO SIR
03F0 A7   88 50             STA    FSB+IRS,X
03F3 30   01                LEAX   1,X
03F5 5A                     DECB             DEC THE COUNT
03F6 26   F6                BNE    DOTR33
03F8 8D   9C                BSR    WRITSS    WRITE SIR BACK OUT
03FA 27   03                BEQ    DIRINT    SKIP IF NO ERROR
03FC 7E   0330      DOTRK4  JMP    REMSE1    GO REPORT ERROR

                    * INITIALIZE DIRECTORY

03FF 8E   0800      DIRINT  LDX    #WORK     SET POINTER
0402 86   0F                LDA    #SMAXS0   GET MAX FOR TRK 0
0404 0D   25                TST    DBSDF     SINGLE SIDE?
0406 27   02                BEQ    DIRIN1    SKIP IF SO
0408 86   1E                LDA    #SMAXS1   SET MAX FOR DS
040A A7   88 1F     DIRIN1  STA    FCS+1,X   SET UP
040D BD   038C              JSR    READSS    READ IN SECTOR
0410 26   EA                BNE    DOTRK4    ERROR?
0412 8E   0800              LDX    #WORK     RESTORE POINTER
0415 6F   88 40             CLR    FSB,X     CLEAR LINK
0418 6F   88 41             CLR    FSB+1,X
041B BD   0396              JSR    WRITSS    WRITE BACK OUT
041E 26   DC                BNE    DOTRK4    ERRORS?
```

```
                    * SAVE BOOT ON TRACK 0 SECTOR 1
                    * (MAY REQUIRE CHANGES - SEE TEXT ABOVE)

0420 8E   C100      DOBOOT  LDX    #BOOT      POINT TO LOADER CODE
0423 4F                     CLRA              TRACK #0
0424 C6   01                LDB    #1         SECTOR #1
0426 BD   DE03              JSR    DWRITE     WRITE THE SECTOR
0429 26   D1                BNE    DOTRK4     SKIP IF AN ERROR

                    * REPORT TOTAL SECTORS AND EXIT

042B 8E   0800              LDX    #WORK      SETUP AN FCB
042E 86   10                LDA    #16        OPEN SIR FUNCTION
0430 A7   84                STA    0,X
0432 BD   D406              JSR    FMS        OPEN THE SIR
0435 26   C5                BNE    DOTRK4
0437 86   07                LDA    #7         GET INFO RECORD FUNCTION
0439 A7   84                STA    0,X
043B BD   D406              JSR    FMS        GET 1ST INFO RECORD
043E 26   BC                BNE    DOTRK4
0440 8E   0513              LDX    #CMPLTE    REPORT FORMATTING COMPLETE
0443 BD   CD1E              JSR    PSTRNG
0446 8E   0527              LDX    #SECST     PRINT TOTAL SECTORS STRING
0449 BD   CD1E              JSR    PSTRNG
044C 8E   0815              LDX    #WORK+21   TOTAL IS IN INFO RECORD
044F 5F                     CLRB
0450 BD   CD39              JSR    OUTDEC     PRINT NUMBER
0453 7E   01A5              JMP    EXIT3      ALL FINISHED!
```

```
                  ************************************************
                  * SECTOR MAPS
                  * ****** ****
                  * THE MAPS SHOWN BELOW CONTAIN THE CORRECT
                  * INTERLEAVING FOR AN 8 INCH DISK.  IF USING 5
                  * INCH DISKS (SINGLE DENSITY) YOU SHOULD USE
                  * SOMETHING LIKE '1,3,5,7,9,2,4,6,8,10' FOR
                  * SSCMAP FOR A SINGLE SIDED DISK.

                  ************************************************
0456 01 06 0B 03  SSCMAP  FCB    1,6,11,3,8,13,5,10
045E 0F 02 07 0C          FCB    15,2,7,12,4,9,14
0465 10 15 1A 12          FCB    16,21,26,18,23,28,20,25
046D 1E 11 16 1B          FCB    30,17,22,27,19,24,29

0474 01 0E 03 10  DSCMAP  FCB    1,14,3,16,5,18,7
047B 14 09 16 0B          FCB    20,9,22,11,24,13
0481 1A 02 0F 04          FCB    26,2,15,4,17,6,19
0488 08 15 0A 17          FCB    8,21,10,23,12,25
048E 1B 28 1D 2A          FCB    27,40,29,42,31,44,33
0495 2E 23 30 25          FCB    46,35,48,37,50,39
049B 34 1C 29 1E          FCB    52,28,41,30,43,32,45
04A2 22 2F 24 31          FCB    34,47,36,49,38,51


                  * STRINGS

04A8 41 52 45 20  SURES   FCC    'ARE YOU SURE? '
04B6 04                   FCB    4
04B7 44 49 53 4B  WPST    FCC    'DISK IS PROTECTED!'
04C9 04                   FCB    4
04CA 53 43 52 41  SCRDS   FCC    'SCRATCH DISK IN DRIVE '
04E0 04                   FCB    4
04E1 46 41 54 41  FATERS  FCC    'FATAL ERROR --- '
04F1 46 4F 52 4D  ABORTS  FCC    'FORMATTING ABORTED'
0503 04                   FCB    4
0504 42 41 44 20  BADSS   FCC    'BAD SECTOR AT '
0512 04                   FCB    4
0513 46 4F 52 4D  CMPLTE  FCC    'FORMATTING COMPLETE'
0526 04                   FCB    4
0527 54 4F 54 41  SECST   FCC    'TOTAL SECTORS '
0537 04                   FCB    4
0538 44 4F 55 42  DBST    FCC    'DOUBLE SIDED DISK? '
054B 04                   FCB    4
054C 44 4F 55 42  DDSTR   FCC    'DOUBLE DENSITY DISK? '
0561 04                   FCB    4
0562 56 4F 4C 55  NMSTR   FCC    'VOLUME NAME? '
056F 04                   FCB    4
0570 56 4F 4C 55  NUMSTR  FCC    'VOLUME NUMBER? '
057F 04                   FCB    4
```

```
***************************************************************
* WRITE TRACK ROUTINE                                         *
***************************************************************
* THIS SUBROUTINE MUST BE USER SUPPLIED.                      *
* IT SIMPLY WRITES THE DATA FOUND AT "WORK" ($0800) TO THE    *
* CURRENT TRACK ON THE DISK.  NOTE THAT THE SEEK TO TRACK     *
* OPERATION HAS ALREADY BEEN PERFORMED.  IF SINGLE DENSITY,   *
* "TKSZ" BYTES SHOULD BE WRITTEN.  IF DOUBLE, "TKSZ*2"        *
* BYTES SHOULD BE WRITTEN.  THIS ROUTINE SHOULD PERFORM       *
* ANY NECESSARY DENSITY SELECTION BEFORE WRITING.  DOUBLE     *
* DENSITY IS INDICATED BY THE BYTE "DNSITY" BEING NON-ZERO.   *
* THERE ARE NO ENTRY PARAMETERS AND ALL REGISTERS MAY BE      *
* DESTROYED ON EXIT.  THE CODE FOR THIS ROUTINE MUST NOT      *
* EXTEND PAST $0800 SINCE THE TRACK DATA IS STORED THERE.     *
***************************************************************

                   *********************************************
                   * WESTERN DIGITAL PARAMTERS
                   * ******* ******* *********
                   * REGISTERS:
             0000  COMREG  EQU    $0000     COMMAND REGISTER
             0000  TRKREG  EQU    $0000     TRACK REGISTER
             0000  SECREG  EQU    $0000     SECTOR REGISTER
             0000  DATREG  EQU    $0000     DATA REGISTER
                   * COMMANDS:
             00F4  WTCMD   EQU    $F4       WRITE TRACK COMMAND
                   *********************************************

                   *********************************************
                   * CONTROLLER DEPENDENT PARAMETERS
                   * ********** ********* **********
             0000  DRVREG  EQU    $0000     DRIVE SELECT REGISTER
                   *********************************************


0580 12            WRTTRK  NOP              ROUTINE GOES HERE
0581 39                    RTS
```

```
**********************************************************
*
* BOOTSTRAP FLEX LOADER
*
* THE CODE FOR THE BOOTSTRAP FLEX LOADER MUST BE IN MEMORY
* AT $C100 WHEN NEWDISK IS RUN.  THERE ARE TWO WAYS IT CAN
* BE PLACED THERE.  ONE IS TO ASSEMBLE THE LOADER AS A
* SEPARATE FILE AND APPEND IT ONTO THE END OF THE NEWDISK
* FILE.  THE SECOND IS TO SIMPLY PUT THE SOURCE FOR THE
* LOADER IN-LINE HERE WITH AN ORG TO $C100.  THE FIRST FEW
* LINES OF CODE FOR THE LATTER METHOD ARE GIVEN HERE TO
* GIVE THE USER AN IDEA OF HOW TO SETUP THE LOADER SOURCE.
*
* IT IS NOT NECESSARY TO HAVE THE LOADER AT $C100 IN ORDER
* FOR THE NEWDISK TO RUN.  IT SIMPLY MEANS THAT WHATEVER
* HAPPENS TO BE IN MEMORY AT $C100 WHEN NEWDISK IS RUN
* WOULD BE WRITTEN OUT AS A BOOT.  AS LONG AS THE CREATED
* DISK WAS FOR USE AS A DATA DISK ONLY AND WOULD NOT BE
* BOOTED FROM, THERE WOULD BE NO PROBLEM.
*
**********************************************************

                   * 6809 BOOTSTRAP FLEX LOADER

C100                       ORG    $C100

C100 20   07       BOOT    BRA    BOOT1

C102 00 00 00              FCB    0,0,0
C105 00            TRK     FCB    0         STARTING TRACK.(AT $C105)
C106 00            SCTR    FCB    0         STARTING SECTUR (AT $C106)
C107 0000          TEMP    FDB    0

             C300  FCB     EQU    $C300

C109 7E   C109     BOOT1   JMP    BOOT1     ROUTINE GOES HERE

                   **********************************************************

                           END    NEWDISK
```

APPENDIX G
Sample Adaptation for SWTPc MF-68

In this appendix we shall give source listings of the code for a sample adaptation of FLEX. This sample is the adaptation of FLEX to a Southwest Technical Products (SWTPc) 6809 computer system using their SBUG monitor and MF-68 minifloppy disk system. SBUG is a simple ROM monitor which assumes a console or terminal is connected to the system via an ACIA located at $E004. SBUG also redirects all interrupts through its own RAM vectors,in the area of $DFC0.

The MF-68 disk system to Which these adaptions apply is a single-sided, single-density, dual drive minifloppy system. The controller board (SWTPc's part number DC-1) employs a Western Digital 1771 floppy disk controller chip as its main logic. Besides the four standard registers for the Western Digital chip, there is one 8-bit, write-only register on the controller called the drive select register. The 2 low-order bits of this register select the drive as follows:

bit 1	bit 0	Selected Drive
0	0	#0
0	1	#1
1	0	#2
1	1	#3

All other bits in the drive select register are ignored.

The Procedure

The source listings of all the code necessary to adapt FLEX to the described system follows. These listings include:

1) The Console I/O Driver Package
2) The Disk Driver Package
3) A ROM Boot Program
4) A FLEX Loader Program
5) A NEWDISK Program

A few comments about each program or package are in order.

1) The Console I/O Driver Package

The most important part of the Console Driver package is the set of routines which perform the character I/O to the system terminal or console. As can be seen, these are written for an ACIA at location $E004. The interrupt vectors (IRQVEC and SWIVEC) are simply those setup by SBUG. The interrupt timer routines for printer spooling assume a SWTPc MP-T timer board installed in I/O slot #4 (PIA at $E012).

2) The Disk Driver Package

This package contains all the routines for driving the disks. It should be noted that these routines will probably not work for an 8 inch disk system running at 1 MHz. The data transfer rate required by the 8 inch disk system is faster than the READ and WRITE routines can handle. The only solution is to increase the clock speed or use a DMA or buffered controller. The INIT routine clears all the temporary storage values. such that the system starts at track 0 on all drives. There is no need for a WARM start routine in this system, so WARM.points directly to a return. With this minifloppy system there is no way for the cpu to determine whether or not the drives are in a "ready" state. As a result, we must assume the drives are always ready. Since the response will be the same for CHKRDY and QUICK (there is no need for a CHKDRDY delay), the jump vectors for the two point to the same routine. This routine always returns a ready condition if the specified drive number is 0 or 1. Any other drive number receives a not-ready condition. This technique has two side effects. First, since drives 0 and 1 are always assumed ready, if either is not ready (no disk inserted or door not closed), the system will "hang" until the drive is put into a ready state or the cpu reset. Second, if there are more than two drives on line, only the first two will be searched by commands which should search all drives. If a user wishes, he can certainly make the check for a valid drive number in CHKRDY include drives 2 and 3.

3) A ROM Boot Program

Nothing fancy about this one. The emphasis here was to keep things short and simple. For the lack of a better place, this sample was orged at $7000. The user will probably wish to reassemble the code into ROM at some high address. If the user has more room in his ROM it might be desirable to perform more complete error checking and recovery.

4) A FLEX Loader Program

This program is an exact copy of the skeletal FLEX Loader given in Appendix E with the exception of the added routine to read a single sector. It may be noted that the,"read single sector" routine used is almost identical to that prepared for the Disk Driver package. If the user has enough room left over (the program should not be over 256 bytes) it might be desireable to add a check to see if the disk has actually been linked. This check would examine the two bytes at $C105 and $C106 to be sure that were changed to some non-zero value (which would imply a LINK command had been performed). If the two bytes were still zero, an appropriate message should be printed and the loading operation aborted.

5) A NEWDISK Program

For this system we need only a single-sided, single-density NEWDISK routine. It is easiest, however, to use the full NEWDISK routine as supplied and default to single-sided, single-density,,by inserting the two branch instructions as pointed out in the listing ("BRA FORM25" and "BRA FORM26"). All the values given in the skeletal NEWDISK for minifloppys have been used for this version. For this example we have used 35 as the number of tracks on the disk, but it could certainly be changed to 40 if the drives were capable of writing 40 tracks. The sector maps have been altered to reflect the number of sectors and proper interleaving for a single-sided, single-density minifloppy. The only code really added to the skeletal NEWDISK is the Write Track routine and the Bootstrap Loader routine. You will note that the Bootstrap Loader is exactly the same as what we have already listed. Only the added code or changed code has been printed in this NEWDISK sample. The remainder of the routine is identical to that of the skeletal NEWDISK listed in Appendix F.

```
                  * CONSOLE I/O DRIVER PACKAGE
                  *
                  * COPYRIGHT (C) 1980 BY
                  * TECHNICAL SYSTEMS CONSULTANTS, INC.
                  * 111 PROVIDENCE RD, CHAPEL HILL, NC 27514
                  *
                  * CONTAINS ALL TERMINAL I/O DRIVERS AND INTERRUPT HANDLING
                  * INFORMATION.  THIS VERSION IS FOR A SWTPC SYSTEM USING
                  * A SBUG MONITOR AND THE MF-68 MINIFLOPPY SYSTEM. THE
                  * INTERRUPT TIMER ROUTINES ARE FOR A SWTPC MP-T TIMER
                  * CARD ADDRESSED AT $E012.

                  *  SYSTEM EQUATES

            C700  CHPR    EQU    $C700     CHANGE PROCESS ROUTINE
            E012  TMPIA   EQU    $E012     TIMER PIA ADDRESS
            E004  ACIA    EQU    $E004     ACIA ADDRESS


                  ***************************************************
                  *                                                 *
                  * I/O ROUTINE VECTOR TABLE                        *
                  *                                                 *
D3E5                      ORG    $D3E5     TABLE STARTS AT $D3E5    *
                  *                                                 *
D3E5 D37B         INCHNE  FDB    INNECH    INPUT CHAR - NO  ECHO    *
D3E7 D3C3         IHNDLR  FDB    IHND      IRQ INTERRUPT HANDLER    *
D3E9 DFC2         SWIVEC  FDB    $DFC2     SWI3 VECTOR LOCATION     *
D3EB DFC8         IRQVEC  FDB    $DFC8     IRQ VECTOR LOCATION      *
D3ED D3BD         TMOFF   FDB    TOFF      TIMER OFF ROUTINE        *
D3EF D3B9         TMON    FDB    TON       TIMER ON ROUTINE         *
D3F1 D3A3         TMINT   FDB    TINT      TIMER INITIALIZE ROUTINE *
D3F3 F814         MONITR  FDB    $F814     MONITOR RETURN ADDRESS   *
D3F5 D370         TINIT   FDB    INIT      TERMINAL INITIALIZATION  *
D3F7 D399         STAT    FDB    STATUS    CHECK TERMINAL STATUS    *
D3F9 D38A         OUTCH   FDB    OUTPUT    TERMINAL CHAR OUTPUT     *
D3FB D388         INCH    FDB    INPUT     TERMINAL CHAR INPUT      *
                  *                                                 *
                  ***************************************************


                  * ACTUAL ROUTINES START HERE
                  ******************************

D370                      ORG    $D370

                  * TERMINAL INITIALIZE ROUTINE

D370 86   13      INIT    LDA    #$13      RESET ACIA
D372 B7   E004            STA    ACIA
D375 86   11              LDA    #$11      CONFIGURE ACIA
D377 B7   E004            STA    ACIA
D37A 39                   RTS
```

```
                    * TERMINAL INPUT CHAR. ROUTINE - NO ECHO

D37B B6   E004      INNECH  LDA    ACIA       GET ACIA STATUS
D37E 84   01                ANDA   #$01       A CHARACTER PRESENT?
D380 27   F9                BEQ    INNECH     LOOP IF NOT
D382 B6   E005              LDA    ACIA+1     GET THE CHARACTER
D385 84   7F                ANDA   #$7F       STRIP PARITY
D387 39                     RTS

                    * TERMINAL INPUT CHAR. ROUTINE - W/ ECHO

D388 8D   F1        INPUT   BSR    INNECH

                    * TERMINAL OUTPUT CHARACTER ROUTINE

D38A 34   02        OUTPUT  PSHS   A          SAVE CHARACTER
D38C B6   E004      OUTPU2  LDA    ACIA       TRANSMIT BUFFER EMPTY?
D38F 84   02                ANDA   #$02
D391 27   F9                BEQ    OUTPU2     WAIT IF NOT
D393 35   02                PULS   A          RESTORE CHARACTER
D395 B7   E005              STA    ACIA+1     OUTPUT IT
D398 39                     RTS

                    * TERMINAL STATUS CHECK (CHECK FOR CHARACTER HIT)

D399 34   02        STATUS  PSHS   A          SAVE A REG.
D39B B6   E004              LDA    ACIA       GET STATUS
D39E 84   01                ANDA   #$01       CHECK FOR CHARACTER
D3A0 35   02                PULS   A          RESTORE A REG.
D3A2 39                     RTS


                    * TIMER INITIALIZE ROUTINE

D3A3 8E   E012      TINT    LDX    #TMPIA     GET PIA ADDRESS
D3A6 86   FF                LDA    #$FF       SET SIDE B AS OUTPUTS
D3A8 A7   84                STA    0,X
D3AA 86   3C                LDA    #$3C       CONFIGURE PIA CONTROL
D3AC A7   01                STA    1,X
D3AE 86   8F                LDA    #$8F       TURN OFF TIMER
D3B0 A7   84                STA    0,X
D3B2 A6   84                LDA    0,X        CLR ANY PENDING INTRRPTS
D3B4 86   3D                LDA    #$3D       RECONFIGURE PIA
D3B6 A7   01                STA    1,X
D3B8 39                     RTS

                    * TIMER ON ROUTINE

D3B9 86   04        TON     LDA    #$04       TURN ON TIMER (10ms)
D3BB 20   02                BRA    TOFF2

                    * TIMER OFF ROUTINE

D3BD 86   8F        TOFF    LDA    #$8F       TURN OFF TIMER
```

```
D3BF B7   E012      TOFF2   STA    TMPIA
D3C2 39                     RTS

                    * IRQ INTERRUPT HANDLER ROUTINE

D3C3 B6   E012      IHND    LDA    TMPIA      CLR ANY PENDING INTRRPTS
D3C6 7E   C700              JMP    CHPR       SWITCH PROCESSES

                    * END STATEMENT HAS FLEX TRANSFER ADDRESS!

                            END    $CD00
```

```
                  * DRIVER ROUTINES FOR SWTPC MF-68
                  *
                  * COPYRIGHT (C) 1980 BY
                  * TECHNICAL SYSTEMS CONSULTANTS, INC.
                  * 111 PROVIDENCE RD, CHAPEL HILL, NC 27514
                  *
                  * THESE DRIVERS ARE FOR A SINGLE-SIDED, SINGLE
                  * DENSITY SWTPC MF-68 MINIFLOPPY DISK SYSTEM.
                  *
                  * THE  DRIVER ROUTINES PERFORM THE FOLLOWING
                  *  1.  READ SINGLE SECTOR - DREAD
                  *  2.  WRITE SINGLE SECTOR - DWRITE
                  *  3.  VERIFY WRITE OPERATION - VERIFY
                  *  4.  RESTORE HEAD TO TRACK 00 - RESTOR
                  *  5.  DRIVE SELECTION - DRIVE
                  *  6.  CHECK READY - DCHECK
                  *  7.  QUICK CHECK READY - DQUICK
                  *  8.  DRIVER INITIALIZATION - DINIT
                  *  9.  WARM START ROUTINE - DWARM
                  * 10.  SEEK ROUTINE - DSEEK

                  *EQUATES

             0002 DRQ     EQU    2         DRQ BIT MASK
             0001 BUSY    EQU    1         BUSY MASK
             001C RDMSK   EQU    $1C       READ ERROR MASK
             0018 VERMSK  EQU    $18       VERIFY ERROR MASK
             005C WTMSK   EQU    $5C       WRITE ERROR MASK
             E014 DRVREG  EQU    $E014     DRIVE REGISTER
             E018 COMREG  EQU    $E018     COMMMAND REGISTER
             E019 TRKREG  EQU    $E019     TRACK REGISTER
             E01A SECREG  EQU    $E01A     SECTOR REGISTER
             E01B DATREG  EQU    $E01B     DATA REGISTER
             008C RDCMND  EQU    $8C       READ COMMAND
             00AC WTCMND  EQU    $AC       WRITE COMMAND
             000B RSCMND  EQU    $0B       RESTORE COMMAND
             001B SKCMND  EQU    $1B       SEEK COMMAND
             CC34 PRCNT   EQU    $CC34

                  ***********************************************
                  * DISK DRIVER ROUTINE JUMP TABLE
                  ***********************************************
DE00                      ORG    $DE00
DE00 7E   DE2E    DREAD   JMP    READ
DE03 7E   DE8E    DWRITE  JMP    WRITE
DE06 7E   DEC1    DVERFY  JMP    VERIFY
DE09 7E   DEDA    RESTOR  JMP    RST
DE0C 7E   DEED    DRIVE   JMP    DRV
DE0F 7E   DF10    DCHECK  JMP    CHKRDY
DE12 7E   DF10    DQUICK  JMP    CHKRDY
DE15 7E   DE23    DINIT   JMP    INIT
DE18 7E   DE2D    DWARM   JMP    WARM
DE1B 7E   DE71    DSEEK   JMP    SEEK
                  ***********************************************
```

```
                      * GLOBAL VARIABLE STORAGE

DE1E 00               CURDRV  FCB    0          CURRENT DRIVE
DE1F 0000 0000        DRVTRK  FDB    0,0        CURRENT TRACK PER DRIVE


                      * INIT AND WARM
                      *
                      * DRIVER INITIALIZATION

DE23 BE   DE1E        INIT    LDX    #CURDRV    POINT TO VARIABLES
DE26 C6   05                  LDB    #5         NO. OF BYTES TO CLEAR
DE28 6F   80          INIT2   CLR    0,X+       CLEAR THE STORAGE
DE2A 5A                       DECB
DE20 26   FB                  BNE    INIT2      LOOP TIL DONE
DE2D 39               WARM    RTS               WARM START NOT NEEDED

                      * READ
                      *
                      * READ ONE SECTOR

DE2E 8D   41          READ    BSR    SEEK       SEEK TO TRACK
DE30 86   8C                  LDA    #RDCMND    SETUP READ SECTOR COMMAND
DE32 7D   CC34                TST    PRCNT      ARE WE SPOOLING?
DE35 27   03                  BEQ    READ2      SKIP IF NOT
DE37 113F                     SWI3              ELSE, SWITCH TASKS
DE39 12                       NOP               NECESSARY FOR SBUG
DE3A 1A   10          READ2   SEI               DISABLE INTERRUPTS
DE3C B7   E018                STA    COMREG     ISSUE READ COMMAND
DE3F 17   00E5                LBSR   DEL28      DELAY
DE42 5F                       CLRB              GET SECTOR LENGTH (=256)
DE43 B6   E018        READ3   LDA    COMREG     GET WD STATUS
DE46 85   02                  BITA   #DRQ       CHECK FOR DATA
DE48 26   08                  BNE    READ5      BRANCH IF DATA PRESENT
DE4A 85   01                  BITA   #BUSY      CHECK IF BUSY
DE4C 26   F5                  BNE    READ3      LOOP IF SO
DE4E 1F   89                  TFR    A,B        ERROR IF NOT
DE50 20   0A                  BRA    READ6
DE52 B6   E01B        READ5   LDA    DATREG     GET DATA BYTE
DE55 A7   80                  STA    0,X+       PUT IN MEMORY
DE57 5A                       DECB              DEC THE COUNTER
DE58 26   E9                  BNE    READ3      LOOP TIL DONE
DE5A 8D   05                  BSR    WAIT       WAIT TIL WD IS FINISHED
DE5C C5   1C          READ6   BITB   #RDMSK     MASK ERRORS
DE5E 1C   EF                  CLI               ENABLE INTERRUPTS
DE60 39                       RTS               RETURN

                      * WAIT
                      *
                      * WAIT FOR 1771 TO FINISH COMMAND

DE61 7D   CC34        WAIT    TST    PRCNT      ARE WE SPOOLING?
DE64 27   03                  BEQ    WAIT1      SKIP IF NOT
```

```
DE66 113F               SWI3                SWITCH TASKS IF SO
DE68 12                 NOP                 NECESSARY FOR SBUG
DE69 F6   E018   WAIT1  LDB    COMREG       GET WD STATUS
DE6C C5   01            BITB   #BUSY        CHECK IF BUSY
DE6E 26   F1            BNE    WAIT         LOOP TIL NOT BUSY
DE70 39                 RTS                 RETURN

                           * SEEK
                           *
                           * SEEK THE SPECIFIED TRACK

DE71 F7   E0 1A  SEEK   STB    SECREG       SET SECTOR
DE74 B1   E019          CMPA   TRKREG       DIF THAN LAST?
DE77 27   12            BEQ    SEEK4        EXIT IF NOT
DE79 B7   E01B          STA    DATREG       SET NEW WD TRACK
DE7C 17   00A8          LBSR   DEL28        GO DELAY
DE7F 86   1B            LDA    #SKCMND      SETUP SEEK COMMAND
DE81 B7   E018          STA    COMREG       ISSUE SEEK COMMAND
DE84 17   00A0          LBSR   DEL28        GO DELAY
DE87 8D   D8            BSR    WAIT         WAIT TIL DONE
DE89 C5   10            BITB   #$10         CHECK FOR SEEK ERROR
DE8B 16   0099   SEEK4  LBRA   DEL28        DELAY

                 * WRITE
                 *
                 * WRITE ONE SECTOR

DE8E 8D   E1     WRITE  BSR    SEEK        SEEK TO TRACK
DE90 86   AC            LDA    #WTDMND     SETUP WRITE SCTR COMMAND
DE92 7D   CC34          TST    PRCNT       ARE WE SPOOLING?
DE95 27   03            BEQ    WRITE2      SKIP IF NOT
DE97 113F               SWI3               CHANGE TASKS IF SO
DE99 12                 NOP                NECESSARY FOR SBUG
DE9A 1A   10     WRITE2 SEI                DISABLE INTERRUPTS
DE9C B7   E018          STA    COMREG      ISSUE WRITE COMMAND
DE9F 17   0085          LBSR   DEL28       DELAY
DEA2 5F                 CLRB               GET SECTOR LENGTH (=256)
DEA3 B6   E018   WRITE3 LDA    COMREG      CHECK WD STATUS
DEA6 85   02            BITA   #DRQ        READY FOR DATA?
DEA8 26   08            BNE    WRITE5      SKIP IF READY
DEAA 85   01            BITA   #BUSY       STILL BUSY?
DEAC 26   F5            BNE    WRITE3      LOOP IF SO
DEAE 1F   89            TFR    A,B         ERROR IF NOT
DEB0 20   0A            BRA    WRITE6
DEB2 A6   80     WRITE5 LDA    0,X+        GET A DATA BYTE
DEB4 B7   E01B          STA    DATREG      SEND TO DISK
DEB7 5A                 DECB               DEC THE COUNT
DEB8 26   E9            BNE    WRITE3      FINISHED?
DEBA 8D   A5            BSR    WAIT        WAIT TIL WD IS FINISHED
DEBC C5   5C     WRITE6 BITB   #WTMSK      MASK ERRORS
DEBE 1C   EF            CLI                ENABLE INTERRUPTS
DEC0 39                 RTS                RETURN

                 * VERIFY
```

```
                    *
                    * VERIFY LAST SECTOR WRITTEN

DEC1 86   8C        VERIFY  LDA    #RDCMND   SETUP VERIFY COMMAND
DEC3 7D   CC34              TST    PRCNT     ARE WE SPOOLING?
DEC6 27   03                BEQ    VERIF2    SKIP IF NOT
DEC8 113F                   SWI3             CHANGE TASKS IF SO
DECA 12                     NOP              NECESSARY FOR SBUG
DECB 1A   10        VERIF2  SEI              DISABLE INTERRUPTS
DECD B7   E018              STA    COMREG    ISSUE VERIFY COMMAND
DED0 17   0054              LBSR   DEL28     GO DELAY
DED3 8D   8C                BSR    WAIT      WAIT TIL WD IS DONE
DED5 1C   EF                CLI              ENABLE INTERRUPTS
DED7 C5   18                BITB   #VERMSK   MASK ERRORS
DED9 39                     RTS              RETURN

                    * RST
                    *
                    * RST RESTORES THE HEAD TO 00

DEDA 34   10        RST     PSHS   X         SAVE X REGISTER
DEDC 8D   0F                BSR    DRV       DO SELECT
DEDE 86   0B                LDA    #RSCMND   SETUP RESTORE COMMAND
DEE0 B7   E018              STA    COMREG    ISSUE RESTORE COMMAND
DEE3 8D   42                BSR    DEL28     DELAY
DEE5 17   FF79              LBSR   WAIT      WAIT TIL WD IS FINISHED
DEE8 35   10                PULS   X         RESTORE POINTER
DEEA C5   D8                BITB   #$D8      CHECK FOR ERROR
DEEC 39                     RTS              RETURN

                    * DRV
                    *
                    * SELECT THE SPECIFIED  DRIVE

DEED A6   03        DRV     LDA    3,X        GET DRIVE NUMBER
DEEF 81   03                CMPA   #3         ENSURE IT'S < 4
DEF1 23   05                BLS    DRV2       BRANCH IF OK
DEF3 C6   0F                LDB    #$0F       ELSE SET ERROR VALUE
DEF5 1A   01                SEC
DEF7 39                     RTS
DEF8 8D   25        DRV2    BSR    FNDTRK     FIND TRACK
DEFA F6   E019              LDB    TRKREG     GET CURRENT TRACK
DEFD E7   84                STB    0,X        SAVE IT
DEFF B7   E014              STA    DRVREG     SET NEW DRIVE
DF02 B7   DE1E              STA    CURDRV
DF05 8D   18                BSR    FNDTRK     FIND NEW TRACK
DF07 A6   84                LDA    0,X
DF09 B7   E019              STA    TRKREG     PUT NEW TRACK IN WD
DF0C 8D   19                BSR    DEL28      DELAY
DF0E 20   0B                BRA    OK
```

```
                  * CHKRDY
                  *
                  * CHECK DRIVE READY ROUTINE

DF10 A6   03      CHKRDY   LDA    3,X       GET DRIVE NUMBER
DF12 81   01               CMPA   #1        BE SURE IT'S 0 OR 1
DF14 23   05               BLS    OK        BRANCH IF OK
DF16 C6   80               LDB    #$80      ELSE, SHOW NOT READY
DF18 1A   01               SEC
DFLA 39                    RTS              RETURN
DF1B 5F           OK       CLRB             SHOW NO ERROR
DF1C 1C   FE               CLC
DFLE 39                    RTS

                   * FIND THE TRACK FOR CURRENT DRIVE

DF1F 8E   DE1F     FNDTRK  LDX    #DRVTRK   POINT TO TRACK STORE
DF22 F6   DE1E             LDB    CURDRV    GET CURRENT DRIVE
DF25 3A                    ABX              POINT TO DRIVE'S TRACK
DF26 39                    RTS               RETURN

                   * DELAY

DF27  17   0000    DEL28   LBSR   DEL14
DF2A  17   0000    DEL14   LBSR   DEL
DF2D  39           DEL     RTS

                           END
```

```
                  * ROM BOOT FOR SWTPC 6809 MF-68
                  *
                  * COPYRIGHT (C) 1980 BY
                  * TECHNICAL SYSTEMS CONSULTANTS. INC.
                  * 111 PROVIDENCE RD, CHAPEL HILL, NC 27514

                  * EQUATES

            E014  DRVREG  EQU    $E014
            E018  COMREG  EQU    $E018
            E01A  SECREG  EQU    $E01A
            E01B  DATREG  EQU    $E01B
            C100  LOADER  EQU    $C100

7000                      ORG    $7000

7000 B6   E018    START   LDA     COMREG    TURN MOTOR ON
7003 7F   E014            CLR     DRVREG    SELECT DRIVE #0
7006 8E   0000            LDX     #0000
7009 30   1F      OVR     LEAX    -1,X      DELAY FOR MOTOR SPEEDUP
700B 26   FC              BNE     OVR
700D C6   0F              LDB     #$0F      DO RESTORE COMMAND
700F F7   E018            STB     COMREG
7012 8D   2B              BSR     DELAY
7014 F6   E018    LOOP1   LDB     COMREG    CHECK WD STATUS
7017 C5   01              BITB    #1        WAIT TIL NOT BUSY
7019 26   F9              BNE     LOOP1
701B 86   01              LDA     #1        SETUP FOR SECTOR #1
701D B7   E01A            STA     SECREG
7020 8D   1D              BSR     DELAY
7022 C6   8C              LDB     #$8C      SETUP READ COMMAND
7024 F7   E018            STB     COMREG
7027 8D   16              BSR     DELAY
7029 8E   C100            LDX     #LOADER   ADDRESS OF LOADER
702C C5   02      LOOP2   BITB    #2        DATA PRESENT?
702E 27   05              BEQ     LOOP3     SKIP IF NOT
7030 B6   E01B            LDA     DATREG    GET A BYTE
7033 A7   80              STA     0,X+      PUT IN MEMORY
7035 F6   E018    LOOP3   LDB     COMREG    CHECK WD STATUS
7038 C5   01              BITB    #1        IS WD BUSY?
703A 26   F0              BNE     LOOP2     LOOP IF SO
703C 7E   C100            JMP     LOADER    JUMP TO FLEX LOADER

703F 8D   00      DELAY   BSR     RTN
7041 39           RTN     RTS

                          END     START
```

```
                    * LOADER - FLEX LOADER ROUTINE
                    *
                    * COPYRIGHT (C) 1980 BY
                    * TECHNICAL SYSTEMS CONSULTANTS INC.
                    * 111 PROVIDENCE RD, CHAPEL HILL, NC 27514
                    *
                    * LOADS FLEX FROM DISK.  ASSUMES DRIVE IS ALREADY
                    * SELECTED AND A RESTORE HAS BEEN PERFORMED BY THE
                    * ROM BOOT AND THAT STARTING TRACK AND SECTOR OF
                    * FLEX ARE AT $C105 AND $C106.  BEGIN EXECUTION
                    * BY JUMPING TO LOCATION $C100.  JUMPS TO FLEX
                    * STARTUP WHEN COMPLETE.
                    *

                    * EQUATES

              C07F  STACK   EQU   $C07F
              C300  SCTBUF  EQU   $C300      DATA SECTOR BUFFER


                    * START OF UTILITY

C100                        ORG   $C100

C100 20   0A        LOAD    BRA   LOAD0

C102 00 00 00               FCB   0,0,0
C105 00             TRK     FCB   0           FILE START TRACK
C106 00             SCT     FCB   0           FILE START SECTOR
C107 00             DNS     FCB   0           DENSITY FLAG
C108 C100           TADR    FDB   $c100       TRANSFER ADDRESS
C10A 0000           LADR    FDB   0           LOAD ADDRESS

C10C 10CE C07F      LOAD0   LDS   #STACK      SETUP STACK
C110 FC   C105              LDD   TRK         SETUP STARTING TRK & SCT
C113 FD   C300              STD   SCTBUF
C116 108E C400              LDY   #SCTBUF+256

                    * PERFORM ACTUAL FILE LOAD

C11A 8D   35        LOAD1   BSR   GETCH       GET A CHARACTER
C11C 81   02                CMPA  #$02        DATA RECORD HEADER?
C11E 27   10                BEQ   LOAD2       SKIP IF SO
C120 81   16                CMPA  #$16        XFR ADDRESS HEADER?
C122 26   F6                BNE   LOAD1       LOOP IF NEITHER
C124 8D   2B                BSR   GETCH       GET TRANSFER ADDRESS
C126 B7   C108              STA   TADR
C129 8D   26                BSR   GETCH
C12B B7   C109              STA   TADR+1
C12E 20   EA                BRA   LOAD1       CONTINUE LOAD
C130 8D   1F        LOAD2   BSR   GETCH       GET LOAD ADDRESS
C132 B7   C10A              STA   LADR
C135 8D   1A                BSR   GETCH
C137 B7   C10B              STA   LADR+1
```

```
C13A 8D   15                BSR    GETCH      GET BYTE COUNT
C13C 1F   894D              TAB               PUT IN B
C13F 27   D9                BEQ    LOAD1      LOOP IF COUNT=0
C141 BE   C10A              LDX    LADR       GET LOAD ADDRESS
C144 34   14       LOAD3    PSHS   B,X
C146 8D   09                BSR    GETCH      GET A DATA CHARACTER
C148 35   14                PULS   B,X
C14A A7   80                STA    0,X+       PUT CHARACTER
C14C 5A                     DECB              END OF DATA IN RECORD?
C14D 26   F5                BNE    LOAD3      LOOP IF NOT
C14F 20   C9                BRA    LOAD1      GET ANOTHER RECORD

                   *  GET CHARACTER ROUTINE - READS A SECTOR IF NECESSARY

C151 108C C400     GETCH    CMPY   #SCTBUF+256 OUT OF DATA?
C155 26   0F                BNE    GETCH4     GO READ CHARACTER IF NOT
C157 8E   C300     GETCH2   LDX    #SCTBUF    POINT TO BUFFER
C15A EC   84                LDD    0,X        GET FORWARD LINK
C15C 27   0B                BEQ    GO         IF ZERO, FILE IS LOADED
C15E 8D   0D                BSR    READ       READ NEXT SECTOR
C160 26   9E                BNE    LOAD       START OVER IF ERROR
C162 108E C304              LDY    #SCTBUF+4 POINT PAST LINK
C166 A6   A0       GETCH4   LDA    0,Y+       ELSE, GET A CHARACTER
C168 39                     RTS

                   * FILE IS LOADED, JUMP TO IT

C169 6E   9F C108  GO       JMP    [TADR]     JUMP TO TRANSFER ADDRESS


                   * READ SINGLE SECTOR
                   *
                   * THIS ROUTINE MUST READ THE SECTOR WHOSE TRACK
                   * AND SECTOR ADDRESS ARE IN A ANB B ON ENTRY.
                   * THE DATA FROM THE SECTOR IS TO BE PLACED AT
                   * THE ADDRESS CONTAINED IN X ON ENTRY.
                   * IF ERRORS, A NOT-EQUAL CONDITION SHOULD BE
                   * RETURNED.  THIS ROUTINE WILL HAVE TO DO SEEKS.
                   * A,B,X, AND U MAY BE DESTROYED BY THIS ROUTINE,
                   * BUT Y MUST BE PRESERVED.

                   * WESTERN DIGITAL EQUATES

             E018  COMREG   EQU    $E018      COMMAND REGISTER
             E019  TRKREG   EQU    $E019      TRACK REGISTER
             E01A  SECREG   EQU    $E01A      SECTOR REGISTER
             E01B  DATREG   EQU    $E01B      DATA REGISTER
             0002  DRQ      EQU    2          DRQ BIT MASK
             0001  BUSY     EQU    1          BUSY MASK
             001C  RDMSK    EQU    $1C        READ ERROR MASK
             008C  RDCMND   EQU    $8C        READ COMMAND
             001B  SKCMND   EQU    $1B        SEEK COMMAND

                   * READ ONE  SECTOR
```

```
C16D 8D   2F      READ    BSR   XSEEK    SEEK TO TRACK
C16F 86   8C              LDA   #RDCMND  SETUP READ SECTOR COMMAND
C171 B7   E018            STA   COMREG   ISSUE READ COMMAND
C174 BD   3E              BSR   DEL28    DELAY
C176 5F                   CLRB           GET SECTOR LENGTH (=256)
C177 BE   C300            LDX   #SCTBUF  POINT TO SECTOR BUFFER
C17A B6   E018    READ3   LDA   COMREG   GET WD STATUS
C17D 85   02              BITA  #DRQ     CHECK FOR DATA
C17F 26   08              BNE   READ5    BRANCH IF DATA PRESENT
C181 85   01              BITA  #BUSY    CHECK IF BUSY
C183 26   F5              BNE   READ3    LOOP IF SO
C185 1F   89              TFR   A,B      SAVE ERROR CONDITION
C187 20   0A              BRA   READ6
C189 B6   E01B    READ5   LDA   DATREG   GET DATA BYTE
C18C A7   80              STA   0,X+     PUT IN MEMORY
C18E 5A                   DECB           DEC THE COUNTER
C18F 26   E9              BNE   READ3    LOOP TIL DONE
C191 8D   03              BSR   XWAIT    WAIT TIL WD IS FINISHED
C193 C5   1C      READ6   BITB  #RDMSK   MASK ERRORS
C195 39                   RTS            RETURN

                   * WAIT FOR 1771 TO FINISH COMMAND

C196 F6   E018    XWAIT   LDB   COMREG   GET WD STATUS
C199 C5   01              BITB  #BUSY    CHECK IF BUSY
C19B 26   F9              BNE   XWAIT    LOOP TIL NOT BUSY
C19D 39                   RTS            RETURN

                  * SEEK THE SPECIFIED TRACK

C19E F7   E01A    XSEEK   STB   SECREG   SET SECTOR
C1A1 B1   E019            CMPA  TRKREG   DIF THAN LAST?
C1A4 27   0E              BEQ   DEL28    EXIT IF NOT
C1A6 B7   E01B            STA   DATREG   SET NEW WD TRACK
C1A9 8D   09              BSR   DEL28    GO DELAY
C1AB 86   1B              LDA   #SKCMN   SETUP SEEK COMMAND
C1AD B7   E018            STA   COMREG   ISSUE SEEK COMMAND
C1B0 BD   02              BSR   DEL28    GO DELAY
C1B2 BD   E2              BSR   XWAIT    WAIT TIL DONE

                  * DELAY

C1B4 BD   C1B7    DEL28   JSR   DEL14
C1B7 BD   C1BA    DEL14   JSR   DEL
C1BA 39           DEL     RTS


                          END
```

```
        * NEWDISK
        *
        * COPYRIGHT (C) 1980 BY
        * TECHNICAL SYSTEMS CONSULTANTS. INC.
        * 111 PROVIDENCE RD, CHAPEL HILL, NC 27514
        *
        * DISK FORMATTING PROGRAM FOR 6809 FLEX.
        * GENERAL VERSION DESIGNED FOR WD 1771/1791.
        * THE NEWDISK PROGRAM INITIALIZES A NEW DISKETTE AND
        * THEN PROCEEDS TO VERIFY ALL SECTORS AND INITIALIZE
        * TABLES.  THIS VERSION IS SETUP FOR AN 8 INCH DISK
        * SYSTEM WITH HINTS AT CERTAIN POINTS FOR ALTERING
        * FOR A SINGLE-DENSITY 5 INCH DISK SYSTEM.  THIS
        * VERSION IS NOT INTENDED FOR 5 INCH DOUBLE-DENSITY.

        **************************************************
        * DISK SIZE PARAMETERS
        * **** **** **********
        * THE FOLLOWING CONSTANTS SETUP THE SIZE OF THE
        * DISK TO BE FORMATTED. THE VALUES SHOWN ARE FOR
        * 8 INCH DISKS. FOR 5 INCH DISKS, USE APPROPRIATE
        * VALUES. (IE. 35 TRACKS AND 10 SECTORS PER SIDE)
        **************************************************

0023    MAXTRK   EQU   35        NUMBER OF TRACKS
        * SINGLE DENSITY:
000A    SMAXS0   EQU   10        SD MAX SIDE 0 SECTORS
000A    SMAXS1   EQU   10        SD MAX SIDE 1 SECTORS
        * DOUBLE DENSITY:
000A    DMAXS0   EQU   10        DD MAX SIDE 0 SECTORS
000A    DMAXS1   EQU   10        DD MAX SIDE 1 SECTORS

        **************************************************
        * MORE DISK SIZE DEPENDENT PARAMETERS
        * **** **** **** ********* **********
        * THE FOLLOWING VALUES ARE ALSO DEPENDENT ON THE
        * SIZE OF DISK BEING FORMATTED. EACH VALUE SHOWN
        * IS FOR 8 INCH WITH PROPER 5 INCH VALUES IN
        * PARENTHESES.
        **************************************************

        * SIZE OF SINGLE DENSITY WORK BUFFER FOR ONE TRACK
0BEA    TKSZ    EQU    3050      (USE 3050 FOR 5 INCH)
        * TRACK START VALUE
0000    TST     EQU    0         (USE 0 FOR 5 INCH)
        * SECTOR START VALUE
0007    SST     EQU    7         (USE 7 FOR 5 INCH)
        * SECTOR GAP VALUE
000E    GAP     EQU    14        (USE 14 FOR 5 INCH)

        **************************************************
                ...
                ...
                etc.
```

```
                           ...
                           ...
                           ...

014E 8D   B6               BSR    OUTIN2    GET RESPONSE
0150 26   51               BNE    EXIT      EXIT IF "NO"
0152 0F   25               CLR    DBSDF     CLEAR FLAG
                   * PLACE A "BRA FORM25" HERE IF CONTROLLER
                   * IS ONLY SINGLE SIDED.

0154 20   0D               BRA    FORM25
0156 8E   04F2             LDX    #DBST     ASK IF DOUBLE SIDED
0159 8D   A8               BSR    OUTIN     PRINT & GET RESPONSE
015B 26   06               BNE    FORM25    SKIP IF "NO"
015D 0C   25               INC    DBSDF     SET FLAG
015F 86   0A               LDA    #SMAXS1   SET MAX SECTOR
0161 97   32               STA    MAX
0163 0F   26       FORM25  CLR    DENSE     INITIALIZE SINGLE DENSITY
0165 0F   027              CLR    DNSITY
                   *  PLACE A "BRA FORM26" HERE IF CONTROLLER
                   * IS ONLY SINGLE DENSITY.

0167 20   09               BRA    FORM26

0169 8E   0506             LDX    #DDSTR    ASK IF DOUBLE DENSITY
016C 8D   95               BSR    OUTIN     PRINT & GET RESPONSE
016E 26   02               BNE    FORM26    SKIP IF "NO"
0170 0C   26               INC    DENSE     SET FLAG IF SO
0172 8E   051C     FORM26  LDX    #NMSTR    ASK FOR VOLUME NAME
                           ...
                           ...
                           ...
                           etc.
```

```
                  ************************************************
                  * SECTOR MAPS
                  * ****** ****
                  * THE MAPS SHOWN BELOW CONTAIN THE CORRECT
                  * INTERLEAVING FOR AN 8 INCH DISK.  IF USING 5
                  * INCH DISKS (SINGLE DENSITY) YOU SHOULD USE
                  * SOMETHING LIKE '1,3,5,7,9,2,4,6,8,10' FOR
                  * SSCMAP FOR A SINGLE SIDED DISK.
                  ************************************************

0458 01 03 05 07  SSCMAP  FCB    1,3,5,7,9,2,4,6,8,10

             0458  DSCMAP  EQU    SSCMAP


                  * STRINGS

0462 41 52 45 20  SURES   FCC    'ARE YOU SURE? '
0470 04                   FCB    4
0471 44 49 53 4B  WPST    FCC    'DISK IS PROTECTED!'
0483 04                   FCB    4
0484 53 43 52 41  SCRDS   FCC    'SCRATCH DISK IN DRIVE '
049A 04                   FCB    4
049B 46 41 54 41  FATERS  FCC    'FATAL ERROR --- '
04AB 46 4F 52 4D  ABORTS  FCC    'FORMATTING ABORTED'
04BD 04                   FCB    4
04BE 42 41 44 20  BADSS   FCC    'BAD SECTOR AT '
04CC 04                   FCB    4
04CD 46 4F 52 4D  CMPLTE  FCC    'FORMATTING COMPLETE'
04E0 04                   FCB    4
04E1 54 4F 54 41  SECST   FCC    'TOTAL SECTORS = '
04F1 04                   FCB    4
04F2 44 4F 55 42  DBST    FCC    'DOUBLE SIDED DISK? '
0505 04                   FCB    4
0506 44 4F 55 42  DDSTR   FCC    'DOUBLE DENSITY DISK? '
051B 04                   FCB    4
051C 56 4F 4C 55  NMSTR   FCC    'VOLUME NAME? '
0529 04                   FCB    4
052A 56 4F 4C 55  NUMSTR  FCC    'VOLUME NUMBER? '
0539 04                   FCB    4
```

```
***************************************************************
* WRITE TRACK ROUTINE                                         *
***************************************************************
* THIS SUBROUTINE MUST BE USER SUPPLIED.                      *
* IT SIMPLY WRITES THE DATA FOUND AT "WORK" ($0800) TO THE    *
* CURRENT TRACK ON THE DISK.  NOTE THAT THE SEEK TO TRACK     *
* OPERATION HAS ALREADY BEEN PERFORMED.  IF SINGLE DENSITY,   *
* "TKSZ" BYTES SHOULD BE WRITTEN.  IF DOUBLE, "TKSZ*2"        *
* BYTES SHOULD BE WRITTEN.  THIS ROUTINE SHO6LD PERFORM       *
* ANY NECESSARY DENSITY SELECTION BEFORE WRITING.  DOUBLE     *
* DENSITY IS INDICATED BY THE BYTE "DNSITY" BEING NON-ZERO.   *
* THERE ARE NO ENTRY PARAMETERS AND ALL REGISTERS MAY BE      *
* DESTROYED ON EXIT.  THE CODE FOR THIS ROUTINE MUST NOT      *
* EXTEND PAST $0800 SINCE THE TRACK DATA IS STORED THERE.     *
***************************************************************

                   ********************************************
                   * WESTERN DIGITAL PARAMTERS
                   * ******* ******* *********

                   * REGISTERS:
            E018   COMREG  EQU    $E018     COMMAND REGISTER
            E019   TRKREG  EQU    $E019     TRACK REGISTER
            E01A   SECREG  EQU    $E01A     SECTOR REGISTER
            E01B   DATREG  EQU    $E018     DATA REGISTER
                   * COMMANDS:
            00F4   WTCMD   EQU    $F4       WRITE TRACK COMMAND
                   ********************************************

                   ********************************************
                   * CONTROLLER DEPENDENT PARAMETERS
                   * ********** ********* **********
            E014   DRVREG  EQU    $E014     DRIVE SELECT REGISTER
                   ********************************************


053A 8E    0800    WRTTRK  LDX    #WORK     POINT TO DATA
053D 86    F4              LDA    #WTCMD    SETUP WRITE TRACK COMMAND
053F 87    E018            STA    COMREG    ISSUE COMMAND
0542 BD    0564            JSR    DELAY
0545 B6    E018    WRTTR2  LDA    COMREG    CHECK WD STATUS
0548 85    02              BITA   #$02      IS WD READY FOR DATA?
054A 26    06              BNE    WRTTR4    SKIP IF READY
054C 85    01              BITA   #$01      IS WD BUSY?
054E 26    F5              BNE    WRTTR2    LOOP IF BUSY
0550 20    11              BRA    WRTTR8    EXIT IF NOT
0552 A6    80      WRTTR4  LDA    0,X+      GET A DATA BYTE
0554 B7    E01B            STA    DATREG    SEND TO DISK
0557 BC    13EA            CMPX   #SWKEND   OUT OF DATA?
055A 26    E9              BNE    WRTTR2    REPEAT IF NOT
055C B6    E018    WAIT    LDA    COMREG    CHECK WD STATUS
055F 85    01              BITA   #$01      IS IT BUSY?
0561 26    F9              BNE    WAIT      LOOP IF SO
0563 39            WRTTR8  RTS              RETURN
```

```
0564 BD   0567      DELAY   JSR    DELAY2
0567 BD   056A      DELAY2  JSR    DELAY4
056A 39             DELAY4  RTS
```

```
**********************************************************
*
* BOOTSTRAP FLEX LOADER
*
* THE CODE FOR THE BOOTSTRAP FLEX LOADER MUST BE IN MEMORY
* AT $C100 WHEN NEWDISK IS RUN.  THERE ARE TWO WAYS IT CAN
* BE PLACED THERE.  ONE IS TO ASSEMBLE THE LOADER AS A
* SEPARATE FILE AND APPEND IT ONTO THE END OF THE NEWDISK
* FILE.  THE SECOND IS TO SIMPLY PUT THE SOURCE FOR THE
* LOADER IN-LINE HERE WITH AN ORG TO $C100.  THE FIRST FEW
* LINES OF CODE FOR THE LATTER METHOD ARE GIVEN HERE TO
* GIVE THE USER AN IDEA OF HOW TO SETUP THE LOADER SOURCE.
*
* IT IS NOT NECESSARY TO HAVE THE LOADER AT $C100 IN ORDER
* FOR THE NEWDISK TO RUN.  IT SIMPLY MEANS THAT WHATEVER
* HAPPENS TO BE IN MEMORY AT $C100 WHEN NEWDISK IS RUN
* WOULD BE WRITTEN OUT AS A BOOT.  AS LONG AS THE CREATED
* DISK WAS FOR USE AS A DATA DISK ONLY AND WOULD NOT BE
* BOOTED FROM, THERE WOULD BE NO PROBLEM.
*
**********************************************************

                  * 6809 BOOTSTRAP FLEX LOADER

                  * EQUATES

            C07F   STACK   EQU    $C07F
            C300   SCTBUF  EQU    $C300      DATA SECTOR BUFFER

                  * START OF UTILITY

C100                       ORG    $C100
C100 20   0A       BOOT    BRA    LOAD0

C102 00 00 00              FCB    0,0,0
C105 00            TRK     FCB    0          FILE START TRACK
C106 00            SCT     FCB    0          FILE START SECTOR
C107 00            DNS     FCB    0          DENSITY FLAG
C108 C100          TADR    FDB    $C100      TRANSFER ADDRESS
C10A 0000          LADR    FDB    0          LOAD ADDRESS
C10C 10CE C07F     LOAD0   LDS    #STACK     SETUP STACK
C110 FC   C105             LDD    TRK        SETUP STARTING TRK & SCT
C113 FD   C300             STD    SCTBUF
C116 108E C400             LDY    #SCTBUF+256

                  * PERFORM ACTUAL FILE LOAD

C11A 8D   35       LOAD1   BSR    GETCH      GET A CHARACTER
C11C 81   02               CMPA   #$02       DATA RECORD HEADER?
C11E 27   10               BEQ    LOAD2      SKIP,IF SO
C120 81   16               CMPA   #$16       XFR ADDRESS HEADER?
C122 26   F6               BNE    LOAD1      LOOP IF NEITHER
C124 8D   2B               BSR    GETCH      GET TRANSFER ADDRESS
```

```
C126 B7   C108            STA    TADR
C129 8D   26              BSR    GETCH
C12B B7   C109            STA    TADR+1
C12E 20   EA              BRA    LOAD1      CONTINUE LOAD
C130 8D   1F      LOAD2   BSR    GETCH      GET LOAD ADDRESS
C132 B7   C10A            STA    LADR
C135 8D   1A              BSR    GETCH
C137 B7   C10B            STA    LADR+1
C13A 8D   15              BSR    GETCH      GET BYTE COUNT
C13C 1F   894D            TAB               PUT IN B
C13F 27   D9              BEQ    LOAD1      LOOP IF COUNT=0
C141 BE   C10A            LDX    LADR       GET LOAD ADDRESS
C144 34   14      LOAD3   PSHS   B,X
C146 8D   09              BSR    GETCH      GET A DATA CHARACTER
C148 35   14              PULS   B,X
C14A A7   80              STA    0,X+       PUT CHARACTER
C14C 5A                   DECB              END OF DATA IN RECORD?
C14D 26   F5              BNE    LOAD3      LOOP IF NOT
C14F 20   C9              BRA    LOAD1      GET ANOTHER RECORD

                  * GET CHARACTER ROUTINE - READS A SECTOR IF NECESSARY

C151 108C C400    GETCH   CMPY   #SCTBUF+256 OUT OF DATA?
C155 26   0F              BNE    GETCH4     GO READ CHARACTER IF NOT.
C157 8E   C300    GETCH2  LDX    #SCTBUF    POINT TO BUFFER
C15A EC   84              LDD    0,X        GET FORWARD LINK
C15C 27   0B              BEQ    GO         IF ZERO, FILE IS  LOADED
C15E 8D   0D              BSR    READ       READ NEXT SECTOR
C160 26   9E              BNE    BOOT       START OVER IF ERROR
C162 108E C304            LDY    #SCTBUF+4 POINT PAST LINK
C166 A6   A0      GETCH4  LDA    0,Y+       ELSE, GET A CHARACTER
C168 39                   RTS

                  * FILE IS LOADED, JUMP TO IT

C169 6E   9F C108 GO      JMP    [TADR]     JUMP TO TRANSFER ADDRESS


                  * READ SINGLE SECTOR
                  *
                  * THIS ROUTINE MUST READ THE SECTOR WHOSE TRACK
                  * AND SECTOR ADDRESS ARE IN A ANB B ON ENTRY.
                  * THE DATA FROM THE SECTOR IS TO BE PLACED AT
                  * THE ADDRESS CONTAINED IN X ON ENTRY.
                  * IF ERRORS, A NOT-EQUAL CONDITION SHOULD BE
                  * RETURNED.  THIS ROUTINE WILL HAVE TO DO SEEKS.
                  * A,B,X, AND U MAY BE DESTROYED BY THIS ROUTINE,
                  * BUT Y MUST BE PRESERVED.

                  * WESTERN DIGITAL EQUATES

             0002 DRQ     EQU    2          DRQ BIT MASK
             0001 BUSY    EQU    1          BUSY MASK
             001C RDMSK   EQU    $1C        READ ERROR MASK
```

```
             008C   RDCMND  EQU   $8C       READ COMMAND
             001B   SKCMND  EQU   $1B       SEEK COMMAND

                    * READ ONE SECTOR

C16D 8D    2F       READ    BSR   XSEEK     SEEK TO TRACK
C16F 86    8C               LDA   #RDCMND   SETUP READ SECTOR COMMAND
C171 B7    E018             STA   COMREG    ISSUE READ COMMAND
C174 8D    3E               BSR   DEL28     DELAY
C176 5F                     CLRB            GET SECTOR LENGTH (=256)
C177 BE    C300             LDX   #SCTBUF   POINT TO SECTOR BUFFER
C17A BD    E018     READ3   LDA   COMREG    GET WD STATUS
C17D 85    02               BITA  #DRQ      CHECK FOR DATA
C17F 26    08               BNE   READ5     BRANCH IF DATA PRESENT
C181 85    01               BITA  #BUSY     CHECK IF BUSY
C183 26    F5               BNE   READ3     LOOP IF SO
C185 1F    89               TFR   A,B       SAVE ERROR CONDITION
C187 20    0A               BRA   READ6
C189 B6    E01B     READ5   LDA   DATREG    GET DATA BYTE
C18C A7    80               STA   0,X+      PUT IN MEMORY
C18E 5A                     DECB            DEC THE COUNTER
C18F 26    E9               BNE   READ3     LOOP TIL DONE
C191 8D    03               BSR   XWAIT     WAIT TIL WD IS FINISHED
C193 C5    1C       READ6   BITB  #RDMSK    MASK ERRORS
C195 39                     RTS             RETURN

                    * WAIT FOR 1771 TO FINISH COMMAND

C196 F6    E018     XWAIT   LDB   COMREG    GET WD STATUS
C199 C5    01               BITB  #BUSY     CHECK IF BUSY
C19B 26    F9               BNE   XWAIT     LOOP TIL NOT BUSY
C19D 39                     RTS             RETURN

                    * SEEK THE SPECIFIED TRACK

C19E F7    E01A     XSEEK   STB   SECREG    SET SECTOR
C1A1 B1    E019             CMPA  TRKREG    DIF THAN LAST?
C1A4 27    0E               BEQ   DEL28     EXIT IF NOT
C1A6 B7    E01B             STA   DATREG    SET NEW WD TRACK
C1A9 8D    09               BSR   DEL28     GO DELAY
C1AB 86    1B               LDA   #SKCMND   SETUP SEEK COMMAND
C1AD B7    E018             STA   COMREG    ISSUE SEEK COMMAND
C1B0 8D    02               BSR   DEL28     GO DELAY
C1B2 8D    E2               BSR   XWAIT     WAIT TIL DONE

                     * DELAY

C1B4 BD    C1B7      DEL28  JSR   DEL14
C1B7 BD    C1BA      DEL14  JSR   DEL
C1BA 39              DEL    RTS

                            END   NEWDISK
```