FLEX NEWSLETTER NO. 2
October 1979

P.O. Box 2574, West Lafayette, Indiana 47906

We have some exciting news for you in this issue. In particular, our first 6809 products are already on the market! But more on that later because we've also got news on extended BASIC, tips on using the 6800 Text Processing System, fixes for a couple of utilities, and more.

1) FLEX NEWS

It seems that FLEX is really catching on - almost all 6800 disk systems have the ability of running FLEX. Of course FLEX is available for the SWTPc hardware; we sell a version which runs on the Smoke Signal Hardware; a company called Great Plains Computer Company (P.O. Box 916, Idaho Falls, ID 83401) sells a version for the TANO Outpost II computer; Midwest Scientific Instruments is offering a version for their hardware (their hard-sectored disks cannot be read on other soft-sectored disk systems and our soft-sectored disks cannot be read on their system); SWTPc is selling a conversion kit which allows Percom disk users to upgrade their hardware to the FLEX operating system and Percom is now offering a program which allows a user to read FLEX disks and to convert FLEX 2.0 to run on their hard-sectored disk systems; and a few other companies offer disk controller boards which they claim to be capable of supporting FLEX. This essentially leaves out only two groups... the Motorola Exorcisor system and homebrew or custom disk systems. Well, we're working on them too! There are no completion dates available at this time, but we have work in progress on a version of FLEX which will run with the Exorcisor hardware and another for general use which will allow the user to write his own driver routines for practically any type of soft-sectored floppy disk hardware he may have. We'll be letting you know about these products when they are available.

We have received numerous calls and letters from users of Percom disk systems who want to run FLEX, TSC BASIC, Sort/Merge, and other FLEX-based software. In response, SWTPc and TSC have prepared a conversion kit which will allow Percom disk owners to do just that! This kit is available from Southwest Technical Products, Inc., 219 W. Rhapsody, San Antonio, TX 78216, for $149.95. It contains a controller board (assembled and tested), cable connector, FLEX 2.0, a copy of SWTPc BASIC 3.5, and a full set of manuals. You simply replace the Percom controller board and cable with the SWTPc ones, place 8K of memory at $A000, and you're ready to boot up FLEX! A very versatile yet fool-proof single drive copy routine is also included since most Percom systems contain only one drive.

Another method of running FLEX on Percom disks has just been announced by Percom. They sell a program which will read a soft-sectored FLEX 2.0 disk and copy it to a hard-sectored Percom format disk. This copy method requires two disk drives. Another program is included which will copy FLEX 2.0 itself onto a Percom disk and convert the FLEX drivers to operate with their hard-sectored disk controller. Thus you can run FLEX on the Percom disk system even though it is hard-sectored. Now the FLEX 2.0 disks we sell will not be directly compatible, but the program Percom sells to copy FLEX disks can be used to copy the software over to a standard Percom hard-sectored disk. This new disk can be used with the Percom version of FLEX while the original TSC disk can be kept as an archival copy.

2) 6809 SOFTWARE

It's finally here! The 6809 chip is no longer a thing of the future and neither is 6809 software. We now have in stock six 6809 programs with more coming soon. The six are 6809 FLEX Disk Operating System, Text Editing System, Macro Assembler, Debug Package, 6809 Standard BASIC, and 6809 Extended BASIC. These are all disk based (under the FLEX operating system) for either 5" or 8" SWTPc disk systems. This software will undoubtedly be available for other disk systems as the manufacturers firm up their 6809 plans.

The 6809 FLEX Disk Operating System (FLEX 9.0) is identical to the 6800 version from a user or programmer viewpoint except for the fact that FLEX 9.0 is located at $C000 instead of $A000. In other words, all calls to FLEX are performed in exactly the same manner except that the addresses must be in the $C000 to $DFFF range instead of the $A000 to BFFF range. Thus existing 6800 programs which are FLEX based can be converted to 6809 by simply adding $2000 to all addresses in 6800 FLEX, changing any ORG statements if necessary, and reassembling with the 6809 assembler since it can accept 6800 mnemonics and produce 6809 code. Changing the addresses can be done quite simply in the editor with two instruction:

```
↑C/$A/$C/!*
↑C/$B/$D/!*
```

You must be careful, however, that every place in the source which has a $A or $B is the start of an address and not an eight bit value. Of course this conversion will not take full advantage of the 6809 instruction set, but it is a quick means of getting going in 6809 until you have time to re-write the software. Another advantage to keeping FLEX 9.0 compatible is that the disks are also compatible so that text files prepared under 6800 FLEX can be directly read under 6809 FLEX. On 5" systems this only applies to FLEX 2.0.

The text editor and assembler are included with the FLEX package but may be purchased separately. The editor is an improved version of the ubiquitous TSC 6800 Text Editing System. It is line and content oriented with character string, local, and global instructions. The resident 6809 assembler supports macros and conditional assembly. It accepts 6809, 6800, and 6801 mnemonics so that existing software can be immediately reassembled to produce 6809 object code.

The standard BASIC is a very fast interpreter with 6-digit precision in its binary floating point math package. Features include random access files via record I/O and virtual arrays, unlimited string length, if/then/else construct, TRACE, ON ERROR GOTO, two-dimensional arrays, and a renumber facility. Also supported is a COMPILE command which places an unreadable, compressed form of the source on disk which can only be executed by a RUN command. This permits the distribution of proprietary BASIC programs.

The Extended BASIC has all the features of the standard BASIC plus 17-digit precision on floating point math, complete PRINT-USING facilities, integer variables (indicated by a % suffix like the $ suffix for string variables), INSTRing for finding an occurrence of one string within another, SWAP for swapping the value of two variables, INCH$ to get a single character without waiting for a carriage return, double peek and poke (DPEEK and DPOKE) for 16 bit values, and more.

The debug package is a powerful tool for assembly language program debugging capable of simulating all functions of the 6809 CPU including interrupts and I/O operations. Multiple breakpoints may be user-defined and may be conditional on several criteria. Tracing is possible as is single or multiple stepping. A "traceback" feature allows the listing of the previous 255 executed instructions. Memory protection may be enabled for multiple blocks of the address space. General features include a mini-assembler, disassembler, memory examine and change, hex calculator, and a machine states counter.

Except for FLEX, all code is fully reentrant and position independent. Cassette versions (with restricted features) are available for the editor, assembler, debug, and standard BASIC. Prices are as follows:

6809 FLEX w/ edit. & asmb.	$90.00
Text Editing System	$35.00
6809 Assembler	$40.00
Debug Package	$75.00
Standard BASIC	$65.00
Extended BASIC	$100.00

No source listing is included at these prices. If you wish to order, be sure to specify 8 or 5 inch disks.

3) BASIC News

By now you may have also heard the news of our new Extended BASIC's for 6800 and 6809. As mentioned above under 6809 software, the Extended BASIC features 17-digit precision, PRINT USING, integer variables, and much more. This is a very accurate and still remarkably fast BASIC for any commercial or high precision scientific work. The only drawback is its size - approximately 19K. We recommend at 32K of user memory to make proper use of the software.

A copy of a recent ad for the BASIC is included later in this newsletter. In it you will also notice mention of a "BASIC Precompiler" for both the standard and extended BASIC's. This precompiler allows you to edit BASIC programs (requires an external editor) in a non-standard BASIC form. By non-standard we mean that you can use any length variable names (up to 255 characters), you don't have to put line numbers on lines, where you do need a line number (for GOTO, GOSUB, or other purposes) you can use an alphanumeric label instead of a number if desired. This results in BASIC programs that are much more easily written, read, and modified. The output of the precompiler is the compressed source format that our BASIC's can execute via the RUN command.

4) SOFTWARE PROBLEMS

No matter how hard we try, a few software "bugs" seem to slip by us now and then. When you are developing programs of the length and complexity of some of our latest releases it is almost inevitable. Here are some fixes to problems which have been reported.

6809 ASSEMBLER - Resident disk version
Early versions of the assembler will hang up if you have an illegal mnemonic of over 5 characters in length. The fix is simple: At location $07BF or $07D4 (depending on your version), change the "16 00 7E" to "16 0C 14". Page 57 of the first 6809 assembler manuals shipped describes the steps to manually setup the tables. An important note was left out at the bottom of that page and is printed here:

> NOTE: The assembler uses a large stack area to maintain all its temporary variables and buffers. This stack requires 0720 hex bytes of RAM starting at FLEX's MEMEND and growing down from there. This implies two things: if manually setting up the tables you must leave this space free and if you want to move where the stack resides, you must set MEMEND accordingly as the assembler will always place its stack so that it sets up against MEMEND.

6809 EXTENDED BASIC

Version #1 of 6809 Extended BASIC has a problem with certain mixed arithmetic and string operands. This can be corrected by changing the byte at $21BB from $1E to $1D.

6809 DEBUG PACKAGE

The following errors have been discovered in the 6809 Debug Package. The version currently being shipped has all of the problems corrected.

1) The mini-assembler cannot force long (using ">") on indexed instructions having a zero or 5-bit offset. No patch is available.

2) The disassembler prints a 5-bit offset of -16 as "-$G". No patch available.

3) The "X" command does not preserve the U-register. Patch as follows:

Address	Old value	New value
$596B	$AF	$34
$596C	$C8	$50
$5977	$AE	$35
$5978	$C8	$50

4) Disassembling an indexed instruction with an illegal offset of $F0 causes Debug to hang. Change the byte at $687F from $C6 to $C8.

TEST UTILITY - 6800 FLEX 2.0

The TEST utility will always report an error for track 0, sector 1 and for all sectors of tracks above track $22. Change the "SECTOR FCB 0" at $A104 to "SECTOR FCB 1" and change the " CMP A #76" at $A180 to " CMP A #34".

RECOVER UTILITY - All versions

Replace the " CLR 34,X" at $A14B with " JSR FIX". Then at the end of the code insert the following code:

FLEX 2.0 or 8" FLEX 1.0

```
FIX  CLR   34,X
     LDAA  #$FF
     STAA  #FCB+59
     STAA  #FCB2+59
     RTS
```

miniFLEX version

```
FIX  LDAA  #$80
     STAA  34,X
     LDAA  #$FF
     STAA  #FCB+59
     STAA  #FCB2+59
     RTS
```

5) PRINT.SYS for an ACIA device

We printed some hints on using the "P" command and the "PRINT.SYS" routines in the last newsletter. However, we receive so many calls from people who can't write a PRINT.SYS routine for a serial printer (connected to an ACIA) that we are re-printing a sample of such code here. This code is for 6800 FLEX 2.0 or 8" FLEX 1.0. You will have to modify the addresses if working with 6809 FLEX or with miniFLEX.

Before using the routines below, you will have to set up the proper value for the ACIA output port you wish to use. This is done with an equate statement. For example to output via an ACIA on port 7 you should use:

```
ACIA    EQU    $801C
```

The output character routine (POUT) should look like:

```
        ORG    $ACE4   MUST START AT $ACE4
POUT    PSH B          SAVE B ACC.
POUT2   LDA B  ACIA    GET STATUS
        ASR B          GET TDR BIT
        ASR B          INTO CARRY
        BCC    POUT2   LOOP IF NOT READY
        PUL B          RESTORE B ACC.
        STA A  ACIA+1  WRITE OUT THE CHAR.
        RTS            RETURN
```

The initialization routine (PINIT) should look like:

```
        ORG    $ACC0   MUST START AT $ACC0
PINIT   LDA A  #$13    RESET ACIA
        STA A  ACIA
        LDA A  #$11    SET 8 BITS & 2 STOP
        STA A  ACIA
        RTS            RETURN
```

The printer check routine (PCHK) should be:

```
        ORG    $ACD8   MUST START AT $ACD8
PCHK    PSH B          SAVE B ACC.
        LDA B  ACIA    GET STATUS
        ROR B          GET TDR BIT INTO
        ROR B          SIGN POSITION
        ROR B
        PUL B          RESTORE B ACC.
        RTS            RETURN
```

6) POTPOURRI

Technical Calls

If you have technical questions or problems and feel a need to call TSC, you must do so between 10 and 12 o'clock EST on Monday through Friday. A software technician should always be on call during those hours and those hours only. Chances are about 1 in 100 of getting through at any other time. The major reason for limiting the times we can accept calls is that it is usually a waste of time trying to debug a software problem over the telephone. If you do experience trouble, first check very carefully to be sure it is really a software problem and not a cockpit error. If you find you are using the software properly, next check your hardware. You would be surprised how many times we get reports of problems that we cannot duplicate in-house. Then we begin to tear our hair out looking for the problem only to receive a call from the same person confessing he found a memory problem or a flakey solder joint on his motherboard or that he was not following instructions found in the manual. Even when it is a problem or bug in the software, chances are nil that we can work it out over the phone. You will get much better response by thoroughly documenting the problem and mailing it to us. That way we can make sure it is routed to the correct people and we have a hard copy of your problem on file to compare to others we may receive. So please, don't call with software problems ... document them and mail them in!

Alphabetized Directory Listing

Ted Wolff of New York City wrote to us with a suggestion for obtaining an alphabetized directory listing. Ted says to use the BUILD command to create an EXEC file consisting of the following single line:

```
CAT,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z
```

This file could be called ALPHACAT.TXT or something similar. Now to obtain an alphabetized listing of the directory simply type "EXEC,ALPHACAT". The same procedure could be used for the DIR utility also. Thanks, Ted!

An Improved Command Processor

John Jordan of Oak Ridge, Tennessee, sent us a copy of a program he has written for FLEX called EX. For various reasons we are not able to add the program to our product line, but it is such a well thought-out and well documented (over 35 pages) piece of software that we wanted to let you know about it. If you're interested you might contact John and let him know. If he gets enough response he may duplicate and sell copies. The "EX" program is a command processor which John says is "kind of like a mini job control language for FLEX". To the beginner it could be used much like the EXEC command supplied with FLEX (in fact it is upward compatible with EXEC). The major advantage over EXEC, however, comes in the ability to pass arguments to the EX procedure file from the calling command line. Other options include dynamic variable definition, user prompting, conditional branching, and a trace feature for debugging an

EX file under development. If you are interested, contact John at the following address:

John K. Jordan
103 Elliott Circle
Oak Ridge, TN 37830

There is no guarantee that John will want to distribute the program, and that will probably depend on how much response he gets.

The '68' Micro Journal

Some time ago in our TSC Newsletter we mentioned a new magazine dedicated to the 6800 family of microprocessors. Since that time we have received several issues and find it to be a good investment. It's called the "'68' Micro Journal", is about 50 pages in length (at this time), printed on excellent quality paper, and is always full of nothing but 6800/6809 related programs, reviews, tips, new product releases, and other goodies. Their turn-around time on new product releases has got to be the fastest of any widely distributed microprocessor related monthly in the country. The eight issues printed to date have included numerous FLEX related items. If you want to keep up-to-date on the latest happenings in the world of FLEX and the 6800 family of micros in general, we suggest you check this magazine out. A one year subscription is $14.50 and can be obtained from:

'68' Micro Journal
3018 Hamill Rd.
PO Box 849
Hixson, TN 37343

MiniFLEX Software Discontinued

Technical Systems Consultants will be fading out its MiniFLEX line of support software by the end of the year. MiniFLEX is the original version of FLEX supplied by SWTPc for their MF-68 floppy disk system. Since that time, we came out with a new version of FLEX for the MF-68 called FLEX 2.0. This version is much improved and is compatible with the 8" FLEX which SWTPc distributes and the FLEX which all other systems run (see section #1 of this newsletter). FLEX 2.0 disks are also compatible with 6809 FLEX disks. You can be assured that TSC will continue its support of FLEX 2.0 for a long time.

6809 Cross Assembler on 6800

We are selling a 6809 cross assembler which runs on a 6800. Thus it accepts 6809 instructions (as well as 6800 and 6801) and produces 6809 object code. It has macros and conditional assembly just like our resident 6809 assembler. A manual and object code disk (FLEX 2.0 or 8" FLEX 1.0) sell for $100.00.

7) TIPS ON USING THE TEXT PROCESSING SYSTEM

Until BASIC came out, we probably received more calls and letters on the text processor than any other program we sell. It is an extremely complex piece of software and not something the casual user can expect to master in a couple of days or even weeks. The almost unlimited combinations of commands and macros which can be setup can cause much confusion. Our strongest suggestion to users is to READ THE MANUAL. When you have done that, read it a couple of more times. Then when you have problems, refer to it freely. We feel that manual is one of the best we have written in terms of explaining the use of a complex program in as concise a form as possible. It's quite vexing when a user calls up with a problem whose solution is clearly stated in the manual.

As with any manual, however, there are certain points which could use further clarification. We will go over a few of those points here.

Getting a Header on Page 1

If you have ever run a document with a header macro which was to be executed at the top of each page (via a ".AT 0 HD" type command), you may have noticed that the header did not get executed at the top of the first page of the document. The text processor does this on purpose, since in many applications you want to have a special title at the top of the first page (see the first page of this newsletter for example). If, however, you do want the normal header macro to execute at the top of the first page, it can be easily accomplished. Simply insert a break command (.BR) after the AT command which specifies where the header should occur and before any text which will be output on the first page.

Stopping Output for Paper Change

When working with paper that is non-continuous (ie. separate sheets), it is necessary to stop the output of the text processor at the end of each page such that a new sheet of paper can be inserted into the typewriter. As you have found in the manual, this can be done with a stop command (.ST) which will stop output, waiting for the user to hit any key before continuing. What might be unclear is just where to place the ST command. We have found that the best place is in the header macro and NOT the footer macro. It may seem more logical to stop output in the footer after doing a page eject, but sometimes the text processor will attempt to execute the header macro before it sees the stop. Putting the stop in the header works perfectly. If you do put a stop command in your header, be sure to use a colon instead of a period so that you will not cause a break and the associated buffer flushing. In fact, you should be certain that NO command in the header or footer causes a break.

Diversions

There seems to be a fair amount of confusion about diversions. They are really quite simple. When a diversion is initiated, the text processor merely routes the characters that would normally be going to the printer into the diversion storage area in memory. Thus instead of seeing the output on the printer, it is diverted and stored in memory IN THE EXACT FORMAT THAT WOULD HAVE BEEN PRINTED. This last point is very important. For example, if you divert some text with justification turned on and then read that diversion back with justification turned on, you might get some wierd looking results. In general you should divert with justification off and read the text back in with justification on or vice versa.

Another cause of confusion with diversion is that terminating a diversion with a .DI command does not automatically flush the buffer. As we found earlier, when diverting we are doing normal text processor output, but the characters are sent to the diversion storage area rather than the printer. When performing output of this nature with the fill mode on, there is an output line buffer in which the words read from the input file are placed until the line is full. At that point the buffer is "flushed" or all printed at once. When printing it looks like a continuous stream of output characters because the filling of this buffer is very fast compared to the time it takes to print the line. The same buffer is used when doing diversion with the fill mode turned on. Let's assume we are doing a diversion with two short words. It might look like this:

```
.di nm
John Doe
.di
```

The words "John Doe" would be stuffed into the output line buffer for flushing when filled. But now we end the diversion before the output line buffer is filled. Since no break occurred to cause a flush of the buffer (as seen in the manual, ".DI" does NOT cause a break), nothing gets output to the diversion area. The words John Doe are left in the line buffer.

The simple solution to this is to perform a break before terminating the diversion. This will cause whatever happens to be in the output line buffer to be flushed. In the case of our example, that means "John Doe" will be sent out to the diversion area. The input text would look like:

```
.di nm
John Doe
.br          THIS IS THE KEY!
.di
```

Note that if the fill mode is not on this break is not necessary. That is because in the no-fill mode, the text processor simply reads a line of input into the output line buffer and immediately outputs it without waiting for the buffer to be full.

Numbered Paragraphs

There is often a need to have a set of paragraphs or blocks of text indented and numbered. An example of this format is shown here with nonsensical text for the three paragraphs:

1) This is paragraph number 1. It is simply a couple of meaningless sentences for the purpose of filling up space. As you can see, it is indented and right justified.

2) This is paragraph number 2. It is simply a couple of meaningless sentences for the purpose of filling up space. As you can see, it is indented and right justified.

3) This is paragraph number 3. It is simply a couple of meaningless sentences for the purpose of filling up space. As you can see, it is indented and right justified.

There are several ways to accomplish this format, but let's look at the one we've found to be quite easy. The basic idea is to set an indent as desired for the paragraphs. Then before each paragraph is started we do a single-indent command in a minus direction. For example:

```
.in 10
.sp
.si -3
1)\ This is paragraph number 1.
It is simply a couple of meaningless sentences for
the purpose of filling up space.
As you can see, it is indented and right justified.
```

There is one tricky thing about this sample which deserves our attention. That is the backslash-space combination just after the paragraph number. The backslash makes the space a non-paddable space character so that when padding occurs to justify the line, we can be sure that no additional spaces will be added between the parend and the first character of the first line of the paragraph. This ensures an even column for the left side of the paragraph.

Some of the excess typing involved in doing this sort of layout can be obviated by the use of macros as shown here.

```
.dm bp    (Begin Paragraph macro definition)
.sp
.si -3
..
.in 10
.bp
1)\ This is paragraph number 1.
It is simply a couple of meaningless sentences for
the purpose of filling up space.
As you can see, it is indented and right justified.
.bp
2)\ This is paragraph number 2.
It is simply a couple of meaningless sentences for
the purpose of filling up space.
As you can see, it is indented and right justified.
.bp
3)\ This is paragraph number 3.
It is simply a couple of meaningless sentences for
the purpose of filling up space.
As you can see, it is indented and right justified.
.sp
.in 0
```

Sometimes it may be a name or word which should be in the indent field and not a number-parend as above. If the words are not the same length, you would have to use the correct number of unpaddable space characters (backslash-space combination) to space over from the end of the word to the beginning column of the indent field. An easier method is to use tabs. For example, look at the following layout:

ONE This is paragraph number 1. It is simply a couple of meaningless sentences for the purpose of filling up space. As you can see, it is indented and right justified.

TWO This is paragraph number 2. It is simply a couple of meaningless sentences for the purpose of filling up space. As you can see, it is indented and right justified.

THREE This is paragraph number 3. It is simply a couple of meaningless sentences for the purpose of filling up space. As you can see, it is indented and right justified.

The input text file to produce the above layout looks like this:

```
.dm bp
.sp
.si -#i
..
.tc
.ta T1
```

```
.in 10
.bp
ONE_This is paragraph number 1.
It is simply a couple of meaningless sentences for
the purpose of filling up space.
As you can see, it is indented and right justified.
.bp
TWO_This is paragraph number 2.
It is simply a couple of meaningless sentences for
the purpose of filling up space.
As you can see, it is indented and right justified.
.bp
THREE_This is paragraph number 3.
It is simply a couple of meaningless sentences for
the purpose of filling up space.
As you can see, it is indented and right justified.
.in 0
```

There are two important things to note about this technique. First we used a ".si -#i" for the single ident in the begin paragraph macro. This means to single indent in a minus direction (to the left) by the value found in number register "i". Now if you check the manual you will find that number register "i" contains the current indent amount. So what we are doing is indenting to the left by the same amount that we are currently indenting to the right. This effectively cancels the current indent for that single line or in other words puts us at the left margin.

The second point is to note the apparent difference between the indent amount and the tab column setting. That is because the tab column setting is the actual column number in which to start printing (in this case column number 11) while the indent amount is the number of column positions to indent or skip over. Thus an indent of 10 means we will skip over 10 columns and thus be ready to print in column 11. This is something to watch out for.

Making the ESCAPE Key Work Consistently

You may have noticed that the disk version of the text processor doesn't always stop the output when an ESCAPE or CTRL-C key is hit. This occurs because of a colision between the text processor and FLEX. The text processor is looking for a CTRL-C to stop output while FLEX is looking for an ESCAPE. If you hit an ESCAPE and the text processor happens to look for a CTRL-C before FLEX looks for an ESCAPE, your ESCAPE character will be lost when the processor finds it is not a CTRL-C. The simple solution is to disable one of the two checks and it is easiest to disable the CTRL-C check in the text processor. This simply means you will have to use the ESCAPE key instead of the CTRL-C to stop output. The point to patch in the text processor is the instruction with the label "TSTBRK". This should be at $1594 in the 8" FLEX 1.0 or 5" FLEX 2.0 version or at $1595 in the miniFLEX version. Change this instruction to an RTS ($39).

8) A "DUMP and REPAIR" UTILITY

We've got another FLEX utility for you in this newsletter. It was submitted by some folks who work at the Collins Avionics Department of Rockwell International in Cedar Rapids, Iowa. These people have done a lot of work with FLEX and have generously donated their "Dump and Repair" Utility. As you will see from the documentation, this utility lets you read any sector on the disk by supplying an absolute disk address (track and sector number). If desired the sector can be modified and written back out. This is a very useful utility to have around but is also quite deadly if not used properly! Make sure you know exactly what you are doing when you modify the data in a sector. We are printing the assembled source listing of the FLEX 2.0 version exactly as we received it. It can be modified for 6809 FLEX or miniFLEX if desired. We have run the utility and experienced no problems whatsoever, but we make absolutely no guarantees on its operation and will not support technical calls regarding the utility.

DUMP AND REPAIR UTILITY - DR

The Dump & Repair utility is an interactive disk sector read, display, and modify routine displaying 16 lines each of 16 bytes of data in hex and ASCII. Any data with values between $20 and $7E (printable) are shown as the printable character. Other values are represented as periods. In the Repair mode, changes are made in the RAM sector buffer and only made on the disk with the Write directive. All input is solicited with wrong responses producing a menu of acceptable responses.

DESCRIPTION The syntax of the DR command is simply DR

Valid inputs are: two digit drive, track, and sector numbers

N	next
P	previous
R	repair
W	write
D	return to DOS
L	last
SP	next in file
ESC	restart
RET	return to read-write mode (from repair)

EXAMPLE

```
+++DR
DRIVE? 01
TRACK? 04
SECTOR? 07
0407
00 040800004455 4D502044454D4FE8860B. . . DUMP DEMO. . .
10 2713811626F68DD1B724078DCCB72408'. . . &. . . $. . . $.
20 FE24076E008DC2368DBF33B7240AF724. $. n. . . 6. . 3. $. . $
30 098DB61627D6378DB033FE2409A70008. . . . '. 7. . 3. $. . . .
40 FF24095A26F020C40900FE01000100F7. $. Z&.  . . . . . .
50 00000000000000000000000000000000. . . . . .
60 00FB2202000001200000000000000000. . ". . . . . .
70 00000000000000000000000000000000. . . . . .
80 00000000000000000000000000000000. . . . . .
90 00FB2202000001200000000000000000. . ". . . . . .
A0 00000000000000000000000000000000. . . . . .
B0 00FB2202000001200000000000000000. . ". . . . . .
C0 00000000000000000000000000000000. . . . . .
D0 00000000000000000000000000000000. . . . . .
E0 00FB2202000001200000000000000000. . ". . . . . .
F0 00000000000000000000000000000000. . . . . .
?
```

NOTE: Since this utility allows the user to actually alter the information recorded on the disk, it is imperative that the user have a full understanding of what is being done or an unrecoverable file or a totally destroyed disk may result!

```
                        *
0001                    VERNO   EQU     1           VERSION NUMBER
                        *
                        * THIS IS THE INTERACTIVE DISK DUMP & REPAIR PROGRAM
                        * FOR FLEX(TM) 2.X  INSTRUCTIONS
                        * FOR USE ARE COVERED IN DETAIL IN A SEPERATE DOCUMENT.
                        *
                        * BASIC FUNCTION IS: READ A SINGLE SECTOR INTO A
                        * BUFFER, DISPLAY THE BUFFER IN HEX AND ASCII, ALTER
                        * THE BUFFER DATA (LIKE MEMORY EXAMINE/CHANGE),
                        * REDISPLAY THE BUFFER, AND THEN (IF DESIRED),
                        * REWRITE THE SINGLE SECTOR.  * USER BEWARE! *
                        *
                        * WRITTEN BY R.J.TONNESON - ROCKWELL INTERNATIONAL
                        * CEDAR RAPIDS, IOWA  52406
                        *
                        * UNCONDITIONAL PERMISSION FOR NON COMERCIAL USE IS
                        * HEREBY GIVEN TO THE FLEX NEWSLETTER AND USER GROUP.
                        *
                        * EXTERNAL REFERENCES - DOS
                        *
AC14                    INBFPT  EQU     $AC14       LINE BUFFER POINTER
AD03                    DOS     EQU     $AD03       WARM START ENTRY POINT
AD15                    GETCHR  EQU     $AD15       GET 1 CHR FROM KBD, RET.  IT IN A
AD1B                    GETLIN  EQU     $AD1B       GETS A COMMAND LINE
AD24                    DCRLF   EQU     $AD24       PRINTS CRLF, KEEPS TTYSET HAPPY
AD0F                    OUTFLX  EQU     $AD0F       OUTPUTS CHAR IN A TO CONSOLE
AD3F                    RPTERR  EQU     $AD3F       REPORT DISK ERROR #NN
B406                    FMS     EQU     $B406       MAIN ENTRY POINT
                        *
                        * EXTERNAL REFERENCES - MONITOR
                        *
E1D1                    OUTEEE  EQU     $E1D1       OUTPUTS CHAR IN A TO CONSOLE
E07E                    PDATA1  EQU     $E07E       OUTPUTS STRING TO CONSOLE
                        *
000F                    SIZE    EQU     15          BYTES DISPLAYED/LINE
                        *
6000                            ORG     $6000
                        *
6000 20 01              START   BRA     START1
6002 01                         FCB     VERNO
                        *
                        * SOLICIT DRIVE, TRACK, SECTOR, THEN HAVE AT IT
                        *
6003 CE 60 00           START1  LDX     #START
6006 FF 63 7E                   STX     ERRORX
6009 CE 62 39                   LDX     #DRVMSG
600C BD E0 7E                   JSR     PDATA1
600F BD AD 1B                   JSR     GETLIN
6012 BD 61 4E                   JSR     INBYTE
6015 B7 63 83                   STA A   FCB+3       DRIVE NUMBER
6018 CE 60 18           DSP05   LDX     #DSP05
601B FF 63 7E                   STX     ERRORX
```

```
601E CE 62 44            LDX    #TRKMSG
6021 BD E0 7E            JSR    PDATA1
6024 BD AD 1B            JSR    GETLIN
6027 BD 61 4E            JSR    INBYTE
602A B7 63 9E            STA A  FCB+30     CURRENT TRACK POSITION
602D CE 60 2D  DSP10     LDX    #DSP10
6030 FF 63 7E            STX    ERRORX
6033 CE 62 4F            LDX    #SCTMSG
6036 BD E0 7E            JSR    PDATA1
6039 BD AD 1B            JSR    GETLIN
603C BD 61 4E            JSR    INBYTE
603F B7 63 9F            STA A  FCB+31     CURRENT SECTOR POSITION
                 *
                 * CALL FMS FOR SINGLE SECTOR READ
                 *
6042 CE 63 80  DSP30     LDX    #FCB
6045 86 09               LDA A  #9
6047 A7 00               STA A  0,X
6049 BD B4 06            JSR    FMS
604C 26 39               BNE    DERROR
604E BD 61 7B  DSP40     JSR    DSPSCT
                 * PROMPT THEN GET NEXT TASK
6051 BD 62 29  DSP50     JSR    CRLF
6054 86 3F               LDA A  #'?
6056 BD 62 26            JSR    OUTCH
6059 BD AD 15            JSR    GETCHR
605C 81 4E               CMP A  #'N
605E 27 35               BEQ    NEXT
6060 81 50               CMP A  #'P
6062 27 46               BEQ    PREV
6064 81 52               CMP A  #'R
6066 27 53               BEQ    REPAIR
6068 81 1B               CMP A  #$1B
606A 27 94               BEQ    START
606C 81 20               CMP A  #$20
606E 27 1D               BEQ    NEXTF
6070 81 44               CMP A  #'D
6072 27 33               BEQ    DOSEX
6074 81 4C               CMP A  #'L
6076 27 D6               BEQ    DSP40
6078 81 57               CMP A  #'W
607A 27 0E               BEQ    WRIT1
607C CE 62 5B            LDX    #INVMSG
607F BD E0 7E            JSR    PDATA1
6082 BD 62 29            JSR    CRLF
6085 20 CA               BRA    DSP50
                 *
6087 7E 61 39  DERROR    JMP    SAYERR
608A 7E 61 3F  WRIT1     JMP    WRITE
                 * PREP TO READ THE NEXT SECTOR IN THE FILE
608D FE 63 C0  NEXTF     LDX    FCB+64     BEGINING OF DATA
6090 FF 63 9E            STX    FCB+30
6093 20 AD               BRA    DSP30
                 * PREP TO READ THE NEXT LOGICAL SECTOR
```

```
6095 B6 63 9F  NEXT    LDA A  FCB+31
6098 4C                INC A
6099 81 11             CMP A  #$11
609B 26 05             BNE    NEXT1
609D 7C 63 9E          INC    FCB+30
60A0 86 01             LDA A  #$1
60A2 B7 63 9F  NEXT1   STA A  FCB+31
60A5 20 9B             BRA    DSP30
                       *
60A7 7E AD 03  DOSEX   JMP    DOS
                       * PREP TO READ THE PREVIOUS LOGICAL SECTOR
60AA B6 63 9F  PREV    LDA A  FCB+31
60AD 4A                DEC A
60AE 26 05             BNE    PREV1
60B0 7A 63 9E          DEC    FCB+30
60B3 86 10             LDA A  #$10
60B5 B7 63 9F  PREV1   STA A  FCB+31
60B8 7E 60 42          JMP    DSP30
                       * ALTER RAM SECTOR - GET STARTING LOCATION
60BB CE 60 BB  REPAIR  LDX    #REPAIR
60BE FF 63 7E          STX    ERRORX
60C1 CE 62 EC          LDX    #OFFMSG
60C4 BD E0 7E          JSR    PDATA1
60C7 BD AD 1B          JSR    GETLIN
60CA BD 61 4E          JSR    INBYTE
60CD CE 63 C0          LDX    #FCB+64
60D0 FF 63 7C          STX    SAVEX
60D3 B7 63 74          STA A  SAVEA
60D6 5F                CLR B
60D7 BB 63 7D          ADD A  SAVEX+1
60DA F9 63 7C          ADC B  SAVEX
60DD F7 63 7C          STA B  SAVEX
60E0 B7 63 7D          STA A  SAVEX+1
                       * DISPLAY PRESENT DATA - GET NEXT TASK
60E3 BD 62 29  REP1    JSR    CRLF
60E6 CE 63 74          LDX    #SAVEA
60E9 BD 62 12          JSR    OUT2HS
60EC FE 63 7C          LDX    SAVEX
60EF BD 62 12          JSR    OUT2HS
60F2 BD AD 15          JSR    GETCHR
60F5 81 4E             CMP A  #'N
60F7 27 25             BEQ    INC
60F9 81 50             CMP A  #'P
60FB 27 2D             BEQ    LAST
60FD 81 0D             CMP A  #$D
60FF 27 32             BEQ    DONE
6101 81 20             CMP A  #$20
6103 27 08             BEQ    REP3
6105 CE 62 F8          LDX    #REPMSG
6108 BD E0 7E          JSR    PDATA1
610B 20 D6             BRA    REP1
                       * CHANGE THE DATA AND INCREMENT THE LOCATION
610D CE 61 0D  REP3    LDX    #REP3
6110 FF 63 7E          STX    ERRORX
```

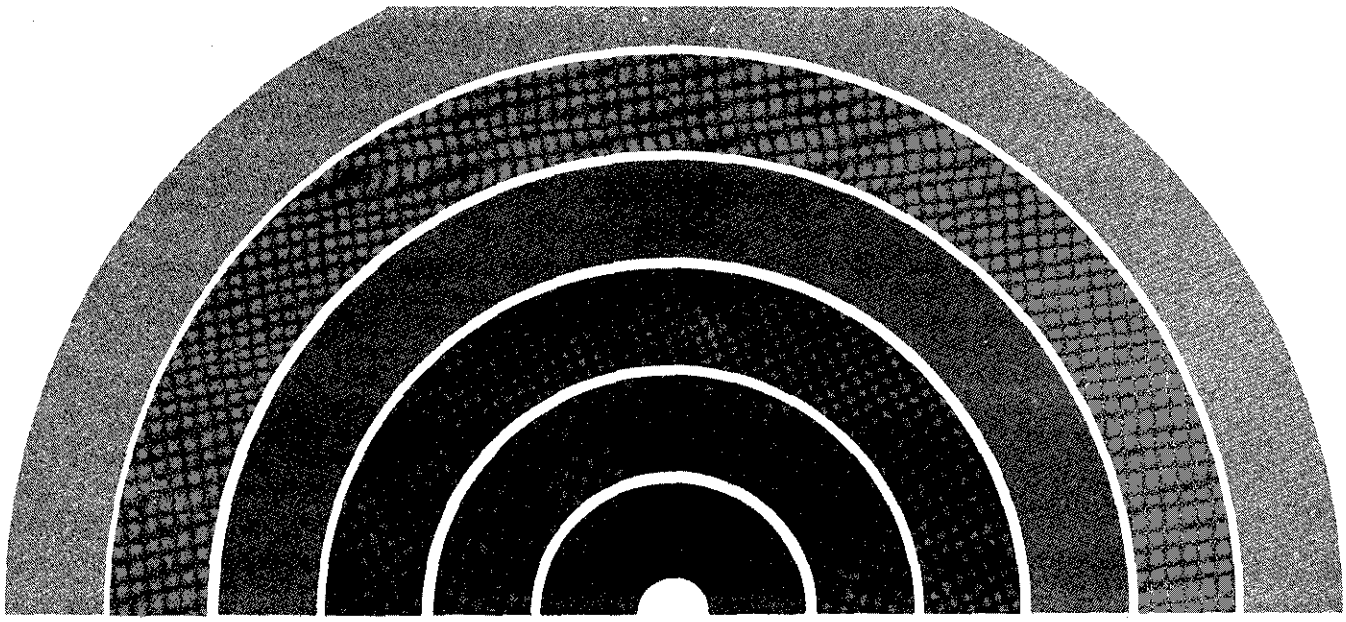
```
6113 BD AD 1B           JSR    GETLIN
6116 BD 61 4E           JSR    INBYTE
6119 FE 63 7C           LDX    SAVEX
611C A7 00              STA A  0,X
611E FE 63 7C  INC      LDX    SAVEX
6121 08                 INX
6122 7C 63 74           INC    SAVEA
6125 FF 63 7C  REP2     STX    SAVEX
6128 20 B9              BRA    REP1
                * BACK UP ONE LOCATION
612A FE 63 7C  LAST     LDX    SAVEX
612D 09                 DEX
612E 7A 63 74           DEC    SAVEA
6131 20 F2              BRA    REP2
                * NO MORE CHANGE
6133 BD 62 29  DONE     JSR    CRLF
6136 7E 60 4E           JMP    DSP40
                *
6139 BD AD 3F  SAVERR   JSR    RPTERR
613C 7E 60 51           JMP    DSP50
                * CALL FMS TO WRITE A SINGLE SECTOR
613F CE 63 80  WRITE    LDX    #FCB
6142 86 0A              LDA A  #10
6144 A7 00              STA A  0,X
6146 BD B4 06           JSR    FMS
6149 26 EE              BNE    SAVERR
614B 7E 60 51           JMP    DSP50
                * GET TWO HEX DIGITS AND FORM A BYTE
614E BD 61 5A  INBYTE   JSR    INHEX
6151 48                 ASL A
6152 48                 ASL A
6153 48                 ASL A
6154 48                 ASL A
6155 16                 TAB
6156 8D 02              BSR    INHEX
6158 1B                 ABA
6159 39                 RTS
                * GET HEX (ONLY) CHARACTER FROM THE CONSOLE
615A BD 62 2F  INHEX    JSR    GETCRT
615D 80 30              SUB A  #$30
615F 2B 0F              BMI    HEXERR
6161 81 09              CMP A  #$09
6163 2F 0A              BLE    GOTHEX
6165 81 11              CMP A  #$11
6167 2B 07              BMI    HEXERR
6169 81 16              CMP A  #$16
616B 2E 03              BGT    HEXERR
616D 80 07              SUB A  #7
616F 39         GOTHEX  RTS
6170 CE 62 DB  HEXERR   LDX    #NHXMSG
6173 BD E0 7E           JSR    PDATA1
6176 FE 63 7E           LDX    ERRORX
6179 6E 00              JMP    0,X
                * DISPLAY THE CURRENT SECTOR BUFFER
```

```
617B BD AD 24  DSPSCT  JSR    DCRLF
617E CE 63 9E          LDX    #FCB+30
6181 BD 61 EF          JSR    OUT4HD
6184 4F                CLR A
6185 B7 63 76          STA A  XBYTE1
6188 B7 63 77          STA A  XBYTE2
618B CE 63 C0          LDX    #FCB+64    BEGINING OF DATA
618E FF 63 78          STX    BFPTR1
6191 FF 63 7A          STX    BFPTR2
6194 BD 61 A0  DSP1    JSR    DOLINE     PROC THE DATA & TEST FOR DONE
6197 B6 63 76          LDA A  XBYTE1
619A 26 F8             BNE    DSP1       < 255?
619C BD AD 24          JSR    DCRLF
619F 39                RTS
               * PROCESS ONE LINE OF THE BUFFER
61A0 BD AD 24  DOLINE  JSR    DCRLF
61A3 CE 63 76          LDX    #XBYTE1
61A6 86 20             LDA A  #$20
61A8 BD 61 F1          JSR    OUT2DD
61AB BD 62 05          JSR    OUTCHD
61AE FE 63 78  DOLIN2  LDX    BFPTR1     DO IT IN HEX
61B1 A6 00             LDA A  0,X
61B3 BD 61 F7          JSR    OUTHLD
61B6 A6 00             LDA A  0,X
61B8 BD 61 FB          JSR    OUTHRD
61BB 08                INX
61BC FF 63 78          STX    BFPTR1
61BF 7C 63 76          INC    XBYTE1     COUNTER
61C2 B6 63 76          LDA A  XBYTE1
61C5 84 0F             AND A  #SIZE
61C7 26 E5             BNE    DOLIN2
61C9 FE 63 7A  DOLIN3  LDX    BFPTR2     NOW IN ASCII
61CC A6 00             LDA A  0,X
61CE 81 1F             CMP A  #$1F       IF NON PRINTING
61D0 2E 02             BGT    DOLIN4
61D2 86 2E             LDA A  #'.        SUBSTITUT A PERIOD
61D4 BD 62 05  DOLIN4  JSR    OUTCHD
61D7 08                INX
61D8 FF 63 7A          STX    BFPTR2
61DB 7C 63 77          INC    XBYTE2
61DE B6 63 77          LDA A  XBYTE2
61E1 84 0F             AND A  #SIZE
61E3 26 E4             BNE    DOLIN3
61E5 39                RTS
               * OFTEN COPIED ROUTINES TO OUTPUT HEX DATA
               * THESE GO THRU DOS FOR USE WITH P.CMD
61E6 A6 00     OUT2HD  LDA A  0,X
61E8 8D 0D             BSR    OUTHLD
61EA A6 00             LDA A  0,X
61EC 08                INX
61ED 20 0C             BRA    OUTHRD
61EF 8D F5     OUT4HD  BSR    OUT2HD
61F1 8D F3     OUT2DD  BSR    OUT2HD
61F3 86 20     OUTSD   LDA A  #$20
```

```
61F5 20 0E              BRA     OUTCHD
                        *
61F7 44         OUTHLD  LSR A
61F8 44                 LSR A
61F9 44                 LSR A
61FA 44                 LSR A
61FB 84 0F      OUTHRD  AND A   #$F
61FD 8B 30              ADD A   #$30
61FF 81 39              CMP A   #$39
6201 23 02              BLS     OUTCHD
6203 8B 07              ADD A   #$7
6205 7E AD 0F   OUTCHD  JMP     OUTFLX
                        * THESE ARE IDENTICAL BUT KEEP THE CHIT-CHAT
                        * OFF THE PRINTER.
                        * COULD USE OUTPUT SWITCH
6208 A6 00      OUT2H   LDA A   0,X
620A 8D 0C              BSR     OUTHL
620C A6 00              LDA A   0,X
620E 20 0C              BRA     OUTHR
6210 8D F6      OUT4HS  BSR     OUT2H
6212 8D F4      OUT2HS  BSR     OUT2H
6214 86 20      OUTS    LDA A   #$20
6216 20 0E              BRA     OUTCH
                        *
6218 44         OUTHL   LSR A
6219 44                 LSR A
621A 44                 LSR A
621B 44                 LSR A
621C 84 0F      OUTHR   AND A   #$F
621E 8B 30              ADD A   #$30
6220 81 39              CMP A   #$39
6222 23 02              BLS     OUTCH
6224 8B 07              ADD A   #$7
6226 7E E1 D1   OUTCH   JMP     OUTEEE
                        *
6229 CE 63 70   CRLF    LDX     #CRLFMS
622C 7E E0 7E           JMP     PDATA1
                        *
622F FE AC 14   GETCRT  LDX     INBFPT
6232 A6 00              LDA A   0,X
6234 08                 INX
6235 FF AC 14           STX     INBFPT
6238 39                 RTS
                        * STRINGS
6239 0D         DRVMSG  FCB     $D,$A,0
623C 44                 FCC     /DRIVE? /
6243 04                 FCB     4
6244 0D         TRKMSG  FCB     $D,$A,0
6247 54                 FCC     /TRACK? /
624E 04                 FCB     4
624F 0D         SCTMSG  FCB     $D,$A,0
6252 53                 FCC     /SECTOR? /
625A 04                 FCB     4
625B 0D         INVMSG  FCB     $D,$A,0
```

```
625E 56                FCC     /VALID INPUTS ARE/
626E 0D                FCB     $D,$A,0
6271 4E                FCC     /N - NEXT/
6279 0D                FCB     $D,$A,0
627C 50                FCC     /P - PREVIOUS/
6288 0D                FCB     $D,$A,0
628B 52                FCC     /R - REPAIR/
6295 0D                FCB     $D,$A,0
6298 57                FCC     /W - WRITE/
62A1 0D                FCB     $D,$A,0
62A4 44                FCC     /D - DOS/
62AB 0D                FCB     $D,$A,0
62AE 4C                FCC     /L - LAST/
62B6 0D                FCB     $D,$A,0
62B9 53                FCC     /SP - NEXT IN FILE/
62CA 0D                FCB     $D,$A,0
62CD 45                FCC     /ESC - RESTART/
62DA 04                FCB     4
62DB 0D        NHXMSG  FCB     $D,$A,0
62DE 49                FCC     /INPUT NOT NEX/
62EB 04                FCB     4
62EC 0D        OFFMSG  FCB     $D,$A,0
62EF 4F                FCC     /OFFSET? /
62F7 04                FCB     4
62F8 0D        REPMSG  FCB     $D,$A,0
62FB 56                FCC     /VALID INPUTS ARE/
630B 0D                FCB     $D,$A,0
630E 4E                FCC     /N - NEXT/
6316 0D                FCB     $D,$A,0
6319 50                FCC     /P - PREVIOUS/
6325 0D                FCB     $D,$A,0
6328 53                FCC     /SP - CHANGE(FOLLOWED BY TWO HEX CHAR)/
634D 0D                FCB     $D,$A,0
6350 52                FCC     /RET - RETURN TO READ-WRITE MODE/
636F 04                FCB     4
6370 0D        CRLFMS  FCB     $D,$A,0,4
               *
6374           SAVEA   RMB     1
6375           SAVEB   RMB     1
6376           XBYTE1  RMB     1
6377           XBYTE2  RMB     1
6378           BFPTR1  RMB     2
637A           BFPTR2  RMB     2
637C           SAVEX   RMB     2
637E           ERRORX  RMB     2
6380           FCB     RMB     320
                       END     START
```

NO ERROR(S) DETECTED



Something New on the Horizon from Technical Systems Consultants

Extended BASIC for 6800 and 6809

Finally, a BASIC for serious business applications or scientific programming is available. All the features of our regular BASIC are supported—and more. Floating point calculations are carried out to an internal accuracy of 17 digits. Most math functions are accurate to 16 digits with a minimum accuracy of 13.5 digits. Integer variables have been included to allow fast execution of control loops and array indexing. Even with the double precision math package, this BASIC is still one of the fastest around.

The business programmer will appreciate the versatile PRINT-USING capabilities which include dollar and asterisk fill, trailing minus sign, imbedded commas, and scientific notation. New string functions have been added for string searching (INSTR) and for creating a string which is the date (DATES$). DPEEK and DPOKE are 16-bit peek and poke type functions. The SCALE command has been included to eliminate the round-off errors typically encountered in binary math packages. The INCH$ function allows single-character input from the terminal. Programmer control of control C breaks is also included.

Overall, the Extended BASIC is the most complete BASIC offered for micro users and is only available on FLEX™ disk. A system with at least 32K of user space is recommended. Specify 8" or 5" media (5" 6800 is FLEX™ 2.0) and either the 6800 or 6809 version when ordering.

AP68-12	**6800 Extended BASIC**	**$100**
SP09-6	**6809 Extended BASIC**	**$100**

BASIC Precompiler

This program allows the creation of BASIC programs without the use of line numbers or restrictive two-character variable names. Alphanumeric line and subroutine labels may be used, as well as variable names of any length. Comment lines are marked with non-alphanumerics for easy readability. The output of the precompiler is in the standard BASIC compiled form. This allows applications programs to be written, precompiled, and then distributed in a non-source form. The precompiler can only be used with one of Technical Systems Consultants' BASICs. Specify 8" or 5" (5" 6800 is FLEX™ 2.0) when ordering.

AP68-13	**Single Precision 6800 Precompiler**	**$40**
AP68-14	**Double Precision 6800 Precompiler**	**$50**
SP09-7	**Single Precision 6809 Precompiler**	**$40**
SP09-8	**Double Precision 6809 Precompiler**	**$50**

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

technical systems consultants, inc.

Box 2570, West Lafayette, IN 47906
(317) 463-2502