FLEX™ NEWSLETTER NO. 5
September 1981

Here's issue number 5! We are quite late with this issue, but we've got lots of good material for you. Thanks for your patience. It might be well to remind you to check the number in the upper right-hand corner of the mailing label on this newsletter. That number is the last issue number you will receive under your current subscription. If it is a "5", you will have to re-subscribe in order to continue receiving the FLEX Newsletter. No subscription renewal notices will be mailed.

1) FLEX™ News

Our base of FLEX users continues to grow. We have experienced phenomenal sales of FLEX and related software outside the U.S. in the past several months. Several of our foreign dealers are really pushing FLEX. SMT of France licensed 6800 FLEX quite some time ago for a personal computer system they were developing called the "Goupil". They recently started delivery of the system and sent us some copies of their sales literature. They have a broad range of configurations ranging from a small cassette based system to a 20 Megabyte hard disk system. Another new licensee of 6809 FLEX is Adtek of Japan. They manufacture and distribute a 6800 and 6809 system aimed primarily at industry and system development. They have translated the FLEX manual set into Japanese.

Speaking of Japan, we see a large potential market there for the 68XX family of microprocessors. Many of you are probably already aware of the Hitachi MB-6890 personal computer based on the 6809 which is already being marketed there. Fujitsu is readying a 6809 based personal computer called the Micro 8. Both these systems have color graphics capabilities. Neither are marketed in the U.S. at this time, but the possibilities for such are very strong. The FLEX Operating System will most likely be available for both these machines.

Another version of 6809 FLEX will be made available soon to users of the SWTPc S/09 system who have their hardware setup to run UniFLEX. Currently, those users must power the system down, remove the UniBUG ROM, insert an S-BUG ROM, and reconfigure the DMF-2 disk controller card. The new FLEX will boot directly on the UniFLEX hardware configuration. Details will be sent to UniFLEX licensees.

2) New Products

Two new products to announce this time: 6809 Pascal and a 68000 Cross Assembler which runs under 6809 FLEX. For further information, see the descriptions printed at the end of this newsletter. Both are available now, on either 8 or 5 inch disks. One point should be made regarding Pascal on 5 inch disks. The compiler is very large and consequently must be supplied on two single-sided, single-density diskettes. If your system supports double-sided, double-density, or double track density diskettes, you can simply copy the programs over to a higher density disk and have no problems. If you are running single-sided, single-density only, there are some special considerations. First, you will have to have two drives. Second, the programs which comprise the compiler portion of the Pascal almost completely fill a single-sided, single-density disk. This implies that to perform a compilation, you must place your source file on a disk in drive 1, remove your normal system disk from drive 0, and insert the Pascal compile disk, before typing the compile command. When the compilation is complete, you must switch back to your system disk in order to run the program you just compiled. Remember, these considerations are not a problem on 8 inch disks or on higher density 5 inch disks.

Three other 6809 FLEX software packages are nearing completion and should be available in the coming two or three months. First is a 6809 Relocating Assembler and Linking Loader package. This is a product we have been lacking for a long time, but will soon have. Second is a Fortran 77 compiler. This project has been "under wraps" for quite a while, but we now feel it's far enough along to let you know about it. This is an ANSI Fortran 77 compiler and will be available for FLEX and UniFLEX. The third product forthcoming is our C Programming Language compiler. This compiler has been delayed several times, but has been given high priority of late. We'll let you know when these products are ready to go.

3) UniFLEX™ News

We know this newsletter is called the "FLEX Newsletter", but many of you subscribers are either current or potential UniFLEX users and are interested in its status. For those of you who don't know, UniFLEX is our multi-user, multi-tasking operating system. The 6809 version of UniFLEX has been in the field for a full year now and it is doing beautifully. We must admit there were a few bugs in the early versions, as in any operating system of this complexity, but these have all been repaired and the system now seems to be solid as a rock. We recently introduced a version which operates on all Gimix hardware (including their new DMA disk controller). The previous version was/is for the SWTPc S/09 system, of course. Our line of support software now includes: BASIC, BASIC Precompiler, Pascal, Text Processor, Sort/Merge, and a 68000 Cross Assembler. Soon to come is an Enhanced Printer Spooler Package. The new products coming for FLEX (Relocating Assembler, Linking Loader, Fortran 77, and C) will also be available

under UniFLEX.

We are very proud of the UniFLEX package. Very few 16-bit micro operating systems are as good, and you won't find any 8-bit operating system that is a match for UniFLEX. If you read any computer magazines, you have undoubtedly seen a lot of excitement over the UNIX™ (trademark of Bell Labs) operating system and the various versions of it (such as Microsoft's XENIX™). Well, 6809 UniFLEX is almost identical to that system, is available now, and runs on less expensive 8-bit hardware. And we dare you to find an operating system with the features of UniFLEX that even comes close to it in cost. OK... enough bragging. We'll leave it up to you to see for yourself.

4) Current Versions

Due to the high demand for publication of current software version numbers, we'll probably make this a regular section in the FLEX Newsletter. Our update policy for FLEX software is as follows: If you have owned a package for under two months an update is free. Beyond two months there is a $10.00 updating fee. To obtain an update you must supply proof of purchase and return the original disk, or supply proof of purchase and an additional $10.00 for us to supply a new disk. The following version numbers are current as of September 1, 1981.

Program Name	6809 Version	6800 Version
6809 Pascal	3	-
Extended BASIC	19	17
Extended BASIC Precompiler	4	2
BASIC	15	13
BASIC Precompiler	3	2
Text Editing System	2	n/a
Assembler	2	n/a
Text Processing System	4	n/a
Sort/Merge	3	3
Debug	17	n/a
6809 Cross Assembler	-	2
68000 Cross Assembler	1	-

5) FLEX Based Products from Other Firms

We love to see outside support of the FLEX Disk Operating System. That's what makes it such a useful system. It is impossible for Technical Systems Consultants Inc. to write all the software necessary for any application. We provide all the systems level tools that we can, but must rely on other firms to publish application software. One such firm is selling a whole line of FLEX application software. If you haven't heard of them, you haven't been paying attention to the 68XX industry. The company is called "Frank Hogg Laboratory, Inc." and specializes strictly in support software for the 68XX family of

processors, mostly under FLEX. Their lead product is a data base management system that is called "Dataman". It is written in 6800 or 6809 Extended BASIC under FLEX. An add-on package, "Datarand", allows the system to access the database randomly, providing much faster access to large databases. Frank Hogg Laboratory is also selling a 6809 Pascal compiler from Dynasoft, a FLEX compatible FORTH called "X-FORTH", a job control program, disassembler, cross assemblers, extended utilities, and lots more. Most programs include source code, and this outfit is dedicated to providing execellent support to its customers. You can check them out at:

Frank Hogg Laboratory, Inc.
130 Midtown Plaza
Syracuse, NY 13210
Phone: (315) 474-7856

6) FLEX™ Tips

Here's a quick tip and some information that may save you some time.

A) Supplying Parameters on MON Command Line

Often it may be desirable to supply some parameters on the command line with the "MON" command such that even though you had exited FLEX to your system's ROM monitor, the parameters would still be available in the input buffer. This can certainly be done and two specific examples will demonstrate just how.

First, let's assume the user wished to run the assembler, but wanted to try a patch in it before it ran. Of course there are many ways to do this, but most would require you to write a modified assembler copy to the disk and run it. In our case, we don't want to have to write a modified copy to the disk (this is the only time we'll run this patched assembler). The ideal situation would be to load the assembler into memory, exit FLEX with the "MON" command, make the necessary patches to the assembler code in memory with the system monitor, and then jump to the starting location in the assembler. There is just one problem with this procedure: the assembler will try to get the filename to be assembled and options from the command line, and they won't be there. The solution? It's simple. When you type the "MON" command to exit FLEX, also type the necessary parameters on that command as if you had typed "ASMB" instead of "MON". For example, if the file to be assembled was called "COPIER" and we wanted no listing or symbol table, the command line typed to exit FLEX before patching the assembler might look like:

```
+++MON COPIER +LS
```

After patching the assembler and then jumping to its starting location the assembler would look for the filename and options in

the FLEX input buffer and sure enough, they will be there!

Another common case arises where a user wishes to copy a file from one disk to another but does not have the COPY command resident on either disk. Now the solution here is a little tricky, but can be used effectively. Let's assume disk A has a file called "INVNTRY.DAT" which we wish to copy to disk B. Further assume that "INVNTRY.DAT" is so large that it uses every available sector on the disk. If we only have 2 drives in the system, there is a problem: neither disk can have room to hold the COPY command. Here's the solution. First, put your normal system disk (which contains a copy of the file "COPY.CMD") in drive 0. Second, get the COPY command into memory but without executing it by typing the command:

```
+++GET 0.COPY.CMD
```

Third, exit FLEX with the MON command, but put the desired parameters after "MON" which would have been on a normal COPY command line:

```
+++MON 1.INVNTRY.DAT 0
```

You should now be in your system's ROM monitor. Fourth, remove your system disk from drive 0, put disk A in drive 1 and disk B (a freshly formatted, empty disk) in drive 0. Fifth, use the jump command in your ROM monitor to jump to the starting location of the COPY command which was loaded earlier. For 6809 FLEX that location is $C100; for 6800 FLEX it is $A100. When the copy is complete, you may remove disk B from drive 0 and re-insert the system disk.

You should exercise a fair amount of caution when performing the above operations. It is not normally advised to remove or swap diskettes at any time other than when the FLEX prompt is displayed, but with caution the above operations produce no problems and can be extremely helpful. One final point. In order to find the starting address of a program (such as the assembler or copy command), you should use the "MAP" command included in the additional utility set available for FLEX.

B) Integers vs. Floating Point in Extended BASIC

In Extended BASIC the user may use either integer variables/constants or floating point variables/constants. A great deal of confusion has arisen from the use of integers and the mixing of integers and floating point. Integer arithmetic will always work in whole numbers; no fractions are used as in floating point. Therefore, the results of integer arithmetic may be different than expected. For example, one may assume that 5/2 would yield 2.5; however, in integer arithmetic the .5 is truncated to leave only the whole part, 2. This means that

```
5/2 * 2  is  4.
```

The answer is 4 because the 5/2 is 2 (not 2.5 in integer arithmetic!) and 2*2 is 4! The expression 10/3 is equal to 3 (not 3.333333...); however, 10./3. is 3.333333... This brings into the discussion the mixing of floating point and integer variables/constants. The computer calculates the result of arithmetic expressions by grouping the operands and operators into groups of three, based on the precedence of the operator, moving from left to right. That is, in A*B+C/D the A*B is calculated first, C/D second, and the addition of the two sub-results last. The type (integer or floating point) of the two operands (or sub-results) determines the type of the result. If one of the operands is floating point, then the result of the operation will be floating point. For example:

5./2 + 5/2 is 4.5.

Because 5./2 is 2.5, and 5/2 is 2, the final answer is 4.5. If using values with only whole numbers, the integer type will be faster and requires less space (only 2 bytes). The user is advised to experiment with the differences in floating point and integer arithmetic and the mixing of the two.

7) Problem Reporting

The number of technical calls we receive at Technical Systems Consultants, Inc. is staggering. You would be amazed at the percentage of time our technical staff spends answering technical phone calls. Many of these calls are justified, many are not. Most could probably be avoided. To relieve some of the burden from our technical programming staff, we have limited the time in which problematic technical calls will be accepted. This time period is between the hours of 10 and 12 am West Lafayette, Indiana time. If you are experiencing difficulties with a FLEX program, we recommend the following procedure:

1) Do everything possible to convince yourself that the problem is in the software. Keep in mind that problems can be in your hardware and/or operator induced. Don't always immediately blame the software. PLEASE, read the manual carefully about the area from which the problem seems to be arising.

2) If convinced the problem is in the software, attempt to produce the smallest test case possible which will generate the error. This is extremely helpful in locating the problem. For example, if the problem is arising in a 400-line BASIC program, try to delete any portion of the program that you can and still repeatedly obtain the error. A 10-line BASIC program which fails in the same way as the 400-line program is much easier to debug.

3) Determine the version number of the software in question. Run the "VERSION" command on the program to determine which version you have. If we are to help you with a program, we must know exactly which program it is. Along the same lines, when making a problem

report, always specify the program as fully as possible. For example, "a problem in BASIC" tells us nothing. What we need to hear is "a problem in version 16 of 6809 Extended BASIC on 8 inch FLEX". You would make things much, much easier for our technical staff if you would conform to the latter.

4) At this point you have two options. One is to call, the other is to write. Often times a call may have to be followed up by writing anyway. If we know about a problem when you call, we may be able to immediately help you over the phone. Chances are, however, problem reports will have to be mailed to us so that we can reproduce the problem here. Also, when you call, there may not be anyone available who can help with your specific problem. When you write, on the other hand, we can always route the problem report to the correct individual for efficient processing. In either case, supply as much information about your problem as possible, with at least the complete program description and version number, a description of your system, a description of the problem, and if possible a short sample case that repeatedly fails. You'd be surprised how much more quickly and pleasantly we respond to properly documented problems.

8) Free FLEX™ Utility!

In issue number 4 of the FLEX Newsletter we provided a utility called "S1DUMP", to dump a FLEX binary file in the Motorola S1/S9 coded record format. Shortly after mailing that newsletter, we received a letter from Douglas Beck of Los Altos, California, with a copy of the antithesis of S1DUMP, called "S1LOAD". As you might guess, it reads a FLEX text file containing an S1/S9 dump (such as would be produced by S1DUMP with the FLEX "O" command) and places the decoded binary data in memory. These two programs are very useful for exchanging binary data between a FLEX system and a non-FLEX system (if there still are any)!

In his letter, Mr. Beck also pointed out that the "S1DUMP" utility needs a slight modification to produce an S1/S9 dump which can be loaded by the Motorola EXORciser "EXBIN.CM" command under MDOS. That change is in the string called "S9". It should be changed to look like this:

```
S9  FCC  'S9030000FC'
    FCB  4
```

Our thanks to Mr. Beck for this utility. His original was in 6800 assembler, we translated it into 6809. It is a simple matter for a user to translate it back to 6800 since we did not use any non-6800 registers. Neither Mr. Beck nor Technical Systems Consultants, Inc. make any guarantees on this code and shall not be held responsible for any possible damages resulting from its use.

The assembled source listing follows.

```
                        *
                        * S1LOAD
                        * BY DOUGLAS K. BECK
                        *
                        * THIS UTILITY LOADS A FILE IN THE S1/S9 RECORD FORMAT
                        * INTO THE PROPER LOCATIONS IN MEMORY.  THE SYNTAX IS:
                        *            +++S1LOAD <FILENAME>
                        * WHERE THE DEFAULT DRIVE IS THE WORKING DRIVE AND THE
                        * DEFAULT EXTENSION IS ".OUT" (FOR FILES CREATED BY THE
                        * S1DUMP UTILITY IN CONJUNCTION WITH THE "O" COMMAND).

                        * EQUATES
                   CD03 WARMS   EQU   $CD03
                   C840 FCB     EQU   $C840
                   CD2D GETFIL  EQU   $CD2D
                   CD33 SETEXT  EQU   $CD33
                   CD3F RPTERR  EQU   $CD3F
                   CD1E PSTRNG  EQU   $CD1E
                   D403 FMSCLS  EQU   $D403
                   D406 FMS     EQU   $D406

                        *
C100                    ORG     $C100
                        *

C100 20   04       S1LOAD  BRA   SLOD1
C102 01            VN      FCB   1          VERSION 1.0
                        *
                        * VARIABLES
C103               BYTECT  RMB   1
C104               ADDR    RMB   2

                        * FLEX FILE SETUP SEQUENCE

C106 8E   C840     SLOD1   LDX   #FCB
C109 BD   CD2D             JSR   GETFIL     GET FILE FROM CMND. LINE
C10C 24   03               BCC   SLOD2
C10E 7E   C1A2             JMP   ERROR
C111 86   0B       SLOD2   LDA   #$B        DEFAULT EXT = .OUT
C113 BD   CD33             JSR   SETEXT
C116 8E   C840             LDX   #FCB
C119 86   01               LDA   #1         OPEN FOR READ
C11B A7   84               STA   0,X
C11D BD   D406             JSR   FMS
C120 27   03               BEQ   SLOD3
C122 7E   C1A2             JMP   ERROR
C125 86   FF       SLOD3   LDA   #$FF       SPACE COMPERSSION OFF
C127 A7   88 3B            STA   59,X

                        * PROCESS "S" FORMAT DATA, TEST FOR SOF AND EOF

C12A 8D   7F       SLOD4   BSR   READ
C12C 81   53               CMPA  #'S        FIND THE "S"
```

```
C12E 26 FA              BNE     SLOD4
C130 8D 79              BSR     READ
C132 81 39              CMPA    #'9         NINE IS END OF FILE
C134 27 34              BEQ     EXIT1
C136 81 31              CMPA    #'1         ONE IS START OF RECORD
C138 26 F0              BNE     SLOD4

                * READ BYTE COUNT

C13A 5F                 CLRB                CLEAR CHECKSUM
C13B 8D 3B              BSR     INBYTE      GET ONE BYTE
C13D 80 02              SUBA    #2
C13F B7 C103            STA     BYTECT      SAVE COUNT

                * READ LOAD ADDRESS
C142 8D 29              BSR     INADDR      GET LOAD ADDRESS

                * MAIN DATA SAVE LOOP

C144 8D 32      SLOD5   BSR     INBYTE      FETCH A BYTE
C146 7A C103            DEC     BYTECT      DECREMENT COUNT
C149 27 16              BEQ     ERROR2      DON'T STORE CHECKSUM
C14B BE C104            LDX     ADDR
C14E A7 84              STA     0,X         SAVE IT
C150 12                 NOP
C151 A1 84              CMPA    0,X         DID IT STORE?
C153 26 07              BNE     ERROR1
C155 30 01              INX
C157 BF C104            STX     ADDR
C15A 20 E8              BRA     SLOD5       LOOP TIL DONE
                *
C15C 8E C1C0    ERROR1  LDX     #ERSTR1
C15F 20 06              BRA     ERR21

                * LAST BYTE READ IS 1'S COMPLEMENT CHECKSUM
                * B HOLDS SUMMATION OF DATA AND CHECKSUM
                * ADDING ONE TO CONTENTS SHOULD EQUAL ZERO

C161 5C         ERROR2  INCB
C162 27 C6              BEQ     SLOD4       GO FOR ANOTHER RECORD
C164 8E C1D4            LDX     #ERSTR2
C167 BD CD1E    ERR21   JSR     PSTRNG
C16A 7E C1A5    EXIT1   JMP     EXIT

                * SUBROUTINES

C16D 8D 09      INADDR  BSR     INBYTE
C16F B7 C104            STA     ADDR
C172 8D 04              BSR     INBYTE
C174 B7 C105            STA     ADDR+1
C177 39                 RTS
                *
C178 34 04      INBYTE  PSHS    B           SAVE B REGISTER
C17A 8D 1B              BSR     INHEX
```

```
C17C 48                     ASLA
C17D 48                     ASLA
C17E 48                     ASLA
C17F 48                     ASLA            DATA MOVED TO M.S. POSITION
C180 1F   894D              TAB
C183 8D   12                BSR     INHEX
C185 34   04 ABE0           ABA             ADD THE TWO HALF BYTES
C189 35   04                PULS    B       RESTORE CHECKSUM
C18B 34   02                PSHS    A       SAVE BYTE
C18D 34   04 ABE0           ABA             MAKE NEW CHECKSUM
C191 1F   894D              TAB             PUT INTO B REGISTER
C194 35   02                PULS    A       GRAB BYTE AND RETURN
C196 39                     RTS

                    * GET CHAR FROM FILE, MAKE INTO HEX DIGIT
C197 8D   12        INHEX   BSR     READ
C199 80   30                SUBA    #$30    REMOVE ASCII OFFSET
C19B 81   09                CMPA    #9      >9?
C19D 2F   02                BLE     INHEX1
C19F 80   07                SUBA    #7      CORRECT
C1A1 39             INHEX1  RTS

                    * ERROR ROUTINE
C1A2 BD   CD3F      ERROR   JSR     RPTERR  REPORT ERROR
C1A5 BD   D403      EXIT    JSR     FMSCLS  CLOSE FILE
C1A8 7E   CD03              JMP     WARMS   RETURN TO DOS

                    * READ CHARACTER FROM FILE
C1AB 8E   C840      READ    LDX     #FCB
C1AE ED   D406              JSR     FMS
C1B1 26   01                BNE     READ1
C1B3 39                     RTS
                    *
C1B4 35   02        READ1   PULS    A
C1B6 35   02                PULS    A       FIX STACK
C1B8 A6   01                LDA     1,X     GET ERROR NUMBER
C1BA 91   08                CMPA    #8      EOF? (NEVER HAPPENS)
C1BC 26   E4                BNE     ERROR
C1BE 20   E5                BRA     EXIT

                    * STRINGS

C1C0 42 59 54 45    ERSTR1  FCC     'BYTE WILL NOT STORE'
C1D3 04                     FCB     4
C1D4 43 48 45 43    ERSTR2  FCC     'CHECKSUM ERROR'
C1E2 04                     FCB     4


                            END     S1LOAD
```

0 ERROR(S) DETECTED

6809 NATIVE-CODE PASCAL COMPILER for FLEX™

The demand for a higher level language that produces fast and efficient code has prompted Technical Systems Consultants, Inc. to develop a 6809 Native-code Pascal Compiler. This Pascal compiler produces actual 6809 assembly language mnemonics, unlike many of the other Pascal "compilers" which only produce interpretive "P-Code". Because of this native-code production, programs developed with the Native-code Pascal Compiler may run from 5 to 10 times faster than those compiled using an interpretive compiler. The specification for the syntax and semantics of Pascal for this compiler are based on the Jensen and Wirth User Manual; the compiler implements nearly all of the features defined in the User Manual. The few exceptions of major features that our Native-code Pascal Compiler does not support includes GOTO statements and labels, procedures and functions used as parameters, the nesting of procedures and functions without the use of FORWARD, and the procedures DISPOSE, PACK and UNPACK. Both integer and floating point math are supported. The floating point arithmetic is double precision containing up to 16.8 digits of accuracy from 1.0 E-38 to 1.0 E+38. This compiler supports the standard trigonometric, exponential and square root functions along with a random number generator for statistical and simulation programming. Integers range from -32768 to +32767, using 16 bits for each integer. All of the ASCII characters from 0 to 127 may be used and written in Pascal programs. Variable names are unique to 160 characters allowing the users greater creativity in programming. PACKED arrays and records are allowed in the syntax; however, a PACKED array or record is no different than an unpacked array or record. Pascal sets may contain up to 128 elements; however, the ordinal value of these elements must be from 0 to 127. Therefore, sets of real numbers are not implemented, but a set of characters is easily accommodated. The Native-code Pascal Compiler allows the users to read the command line in FLEX through a record structure called PARAM. In this way the user may pass parameters from the command line to the user's Pascal program. The passing of file names and options is a common application of this parameter passing feature. Furthermore, the Native-code Pascal Compiler allows the users to redefine the standard Pascal input and output files as external files residing on the disk. The Native-code Pascal Compiler supports dynamic storage allocation using the standard procedure NEW and procedures MARK and RELEASE for dynamic deallocation of storage. Pointer type variables are fully supported; therefore, true file I/O using file buffer pointers and the procedures GET and PUT is implemented. FLEX Pascal files are all sequential access. The Native-code Pascal Compiler allows a Pascal program to call other separately written and compiled Pascal programs or assembly language programs. Furthermore, parameters may be passed to these other programs in the same fashion that parameters are passed from the command line. Additional procedures exist for the users to interface with the operating system itself. These routines include SYSTEM_DRIVE and WORK_DRIVE, BUFFER and UNBUFFER for single character input, and various other routines tied to FLEX. Instructions for trimming the run-time package for the FLEX version of Pascal are included. Trimming the run-time package may be helpful if a program does not need many of the functions but requires a great amount of memory for execution. By trimming the run-time package, a program may be able to reclaim the memory space allotted to the useless run-time procedures. Overall, the 6809 Native-code Pascal Compiler produces very fast and efficient code. Pascal requires a 56K system in order to function. The FLEX version on minifloppies requires two diskettes. The package include our user's manual, a copy of the Pascal User Manual and Report, by Jensen and Wirth, the compiler and run-time object code programs and about ten example Pascal programs in source form.

FLEX 6809 Native-code Pascal Compiler $200.00
Manual only - $40.00

68000 CROSS ASSEMBLER

This is a full 68000 assembler which runs on a 6809 microcomputer under the FLEX™ operating system. It accepts all of the standard Motorola instruction mnemonics with the exception of certain "variations" to standard root mnemonics. Those root mnemonics are ADD, AND, CMP, EOR, MOVE, OR, and SUB. Our 68000 Cross Assembler does not allow the programmer to explicitly append the addressing suffix on these instructions. Instead, the assembler infers the correct type of instruction or opcode to generate from an analysis of the operand. The suffixes not permitted are A (address), Q (quick), and I (immediate). Note that the X (extended) suffix is still permitted. This feature makes programming somewhat easier as the programmer need not be concerned with the variation - the 68000 Cross Assembler does that work for him. Labels may be of any length with eight characters significant. Expressions may contain decimal, octal, hexadecimal, or ASCII constants and permit the following operators: add, subtract, multiply, divide, AND, OR, NOT, shift right, shift left, and relational operators. All expressions are evaluated to a full 32 bits before any required truncation. Numerous directives or options permit page formatting, titles, subtitles, listing control, object code output control, sorted symbol table listing, line numbering, auto field formatting, warnings, command line parameters, inclusion of separate source files, and setting of even word boundaries. The 68000 Cross Assembler also supports conditional assembly and macros. Object code is output in Motorola S1/S2/S8/S9 records of ASCII hexadecimal data. The 68000 Cross Assembler is available on either 5 or 8 inch disk. The manual fully describes the use of the assembler, but assumes the user is already familiar with the standard Motorola 68000 assembler instruction set as defined in the "MC68000 16-Bit Microprocessor User's Guide" published by Motorola Semiconductor Products, Inc.

68000 Cross Assembler for 6809 FLEX $250.00