FLEX USER GROUP
NEWSLETTER

3540 STURBRIDGE COURT
ANN ARBOR, MI 48105
ISSUE 3

STRUBAL+ CHAPTER 2

As promised last time, here is a further report on STRUBAL+. At the risk of spoiling my relationship with Jack Hemenway, I am not very impressed with STRUBAL+. Contrary to Jack's words regarding it, there are only very minor improvements in it. There is an added FUNCTION feature like the DEF FNA in BASIC. Also added is a means of specifying the precision of the arithmetic in increments of two digits, from 4 to 14! The less precise, the arithmetic, the faster the results, says Jack. I was not able to see the difference in calculation time from 6 to 14 digits. My test was admittedly not very precise, and the program contained a mix of arithmetic and scientific functions, which may have been so slow as to obscure the difference in speed of the arithmetic.

At any rate, I translated one of my favorite programs having to do with my work into STRUBAL. This program requires about 1.5K in Assembler, and the source code for a BASIC version is about a K, so the 9K BASIC plus the 1K program take about 10K. The program in STRUBAL is 5K long, but requires about 6K of "runtime package", so the program uses 11K total. In STRUBAL it runs in 6 seconds. In SWTPC BASIC it also runs in 6 seconds! In Computerware's SUPERBASIC it runs in 3 seconds!

The linking loader is set up to load at $3000, right in the middle of my 32K memory, so the 11K program wouldn't fit below it, but just squeezed in above it. BY the way, the source listing for this program is just two pages with liberal comments. I asked Jack why on earth he didn't supply the linking loader in a relocatable file too, so the user could relocate it to suit his own system. Jack responded by sending me the source file so I could make my own relocatable file. I set it up to load at $6400, so I have a lot more space for programs. While I was at it, I also modified LINK to be compatible with the FLEX P command. That is, it prompts for input on the terminal in all cases, but if P,LINK is entered, the load map is output to the printer. While I was at it, I also modified it to honor the pause feature of TTYSET. By the way, in order to avoid confusion I changed the name of the Hemenway linking loader from LINK to LINKR. The R is for relative, and it avoids a conflict with the FLEX LINK utility. Now the load address limits don't scroll off the screen before I can read them. I sent the listing as modified back to Hemenway Associates just a few days ago, and am waiting for a reply.

One of my initial negative reactions was to the fact that LINK produces an object file on disk in the Motorola Punch format. Jack tells me that FLEX is about the only system that won't work with such a file. Obviously to the people at TSC such a file takes about 2.5 times as much disk space as a binary file, so they didn't make provision for it. I had initially written a loader utility for the punch format file, but now will be able to throw away the

punch format subroutine and insert a straight binary file save routine of my own. I must say that the program is very well commented and nicely written. If any of you have the linking loader, I will supply the patches for use with FLEX and the binary file save routine (which I have yet to write).

One of the parts of the package that I mentioned last time is the Relocatable Assembler. Hemenway Asociates are to be congratulated on this nice bit of software. It is very easy to use, sticks to all the Motorola mnemonics, and even uses the same error numbers (these are the same as those of SWTPC's original CO-RES assembler.) Of course they had to make a little change to make it not work with all of.our soruce files (maybe) If you are like me, you probably have adopted the convention of entering LDAA rather than LDA A to save file space in your source file. Both Motorola and TSC allow either in their assemblers. Hemenway only allows the LDA A structure.

I was able to modify one of my source files and make a relocatable file on the first try. This is really nice software, and I highly recommend it. Since I don't have the source file for the assembler, I'll have to do some exploratory surgery (disassembly and dump) to find the jumps to input and output routines, and make it compatible with the P command. This is likely to take a little longer but I will prepare a patch overlay if anyone is interested.

One last point here. Hemenway allows two program sections in the relocatable assembler, a common, and a base section. If you are familiar with the Motorola Relocatable Macro Assembler, you know that it allows 4 sections, that is to say that it keeps track of four program pointers. It allows what Motorola calls a BASE section, which is simply zero page. In this section, direct addressing is used. They allow a DATA section for variable data, and a COMMON section for data to be shared by other program modules. Last but not least, they allow a PROGRAM section for the main body of the program. Hemenway's COMMON may be located on zero page, but direct addressing is not used, since the assembler doesn't know where the COMMON will be located. Hemenway's BASE section corresponds to Motorola's PROGRAM section. The only way in the Hemenway Assembler to take advantage of direct addressing is to assign zero page addresses to often referenced variables with equates. These locations may of course be shared by other program modules if the same equates are used in them also.

Does this make a large difference? It doesn't seem to change the program length by more than about 3% for a program that I have been working on. However, many of the possible zero page references are in often used loops, so I would expect a slowing of the program by a factor larger than this. Back to the main subject, STRUBAL+. As you might guess by now, I'm no longer too excited about STRUBAL+ as a useful development tool. I was told by Bob Grappel that STRUBAL could indeed be made more efficient, but that it was written to be compatible with all of the 6800 systems, and as such didn't take advantage of the "tricks" allowed in any of the systems. This would of course make it quite different for each system. So we have a "universal" tool which doesn't work well with any particular system. Bob indicated that a great deal of the runtime package is devoted to output formatting, which STRUBAL does very nicely. Unfortunately in the application that I had in mind, input is through a PIA and output is to LED displays through another PIA. All the nice formatting is excess baggage that takes up ROM and never is called upon.

The runtime package is written in such a way that it can't be searched as a library and only those routines included that are called by the user program. I hear rumors that perhaps that might be done, as well as some other "efficientizing" to work with a 6800 operating system being developed by Hemenway.

On the other hand, Strubal does work, uses the forms and syntax that alow structured programming, and would be a start toward learning how to understand such languages as Fortran and the various PL/X languages. I was able to get all the programs running that I tried in STRUBAL, one with some difficulty. Most of the 12 errors per page that I made were syntax errors that were obvious on study of the manual. Incidentally, the manual supplied with STRUBAL+ is very much improved over the original STRUBAL manual. There is a page devoted to each command, showing how it is used, and giving examples.

To sum it all up, STRUBAL+ is too inefficient in terms of memory used, and too slow to be any real advantage over a BASIC interpreter. As such it is not a very useful development tool. It is a working system, and might be considered a nice new "toy" for the hobbyist. The original STRUBAL was $100. STRUBAL+ is $280, and as such is a very expensive toy. Frankly, if the old version was worth $100, the new is worth about $125. I would not have spent the $280 for STRUBAL+, and would not have been able to report had Jack Hemenway not decided to send it to me on the basis of my complaints about the old version. I think Hemenway has taken a compiler that has only hobbyist interest, and priced it right out of the hobbyist market! Jack, if you see this, I'm sorry, but that's how I see it.

OTHER SOFTWARE

I just saw an ad last week from Smoke Signal, for their Fortran compiler for 6800. If you received the first issue of the 6800 Microjournal as I did, you saw the article about compatibility between FLEX and SSB. If any of you get the Fortran and successfully operate it with FLEX I would like an article from you about it.

If you've been waiting eagerly for the 6809 processor board from SWTPC as I have, the rumor is that we don't have much longer to wait. I wonder what software will be supplied with the new processor? How much of what we have will have to be scrapped when we go to the 6809?

A BRAINSTORM

The other day I was trying to figure out how to improve and increase the size of our newsletter without increasing the costs to you, the subscribers. I had an idea which, I think merits some consideration. Several of you have sent me utilities that have been published previously. If I were to put together a package of utilities on a disk, and offer them for sale to non-User Group members for a nominal charge, (say $8.00 a disk), we could subsidize our printing operation, and spread all this good software around a bit. You would receive full credit for your program as your name and address would be included in the source listing on disk. The best part is yet to come. Should we receive enough orders to cover the cost of the newsletter publication and

mailing, and have some positive profit left over, I am willing to share that profit at the end of the year with the contributors of software. The dividing of profits would have to be determined by me, on the basis of number of bytes, relative complexity, uniqueness of program etc. How does that strike you authors of programs that we've published so far? Maybe the possibility of some financial gain would inspire some of the others to contribute some programs.

Please note that I am not asking for major software that is of considerable value to be donated free. If you have some major things, don't contribute them free. Tell us what you have and what you think it is worth. maybe we can give it some exposure and make people aware of its availability. I expect to contribute a program or two each issue myself, and so will indirectly benefit from my own efforts. Please remember that the idea is to make membership in the group affordable by anyone who wants to join, and at the same time, keep me out of the poorhouse or worse yet, the doghouse with my family who resent me underwriting a project like this in addition to spending so much time at it! Your responses would be appreciated.

THE DISK ISSUE

With no pun intended, this issue is not yet settled. Several of you have sent me disks, empty or with utilities on them for consideration for publication in this Newsletter. Early along I brought up the possibility of an issue of this on disk, if each of you would send me a disk. As the membership grows, I can see that such an issue would quickly get impractical unless we could get "regional distributors" who would make 5 or 6 copies of a disk I would send to them, and distribute it to others. That way we could keep the wear and tear on disk drives and heads down to a nominal amount by spreading it around among us. Not very many of you have responded with a disk. I will hold the ones I have for a while, and then, if the responses don't come, will fill your disk with the programs to date that we have published, and return them to you, and the "disk issue" will be dead.

TSC BASIC

This issue has again been delayed due to lack of time on my part and the wait for a response from one of our group concerning pulbication of some utilities. During the delay I have received a copy of the TAPE VERSION of TSC's new BASIC. The disk version will soon be available, but TSC wanted to get the BASIC into the hands of some users to see if some bugs show up. I found none at all, having run a dozen or so programs of various types. All I can say at this point is WOW! This BASIC is FAST. Of course, this is what TSC was trying to do.

First of all, a rundown of the features. This BASIC is very similar to all the standard ones. Syntax is essentially unchanged from the older SWTPC BASIC. There is where the resemblance ends. Those of you who work with several systems and BASIC interpreters will appreciate the fact that this BASIC is much closer to the ANSI standard. It allows subscript (0) in arrays. This has been a sore point in SWTPC BASIC when trying to adapt programs from other interpreters. The zero subscript need.not be used, but is there to be compatible with programs written for other BASIC interpreters. There is an

addition in that the IF-THEN-ELSE structure is supported. I had wondered what advantage this would be, but soon found a use for it that saves a couple of lines of program, and is very easy to understand. The final disk version, I am assured, will be compatible with source files prepared for the older BASIC, as well as allowing the saving of the "intermediate" file. This file has had some of the text "translated" into an intermediate code, and though it is not much shorter than the original text file, will load and run faster. The file may be saved in either form, as the interpreter is capable of reconstructing the original text file from the intermediate code. This interpreter does some syntax checking when the text is entered, and will give you error messages when the input line has certain errors.

There is one new command, that I haven't used, the CLEAR instruction. I am now assuming that this clears all variables to zero, which is useful in a program that is to be run several times without exiting. Previously it was necessary to set all the variables to zero by using an assign (LET) statement. Setting a whole array to zero could take considerable time.

Dan Vanada of TSC indicates to me that the disk version will allow use of all the FLEX utilities without exiting BASIC. There will be a problem with a few utilities in MINI FLEX but FLEX2 (see below) has all utilities located out of the way so they won't confilct with BASIC.

By now you are wondering how fast this new BASIC really is. If you have the October 1977 issue of Kilobaud Magazine, refer to the benchmark tests on page 23. This new BASIC falls between number 1 and number 2 in the speed ratings. It must be noted that the number 1 Osi BASIC is running in a 6502 at 2 Megahertz. If any of you has a 6800 system running at that speed (which is more than twice the standard SWTPC operating speed), you will have the fastest BASIC around on an 8 bit Microcomputer. For those of you who may be newcomers to computing, the benchmarks refered to are some simple programs with many repeated instructions in the form of loops. The running time was tested on many systems and the results tabulated. The results for the longest test are a fair indication of the whole series, and some are as follows:

1. OSI 8K BASIC at 2 MHz	21.6 seconds
2. NEW TSC BASIC at .9 MHz	30.0 "
3. CROMEMCO Z-80 at 4 MHz	32.7 "
4. ALTAIR 680 8K V. 3.2	81.8 "
5. SWTPC 8K 1.0	204.5 "

So you can see that this BASIC is a real contender to put the 6800 in the respectable category. We have TSC to thank for most of the software that is making the 6800 a winner at this point. I am looking forward to having the final version, and will order it as soon as it is available.

Are we giving anything up for all this speed? Yes, just a little bit. This version of basic has just 6 digit floating point arithmetic, whereas the old BASIC had 9 digit arithmetic. Actually, the arithmetic is nearly 7 digits internally, but the output is 6 digits. This is an advantage over similar BASIC interpreters that use binary arithmetic as this one does. Many of the others return strange things like 2+2=3.99999. TSC went out of their way to take care of rounding so that the answers that should be integers are. Unlike

some of the other binary arithmetic BASICS this'one allows such things as a program to find prime numbers by testing the quotient of two numbers to see if it is an integer. Some of the binary arithmetic BASIC interpreters give screwy results when you try to use them this way! Perhaps some of you are by now confused. The output of this BASIC is decimal just like the old one. When I say they use binary arithmetic rather than binary coded decimal like the old one, I am refering to internal operations only. The numbers are converted to decimal before they are output. Binary arithmetic is inherently faster. All that is required is that the results be converted to decimal before we see them.

Did we lose anything but digits? Yes, there is a little more. This BASIC has a default line length of 80 characters. You can, however get in and change this number by a POKE. We also lose the DIGITS= command. There is no way to adjust the output format as to number of digits directly. Leading and trailing zeros are suppressed, however, and it is possible to output dollars and cents by using a rounding routine as below:

```
PRINT "TOTAL ";INT(N*100+.5)/100
```

This will give you two digits after the decimal point for the value of N. Other than these, there are no less capabilities in the new BASIC. There is another great gain, however; the array dimensions are not limited to 255, but are limited only by the amount of memory available. This eliminates a difficulty that could previously be "programmed around" with some difficulty.

All in all, I'd say that this is a great new BASIC and you'll want it as soon as it is released. The price with full manual on disk will be $55 approximately (see the TSC Newsletter of recent date.

FLEX2

Since my employer has a SWTPC system going as a text processor using TSC's text editor and processor, we ordered FLEX2 a few weeks ago. It arrived the other day, and we are truly impressed with its additional capabilities. Nearly all of the features for which I had just finished writing utilities, are included in FLEX2. The best part is the "print spooling" function. This allows you to do a function such as assemble a program, and list the output rather than to a printer, to an output file. You can then instruct the system to output this file to the printer, and proceed to do something else with the system while the printer is working. This works by means of a "multi-tasking system". You must have the SWTPC interrupt timer board. FLEX2 sets this up to interrupt the computer every 10 milliseconds. When it is interrupted, it does a quick check to see if the printer busy signal is off, and if it is, fills the printer's buffer with output until the printer is again busy. It then returns to the main activity of running your program, whether you are assembling another program, listing a file, or whatever. There are utilities to allow you to look at the status of the files waiting to be printed, modify these, kill the current print "queue", (in case of a printer jam etc).

Some of the other new features include the possibility to create a command file that can supply input just as though it were being entered from the terminal. For example you can set up a file called YES.TXT that contains a single line containing "YY". This file can be called to enter the responese for a DELETE command. This is only a very simple example. If you have the Hemenway software, ie the Relocatable Assembler and/or the Linking Loader, you can set up all the necessary input in a command file and (if you are like me) assemble and load a new file dozens of times as you debug it and chage the source listing to reflect the corrections, without going through the pain of entering the answers to all the input questions each time. I had written a pair of utilities for MINI-FLEX to do just this, but FLEX2 has it all built in. This letter contains a "patch" to the TSC assembler to allow it to print out the date on the top of each page. The Assembler for FLEX2 (which, along with the TSC text editor, is supplied with FLEX2), already has this feature built in. I was surprised to see the date nicely printed on the top of each page of the first output from the new assembler.

What are the pains of converting? It depends on how old your SWTPC system is. If you have the "2" version of the mother board and the processor board, you have very little to do. If you have the older versions as I do, you will have to make a very small modification on the mother board and the processor board. You will also have to move some of your memory, or add some in the address range from $A000 to $BFFF. FLEX2 takes 8K of memory for the DOS, twice as much as the MINI-FLEX. However, the relocation to the higher address allows you to have a full 32K from $0000 to $7FFF for your "user memory". TSC does not automatically supply the Text Processor in FLEX2 format. I have converted it (I'm getting ahead of myself a bit, see below) with complete success.

How about converting all the old MINI-FLEX files to FLEX2? That is not too bad a process. If you have the source files, you may use a utility supplied with FLEX2 to move the source from a MINI-FLEX disk to a FLEX2 disk, and then edit it, changing all the FLEX EQUATES to agree with the new FLEX2. You can then assemble the program and have the FLEX2 version. If you don't have the source listing, but only the binary file, you may use the SEARCH utility from our last newsletter, and find all occurrences of references to FLEX, substituting the FLEX2 references for the old ones. I have already converted many of my own personal utilities with success on the first try. There is a difference in disk format between the two systems that causes the complication of having to use the MOVE utility to get the file on disk in the FLEX2 format, which has 256 byte sectors rather than the 128 byte sectors of MINI-FLEX. The new format is more efficient, allowing more than 10% increase in the disk storage capacity. The limit of 75 file names in the MINI-FLEX directory no longer holds in FLEX2. It is now possible to protect files so they can not be renamed or deleted accidentaly.

In order to use the MOVE utility for the conversion of programs it is necessary to have both the FLEX2 and the MINI-FLEX operating systems in memory at the same time. With regard to memory allocation, both systems require $0000 to $1FFF. MINI-FLEX though it doesn't use memory in this range, requires some memory in the $2000 range for the bootstrap loader. Debug uses $4000 to $5FFF. MINI-BLEX uses $6000 to $6FFF, and FLEX2 uses $A000 to $BFFF. If you have 4 8K boards as I do, you can move the $4000 to $5FFF board to $A000 to $BFFF, and be able to use both systems, having lost the capablilty to

use DEBUG only. The editor and assembler will run in this configuration, as will both "FLEXes".

I was so impressed that I have ordered FLEX2 for myself, but couldn't wait to get started, so I brought FLEX2 home from work and have started the process of converting files. I'm glad that I just bought a new supply of disks since it will take some extras to hold both versions of some programs until I am completely switched over. I don't think the dual systems will be much complication for this group, since the equivalent addresses may be added as comments on source listings to ease the work of changing from one system to the other. Oh, yes, there's one more advantage. The Utility area in FLEX2 extends from $A100 to $A6FF. This gives us room for a 1.5K long utility in that area. All of the TSC supplied utilities fit up there, so that there is no problem with running any of them from the new BASIC. Try it, you'll like it.

BEGINNER'S CORNER

I was reminded by a new member of the group, that all of you out there are not experienced programmers, and that there should be something for the beginner as well. I guess I had kind of forgotten that point, and I will try to include something along that line each time. I thought I might start by giving you a sample of an introduction to Machine Language and Assembler programming that I wrote for my company to use as notes for a course. This text is rather long, and I will just give it a start here for a few pages. If some of you think it is worthwhile we will continue it over several issues, until it is completed.

6800 MACHINE LANGUAGE AND ASSEMBLER PROGRAMMING

This text will assume that the reader knows nothing about Microprocessors or programming. I will try to introduce all new terms by giving a short definition or synonym.

Parts of a System

In order to understand programming it is necessary first to have a basic understanding of the parts of a microprocessor system. We'll begin a lttle unconventionally, and first look at the memory.

Memory may be thought of as a large number of boxes or "pigeon holes" like the message or mail boxes at a hotel desk. There is a further similarity in that each box has a label called its address. As in the case of house addresses, no two are alike.

Memory is used by the computer system to hold the program (a sequence of operations to be performed on data). Memory also holds the data on which operations are to be performed. The purpose of all the other parts of the system is to perform the sequence of operations directed by the program on the data in the memory.

Input Output

The next part of the computer system is the input/output device.

In order for the system to function, it needs:
A) To be given some data on which to run
B) To be told when it is to run the program
C) To have some way to tell you the results

These functions are performed by an Input/Output device, or I/O. In home computer systems, the I/O may be a "hexadecimal" keyboard and some "LED" displays as on the motorola evaluation kit, or a keyboard and video monitor (or TV set), or a @hard Copy" device such as a teletype, typewriter, or line printer that gives you a piece of paper with the program or its result printed on it. This I/O device is usually called a terminal.

Processor

The next part of the system is caled the processor. The processor is the smart part of the system. It understands all valid program commands, and performs operations on the data from memory. It may drive one or several Peripheral Interface Adaptors (PIA's). These devices allow the processor to communicate with the terminal and other devices such as a tape cassette recorder or disk drive.

In order to perform or "execute" the program, the system makes use of "registers". A register is a special memory location (sometimes several locations), that is part of the processor, and separate from the main memory. The registers are special and have names, unlike the main memory whose locations have address "numbers".

The 6800 has the following registers:

1. Program counter (PC)
2. Accumulator A (ACCA)
3. Accumulator B (ACCB)
4. Index (X)
5. Stack Pointer (SP)
6. Condition Codes (CC)

The program counter is used when the program runs, to keep track of where the current program instruction is in the memory. The accumulator is used in performing the operations on the data in the memory. The results of arithmetic operations are found in the accumulator. The index register has various uses, one of which is to point at a memory location from which data is to be obtained. This will be discussed fully when we get to the subject of indexed addressing. The status and stack registers will also be discussed later.

Mass Storage

This is the name given to devices that can store large quantities of program and data information. Small systems almost exclusively used tape cassettes until recently. At this point, the larger home systems and small business systems use disk. Since this is being written for a disk system user's group, I need not go into detail about what a disk looks like. For those of you who never had to suffer through loading 8K BASIC from a tape, it took about 8 to 10 minutes, if my recollection is correct. Now I sometimes get impatient waiting the 10 seconds or so that the disk takes!

Programs

A program consists of a list of instructions and directions as to the location of the data on which to perform the instructions. Ultimately these instructions must be put into the memory as "binary codes". while it is not the purpose of this article to go deeply into a discussion of binary and hexadecimal number systems and their theory, the work binary means "two valued". The numbers used in the binary system are 0 and 1. All present day computers use binary numbers because the two values can be represented by tow "states" such as on and off, voltage and no voltage, (or positive and negative voltage), magnetized and demagnetized, etc. Perhaps a comparison table would help to explain the relationships.

DEC.	HEX.	BIN.
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111
16	10	10000
17	11	10001
31	1F	11111

As you can see, all of the possible combinations of 1 and 0 that can be made using four places or "bits" (bit is short for binary digit) are used in counting from 0 to decimal 15. Hexidecimal may be thought of as a kind of shorthand notation that replaces four bits of binary with one digit. For example, entering a machine code as 3A is far easier than 00111010, the binary equivalent. The job can't be done with decimal numbers, because there aren't enough symbols. A notation known as "binary coded decimal" uses four binary bits to represent the numbers 0 to 9, so that each four bits represents a

decimal digit. This is very wasteful of memory as compared to binary notation.

Well, that's about it for this time. I hope that has been enough to bring some comments back to me regarding the content and level of presentation. I hope to catch up a bit with this issue, getting the next one prepared relatively soon so that it can still be mailed in April. Thanks to all of you for your interest and your subscriptions. Please feel free to duplicate this letter in any way you like. Some one or more of you has laced a copy in a computer store, and several people have subscribed as a result. Should this copy reach someone who needs the details, I am asking $12 for a 12 issue subscription. It looks as though we are going to have a hard time keeping this monthly, but we're going to try to catch up and stay on schedule. Please make your check payable to Ronald W. Anderson. We don't have a bank account set up in the name of the user's group, and checks to Flex User's Group might be hard for me to cash. If new subscribers want to get in on the letters from the first issue in January 1979, send me a disk, and I will put the first issues on it in formatted form. If you will indicate the width of your printer, I will format your copies to fit. I'll also throw in the source listings of all the programs we've published to date. In order to fit all this on one disk, I'll have to "double side" it for you if you have not already done so. Until I get my Text Processor bugs out, the disks will have to be in MINI-FLEX format. I hope by next month to be able to send either. Should you desire to start with the current issue, please so indicate. I have received letters from some of you saying that you would "like to take advantage of my offer". I have made several offers as you have written for information. Please spell out in your letter what it is that you want so I can get it right the first time. Again, I call for programs, reviews, articles from you. This thing will only stay off the ground as you send material for publication.

HERE ARE SOME UTILITIES

Last time I promised some utilities again. Here is one from Gary Caudell, that is not really for FLEX. If you have wanted the A/BASIC compiler from Microware, but couldn't manage the price, Gary may have a solution for you. The cassette version is quite a bit less expensive, but requires the RT-68 ROM. You can convert the A/BASIC to operate compatibly with MIKBUG OR SWTBUG with this patch program. Note that this does not make A/BASIC compatible with FLEX. It is still a cassette version, but you don't have to buy the RT-68 operating system to use it.

I received quite a group of utilities from Milan Konecny from the Province of Quebec Canada. Some of them are modifications of TSC utilities, mostly to add the date to directory and listing outputs. Milan has agreed to supply these in Append or Overlay form so that we will not be publishing listings of TSC utilities, but only the changes. one of those he submitted was just an overlay or append, and it is given here. The comments make it self documenting. It is a change to make the TSC assembler print the date at the top of each page (unfortunately starting at page 1). I should note here that FLEX2 version of the assembler has been set up to do this. If you are planning to stay wih MINIFLEX for a while, you might want to do this patch. I must say that Milan's programs are very nicely commented and beautifully formatted.

The other night I was doing some "homework" consisting of a 30 page assembler program for my company. I had debugged about 20 pages of it, and it occurred to me that every time I made a few corrections, I had to relist the whole thing in order to get the correct addresses for debugging in the area where I was working on the debug. I decided to write a short overlay to allow me to specify the page where printing is to start. That way, I don't have to relist all the early pages to continue debugging at the end of the program. Of course references early in the program to routines late in the program will be wrong but I am working only on the end of the program. When the debugging is finished, it is of course possible (and necessary) to make a listing of the whole program with all the corrections. This has already saved me a great deal of time and paper. If you don't ever write or debug programs longer than a few pages, this may sound superfluous. In that case, just ignore it. The assembler, if you add this, will prompt only if you are in the print mode ie P,ASMB,ETC. You can specify any page including 0 for the printout to start. Output will be to your terminal until the starting page is reached, where output will switch to the printer.

```
                    *
                    *
                    * GARRY O CAUDELL
                    * 3125 ROBIN LYNN DR.
                    * ASHLAND, KY 41101
                    * PERMISSION TO COPY GRANTED
                    * DECEMBER 1978
                    *
                    * THIS PROGRAM IS TO PATCH THE MICRO-WARE COMPILER
                    * WORK WITH MIKBUG/SWATBUG SYSTEMS
                    *
                    * MIKBUG EQUATES
                           NAM    PABASIC
                           TTL    PATCH ABASIC CASETTE VER.
                    *
                           OPT    PAG
E1AC                INEEE  EQU    $E1AC
E1D1                OUTEEE EQU    $E1D1
E0D0                MON    EQU    $E0D0
E0BF                OUT2H  EQU    $E0BF
                    *
16E5                       ORG    $16E5
16E5 E1 AC                 FDB    INEEE
18E5                       ORG    $18E5
18E5 E1 AC                 FDB    INEEE
18ED                       ORG    $18ED
18ED E1 AC                 FDB    INEEE
1917                       ORG    $1917
1917 E1 AC                 FDB    INEEE
                    *
16EE                       ORG    $16EE
16EE A0 4A                 FDB    CRLF
17D0                       ORG    $17D0
1700 A0 4A                 FDB    CRLP
1951                       ORG    $1951
1951 A0 4A                 FDB    CRLF
                    *
16E2                       ORG    $16E2
16E2 E1 D1                 FDB    OUTEEE
17C9                       ORG    $17C9
17C9 E1 D1                 FDB    OUTEEE
1902                       ORG    $1902
1902 E1 D1                 FDB    OUTEEE
190B                       ORG    $190B
190B E1 D1                 FDB    OUTEEE
1925                       ORG    $1925
1925 E1 D1                 FDB    OUTEEE
                    *
0827                       ORG    $0827
0827 E0 D0                 FDB    MON
                    *
A04A                       ORG    $A04A
A04A 86 0A          CRLF   LDAA   #$0A
A04C 8D 02                 BSR    JOUT
A04E 86 0D                 LDAA   #$0D
A050 7E E1 D1       JOUT   JMP    OUTEEE
                    *
```

```
                * THIS WAS A TOUGH ONE
                *
 16D1                   ORG    $16D1
 16D1 BD A0 4A          JSR    CRLF
 16D4 96 42             LDA A  $42
 16D6 26 2C             BNE    CONT
 16D8 39                RTS
 16D9 EB 00     PATCH   ADD B  0,X
 16DB 7E E0 BF          JMP    OUT2H
 16DE 7E 16 D9          JMP    PATCH
 1704           CONT    EQU    $1704
                *
                * THIS WAS EVEN TOUGHER
                * FIXES OPT S
                *
 0F11                   ORG    $0F11
 0F11 01                NOP
 0F12 01                NOP
                *
                *
                * THE ABOVE IS ALL THAT IS NECESSARY IF YOU
                * WANT TO USE THE AC-30
                * THE FOLLOWING WILL ALLOW THE SOURCE TO STAY IN
                * MEMORY WHILE THE COMPILER IS BEING LOADED
                * THE COMPILER OUTPUT WILL STILL BE TO CASSETTE.
                * NOTE SOME OF THE ABOVE PATCHES WILL NOT BE
                * NECESSARY IF YOU DO THE FOLLOWING. ($18E5, $18ED)
                *
                *
                * SECTION TO REWIND MEMORY
                *
 00FB                   ORG    $00FB
 00FB BD 20 0D  BACK    JSR    REWIND
 00FE 20 10             BRA    FWD
 0100 20 F9             BRA    BACK
 0110           FWD     EQU    $0110
                *
                *
                *
                * SECTION TO LOAD NEMORY TO COMPILER
                *
 18DE                   ORG    $18DE
 18DE 5F                CLR B
 18DF 8D 15     IN1     BSR    JIN
 18E1 81 02             CMP A  #$02      START OF DATA
 18E3 26 FA             BNE    IN1
 18E5 8D 0F     IN2     BSR    JIN
 18E7 81 03             CMP A  #$03      END?
 18E9 27 08             BEQ    JEND
 18EB A7 00             STA A  0,X
 18ED 08                INX
 18EE 5C                INC B
 18EF C1 80             CMP B  #$80      128 BYTES YET?
 18F1 25 F2             BCS    IN2
```

```
 18F3 6F 00     JEND    CLR    0,X
 18F5 39                RTS
 18F6 7E 20 00  JIN     JMP    DATAIN
                *
                *
                *
 2000                   ORG    $2000
 2000 FF 20 16  DATAIN  STX    XSAV+1
 2003 CE 20 3B  LOAD    LDX    #DATA
 2006 A6 00             LDA A  0,X
 2008 08                INX
 2009 81 1A             CMP A  #$1A      END?
 200B 26 05             BNE    STORE
 200D CE 20 3B  REWIND  LDX    #DATA
 2010 86 03             LDA A  #$03
 2012 FF 20 04  STORE   STX    LOAD+1
 2015 CE 00 00  XSAV    LDX    #0000
 2018 39                RTS
 203B           DATA    EQU    $203B
                *
                * IT WILL BE NECESSARY TO KEEP THE CRLF IN
                * $A04A ANYTIME YOU ARE RUNNING PROGRAMS THAT
                * HAVE BEEN COMPILED BY THE A/BASIC COMPILER
                * A BETTER METHIOD IS TO INSERT THE CRLF ROUTINE
                * AT THE END OF THE COMPILED CODE.  THE COMPILER
                * TELLS YOU WHERE THE END IS.  THEN SEARCH OUT
                * THE JUMPS TO $AO4A AND PATCH TO THE NEW CRLF
                * ROUTINE. (FOR AN EXCELLENT SEARCH ROUTINE SEE
                * MAR 1978 73'S MAGAZINE)
                       END
```

NO ERROR(S) DETECTED

SYMBOL TABLE:

BACK	00FB	CONT	1704	CRLF	A04A	DATA	203B	DATAIN	2000
FWD	0110	IN1	18DF	IN2	18ES	INEEE	E1AC	JEND	18F3
JIN	18F6	JOUT	A050	LOAD	2003	MON	E0D0	OUT2H	E0BF
OUTEEE	E1D1	PATCH	16D9	REWIND	200D	STORE	2012	XSAV	2015

```
                         NAM     ASMPGE
                         TTL     START PAGE MOD FOR ASMB
                         OPT     PAG
                *
                *
                *
                * THIS PROGRAM IS APPENDED TO THE TSC
                * ASSEMBLER AND THE DATE MODIFICATION
                * IT PROMPTS FOR A PAGE NUMBER
                * AT WHICH THE LISTING IS TO START
                *
                * IT IS USEFUL FOR NEW LISTNGS OF A
                * LONG PROGRAM WHERE A CHANGE IS MADE
                * FAR ALONG IN THE PROGRAM, THAT DOES
                * NOT HAVE AN EFFECT ON EARLIER PARTS
                * OF THE LISTING.
                *
                * EQUATES
7118            PSTRNG  EQU     $7118
711E            PCRLF   EQU     $711E
710F            GETCHR  EQU     $710F
70A3            SWITCH  EQU     $70A3
00AC            PAGENO  EQU     $AC
07AB            PDATA   EQU     $7AB
                *

160C                    ORG     $160C
160C 1D 37              FDB     BUFBEG

113A                    ORG     $113A
113A 1D 15              FDB     PRTEST

1CA4                    ORG     $1CA4

1CA4 20 42      START   BRA     BEGIN
1CA6            XTEMP   RMB     2
1CA8 00 00      SWPG    FDB     0
1CAA 46         MSG     FCC     /FIRST PAGE TO BE PRINTED?/
1CAB 49 52
1CAD 53 54
1CAF 20 50
1CB1 41 47
1CB3 45 20
1CB5 54 4F
1CB7 20 42
1CB9 45 20
1CBB 50 52
1CBD 49 4E
1CBF 54 45
1CC1 44 3F
1CC3 00 0D              FDB     $D,$A,,,,0
1CC5 00 0A
1CC7 00 00
1CC9 00 00
1CCB 00 00
1CCD 00 00
1CCF 54                 FCC     /TWO DIGITS PLEASE, EG 06/
```

```
 1CD0 57 4F
 ICD2 20 44
 1CD4 49 47
 1CD6 49 54
 1CD8 53 20
 1CDA 50 4C
 1CDC 45 41
 1CDE 53 45
 1CE0 2C 20
 1CE2 45 47
 1CE4 20 30
 1CE6 36
 1CE7 04                FCB    4
 1CE8 FE 71 0D  BEGIN   LDX    $710D
 1CEB 8C E1 D1          CPX    #$E1D1
 1CEE 27 22             BEQ    SKIP       NO PRINT COMMAND
 1CF0 C6 FF             LDA B  #$FF
 1CF2 F7 70 A3          STA B  SWITCH
 1CF5 CE 1C AA          LDX    #MSG
 1CF8 BD 71 18          JSR    PSTRNG
 1CFB BD 71 1E          JSR    PCRLF
 1CFE BD 71 0F          JSR    GETCHR
 1D01 16                TAB
 1D02 BD 71 0F          JSR    GETCHR
 1D05 58                ASL B
 1D06 58                ASL B
 1D07 58                ASL B
 1D08 58                ASL B
 1D09 84 0F             AND A  #$0F
 1D0B 1B                ABA
 1D0C B7 1C A9          STA A  SWPG+1
 1D0F B7 70 A3          STA A  SWITCH
 1D12 7E 03 00  SKIP    JMP    $300


 1D15 7D 70 A3  PRTEST  TST    SWITCH
 1D18 27 1A             BEQ    TEST3
 1D1A FF 1C A6          STX    XTEMP
 1D1D DE AC             LDX    PAGENO
 1D1F 08                INX
 1D20 BC 1C A8          CPX    SWPG
 1D23 26 05             BNE    TEST1
 1D25 7F 70 A3          CLR    SWITCH
 1D28 20 07             BRA    TEST2
 1D2A 37        TEST1   PSH B
 1D2B 5F                CLR B
 1D2C 53                COM B
 1D2D F7 70 A3          STA B  SWITCH
 1D30 33                PUL B
 1D31 FE 1C A6  TEST2   LDX    XTEMP
 1D34 7E 07 AB  TEST3   JMP    PDATA
 1D37           BUFBEG  EQU
                        END    START
```

NO ERROR(S) DETECTED

```
                    *
                    * MILAN KONECNY
                    * 193 CHAPLEAU AVE.
                    * DOLLARD-DES-ORMEAUX,P.Q.
                    * CANADA H9G1C3
                    * (514) 620-2263
                    *
                    * FEB 4, 1979
                    *
                    * TO INSTALL THIS FEATURE, THE TSC DATE
                    * UTILITY IS A PREREQUISITE.
                    *
                    * INSTALLATION INSTRUCTIONS:
                    *
                    * APPEND,ASMB.CMD.0,ASMDATE.BIN.1,ASM.CMD.0
                    *
                    * IF DESIRED, RENAME ASM.CMD.0,ASMB.CMD.0,
                    * AFTER DELETING ORIGINAL ASMB.CMD.0
                    *

                    * GLOBAL VARIABLES

 708E               DATE    EQU    $708E
 7112               PUTCHR  EQU    $7112
 7118               PSTRNG  EQU    $7118
 07BA               PCRLF   EQU    $07BA
 7133               OUTDEC  EQU    $7133
 07AB               PDATA   EQU    $07AB

 1188                       ORG    $1188
 1188 BD 1B F4              JSR    DATEO      SET ASSEMBLER JUMP TO DATE

 160B                       ORG    $160B
 160B CE 1C A4              LDX    #BUFBEG    CORRECT END OF ASSEMBLER

 1BF4                       ORG    $1BF4

 1BF4 20 03         DATEO   BRA    PDAT       BRANCH AROUND WORK BYTES
 1BF6 01            VN      FCB    1
 1BF7 00 00         VALUE   FDB    0
                    *

                    * PRINT DATE

 1BF9 B6 70 8E      PDAT    LDAA   DATE       GET MONTH
 1BFC 27 47                 BEQ    NODATE
 1BFE 81 0C                 CMPA   #12        TEST IF VALID MONTH
 1C00 22 43                 BHI    NODATE     JUMP IF MONTH BAD
 1C02 CE 1C 4E              LDX    #MONTH     POINT TO TABLE
 1C05 C6 04                 LDAB   #4
 1C07 4A            PDAT1   DECA               CHECK DATE
 1C08 27 08                 BEQ    PDAT3
 1C0A 08            PDAT2   INX
```

```
1C0B E1 00              CMP B  0,X
1C0D 26 PB              BNE    PDAT2
1C0F 08                 INX
1C10 20 F5              BRA    PDAT1
1C12 BD 07 AB  PDAT3    JSR    PDATA     GO POINT IT
1C15 86 20              LDA A  #$20      OUTPUT SPACE
1C17 BD 71 12           JSR    PUTCHR
1C1A 7F 1B F7           CLR    VALUE
1C1D B6 70 8F           LDA A  DATE+1    GET DAY NUMBER
1C20 B7 1B F8           STA A  VALUE+1
1C23 CE 1B F7           LDX    #VALUE    POINT TO IT
1C26 5F                 CLR B            CLEAR FLAG
1C27 BD 71 33           JSR    OUTDEC    PRINT DAY
1C2A CE 1C 49           LDX    #CST      POINT TO STRING
1C2D BD 07 AB           JSR    PDATA     PRINT IT
1C30 B6 70 90           LDA A  DATE+2    GET YEAR
1C33 B7 1B F8           STA A  VALUE+1
1C36 5F                 CLR B
1C37 CE 1B F7           LDX    #VALUE    POINT TO VALUE
1C3A BD 71 33           JSR    OUTDEC    PRINT YEAR
1C3D 86 04              LDA A  #$4       SET TERMINATOR
1C3F BD 71 12           JSR    PUTCHR
1C42 BD 07 BA           JSR    PCRLF
1C45 BD 07 BA  NODATE   JSR    PCRLF
1C48 39                 RTS


               * TEXT STRINGS

1C49 2C         CST     FCC    /, 19/
1C4A 20 31
1C4C 39
1C4D 04                 FCB    4

               * MONTH STRINGS

1C4E 4A         MONTH   FCC    /JANUARY/
1C4F 41 4E
1C51 55 41
1C53 52 59
1C55 04                 FCB    4
1C56 46                 FCC    /FEBRUARY/
1C57 45 42
1C59 52 55
1C5B 41 52
1C5D 59
1C5E 04                 FCB    4
1C5F 4D                 FCC    /MARCH/
1C60 41 52
1C62 43 48
1C64 04                 FCB    4
1C65 41                 FCC    /APRIL/
1C66 50 52
1C68 49 4C
```

```
 1C6A 04                FCB    4
 1C6B 4D                FCC    /MAY/
 1C6C 41 59
 1C6E 04                FCB    4
 1C6F 4A                FCC    /JUNE/
 1C70 55 4E
 1C72 45
 1C73 04                FCB    4
 1C74 4A                FCC    /JULY/
 1C75 55 4C
 1C77 59
 1C78 04                FCB    4
 1C79 41                FCC    /AUGUST/
 1C7A 55 47
 1C7C 55 53
 1C7E 54
 1C7F 04                FCB    4
 1C80 53                FCC    /SEPTEMBER/
 1C81 45 50
 1C83 54 45
 1C85 4D 42
 1C87 45 52
 1C89 04                FCB    4
 1C8A 4F                FCC    /OCTOBER/
 1C8B 43 54
 1C8D 4F 42
 1C8F 45 52
 1C91 04                FCB    4
 1C92 4E                FCC    /NOVEMBER/
 1C93 4F 56
 1C95 45 4D
 1C97 42 45
 1C99 52
 1C9A 04                FCB    4
 1C9B 44                FCC    /DECEMBER/
 1C9C 45 43
 1C9E 45 4D
 1CA0 42 45
 1CA2 52
 1CA3 04                FCB    4
 1CA4           BUFBEG  EQU    *

                        END
```

NO ERROR(S) DETECTED