WHAT IS IT?

So what is this little publication titled SCHELAL UPDATE supposed to be? Well, first of all it is just what its title denotes. A means of keeping registered SCHELAL owners up to date on the status of the program in regard to the correcting of 'bugs' that might appear, additional operating information that may be of interest to owners, clarification of points raised by users and so forth. More than that, however, this publication is sort of an experiment. It is an experiment to determine just how much our readers would like to participate in the process of refining the fundamental program as it has been presented in the SCHELAL manual, or participate in the creation and sharing with others, of application programs written to run using the SCHELAL interpreter.

The potential for tailoring a package such as SCHELAL to a wide variety of applications, of adding additional features, of improving its operating efficiency, is virtually endless. Are you, the user, interested in seeing this done? Do some of you want to participate in the arena? Would you like to have a vehicle such as this through which you could communicate with other users? Would you like to join with the program authors in improving and adding to the program? What groups of you like to work on specific sections? Would you like to have a medium for the presentation of application programs that use the language? Do you want to see application programs for games, or would you prefer programs that have more practical applications - such as programs for handling business, scientific and engineering problems?

You, the individual readers, are the ingredients in this experiment. It is you who will determine what direction(s) the experiment goes and what conclusions may be arrived at!

Write us, tell us what you think, send us your suggestions, tell us what you are interested in, remit your program ideas, and send us application programs written in SCHELAL!

(To avoid any possible squabbles, let us have it understood that submissions do become the property of SCHELAL C.C., INC. However, we shall point out that to sort of provide a little incentive, submissions we find worthy of publication receive an honorarium payment, which will, we are sure, more than cover the postage for such submissions.)

How far could this thing go? As has been said, that is up to you. We are simply providing the opportunity. We will be providing three or four issues during the next six months or so as a service to our SCHELAL customers. If, at the end of that time it appears there is a sufficient basis to support the concept, we are prepared to implement it on a subscription basis. If not, then, at least, we will have learned something from the SCHELAL UPDATE experiment, and, we are sure, so will you have!

You may address your comments on this matter, along with submissions to be considered for publication, to:

SCHELAL UPDATE EDITOR
SCHELAL C.C., INC.
1222 Rear - Boston Post Road
Milford, CT 06460

IS OUR FACE RED!

We pride ourselves at SCHELAL on accuracy. It is tough - preparing complex programs in the form of books - making sure that source listings and object listings get transmitted from computer print out to typeset without errors. For instance, three separate "proofs" against countless hours checking to ensure that the critical object code listings in chapters 12 and 13 of the SCHELAL publications were absolutely perfect. After that, the typesetting was used to verify proper operation of the program and to get an idea of how long it might take readers to implement the program on a computer using a keyboard loader. Six to twelve hours for most, depending on how well they can handle a keyboard.) Even after that checking it is a long wait between sending the copy to the printers and getting the first reports from readers!

At this time, a number of customers have already reported that they have SCHELAL up and running fine - so we are finally satisfied with the "proofing" part of the job. The printed copy does agree with our original.

Unfortunately, no matter how good a job our editorial staff does in preparing a program publication, the program authors can blow it all when they go!

Well, SCHELAL has been producing such publications long enough to know that it is down-right impossible to create a program the size and nature of SCHELAL and not find a few "bugs" or disagreeable features down the road after publication. That is the reason for providing some blank pages at the back of the book marked "NOTES." And, of course, a few bugs have shown up in SCHELAL at this point. These have been corrected by PATCH1 and PATCH2 which are pasted into the first edition of SCHELAL on the NOTES pages in the rear of the book prior to shipping.

The problem that necessitated PATCH1 did not show up until just a few days before the first lot of books were due to arrive from the printer. This meant, in order to ship promised books on time, that PATCH2 had to be created and rushed to print quite hastily! The program authors, in conference, quickly arrived at a suitable solution to the problem and created PATCH2. Author Arnold suggested that the patch be placed at the end of memory page 32 where there was plenty of room for such a patch. Author Wadsworth, aiming to "save such a large unused area for a REAL EMERGENCY!" thought he saw another location that the patch seemed to just perfectly fit into starting at location 224 on page 32 in memory! Since author Wadsworth had been designated as overall program manager for SCHELAL, the editorial staff hastily arranged to have the patch printed up to reside starting at that location 224 on memory. Then PATCH2 arrived from the printer the same day that SCHELAL books arrived and were duly packed in books were packed for shipment.

Also, as a number of our ever alert customers quickly noted, (cont. pg. 3)

1
SCHELBAL AVAILABLE ON PAPER TAPE!

For several years now the company has been producing programs in the form of books - leaving it up to individual users to load programs into memory using keyboard loaders. In the past, with the majority of programs falling into the under 8K category, most readers were content with the “book only” delivery method. Apparently, going to a 8K program has been a number of customers finge out of joint. We have had quite a few request for paper tapes of the object code, and a number for the source listing.

We are going to start with making the object code available. (The source listing may be made available at a later date? )

One of the reasons the company has not been in any great hurry to start providing programs on paper tape was because of the lack of standardization of format. While there are still many formats in use, it is the consensus here at SCHELBI that the Hexadecimal Paper Tape Format promuligated by Intel Corporation for use in their INTELLEC MCS* (*TM) is a suitable compromise among the many possibilities and one that is most familiar to industry and university users where the majority of the requests for such tapes appear to be coming from in our analysis.

Several features that the firm’s staff considered worthy in this format include its frequent testing for reader errors and capability to recover from an error condition by simply backing up a few inches to the last block read successfully (instead of having to re-read an entire tape); the header style block format that allows different areas in memory to be loaded, and the fact that, when used with a typical ASCII tele-type system, the tape itself can generate a hexadecimal listing of the data on the tape for checking and reference purposes.

Thus, it is being announced that the official standard at SCHELBI for core images produced on paper tape for the firm’s products will be the Hexadecimal format which is detailed below.

HEXDECIMAL FORMAT FOR PAPER TAPE

The hexdecimal paper tape format that will be used by SCHELBI for core images consists of the following.

A paper tape will contain one or more blocks of information. Each block will be a self-contained unit that includes a header containing information regarding the location of the information in the core area (an address), the amount of data contained in a block (a data byte count), a record type indicator, the actual data in hexadecimal notation, and a checksum. The start of each block of information will be indicated by a special character. All of the information within a block will be arranged in the order illustrated next on a row-by-row basis.

ROW 1 - Start of block mark consisting of the ASCII character code for the colon sign (:).
ROW 2,3 - Block length count consisting of two hexadecimal
character (0SD then LSD). The block length count refers to the number of actual data bytes in a block. This value may be in the range 00 to 0P (0 to 255 decimal). However, a count of zero (00) will indicate an END CF FILE block.

ROW 4 - Address at which data will begin to be loaded in memory expressed as four hexadecimal ASCII encoded characters.

HIGH address then low address.

ROW 8.9 - Type of block indicator. For standard core images this indicator will consist of the two ASCII encoded characters 00. Other types of indicators may be used in the future.

ROW 10...X - Data. Each byte of data to be loaded into memory will be expressed as two ASCII encoded hexadecimal characters (0SD,0LSD) requiring two rows on the paper tape.

ROW X+1, X+2 - Checksum. Expressed as the negative of the sum of the value of all rows in the block since the start of block marker (neglecting carries).

NOTE: Paper tapes punched in hexadecimal format will use the convention of not using the parity bit (eighth bit). This is opposite to the convention established for most SELB1 programs. The decision to follow the convention for the paper tape format was based on fostering compatibility and increased standardization, at least in the area of program loading capability.

PLEASE!!!

Do NOT write and ask us for SELB1 on magnetic tape! We will NOT be supplying magnetic tapes until such time as we are satisfied that there is a fairly stable agreement concerning recording methods and formats.

At this time we are watching the progress of the "K.C." standard closely. However, we feel it will be at least six months to a year, and possible longer, before standardization has set in to the degree that we will invest in the necessary equipment, personnel, etc., to start providing programs on magnetic tape.

BUT - you may write and ask for information concerning paper tapes of other SELB1 programs. We will soon be making paper tapes available for most of the programs presented in previous SELB1 publications - such as our Editors, Assemblers, Monitors, Games, etc.

NOTE - paper tapes supplied by SELB1 will be virtually unaltered if you do not have the corresponding publication! They are being made available as an optional supplement to the book - not as a replacement. Users will still have to provide I/O routines etc., as described in the related books and information regarding the locations of such routines, operating instructions, etc., will NOT - except - NOT be supplied with the paper tape!

FEEL RESTRICTED BY BEING LIMITED TO 20 VARIABLES!

You shouldn't... when it is so easy to essentially quadruple this capacity by using a set of elements in an array as individual variables! For instance, instead of using a group of variables such as N1, N2, N3, N4 simply DIMENSION an array (in this case having nine elements) named N.

DIM N(N)

Then use the elements N(1), N(2), N(3), etc. as different variables. Using this technique you can add up to 64 more variables in a program for a total of 84.

A program utilizing 64 variables will be a very "busy" program.

FA? FA? FA? FA!

Out! We forgot to tell you something. While it is not mentioned in chapter four (see the list on pages 19 and 20 in that chapter), nor is it shown on the handy pocket reference card included with the book (bound in the back with the registration card), the symbol FA is a valid SCERL1 error code! It means that the interpreter has encountered a Function or Array error condition.

Why not pencil in a note to that effect on your pocket reference card? The error code is especially likely to come up if you do not have the DIMENSION capability included in your version of the program (and have substituted NOPl for the localized memory locations) and then attempt to perform an operation that specifies an array element.

THINKING OF ALTERING PORTIONS OF SCERL1?

Individuals planning to modify small sections or subroutines can probably do well enough using hard assembly methods. However, those who plan to undertake extensive revisions - such as, for example, compacting the program by taking advantage of the 8000X's extra instructions - would do well to remember that SCERL1 has numerous programs suitable for such tasks that operate in just 4K of memory (and can use memory beyond that amount to provide extensive symbolic storage). The SCERL1 8020 ASSEMBLER program is designed to process the mnemonics as they appear in the SCERL1 manual (original NTEL mnemonics for the 8080) as well as providing for the extended instruction set of the 8088 CPU. See SCERL1 advertising literature for additional information.

(from pg. 1)

author Waddsworth's choice of location for PATCH2 overlooked the fact that locations 324 and 225 on page 32 were already occupied by the address bytes of the instruction JMP ERROR that would be executed if a Square Root error (negative argument) condition was encountered. Author Waddsworth, after mumbling something about "it was just a test to see if the readers were awake" agreed to relocate the patch to start at location 364 on page 32. A new "PATCH2 - Revised" was printed to replace the original patch number two. The revised version is included in books currently being shipped. Early customers who received the original patch will find a copy of the revised (simply relocated) PATCH2 enclosed with this literature which may be pasted over the original version. To erase all evidence ...

...as though the whole thing never occurred!
MODIFIED SCHELAL

This is the beginning of a section that we plan to have on a regular basis in SCHELAL UPDATE. The purpose of this column will be to present modifications to SCHELAL that will provide some improved operation, or desirable features to the fundamental program. Users are invited to contribute to this column.

In order to maintain some kind of overall organization of the fundamental program as various improvements are thought of, and suggestions for implementing those improvements made and/or contributed, it would be wise to lay out a few rules for contributions to follow. While these rules may not be considered as hard and fast at this point, they will at least serve as an initial guide. More "rules of the game" may become necessary as other join in the fun.

In the example modification to be described in this issue, the following rules were adhered to:

1. The improvement was made by altering the machine code within the machine range defined by the label.
2. The modification is essentially complete and well-contained within the boundary established in item number 1 above. That is, it was not necessary to "patch" the program by establishing subroutines external to the area modified.
3. The improvement does not rely on any improvement or modification. Adherence to this rule will insulate that readers do not end up with a problem of having to refer to previous modifications ad infinitum. Note that this does not mean that a new contributor cannot modify an improvement. It simply means that the presentation should include all modifications and references to the original version of SCHELAL, and not the modifications. Of course, if in doing so one wants to reference an improved subroutine for purposes of discussion or to indicate a point of inspiration, one should certainly do so.
4. This column will relate only to improvements that can be implemented on a 8080 CPU based system. The optimization of SCHELAL for an 8085 is an entirely different matter which will be discussed at a later date.
5. The improvement does not alter the starting address or the size of the program so that it can be implemented on the original version of SCHELAL for an 8080 is an entirely different matter which will be discussed at a later date.
6. If the improvement or alteration does not alter the starting address or the size of the program so that it can be implemented on the original version of SCHELAL for an 8080 is an entirely different matter which will be discussed at a later date.

The modification to be described in this issue is a minor one. It involves changing a few lines of code to make the program more efficient. The change is made by inserting a new subroutine that reduces the number of instructions needed for a particular task. The new subroutine is called "HELAL," and it replaces the original subroutine with the same name.

Following these initial guidelines, you should be prepared to prevent confusion as contributors with various interests begin to point out ways in which the program may be improved, incorporate additional features, or possibly correct any parasitically troublesome situations.

As pointed out in chapter 15 of the book, SCHELAL was deliberately published, not as a highly compacted, intricate, ultracomplexed program that would have been too difficult to safely modify, but rather in a format that was more conducive to expansion and alteration. The reader with a minimal amount of machine language programming capability will be able to find all kinds of ways in which various portions of SCHELAL might be modified to suit individual taste. The range of modifications that one can envision are virtually too numerous to enumerate. Some readers might be interested in finding ways in which to speed up the operation of various sections of the program. Other users might be interested in expanding the program in ways which may be considerably compact the amount of memory the program will use. (Again, reference here is made to the 8080 version. Obviously, SCHELAL can be considerably compacted if the 8086 instruction set is utilized. As pointed out earlier, however, that matter will be handled separately from this column.)

The modification to be described in this issue is a minor one. It involves changing a few lines of code to make the program more efficient. The change is made by inserting a new subroutine that reduces the number of instructions needed for a particular task. The new subroutine is called "HELAL," and it replaces the original subroutine with the same name.

Have you ever created a SCHELAL program and found that it used far more variables than it ever will in the future? If you have, you probably did not discover your error until you attempted to run the program and received a BG error message. After some head scratching, when you finally figured out that the problem was caused by too many variables, you attempted an easy solution by combining mathematical statement lines to reduce the number of variable names. Also, however, you discovered that after modifying the program you were stuck in a nasty situation. Every time you tried to run the program that BG error message came back again.

Why? Because eliminating a variable name from a program statement does not eliminate that variable name from the variables table. The variables table remains filled. How does one normally get rid of that situation? By use of the SCR command.

Unfortunately, while this continued does indeed clear out the variable names table, it also clears out the user program buffer, making it necessary for the programmer to re-enter the revised program. This may not be so difficult if the user has high speed disk storage facilities and can utilize the LOAD command. Nor is it tough if the program is relatively small. However, in some cases a program overflowing from excess variable names will have been a relatively large program and re-reading it by keyboard may be a little frustrating.

A user that has really studied SCHELAL, and that has a resident Monitor facility on their computer system might discover that a shortcut to getting out of that type of situation would be to use the DEINIT command to initialize the variables table to the effectively empty condition. This can be accomplished by placing a zero byte at the start of the regular variable control table (which is at address PG 27 LOC 2312), and, re-initializing the value in the variables counter at PG 27 LOC 077 to a value of 001.

That action is one of several that is performed when a SCR command is issued. But, the SCR command also results in the user program buffer being effectively erased. It might be possible to find how this could have two types of initializing commands. One would be an all-inclusive
initializing command just like the SCR command; the other would be a special command that only initialized the variables symbol table.

The modification presented herein provides that capability by replacing the SCR command with two single letter commands. One single letter command signifies the letter E for "scratch" provides the all-inclusive initializing capability for the interpreter. The second command signifies by the letter R for "reset" allows the programmer to effectively erase just the variables symbol table while leaving the user program buffer intact.

This improved capability can be provided by modifying the section of SCBELAL that starts at the label NOLIST and ends with the label NOSCR. The source listing for the original version of this section is discussed in chapter 4 on pages 6 and 6. The area in the assembled listing starts on PG 10 LOC 354 and ends at PG 11 LOC 066.

The source listing of the modification that follows illustrates how the improvement was affected by re-organizing the order in which specific initializing actions were taken; splitting the original SCR command in the command look-up table into two character strings, one containing E R; the other an E and "lightening up" the program a little by sacrificing the possible contents of the D and E and the M and CPU registers wherever the program proceeds from the STRCP subroutine.

Assembled object code listings of a modification for both the 8208 and 8080 processors are presented on the following page.

To operate the modified version, simply remember that the SCR command has been replaced by the single letter command S. Additionally, a new command, invoked by entering an E followed by a carriage return when in the executive mode, will cause the array and regular variable symbol tables to be effectively erased without disturbing the contents of the user program buffer.
The first two issues of SCIRL’s UPDATE were sent to all purchasers at their purchase addresses. It cost a considerable amount of money to send out copies of SCIRL’s UPDATE. Future copies will only be sent to those purchasers who have had their names or addresses printed on the last page of your SCIRL book.

Now you can cook-up hot programs on your "8080"


Order your copy of SCIRL’s "8080" Software Gourmet Guide & Cookbook today! Only $4.95 post. Bon appetite!

A PLUG FOR CREATIVE COMPUTING

The game presented on the next page is a slightly revised version of a program that appeared in an excellent magazine that is appropriately named CREATIVE COMPUTING. The magazine is published by an enthusiastic and creative organization headed by David H. Ahl. In addition to games such as that shown in this issue, the magazine regularly presents a variety of articles, book and product reviews, educational material, and a good selection of general information which we feel most of our customers would find highly interesting. Recent issues of the magazine contained 88 pages (95 x 13). Considering the fact that relatively little advertising appears in those 88 pages, the amount of text and editorial material per issue far exceeds any other computer-related publication that has come to our attention to date. Individuals interested in subscribing to the publication may do so at the following rates. 1 year - $8.00, 3 years - $21.00. If you have any doubts, you can sample one copy for $1.50. (The magazine is issued bimonthly.) Subscription orders should be forwarded directly to:

CREATIVE COMPUTING
P. O. Box 729-M
Morristown, N.J. 07960

3
Mr. R. J. Toy is one of those hearty souls who utilize a Baudot encoded teletypewriter with his computer system. These machines are generally considerably less expensive than the sought after ASCII encoded devises. We don't know how many other SCBELAL users may be using the same type of machine but we thought Mr. Toy's comments related to the use of such a machine - and other matters, would be of interest to all. (Users with Baudot machines might be interested in communicating directly with Mr. Toy on mutual grounds.)

When Mr. Toy originally received his copy of SCBELAL he was apparently a little crest-fallen when he discovered the limitations on the use of CPU registers specified in the book. The recommendation that only CPU registers A and B be used for I/O routines met with the following comments...

"Since the accumulator is loaded with the data to be inputted or outputted this really leaves only register B. I normally need H and L for each input and output operation.

After reading about for several days trying to decide what hardware modifications had to be made, I finally decided to look into the possibility of program modification. To my surprise I found that the I/OFO routine leaves H and L free, so there is no problem on output.

The input situation, however, was not as easy. After considerable study I concluded tentatively that D and E were free. So I went ahead and developed several I/O routines on this basis. The results so far indicate apparent success. (But wait - read on some more! Ed.) I have now tried everything in the chapter on operating SCBELAL up to and including program modification with the correct results, with one exception. In addition, simple problems in addition, subtraction, multiplication, and division yield the correct answers.

The one exception mentioned above was that the TAB function did not work properly. Instead of all spaces between "HELLO's", the first character was a space as expected but the rest were something else. A study of this problem revealed that at least for TAB the content of the accumulator must also be saved on output. To make a long story short, the simplest solution was to change the contents of 015 010 from 003 to 021. This reloads the accumulator with a "space" each time a space is supposed to be sent. Don't change your system yet - read on Ed!"

A few days later another letter was received from Mr. Toy and the discussion started above was continued...

"On the matter of the TAB function, my original quick fix turned out to be for the commas controlled routine only, PCOM1. It is also necessary to similarly modify TABF03 for the numerically controlled spacing, and the BACKSP for backspacing. The latter would require a patch so I gave up on this tack, modified my output routine to save and restore A, ... etc. Inclined, PCOM1 and TABF03 are identical except for addresses so one of them can be eliminated if memory space is needed."..."

Mr. Toy then went on to a new topic...

"I have tried all the example programs in the SCBELAL manual except for the last one. They all appear to operate properly except the two programs involving the CHR function on pages 14-24 and 14-29. In the table program the last character of the decimal number comes out as a letter. In the line printing program only the first character in the line comes out correctly. Unless my I/O routines are associated with these programs, which seems unlikely, it would appear that registers B, D and E are free on input, and B, H, and L free on output. In addition, on output the data should be stored in the correct memory location.

Mr. Toy must really be working his system out because in a few more days he has the following comments...

"After several hours of hard labor I finally found out why the CHR program on page 14-29 is so complicated that it requires about half a second for each character to be processed before the program looks for the next character! This delay seems to be unusual, so readers may well be advised of this fact in connection with this particular program, especially if they are using an 8080." (True - the delay is rather disconcerting on an 8080 based system. 8080 users, however, will find the delay barely perceptible. Ed.)

"I still have not determined why the decimal numbers in the CHR table program on page 14-24 do not come out correctly. However, I am now reasonably satisfied that my I/O routines work properly on all functions, so I will not spend much more time on this. For your information I am enclosing a printout of my results..."

Please note that I have substituted a dash for the READY message. This involved changing only two bytes in SCBELAL: 001 352 is replaced with 021 and 001 353 is replaced with 210. The result is a single line space for "READY" instead of three. This uses up less paper, especially when operating in the "calculator" mode.

Not one to give up, Mr. Toy soon followed up with...

"I finally discovered why the program on the Table of ASCII characters would not work. An 'X' in statement 130 was missing. A printout of the correction and a RUN enclosed. You may also be interested in the substitution of characters to use the model 15 TTY..."

THE EDITOR REPLIES

Communications of the type Mr. Toy has submitted are exactly why we established the support publication SCBELAL UPDATE. It is through such communications that SCBELAL itself can be improved and tailored to suit the requirements of individual users or groups of users. Mr. Toy's letters are the first of what we hope become a flood of helpful tips from users engaged in disseminating information about SCBELAL amongst its users.

Now, to answer or explain a few of the questions raised by Mr. Toy.

Mr. Toy has apparently made some very useful discoveries in regards to the availability of certain CPU registers during I/O operations. His observations should be of considerable interest to users with special I/O devices who find they need more CPU registers available. The stipulation made in the publication regarding the use of CPU registers to just A and B was given on the basis of design guidelines that the program authors established. In other words, the program authors, during the development stages, reserved those two registers for use during I/O operations, so that they would have the freedom of using all other CPU registers if desired. They did not, during the development process, keep track of whether every other possible register was thus actually in use during I/O operations. Mr. Toy's observations are as interesting to the authors as they may be to others and may be taken for what they are worth. (Which is a lot if your running a Baudot machine!)

Mr. Toy's observation regarding the saving of the accumulator's original status during an output operation that utilizes a TAB is correct. The users output routine should exit with the original character in the accumulator still present.

Our thanks to Mr. Toy, (and our apologies to all readers) for discovering the clerical error on line number 130 of the example SCBELAL program on page 14 of chapter 14. The line should read:

130 Q=INT(N - 64Q1: 8Q2)

The suggestion regarding the use of a hyphen to shorten the READY seems like a good one for those that want to implement it.

Users who anticipate using a Baudot coded device might be interested in contacting Mr. Toy directly to discuss I/O routines etc. His address is...

Mr. S. Joseph Toy
Route 3, Box 73
Chico, CA 95926

5
SCELBAL II UNDER DEVELOPMENT!

As SCELBAL owners know, SCELBAL was developed primarily for 8008 system owners. There were several reasons for doing so. First, when SCELBAL COMPUTER CONSULTING, INC., first went into business, it produced a microcomputer based on the 8008 CPU. A number of those systems are still out in the field and many owners had indicated a desire to have the capabilities of a high level program available. We no longer manufacture microcomputer systems, but we felt an obligation towards those who had helped us pioneer in the field of the personal computer.

Second, in addition to those 8008 microcomputer systems sold by SCELBAL, there were several thousand similar systems (8080 based) known to be in existence produced by other early microcomputer system manufacturers along with numerous personal systems based on the MARK-8 article that appeared in RADIO-ELECTRONICS magazine some two years ago. Many people had written to us indicating that they felt the rapid growth of the acceptance of the 8080 and other more advanced CPUs, and the attention they were receiving, would mean that early 8080 users high and dry without ever having a high level language developed for it.

Third, we felt that developing such an interpreter for a micro CPU as primitive as the 8080 is now considered, instead of being a waste of time (= apparently everyone else thought it would), would be a valuable experience. After all, if it could be accomplished for such a primitive CPU, upgrading the fundamental concepts and routines from that point to take advantage of the increased power of instruction available on more advanced CPUs would be a pretty straightforward task.

Additionally, we of course knew that an interpreter written for an 8080 could be directly assembled to operate on an 8080 even if it was not "efficient" in making use of that CPU's capabilities. This meant that, though many users were planning on eventually upgrading their personal systems from an 8008 to an 8080, with the existence of SCELBAL, could do so without having to modify a single one of their SCELBAL higher level programs.

Finally, it was felt that presenting SCELBAL in detail, with complete source listings, flow charts, etc., for the 8008 CPU, in the manner in which it was done (not using any page zero, not trying fancy page packing tricks, etc.) would result in an information source which could have fun! One can pick almost any section one might be interested in and find ways of incorporating it, by using better coding techniques, etc. 8080 owners, as pointed out in chapter fifteen, could go to work with vigor on improving the program if they so desired because it is that those upgrading from an 8008 to an 8080 do NOT have to modify the interpreter to increase its efficiency if it is not interested in doing so!

More than all those factors combined, however, SCELBAL was developed with the intention to become an ever-evolving program. As new machine types became available, SCELBAL could be adapted, as users became more sophisticated in their demands for program performance, SCELBAL could be upgraded. Since the entire fundamental organization and logic of the interpreter had been presented, users would not be forced to wait for such advances to come from SCELBAL if they had the desire and capabilities to proceed on their own.

Naturally, many users of SCELBAL do not wish to become involved with the intimate details of the interpreter's operation. They just want to be able to use the end result. Fine. SCELBAL intends to continue to improve the program as well as to provide the language for other types of microcomputers when it appears that there is a market sufficient enough to justify the expense. It is hoped that by listening to the thoughts of many other users, and by providing an opportunity for others to communicate their needs, the overall quality and capability of SCELBAL can be improved. Indeed, there is no end in sight to the potential. The limited factor, w in most endeavors, is time and money.

Even as the first copy of SCELBAL was published, work was underway to produce a revised version that would capitalize on the increased power of the 8080 instruction set (that of the 8080). Work is proceeding smoothly. Feedback from SCELBAL customers who are 8080 system owners indicate they are highly interested in such a revised package.

Essentially, the revised version titled SCELBAL II will simply be a compiled version of the original program. It will remain organized in essentially the same manner, using the same subroutine names, etc., so that the original publication will initially re- main as the prime reference. Preliminary indications are that the 80805 customized version, with DMA/enuence capability, will reside in about 8K of memory (without using page zero). A few minor operating improvements (such as increasing the number of variable names allowed) are planned. The possibility for the inclusion of other features remains open at this point pending feedback from users. (By this it is meant operating improvements. The addition of extended functions such as data storage, routines, exception, string handling capabilities and so forth constitute not merely improvements, but actually the creation of additional features. More has and will be said about such matters in other articles.)

How long before SCELBAL II will be released? Probably another few or six months. We will provide time for plenty of feedback from users to try and catch any gremlins or add needed improvements. Registering with SCELBAL owners will be notified when SCELBAL II is available. Chances are you will hear more about its development in these pages as it progresses.

In the meantime, if your interest is evicted (now dormant) to work on such a project yourself, the following information may help you get off to a sound start. Reversing the storage format for those critical double-byte values used in SCELBAL will enable you to capitalize on a number of the disk file system manipulating instructions. These storage locations are all on page 16 (octal). They are the locations used to hold the User Program (Line Pointer (160 & 361), the Auxiliary Program Buffer Pointer (362 & 363), and the End Of Buffer Pointer (364 & 361).
Letters

I don't know how many people might be interested in the following modification for SC/68, but it may have some useful utility.

Since time to time I find desirable to rearrange a table of data so that the rows are arrayed in a string that is the same thing.

From time to time I find desirable to rearrange a table of data so that the rows are arrayed in a string that is the same thing.

One advantage of this method is the large buffer space available. Another advantage is that the data is easily stored by using the SAVE command.

One advantage of this method is the large buffer space available. Another advantage is that the data is easily stored by using the SAVE command.

The ROADRACE program presented in ISSUE 03 of SC/68 was provided courtesy of CREATIVE COMPUTING. The magazine CREATIVE COMPUTING is published by an enthusiastic and creative organization based in New York. In addition to games such as that presented in ISSUE 03, the magazine regularly presents a variety of technical, book and product reviews, educational material, and a good selection of general information which we feel most of our customers would find highly interesting. Recent issues of the magazine contained 88 pages or more in an 8 1/2 by 11 format. Considering the fact that there is relatively little advertising space allotted in those 88 pages, the amount of text and editorial material per issue far exceeds most other computer-related publications that we have seen of late. Individuals interested in subscribing to CREATIVE COMPUTING may do so at the following rates: 1 year: $8.00, 3 years: $22.00. If you have any doubts, you may obtain a sample copy of a recent issue for $1.50. (The magazine is published bimonthly.) Subscription orders may be forwarded directly to the publisher:

CREATIVE COMPUTING
P.O. Box 769-M
Morristown, N.J. 07960

HEY! WE FORGOT TO TELL YOU....
STRINGS SUPPLEMENT
NOW AVAILABLE

The Strings Supplement to SCHELP is now available. The 68 page booklet (8 1/2 X 11) may be obtained for $10.00 from the publisher at the address shown below. The booklet provides the source code and assembled object listings for both 8088 and 8086 systems for routines that will enable SCHELP users to add String Function capabilities to their systems. Users intending to add the Strings capabilities should have a minimum of 12K memory (read and write) available in their system.

Details of the Strings Supplement capabilities were provided in Issue 03 of SCHELP UPDATE.

The $10.00 price for the Strings Supplement includes postpaid delivery by U.S. Mail service. Address orders to:

ORDER DEPARTMENT
SCHELP INC.
PO BOX 133
PT STN
MILFORD, CT 06460

COMING SOON
EXTENDED MATHEMATICAL FUNCTIONS FOR SCHELP

New in the final documentation stages are five extended mathematical functions soon to be made available for SCHELP users. The new functions, which will be made available as a supplemental publication, will provide users with the following additional capabilities when installed: SIN, COS, EXP, LOG10, and ATN. The SIN and LOG functions are calculated using Ceyhun optimized Taylor series. The EXP and ATN are calculated using continued fractions. The COS function is calculated using the SIN function. The argument range for the functions will be as follows:

\[
\begin{align*}
\text{SIN} & : -1943803 < x < 1943803 \\
\text{COS} & : -1943803 < x < 1943803 \\
\text{EXP} & : -89 < x < 89 \\
\text{LOG} & : 0.001 < x < \infty \\
\text{ATN} & : 0.001 < x < \infty \\
\end{align*}
\]

The soon to be available booklet will contain source and object listings as in all other publications related to SCHELP. Prospective String Function users should note that assembled object listings for the mathematical functions will reside in the same memory locations (pages 50 through 54 octal) as various string routines. The overlapping was based on the premise that from memory space considerations (particularly for 8089 based systems) users would not find it practical to have both string functions and mathematical functions installed at the same time. (String function use theoretically is less likely to be concerned with extended mathematical functions it seems.) Users who might desire to have both types of capabilities installed simultaneously would need to relocate one set of routines and would probably want to have 16K or more of read and write memory available in the system.

It is anticipated that the extended mathematical function routines will be available in the form of a SCHELP Update-52 supplement at a price of $5.00 including postpaid delivery by U.S. Mail.

PREMIUMS FOR YOUR PROGRAMS
APPLICATION NOTES
ARTICLES
COMMENTS

If you have developed your own, original programs to perform tasks that may be of interest to other SCHELP users, chances are you are in a position to pick up a bit of cash! User submitted programs accepted for publication by SCHELP earn an honorarium check and a nice certificate blessing the author's performance! We are particularly interested in programs that may be of value to scientists, engineers, and small businesses. However, games, and general purpose routines are frequently accepted.

But, you don't have to be a SCHELP programmer to earn some cash. We are also interested in seeing articles of general interest to SCHELP users, as well as application notes, and even comments or suggestions!

You may submit your efforts to the address given below. Material accepted for publication earns the author an honorarium check based on originality, usefulness to readers, length, completeness and quality of presentation etc. Submissions accepted for publication become the property of SCHELP Inc., Inc.

The set of submitting for publication is certification that the material is original and that the author agrees to the terms of this announcement. While every attempt will be made to return rejected material accompanied by a SASE (self-addressed, stamped envelope) SCHELP Inc. assumes no responsibility for submitted material.

Material to be considered for publication should be forwarded to:

SCHELP UPDATE EDITOR
SCHELP INC.
PO BOX 133
PT STN
MILFORD, CT 06460
Several notes of caution are in order. First, the modification as shown in the accompanying listings is for the essentially unmodified version of SCBELAL as presented in the basic publication. If you have made modifications to your version, be careful. Same goes if you have implemented any of the supplementary listings.

In particular, if you are playing around with comparing SCBELAL for an 8080 machine and have changed the order of the bytes stored in the END of User Program Buffer Pointer (Page 26, Locations 364, 365) as mentioned in SCBELAL UPDATE Issue 04, you will have to change things around a little bit in the accompanying listing in the vicinity of the LOOKUS subroutine at Page 05 Location 157 etc.

If you have installed Strings or Mathemetical Supplements or if your User Program Buffer storage area does not end at Page 54 Location 377 in your system, you will need to alter the values in the accompanying listing marked with a "**" notation in the comments section (such as Page 05 Location 54 and Page 11 Location 44) so that the end of the User Program Buffer storage area is set up properly by the newly unmodified variable modifications.

It is assumed that those who have otherwise modified SCBELAL or relabeled the program, will know how to proceed to adapt the modification.

Finally, a note of caution. The modification checks to see that variables do not run into a user's source listing. However, no check is made to ensure that the user buffer does not run into the variables table. It is thus theoretically possible to "bom" the variables table if one was, for instance, inserting new lines into a source listing and altering with the RUN mode to test the operation of the program being developed. If it looks like storage will be tight in a program, load the sources before executing a RUN command! Since variable names are added to the variables table as a program is executed, the modified program will indicate if buffer space is exhausted.

Have fun with the new capability!
The supplement contains source and object listings as in other publications related to SCRELB. The assembled object listings provided reside in locations on pages 50 through 54. They may be reassembled to reside elsewhere to suit the user's desired string Function needs. We should note that those same pages are used by sections of the String Functions.

The price of the Mathematical Supplement to SCRELB is $3.00 in the U.S. including mail delivery. Foreign purchasers should include $2.00 for normal delivery of the supplement.

A FEW CORRECTIONS

C. A. Bannister of Richmond, VA, was the first to report some object code errors in the listing for modified SCRELB shown on page 3 of SCRELB UP. DATE issue 02. The object code errors only occurred in the 8008 listing.

It seems that the object codes for XRA, LMA and LLA directives got fouled up in the listing. The code for XRA should be 200, for LMA it is 250 and for LLA it is 350.

Alert Bannister also noted a typographical error on the first line of Mr. Toy's routine shown on page 2 of issue 04. The code for LLI should be 068 not 066 as printed.

Thanks for the use of your sharp eyes - and our apologies to our readers for letting those errors get by. - Ed.

STIRNGS PATCH

Mr. H. J. Lewis of Canada has spotted a glitch in the Strings Supplement. The following patch, named in his honor, was installed at Page 50 Section 37:

JZPF HJLFIX

It will replace the JZPF SSTECL instruction. The patch, which may be placed on Page 54 at Location 301, is just two instructions:

HJLFIX, CAL SWITCH JMP SSTECL

This patch will correct an anomaly in the string comparison routines that can effect string comparison operations.

Many thanks to Mr. Lewis for his persistence in analyzing and solving this problem and bringing it to our attention! - Ed.

What is the VALUE of VAL?

String functions are designed to allow the user to manipulate "strings" of alphanumeric characters instead of mathematical quantities.

However, there may be times when it is desirable to manipulate information in essentially two forms - as a string of characters, and as a numerical value.

Suppose, for instance, one wanted to have the computer make a list of groceries showing the price for each item, and then also mathematically sum the prices to obtain a total.

| TOMATOES | 24 |
| LETTUCE | 79 |
| CARROTS | 38 |
| ORANGES | 98 |

One could use string capabilities to list the items and their prices. But the character strings themselves are useless for calculating mathematical information unless one has the special capability to convert between one mode and the other. That is what the VAL function in the SCRELB String Supplement provides!

The VAL function converts characters in a string from an ASCII representation of a decimal number to its numeric value. In other words, the prices in the example can be converted from character string format to actual decimal values that can be mathematically manipulated by SCRELB!

Assume the lines in the above example are each composed of two strings "A" (item) and "B" (price). The "price" strings in the example could be converted in string arrays B(1) through B(4). One could obtain a numerical value for the total of all the prices in the list with a routine such as:

FOR 1 = X TO 4
LET T = VAL(B(X)) + T
NEXT X
PRINT T

This is because the VAL function would convert the numerical character strings to mathematical VALUES!

If reader interest warrants, we will discuss capabilities of the String Supplement for SCRELB and some more in the next issue of this publication.
SCELBAL READY FOR RELEASE

For sometime there has been a question as to whether or not SCELBAL-II would ever be released in source form. In appreciation of our early customers, a compromise has been reached. As detailed in a separate flyer that will accompany this edition of SCELBAL UPDATE, the revised edition developed specifically for 6800/2/80 systems will be made available to registered SCELBAL owners for a modest fee as an unommented assembly source listing. Since SCELBAL-II essentially follows the general structure of the original version, SCELBAL owners with 8080 or 2/80 systems should find the improved version attractive and understandable. Those not having the original SCELBAL documentation would likely find it somewhat discouraging to attempt to decipher the un-commented listing of SCELBAL-II. In any event, SCELBAL-II will only be made available to purchasers of the original SCELBAL documentation.

THIS TO BE LAST ISSUE OF SCELBAL UPDATE

As we indicated what we began publication of this journal, the objectives of this supplementary publication were multiple-fold. First, it would provide a vehicle for informing SCELBAL customers of program corrections that were liable to be required in a program and scope of an interpreter. Second, it would be an experimental publication to determine if users wanted to work through the publication to amplify the package in any way. We said we would provide this publication, free for a limited period of time, and possibly on a subscription basis thereafter, if users showed this is what they wanted.

Well, this free period is over, and support for such a publication on a subscription basis has not been demonstrated. Only a handful of readers have submitted material for publication-even though an honorarium is presented for published material. Only a fraction of 1 percent or readers have expressed any interest in having the publication continue on a subscription basis.

The journal has lived up to its task of informing SCELBAL users of program bugs discovered by users over a more than sufficient time span. SCELBAL, with minor alterations pointed out in this journal, is a proven interpretative language. Best wishes to all its users!

BOWLING HANDICAPPER IN ONLY 512 BYTES!

Harold F. Bower has been running SCELBAL in an eight-bit 8080 system for some time so he had a limited 512 bytes of user program storage room. That didn’t stop him though. He sent in the following program that has been helping him calculate information used by bowling leagues.

Input total games to date
Input scratch scores
Input previous scratch total
Input previous total pins-keeping this list saves problems with changing players in singles leagues
Input player’s previous handicap

The above three lines give formatted output of scratch total, handicap total, and cumulative total sortable for a 32 column TV display.

Goto 20

End

Harold says that while the above program requires quite a few more manual entries than would be required if master files were maintained in string format, and could be saved then later loaded and modified with the new results being saved for the next time, the program does save a considerable amount of work and can be run on a minimal system.

Howard’s stationed in Germany at Ft. 5th SFG CMD, DCOF Sept-79, APD New York, NY 09056. He has recently upgraded his system to a 12K 2-80 so he should really be cracking out handicaps by this time!
not be used to terminate the entry. Instead, I use another key, which in my case is the Blank key on my model 15 TT. The STRNF routine is re-oriented so that CRFL is skipped when the Blank key is used. My previous changes on page 91 that substitute a semicolon for the comma have been removed, and all routines there are restored to their original form. While this allows more than one input per line on the YT, it also requires that the end of the line be terminated by a following PRINT statement. This seems to be a good tradeoff. The CR key can be used at the end of the line but it is probably better to use a PRINT statement, which makes the carriage return automatic. My modifications to INPUT now consist only of the following:

Code for Blank key which replaces code for Control/C.

Address in re-arranged STRNF routine to skip CRFL op.

CAL CRFL
CAL SUBHL

If one wishes to retain Control/C the test for Line Feed can be sacrificed instead, since LF is not normally used during input of data.

To input data into the same line as data being printed out from memory under TAB control, it is necessary to increment the COLUMN COUNTER each time a digit is input. This is accomplished by inserting a column counter incrementing routine into CINPUT which is provided by the user for his own particular input device. By adding a test for the Blank key and the Delete key, which are both non-printing, the column counter incrementing routine can be skipped. If this is not done, the position of the column will be displaced by one character, although this can be compensated for by changing the TAB value. Skipping the column counter incrementer, however, is better, as it simplifies programing. The complete routine to be inserted into CINPUT that I use... is as follows:

704 CPI
150 JTZ
704 CPI
150 JTZ
066 043 LLI 043
056 001 LMI 001
317 LBM
010 INB
371 LMB

The code for the Blank key or the Delete key is in the accumulator when the routine is entered. If either JZT is true, the jump is to the byte immediately following the end of the routine, which effectively bypasses the column counter incrementer. Incidentally, the Delete key, in my case, is the BEL key of the model 15 TT.

One needs to be careful that registers B, H, and L are free when the routine is used. Locating the routine here covers both numerical and CHR inputs. This addition is useful only if the preceding modification to INPUT is made.

Another improvement I have made to SCELBA is to add a function to limit the number of digits printed out. This has been a problem in printing tables of data where either allowances must be made for printing out the full 7 digits or accept an occasional overlap between columns. The INTEGER function does not seem to work for numbers with more than 4 digits (a result of binary rounding operations that start to show their affect when numbers exceed 4 digits — Ed.), and in any case can be used only with whole numbers. Even a number-rounding routine does not always work because the last stage of division frequently results in the value extending back out to 7 digits.

My new function changes the value at location 025 035 which specifies the number of digits to be printed. It replaces the RGN function, which I have never used, and occupies the same space with one byte left over. The Function Names Table is also changed to D52. The subprogram of D52 is the number of digits to be printed. A user program statement would take the form of:

100 PRINT D52(1)

This will limit all values to three significant digits, until a subsequent statement changes the limit. Besides the 3 digits, allowance must be made, of course, for a possible minus sign and a decimal point. A listing for the D52 Function follows.

Test for Blank key.

Skip col incr if increment by Block.

Test for Delete key.

Skip col incr if increment by Delete. Point to Column Counter.

Load column counter into B.

Increment column counter.

Restore column counter to memory.

Robert Pearce of 504 McGow Fork Rd., Clayton, KY 40104, says he is not a technical writer but he took the time to send in some pretty clear explanations of how he added some extra capabilities to SCELBA. We think his additions will be of interest to many SCELBA users.

The first improvement he discusses is a modification to the TEXTC routine that he has named TEXTCM. The modification provides the user with the capability of halting a listing of a program at any time by depressing any character on the input keyboard (except CR or CTRL/C). Doing so places the program in an "input loop." effectively halting operations while the user inspects the system's display. To continue the display the user may type
TEXTCM, LCM Fetch (cc) from the first location in
LAM The buffer (H&L putting there)
NDA Into Reg 7 & A. Test the (cc) value.
RTZ No display? If (cc) is zero.

TEXTCL, CAL ADV Advance pointer to next location.
LAM Get character from buffer.
CAL ECHO Display character.
IN * Get input from keyboard.
CPI 000 Test for 0.
JZZ END If yes, continue with TEXTCM rtn.

INLOOP, CAL INPUT (User subroutines without echo) stop here.
CPI cr And wait for a C/R or a CTRL/C.
JZZ END If get C/R, continue with display.
CPI ctrl If get CTRL/C exit to
JZZ XEXEC Start over.
JMP INLOOP ELSE cycle.

END, DCC Decrement (cc).
JZJ TEXTCL II (cc) is not zero continue display.
RTF Exit to calling routine.

[AT PAGE 02 LOCATION 061 CHANGE:]
SYXT3, CAL TRACE Insert TRACE patch call.

[AT A SUITABLE PATCH AREA ADD:]
TRAC, LLI 201 Replace SYXT3 instructions.
LBM

SWITCH, RET/NOP RET = NO Trace, NOP = Trace
(Editors note: be careful here, the
label SWITCH has been used else-
where in SCELBAL).

LLJ 310 Point to line number buffer.
CAL TEXT Display line number.
LAI 001 Set up number of blanks.
CAL TABC Display blank.
LLJ 201 Replace SYXT3 instructions.
LBM

RET Return to SYXT3.

[AT PAGE 07 LOCATION 074 SET UP:]
JZZ UDF(*) Jump to UDF function.

[AT A SUITABLE PATCH AREA ADD:]
UDF(*), LLI 196 Point to MSB of FPACC.
LHI 001 LAM Get MSB.
CPI 100 Compare for a FPBIT "1."
LLI *** Address of SWITCH point
LHI *** For TRACE switch.
JZZ TRAC If comparison = 0 move a NOP
LMI 007 To the switch, else move a RET
RET to the switch. Then exit.

TRAC, LMI 209 RET Set up a NOP for the switch.
Exit.

ONE MORE TIME

In SCELBAL UPDATE. Issue 04 of 1/77 on page 03 Mr. James Tucker of 7 Goose Street, Exeter, N.H. 03833 discussed a problem with storage of the first variable in the variables symbol table. He recently wrote to notify us of a related problem and a proposed correction.

"The program as it now functions skips the first storage cell when the first variable encountered is a "FOR-NEXT" variable.

Present program:

010 132 108 356 022

Change to:

010 132 104 053 075 **

JMP PATCH (or suitable loc)

And put in the following patch:

075 052 106 356 022 CAL SWITCH
075 055 201 LAM
075 056 074 000 CPI 000
075 060 113 135 010 JZJ 010 135 (return)
075 063 106 356 022 CAL SWITCH
075 066 104 291 010 JMP STOYSYA

Present program:

005 065 106 356 022 CAL SWITCH

Change to:

005 065 104 017 015

JMP PATCH (or suitable loc)

And put in the following patch:

075 017 106 356 022 CAL SWITCH
075 021 207 CH 000
075 023 074 000 JZJ 005 070 (return)
075 030 106 356 022 CAL SWITCH
075 033 104 134 005 JMP LOOKYA