REMark

Issue 7 • 1979

Official magazine for users of Heath computer equipment.

Cover shot: This is the view just a few feet from our office. It is however, radically different just three short months from now. BRRR!

on the stack

>CAT

NOTICE
CANADIAN MEMBERS

To avoid excessive shipping costs and to provide more expedient delivery to Canadian members, you may now place your orders for HUG software directly with Heath Canada at 1480 Dundas East Highway, Mississauga, Ontario Canada L4X 2R7.

"REMark" is a HUG membership magazine published quarterly. A subscription cannot be purchased separately without membership. The following rates apply.

	U.S. Domestic	Canada & Mexico	International
Initial	$14	$16	$24
Renewal	$11	$13	$18

Membership in England, France, Germany, Belgium, Holland, Sweden and Switzerland is acquired through the local distributor at the prevailing rate.

Back issues are available at $2.50 plus 10% handling and shipping. Request for magazines mailed to foreign countries should specify mailing method and add the appropriate cost.

Send payment to:

Heath Users' Group
Hilltop Road
St. Joseph, MI 49085

HUG Manager and Editor ..... Jim Blake
Graphics .....Ron Hungerford
HUG Secretary .....Susan Gilfoyle
Software Developer .....Gerry Kabelman

REMark

Organization! Confusion with a Structured Definition

Gene Bellinger
4929 Red Fox Dr.
Annandale, Va. 22003

Some time ago I mentioned to our illustrious editor that I was considering writing a series of articles for HUG concerning Top Down Structured Programming. It was felt that this was a necessary undertaking after surveying the programs in HUG SOFTWARE VOLUME I and my own program library, the total definition of which could be none other than mass confusion.

Don't get me wrong! The authors of the submitted software have presented some very worthwhile ideas. The problem seems to stem from the fact that it is almost impossible for one person to author a program which exactly suits the needs or desires of another person. Also, some of us are hesitant to use programs that we are not intimately familiar with. There is a very strong desire to understand just how the program accomplishes its function. Many people also attempt to learn from the programs of others; which is quite difficult when the logic is obscured in the optimization.

Thus began the conceptualization of a series of articles; but with no great deal of urgency. Then on 15 April 1979, another bug appeared in one of my own undocumented, unstructured, and optimally optimized BASIC programs; which in the final analysis could not be eliminated without rewriting about 50 lines of code. This was just too much, the last bug you might say, for the human system crashed and in a fit of rage ten disks were scratched simply to ensure total bug elimination. This may have been a bit drastic but it is certain all bugs have been eliminated. Of course all the programs went the way of the bugs, but it is hoped that the future may prove to be the better for it.

In addition I have vowed not to write another program until this series of articles is finished. Thus what follows is designed to instantiate my own concepts and offer them as an assistance to others, least they arrive at my state of turmoil.

The intention is to proceed as follows:

1. Define allowable structural elements [The Ideas]
2. Implement these structures in HDOS BASIC [The Means]
3. Define a Program Design Language [The Method]
4. Develope a RENUMBER BASIC program using 1-3 [The Proof]

Notice that there is no discussion of the pros and cons of Top Down Structured Programming included in this list. The decision to use Top Down Structuring is one you must arrive at through your own experience and no volume of words will entice you to begin any sooner. I simply propose to offer The Ideas, The Means, The Method, and the Proof. The decision is totally yours.

The Ideas

Although it has been mathematically proven that any program can be written with only three structural elements (structures 1-3 below) two additional ones will be offered as they are found to be quite useful.

Each structure is designed with two specific concepts. First, each structure represents a process for performing a specific function, and second, each structure is designed with a single entry, single exit structure. The single entry, single exit idea forces each structure to be a complete unit in that execution begins at the top or beginning and finishes at the bottom or end, thus maintaining the top down flow of control.

Structural Elements

1. SEQUENCE-OF-STATEMENTS;
2. IF-THEN-ELSE;
3. DO WHILE;
4. REPEAT UNTIL;
5. CASE STRUCTURE;

1. SEQUENCE-OF-STATEMENTS:

This is the basic building block and is just what it says; a sequence-of-statements. This is meant to include all legal BASIC statements including the structural elements 2 through 5 but not the unrestricted GOTO statement. I could write a book on how much trouble the GOTO statement has gotten me into but I will refrain, least you be bored to death. Later in the implementation of structures 2 through 5 specific uses of the GOTO will be necessary and these will be explained as presented. These GOTO statements will be classified as restricted GOTO statements.

2. IF-THEN-ELSE:

The traditional IF-THEN-ELSE structure has been expanded to an IF-THEN-ELSEIF-ELSE-END IF structure; but so as to eliminate immediate confusion we will work up to an explanation of this construction.

In its simplest form the structure is as follows:

```
IF conditional-statement THEN
    sequence-of-statements;
END IF;
```

In this form the conditional-statement is evaluated as being either TRUE or FALSE. When the evaluation is TRUE the sequence-of-statements is executed and control is passed to the next executable statement after the END IF statement. When the conditional-statement is evaluated as FALSE control is passed to the next executable statement following the END IF statement with the sequence-of-statements being bypassed completely.

From this we move on to the traditiional IF-THEN-ELSE:

```
IF conditional-statement THEN
    sequence-of-statements-1;
ELSE
    sequence-of-statements-2;
END IF;
```

Again the conditional-statement is evaluated as either TRUE or FALSE. When the evaluation is TRUE sequence-of-statements-1 is executed and control is passed to the statement following the END IF statement. If the evaluation is FALSE sequence-of-statements-2 is executed and control falls through the END IF statement to the next executable statement. Thus only one of the sequences are executed.

Now we come to the ELSEIF portion of the structure. This construct is intended to assist in the elimination of the ever annoying and confusing nested if statement. The ELSEIF portion of the structure may be repeated as many time as necessary and functions as follows:

```
IF conditional-statement-1 THEN
    sequence-of-statements-1;
ELSEIF conditional-statent-2 THEN
    sequence-of-statements-2;
ELSEIF conditional-statement-3 THEN
    sequence-of-statements-3;
ELSE
    sequence-of-statements-4;
END IF;
```

This structure should be easily understood from the previous structures. It should suffice to say that the sequence-of-statements 1 through 4 are mutual exclusive (i.e. only one sequence will be evaluated and then control passed to the statement following the END IF statement). In addition the structure may only contain one ELSE segment which is executed when all conditional-statements evaluated as false.

3. DO WHILE:

```
DO WHILE (conditional-statement);
    sequence-of-statements;
END;
```

This structure is designed to repeatedly execute a sequence-of-statements as long as the conditional-statement evaluates to TRUE. The conditional-statement is evaluated prior to the execution and when it evaluates to FALSE control is passed to the next statement following the END statement. Thus the sequence-of-statements may actually be executed zero (0) times.

4. REPEAT UNTIL:

```
REPEAT
    sequence-of-statements;
UNTIL (conditional-statement);
```

This structure is designed to repeatedly execute a sequence-of statements until the conditional-statement evaluates to TRUE. Note the specific difference from the DO WHILE. The sequence-of-statements in the REPEAT structure must be executed at least once and the repeat continues UNTIL the conditional-statement is TRUE; whereas the DO WHILE continues WHILE the conditional-statement is TRUE.

5. CASE Structure:

```
CASE expression
1:  sequence-of-statements-1;
2:  sequence-of-statements-2;
3:  sequence-of-statements-3;
END;
```

For this structure the expression is evaluated and for a resultant value of 1, sequence-of-statements-1 is executed and then control is passed to the next executable statement following the END statement. Similar results occur for the expression having a value of 2 or 3. When the expression evaluates outside the defined range (i.e. less than 1 or greater than 3) control is passed to the first statement following the END statement and no sequence is executed.

Summary:

Five structural elements have been defined which actually provide more than a means of creating Top Down Structured Programs. In fact using these structures allows for the creation of nothing but Top Down Structured Programs. The only backward flow of control is provided for by the DO WHILE and REPEAT UNTIL structures. With a little thought you will see that this is a more than adequate defintion of a set of tools to use in the creation of programs.

The Means

The next step is, to define a clear and concise implementation of these structures in HDOS BASIC and still maintain the concept of single entry, single exit functional structures.

One specific set of implementations is as follows:

1. sequence-of-statements:

The sequence-of-statements structure requires no specific implementation but two points may be mentioned. These points are designed to improve program readability, which also does a great deal to improve ones understanding. First, the 'REM' statement is used to insert meaningful comments into the program and second, the colon (:) is used to provide statement offset as:

```
10 REM % sample sequence-of-statements;
20 : A = 1
30 : B = A + B
```

The readability afforded by the alignmnt will prove more useful as we continue. The percent sign (%) in the 'REM' statement is simply to allow rapid identification of comments without looking back to the left margin for the 'REM' identification. The worth of this convention will become more obvious as statements are offset further to the right.

2. IF-THEN-ELSEIF-ELSE-END IF:

The implementation of this structure will be explained using two examples. First, suppose we wish to set C = 1 when A < B; otherwise set C = 2.

```
10 REM % sample IF-THEN-ELSE structure;
20 : IF NOT (A<B) THEN 60
30 : C = 1
```

```
40 : GOTO 80
50 REM <ELSE>
60 : C = 2
70 REM <END IF>
80 :
```

There are several points of specific interest in this example and a little justification necessary. The colon (:) and the 'REM' statements have been used to create an implementation that looks like the IF-THEN-ELSE structure. Less than and greater than symbols are used to enclose dummy substructure elements simply to make them readily differentiable from comments; which describe operations. Thus the portions of the structure are immediately obvious on sight, as well as where the structure ends.

Now for a justification of the NOT (A<B). Admittedly this is equivalent to (A> = B) but in the initial statement of the operation we wanted to do something if (A<B). Thus the NOT is employed to eliminate the necessity of reversing all the logic to maintain the original structure. Reversing the logic has a very definite tendency to obscure meaning and result in logic errors sooner or later. All that is necessary is for one to adopt a new method of reading IF statements. If the parenthetical expression is TRUE control flows downward as it should. When the parenthetical expression is FALSE, it is negated by the NOT, making the entire conditional-statement TRUE and a branch to the ELSE clause is effected. Repeated consistant use of this structure leads to a very nonambiguous interpretation. Note the GOTO statement in line 40. This is a restricted GOTO in that it is only allowed to pass control to the first executable statement following the END IF statement. Later I will explain just why the reference is made to the next executable statement and not the END IF statement which is actually the end of the structure.

A full IF-THEN-ELSEIF-ELSE-END IF example:

```
005 REM % FULL EXAMPLE;
010 : IF NOT (A<B) THEN 50
020 : C = 1
030 : GOTO 110
040 REM <ELSEIF (A = B) THEN>
050 : IF NOT (A = B) THEN 90
060 : C = 2
070 : GOTO 110
080 REM <ELSE>
090 : C = 3
100 REM <END IF>
110 :
```

Notice that all the GOTO's transfer control to the same line. This implementation maintains the downward flow of control and gives the structure a single entry and single exit point. Be maintaining this single entry, single exit concept a structure becomes a logical block; sort of a complete thought. This specific point will be addressed in detail later.

3. DO WHILE:

```
10 REM <DO WHILE(A<B);>
20 : IF NOT (A<B) THEN 60
30 : A = A + 1
40 : GOTO 20
50 REM <END>
60 :
```

This structure is consistant with previous concepts and implements a DO WHILE(A<B). Line 40 contains another restricted GOTO, in that control may only be passed to the beginning of the structure. Notice that the loop testing is specifically at the beginning of the loop so the structure may actually be executed zero times.

4. REPEAT UNTIL:

```
10 REM <REPEAT>
20 : A = A-1
30 : IF NOT (A<B) THEN 20
40 REM <UNTIL (A<B);>
50 :
```

Thus we have a structure which repeats itself UNTIL a condition is TRUE, in which case control falls through to the next statement in the sequence. Note that the test is specifically at the end of the loop so the structure must execute at least once.

5. CASE structure:

```
010 REM <CASE A>
020 : ON (A) GOTO 50, 70, 90
025 REM <OUT OF RANGE>
030 : GOTO 110
035 REM <1:>
050 : B = 3
060 : GOTO 110
065 REM <2:>
070 : B = 1
080 : GOTO 110
085 REM <3:>
090 : B = 7
100 REM <END>
110 :
```

Now (A) is evaluated and if the result is 1, 2, or 3 the specified branches 50, 70, or 90 are taken; after which control is passed to statement 110 which is intended to indicate the next executable statement. Note that all the branches are to the same statement thus maintaining the one entry, one exit concept. If (A) evaluates to any value out of range (i.e. less than 1 or greater than 3) control falls through to statement 30 which may be used as an error handling section or simply pass control to statement 110. The implementation of the CASE structure maintains the desired or required downward flow of control.

Summary:

The five designated structures have been given a specific implementation in HDOS BASIC which maintain the single entry, single exit concept and produce a continuing downward flow of control. They have also been implemented in a manner which preserves their original form (i.e. the use of the 'REM' statement, the colon (:), and the bracket <> notation), which makes them readily identifiable on visual inspection.

The Method

Now that all the means to develop Top Down Structured Programs are at our disposal, we need to develop a method by which to employ the means. Along with the method evolves several additional restrictions. These restrictions are offered as helpful

hints on things to avoid but may be used if you so desire. You will decide to stop only when you realize you have suffered enough.

1. Basic allows multiple statements per line (don't)
2. Basic allow 'REM' statements on the same line with another statement (don't)
3. Basic allows GOTO LINO (expression) (don't!!!!!!)
4. Basic understands THEN GOTO (don't)

Let us continue. You may have noticed that the implemented structures were presented without the use of conventional flow charts. This was intentional, as conventional flowcharting is as unrestricted as the GOTO. It allows one to transfer the flow of control anywhere desired, and one usually does, resulting in some very obscure logic.

Thus an alternative is proposed which will serve the purpose of the flow chart and also assist in eliminating the difficult resulting from the restriction of using variable names which are composed of a letter or a letter and a number.

The alternative is rather straight forward and almost obvious enough to be bizzare. Simply write the program in a convenient version of ENGLISH!!!!! By this I mean write a literal description of the program function in English type statements while adhering to the allowable structures as provided. The individual statements may be written in any form desired which is capable of making their function intuitively obvious by inspection.

As a short example, suppose we want to search a specified string for the occurrent of the first non-space character:

```
pointer = 1;
    DO WHILE (string-character(pointer) = space);
    pointer = point + 1;
    END;
```

This sequence of statements defines the desired operation in an understandable manner and is directly translatable into BASIC as follows:

```
10 : P = 1
20 : IF NOT (MID$(S$,P,1) = " ") THEN 50
30 : P = P + 1
40 : GOTO 20
50 :
```

The line for line translation to BASIC has produced an executable sequence of statements but at the same time has eliminated the majority of the self explanation contained in the original description. Thus what is proposed is something on the following order:

```
10 REM % pointer = 1;
20 : P = 1
30 REM  <DO  WHILE  (strong-character(pointer)
        =(space);>
40 :  IF NOT (MID$(S$,P,1) = " ") THEN 90
50 REM  % pointer=pointer + 1;
60 :  P = P + 1
70 :  GOTO 40
80 REM  <END;>
90 :
```

Your first complaint will probably stem from the idea that you must work overtime writing two complete programs, one descriptive and one actual. This is partially true and partially false. This literal English description is a PDL (Program Design Language) specification of the program and may be maintained as such using the HDOS EDIT program. One can create and maintain the PDL description in a file (i.e. pgmname.PDL). Once the PDL description is complete it may be transformed into a 'FULL' set of comments for the actual program. Transformed, not by hand, but, by another BASIC program of course. The PDL description of just such a transformation program will appear at the end of this article.

Now you may complain that all these comments will take up very valuable memory space and slow down actual program execution. In this we are in agreement, thus enter another BASIC program. This one is designed to remove all the 'REM' statements from a BASIC program. A PDL description of this program will also follow. Thus there exists not two but three copies of any given program:

1. pgmname.PDL — A program design language specification of the program with additional comments as needed.
2. pgmname.DOC — A combination PDL and actual BASIC statements which exists as a fully documented and executable BASIC program.
3. pgmname.BAS — A 'REM' stripped version of pgmname.DOC

One additional concept necessary for the creation of meaningful TDS programs is the concept of modularization. The TDS concepts proposed are fine as is, but you must realize that the flow of control of a structure (i.e. DO WHILE, REPEAT UNTIL) may range over several hundred lines. If the structure also contains numerous nested structures you begin to loose the ability to keep track of all the processes in effect. Thus by creating the program in functional modules, each of which performs a function or set of related functions, it is easier to keep track of things. Many opinions have been put forward as to just how large these modules should be but personally I prefer the one page convention. Using this convention the scope of control of all structures is readily obvious on visual inspection. Of course modules may be much smaller (i.e. 3 or 4 lines) for some functions.

Summary:

What has been presented is an implementation of MTDSP (Modularized Top Down Structured Programming) which is in no way meant to be the last word. The individual structures and conventions are simply a basis to work from which may be modified as necessary to satisfy personal individuality. Nothing is effectively lost as long as one maintains a specific set of structures with a single entry single exit feature and some sort of modularization is done.

I suppose you might say this article is a plea to resources. The knowledge which might be acquired from resourceful software development specialists (i.e. programmers) is extremely vast; if they would simply cease being so obscure.

Part II of this article will attempt to develop an MTDS program, with non-obscure logic, designed to renumber HDOS BASIC programs.

remark-pd1:PROCEDURE;1

```
REMO.PDL = > remark-pd1.pd1
REMPDL.PDL = > all remark-pd1 associated pd1's

remark-pd1 is designed to read a pd1 file and convert it to a
sequence of numbered statements for use in further program
development.

CALL initalize;
first-char : = CIN(input-file);

  DO WHILE(first-char <> end-of-file);
    IF first-char <> line-feed THEN
      INPUT line;
      line : = CHAR(first-char) + line;
      CALL check-and-insert;
    ELSE
      line : = null;
    END IF;
  line-number : = line-number + increment;
  line : = CHAR(line-number) + 'REM' + line;
  OUTPUT line;
  END;

CLOSE input-file, output-file;

END remark-pd1;
```

initalize:PROCEDURE;1

```
  REM1.PDL => initalize,pd1

  initalize inputs user specification and sets up program vari-
  ables.

FUNCTION MOD8(x) :: = x - 8 * INT(x/8)
DIMENSION keys (7)
  INITIAL('IF','ELSE','END','OO','REPEAT','UNTIL','CASE');
space-8 : = (8)' ';

INPUT input-file-name,output-file-name;
CALL file-check;
OPEN input-file,output-file;

INPUT first-line-number; increment;
line-number : = first-line-number - increment;

RETURN;
END initalize;
```

check-and-insert:PROCEDURE;1

```
  REM2.PDL => check-and-insert.pd1

  check-and-insert searches a line for key words and if found,
  encloses the key word structure in brackets. A ' % ' is inserted
  prior to the first non space character if no key word is found.

CALL index-non-space;

  IF non-space-index <> zero THEN
    not-found : = true;
    test-string : = MID(line,non-space-index,6);
    DO key-index : = 1 to 7 WHILE(not-found);
    key : = keys(key-index);
    key-length : = LEN(key);
      IF key LEFT(test-string,key-length) THEN
      not-found : = false;
        CALL bracket;
      END IF;
    END;

    IF not-found THEN
      IF MID(test-string,2,1) = colon THEN

        this line is part of a case structure so CALL bracket;
      ELSE
        insert ' % ' before first non space character line :
        LEFT(line,non-space-index-1)
        + ' % ' + MID(line,non-space- index);
                END IF;
              END IF;

        END IF;

      RETURN;
      END check-and-insert;
```

file-check:PROCEDURE;1

```
  REM3.PDL = > file-check.pd1

file-check looks at input $ output file specifications and assigns
default values as required.

comma-index : = MATCH(input-output,comma,1):
  IF comma-index = 0 THEN
    input-file-name : = input-output;
    output-file-name : = null;
  ELSE
    input-file-name : = LEFT(input-output, comma-index-1);
    output-file-name : = MID(input-output,comma-index + 1);
  END IF;
period-index ; = MATCH(input-file-name,period,1);
  IF period-index = 0 THEN
    input-file-name : = input-file-name + '.PDL';
  END IF;
IF output-file-name = null THEN
    period-index : = MATCH(input-file-name,period,1);
    output-file-name : = LEFT(input-file-name,period-index-
                         1)
                         + '.DOC';
  ELSE
    period-index : = MATCH(output-file-name,period,1);
      IF period-index = 0 THEN
        output-file-name : = output-file-name + '.DOC';
      END IF;
    END IF;

RETURN;
END file-check;
```

index-non-space:PROCEDURE;1

```
  REM4.PDL = > index-non-space.pd1

  index-non-space searches a line for the first non space charac-
  ter. In this respect control-I characters are treated as spaces
```

and their occurrence is replaced by an appropriate number of spaces. This replacement is necessary as the prefix 'nnnnn REM' will mess up any offsets created in the PDL using control-I characters. The number of spaces to be inserted is computed as:

8 — MOD(control-E-index-1,8)

```
non-space-index : = 0;
looking : = true;
  DO index : = 1 BY 1 WHILE(looking & index < LEN(line));
  char : = MID(line,index,1);
    IF char = space THEN
      do-nothing;
    ELSEIF char = control-I THEN
      line : = LEFT(line,index-1)
            + LEFT (space-8,8-MOD8(index-1))
            + MID(line,index + 1);
    ELSE
      looking : = false;
      non-space-index : = index;
    END IF;
  END;
RETURN;
END index-non-space;
```

bracket:PROCEDURE;1

REM5.PDL = > bracket.pd1

bracket places the required brackets around a structural element when called.

```
line : = LEFT(line,non-space-index-1)
      + '<' + MID(line,non-space-index) + '>';

RETURN;
END bracket;
```

unremark-doc:PROCEDURE;2 05-May-79

UNREMO.PDL = > unremark-doc.pd1
UNREMDOC.PDL = > all unremark-doc associated PDL's

unremark-doc is designed to remove 'REM' statements from a specified documentation file (i.e. one created from a PDL with the executable statements inserted). In addition offset lines are left justified by the removal of the colon, spaces, and control-I characters which prefix the first interpretable character. Line numbers remain unaltered.

```
CALL initalize;
first-char : = CIN(input-file);
  DO WHILE(first-char <> end-of-file);
  INPUT line;
  line : = CHAR(first-char) line;
    IF line <> rem-line THEN
      CALL deblank;
      OUTPUT line;
    END IF;
  first-char : = CIN(input-file);
  END;

CLOSE input-file,output-file;

END unremark-doc;
```

initalize:PROCEDURE;1 05-May-79

UNREM1.PDL = > initalize.pd1

initalize looks at input & output file specifications and assigns default values as required.

```
INPUT input-output;
comma-index : MATCH(input-output,comma,1):
  IF comma-index = 0 THEN
    input-file-name : = input-output;
    output-file-name : = null;
  ELSE
    input-file-name : = LEFT(input-output,comma-index-1);
    output-file-name : = MID(input-output,comma-index + 1);
  END IF;
period-index : = MATCH(input-file-name,period,1);
  IF period-index = 0 THEN
    input-file-name : = input-file-name + '.DOC';
  END IF;
  IF output-file-name = null THEN
    period-index : = MATCH(input-file-name, period,1);
    output-file-name : = LEFT(input-file-name,period-index-
                          1)
                          '.BAS';
  ELSE
    period-index : = MATCH(output-file-name,period,1);
      IF period-index = 0 THEN
        output-file-name : = output-file-name + '.BAS';
      END IF;
    END IF;

OPEN input-file,output-file;

RETURN;
END initalize;
```

deblank:PROCEDURE;2 05-May-79

UNREM2.PDL => deblank.pd1

deblank looks at an unremark line and if there is a colon in column 7 the line is deblanked as possible.

```
  IF MID(lin,7,1) = colon THEN
    pointer : = 8;
    not-found : = true;
      DO WHILE(not-found & pointer < LEN(line));
      char : = MID(line, pointer,1);
        IF (char = space) OR (char = control-I) THEN pointer :
          = pointer + 1;
        ELSE
          not-found : = false;
        END IF;
      END;
  ELSE
    pointer :=7;
  END IF;

line : = LEFT(line,6) + MID(line,pointer);

RETURN;
END deblank;
```

EOF

WHAT! FTS AGAIN?!

Techniques for large file manipulation on the H-11

Douglas H. McNeill, M.D.
Poynette, Wisconsin

Although the H-11 offers a great deal of storage capacity with two full sized drives, certain large data files may rapidly exhaust the available space and yield increasing frustration with the increasing emergence of the FTS (File Too Short) error message. Some techniques are described to circumvent this problem pending a hardware modification (such as hard disk).

Our operation uses the H-11 for the storage of patient records and account status information in a general medical practice setting. Because density of information is of far greater importance than speed of access, our data files are organized as sequential files rather than virtual files. This compromise requires a larger working area on the disk for updates, however, since introduction of new information to a sequential file (particularly where the information may be added in the **middle** of the file) requires that the file be written to a work area with corrections added and then re-written to its original location in the updated form. Unless some tricks can be devised to stretch the available space, storage overflow is both inevitable and prompt.

The best way to stretch storage is to use more than one disk. HT-11 easily handles the dual drive H-27 so that the system disk (SY:) can be used for many purposes as a scratch area for the major files manipulated on the scratch disk (DK:). A better (less constraining) approach uses an **infinite** number of disks; but, you say, HT-11 forbids switching disks while in BASIC! A brief catalog of our disk allocation and discussion of programming techniques will demonstrate how this is permissible.

Our programs are distributed in the following manner:

for SY:	Disk #1:	XBASIC.SAV	Function:	DOS
		PIP.SAV		
		EDIT.SAV		
		LINX.DAT		prevents dup. run (**vide infra**)
for DK:	Disk #2:	MEDSY0.BAS	Function:	generate updates
		+		
		data files for existing accounts		
		IDENT.BAS		identify disk
for DK:	Disk #3:	MEDSY1.BAS	Function:	update data files
		+		on this disk
		data files for financial data		for accounts
		IDENT.BAS		identify disk
for DK:	Disk #4:	MEDSY2.BAS	Function:	update data
		+		files this disk
		data files for medical information		
		for individuals		
		IDENT.DAT		identify disk

Conceptually, this list could be extended infinitely; the division into three disks for DK: is arbitrary and based on local requirements for our data set.

All programs MEDSY0.BAS, MEDSY1.BAS and MEDSY2.BAS share the same first few program lines:

```
10 REM: (RESERVED FOR PROGRAM IDENTIFICATION)
20 REM: (DITTO)
30 REM: (DITTO)
40 GO TO 80
50 OPEN "IDENT" FOR INPUT AS FILE #1
60 INPUT #1:Q$
70 CHAIN Q$
80 REM: PROGRAM BEGINS HERE
. . .
```

Each disk also carries a file "IDENT.DAT" which consists of the **name** of the update program resident on **that** disk.

The procedure is as follows: first the program (MEDSY0) to generate updates is executed. This produces a file of all updates that is written to the **system** drive and then terminates. The operator is then instructed to exit via Control-C, returning the H-11 to the control of the monitor program KMON.**Then and only then** the DK: drive is unloaded and the disk switched to another in the series. The **old** program is entered using the KMON command RE (re-enter), the BASIC prompt "READY" appears and **in immediate mode** the command GOTO 50 is entered. This causes the old program to determine the contents of the **new** disk, load and execute any relevant programs on that disk. This procedure effectively chains across an infinite number of disks and permits access to a truly huge data set.

One problem needs some amplification. With a large number of diskettes it is possible for the operator to forget which have been updated during this session; either running an update through a sequential file twice or not at all will quickly reduce the file to so much garbage. The file LINX.DAT serves to prevent this.

The first program (MEDSY0) verifies that there are no outstanding updates remaining before a new update file is generated and then produces the file LINX.DAT, stored on the SY: disk, which represents a worklist of remaining updates that must be performed before the next pass. This is accomplished with the following code:

```
100 OPEN "SY:LINX" FOR INPUT AS FILE #1
110 INPUT #1: Q$
120 IF Q$ = "DONE" THEN 200
130 PRINT "LAST BATCH NOT COMPLETED: PROGRAM(S)
        MISSING: "; Q$;" ";
140 INPUT #1: Q$
150 IF Q$ = "DONE" THEN 170
160 PRINT Q$
170 PRINT
180 PRINT "PROGRAM TERMINATING TO PREVENT ER-
        ROR"
190 STOP
200 CLOSE #1
```

```
210 OPEN "SY:LINX" FOR OUTPUT AS FILE #1
220 PRINT #1: "MEDSY1"
230 PRINT #1: "MEDSY2"
240 PRINT #1: "DONE"
250 CLOSE #1
```

Lines 130-150 and 224-240 may be expanded if more than three disk transfers are required. Each new disks scrubs its name from the worklist or aborts with the following code:

```
100 OPEN "IDENT" FOR INPUT AS FILE #1
110 INPUT #1:Q$
120 CLOSE #1
130 OPEN "SY:LINX" FOR INPUT AS FILE #1
140 FOR I = 0 to 2
150 IF END #1 THEN 190
160 INPUT #1: Q$(I)
170 NEXT I
180 I = I + 1
190 I = I - 1
200 FOR J = 0 TO I
210 IF Q$ = Q$(J) THEN 240
220 NEXT J
230 GO TO 320
240 CLOSE #1
250 OPEN "SY:LINX" FOR OUTPUT AS FILE #1
260 FOR J = 0 TO I
270 IF Q$(J) = Q$ THEN 290
280 PRINT #1:Q$(J)
290 NEXT J
300 CLOSE #1
310 GO TO 370
320 GOSUB 1030 [SUBROUTINE TO ALERT OPERATOR]
330 PRINT
340 PRINT Q$; "SUBPROGRAM NOT IN REMAINING WORK-
                LIST"
350 PRINT "- -PROGRAM TERMINATING"
360 STOP
370 REM: PROGRAM BEGINS HERE
```

These techniques permit full utilization of the space of each diskette and facilitate monstrous manipulations without overwriting a file twice or skipping it in an update. The division of files onto each disk is determined on the basis of the utilization to be made of them in subsidiary programs not discussed above. The use of the CHAIN command in line 70 of the common first portion cleans the volatile memory of the H-11; an OVERLAY command could also be used with care to continue a long calculation requiring multiple diskettes for entry while retaining the variables intact. Much care is required, however, to prevent unwanted strange bugs from creeping into the calculation because of reuse of old line numbers.

With a bit of fancy footwork and additional operator care, these techniques effectively make the H-11 system truly high-powered. Only the addition of hard disk drive or conversion to a mainframe computer could equal this performance. Both these alternatives are far less pleasant than the above procedures.

EOF

ADDENDUM

One further trick is worth using to prevent diskette fragmentation. When a data file OLD.DAT is updated to NEW.DAT and NEW.DAT is then re-written to OLD.DAT for the next pass, the file NEW.DAT becomes superfluous and serves only to limit the file size possible on that disk through fragmentation. This occurs because HT-11 assigns one half of the largest available space on a diskette for an new file unless otherwise instructed. Two BASIC statements will free unproductive work area as follows:

```
10000 OPEN "NEW" FOR OUTPUT(1) AS FILE #1

10010 CLOSE #1
```

By opening and immediately closing the completed working file, its size is reduced to a minimum. The specification of the output number of blocks as one (1) will force the allocation of space for this file to the first sector that is free on the diskette, effectively removing the file from the central portion of the diskette and clearing the decks for the largest possible data files to come in future updates. The name is obligated to remain but it cannot block larger files under this disk-cleaning procedure.

EOF (really)

Here are changes to calender program appearing in Vol. II for 10.05 Ex. B.H. Basic :JB:

```
1 REM APPOINTMENT/CALENDAR VER.1.1  JUNE 18,1979
2 REM BY WILLIAM A. WILKINSON
3 REM THIS VERSION WILL WORK ON EXTENDED BASIC #10.05.00.
401 REM ***** THESE FOLLOWING LINES NEED TO BE CHANGED TO
402 REM PREVENT THE CALENDAR FROM DISPLAYING EIGHT DAY
403 REM WEEKS.  APPARENTLY BASIC VER.10.02.XX WOULD START
404 REM THE SPACE FUNCTION FROM THE CURSOR+1 POSITION, WHILE
405 REM BASIC VER.10.05.00 STARTS DIRECTLY FROM THE CURSOR
406 REM POSITION.
440 PRINT SPC(Z7*10)D;:GOTO 445
442 IF D>10 THEN S=6:GOTO 444
443 S=7
4060 D$=STR$(D):S=6
4075 IF D=10 THEN S=7
4104 IF VAL(MID$(P$(X2),5,4))=D THEN PRINT SPC((Z7*10)-1)LEFT$(D$,3)+"*";:F4=0:GOTO 445
4170 PRINT SPC((Z7*10)-1)D$;:F4=0:GOTO 445
65010 POKE 8272,85:POKE 8273,78:POKE 8274,13
65020 POKE 8275,71:POKE 8276,69:POKE 8277,90:POKE 8278,36:POKE 8279,13
65030 POKE 8280,89
65040 POKE 8281,67:POKE 8282,79:POKE 8283,78:POKE 8284,13
65050 POKE 8271,13
65060 RETURN
65110 POKE 8272,80:POKE 8273,85:POKE 8274,90:POKE 8275,36:POKE 8276,13
65120 POKE 8277,67:POKE 8278,79:POKE 8279,78:POKE 8280,13
65130 POKE 8271,9
65140 RETURN
```

THE HDOS TYPE AHEAD BUFFER

J.J. Thompson

One of the features of the console driver used with H8 tape software that many people have found useful is the Type-Ahead Buffer. Knowing the location of this buffer allows programmers to write programs which "POKE" command mode commands such as "LIST", "RUN", "PUT", and "GET" into the buffer. This has the effect of expanding the program mode command set because these commands are illegal in normal program statements.

HDOS also uses a Type-Ahead Buffer, but since its location and operation has not been documented until now it could not be used. I have disassembled HDOS and figured out the location and operation of the Type-Ahead Buffer.

The HDOS Type-Ahead Buffer is a buffer of 100 bytes maintained as a circular queue located in high memory. As HDOS is a relocatable program, the exact location of the buffer depends on the amount of memory in the H8. The starting address for HDOS is kept in memory locations 040 320 (low byte) and 040 321 (high byte). The start of the buffer is 2040 bytes above this starting address and the end of the buffer is 2140 bytes above this starting address.

HDOS treats this buffer as a circular first in first out (FIFO) queue. This is done with two 2 byte pointers. The TAIL pointer located at 2029 (low byte) and 2030 (high byte) bytes above the HDOS starting address contains the address in the buffer where the next character will be placed. The head pointer located at 2031 (low byte) and 2032 (high byte) bytes above the starting address contains the address of the next character to be removed from the queue. Since HDOS can operate in either line or character mode, there is also a 1 byte counter at 2025 bytes above the starting address which keeps a count of the lines (each terminated with a New Line character) in the queue.

When a character is typed on the terminal HDOS first checks to make sure it isn't one of several control characters that aren't stored in the queue. If it isn't one of these, HDOS stores the character and the current tail pointer on the stack. It then increments the tail pointer. If this causes it to point beyond the end of the buffer, it is changed so that it points to the start of the buffer. If the incremented tail pointer, (which points to where the next character is to be stored,) is pointing to the same location as the head pointer, then the queue is full and the terminal bell is sounded. If everything is OK, then the character and the preincremented tail pointer are retrieved from the stack and the character is stored in the queue. If the character was a carriage return, it is changed to a New Line character and the line counter is incremented. This process is repeated for every character entered.

When HDOS wants to retrieve a character from the queue, the following occurs. If HDOS is operating in the line mode, it checks the line counter byte. If this is zero, it indicates that a full line has not been input and HDOS must wait until the counter is incremented. If HDOS is operating in the character mode, it compares the head pointer with the tail pointer. If they point to the same address, it means that the queue is empty and HDOS must wait until a character is input.

When either a character or a line is available HDOS retrieves a character from the location pointed to by the head pointer. The head pointer is then incremented. If it points beyond the end of the buffer, it is changed to point to the start of the buffer. If the retrieved character was a new Line character, the line counter byte is decremented.

Below is given a table of the memory locations discussed above and a subroutine written in BASIC which can be used to put a string into the queue. In this subroutine, the actual addresses of the start and end of the buffer are not calculated as there are two 2 byte pointers in HDOS pointing to the start of the buffer and to the end of the buffer + 1. All comparisons are made using only the low byte pointer as all locations are on the same page of memory so the high bytes don't change.

See *Program Listing on Page 23* —

TABLE OF MEMORY LOCATIONS FOR HDOS TYPE AHEAD BUFFER

Location	Octal Value	Decimal Value
HDOS Starting Address (low byte)	040 320	8400
HDOS Starting Address (high byte)	040 321	8401
Offset to Line Counter Byte	005 351	2025
Offset to Queue Tail Pointer(Low Byte)	007 355	2029
Offset to Queue Tail Pointer(high byte)	007 356	2030
Offset to Queue Head Pointer(low byte)	007 357	2031
Offset to Queue Head Pointer(high byte)	007 360	2032
Offset to Pointer to Buffer Start(low byte)	007 361	2033
Offset to Pointer to Buffer Start(hi byte)	007 362	2034
Offset to Pointer to Buffer end + 1(low byte)	007 363	2035
Offset to Pointer to Buffer end + 1(hi byte)	007 364	2036
Offset to Buffer Start	007 370	2040
Offset to Buffer End	010 134	2140

EOF

The Secret HDOS

Chesney E. Twombly
15 Storer Street,
Kennebunk, Maine 04043

Being curious of mind and devious of soul and unable to take anything at face value, I set out to discover if there was anything which Heath had provided in HDOS and "forgotten" to document. A little reflection indicated to me that if I were to dump the HDOS program files as ASCII text, I might encounter character strings which would be that program's command vocabulary. I was able to obtain the ASCII dump I wanted by using the TYPE command. I TYPEd HDOS.SYS, HDOSOVL.SYS, SYSCMD.SYS, EDIT.ABS, ASM.ABS, and BASIC.ABS, since any un-documented features in these programs would be most valuable to me. A few minutes of staring at my video terminal and voila! I'd discovered a secret HDOS. The Users' Group should hear of this, I thought. And so I'm sharing what I've discovered with you.

Please take this information as subject to change without notice (although it didn't change when Heath released their latest version of HDOS.) since it wasn't documented. But here are some of the things I've discovered:

- Nothing of significance was found in HDOS.SYS, HDOSOVL.SYS, EDIT.ABS, or BASIC.ABS — at least nothing I could recognize.

- The HDOS command mode interpreter, SYSCMD.SYS, understands several commands which weren't written up. Unfortunately, there are no new commands — the commands which were documented, notably,

INDEX	a synonym for CAT
DIRECT	another synonym for CAT
IND	still another
LIST	a synonym for TYPE

- The assembler has two additional pseudo-operations which were not documented. One of these, ENCLU, seems to be equivalent to the XTEXT pseudo; the other, CODE, I still haven't figured out. If anybody out there has more time to work on this than I do, it might prove very rewarding . . .

- There is also an un-documented switch, /WIDE, which might allow you to control the number of characters per line in the .LST file — again, I haven't had time to figure it out.

Finally, we come to a program which Heath supplied on their distribution disk for both releases of HDOS. This program is called PATCH and no explanation was written up for it. The name itself was enough — I just had to figure out what it did. I experimented with it using the .ABS file of one of my assembly-language programs as the data file. After about an hour of trying reasonable (and sometimes unreasonable) things, I found out how to use the program — at least in part. The results of this research have gone to make up a two-page insert into the HDOS manual given on the next two pages.

I hope other users can benefit from my research and I hope this article brings out from the woodwork any other users of like mind who may have discovered other buried treasure in the cavers of the secret HDOS.

A MENU FOR BASIC PROGRAMS

By: Kevin Hauser

This article describes a program which prints a menu of all the BASIC programs on your disk and then allows the user to simply enter a number to load and run a program. Since I have only one drive, the program will require some modification for a two-drive system. The program provides a good example on how the 'CHAIN' instruction can be used.

The program begins with a clear screen which is created at line 3. Variable P1 is equal to the amount of BASIC programs on the disk. Lines 20 through 50 load the names of the BASIC programs into memory. Lines 60 through 80 print the memory on the terminal. Line 85 prints a carriage return if there is an odd number of programs. Lines 90 and 100 get the number of the program to be run and then, using the 'CHAIN' instruction, loads and begins execution of the program. Line 120 is the DATA line and should conta all the names of your BASIC programs EXACTLY AS THEY APPEAR IN A CATALOGUE. For example:

120 DATA "SPACEWAR.GAM", "CHECKBAL.BAS" and so on.

If it is preferred, more than one DATA line can be used. Also, programs that end in .BAS can be entered into the DATA line without the .BAS.

To set up the program, change line 1 so that P1 = the amount of BASIC programs which you have on this disk. Then enter the names of the programs into the DATA line as previously discussed. After loading BASIC, type CHAIN "MENU" and 'RETURN'. Then simply enter the number of the program you want to run.

LISTING 1 FOLLOWS:

```
00001 P1 = 3: REM P1 = # OF BASIC PROGRAMS ON DISK
00002 DIM P$(P1)
00003 FOR X = 1 TO 16: PRINT: NEXT X
00010 PRINT TAB(28);"MENU"
00012 PRINT :PRINT
00015 REM LINES 20 - 50 LOAD THE MENU INTO MEMORY
00020 FOR X = 1 TO P1
00040 READ P$(X)
00050 NEXT X
00055 REM LINES 60 TO 80 PRINT THE MENU
00060 FOR X = 1 to P1 STEP 2
00070 PRINT X;P$(X);:IF X1+P1 THEN PRINT TAB(40);X+1'P$(X1)
00080 NEXT X
00085 IF INT (P1/2)<>0 THEN PRINT
00086 PRINT
00087 REM LINES 90 & 100 GET AND RUN REQUESTED PROGRAM
00090 PRINT TAB(20);"ENTER PROGRAM #";: INPUT " ";P2
00100 CHAIN P$(P2)
00115 REM LINE 120 HOLDS NAMES OF PROGRAMS
00120 DATA see text
00130 END
```

EOF

Operating System

DISKETTE FILE PATCH (PATCH)

David A. Wallace
146 Westford St.
Chelmsford, MA. 01824

A file patch program has been provided by Heath Co. so that data on diskette files can be examined or modified by the user. Heath Co. does not provide documentation for its use; the information contained on this page is provided by experimentation with the program by a user and the procedures given for the use of PATCH may not be optimum or complete.

NOTE: Do not attempt to patch a program file by means of this utility unless you are sure you know what you are doing. Patching a program file requires that you know assembly- and machine-language programming for the 8080 microprocessor and Heath Co. will not support any programs which you modify. It is preferable (when source code exists) to re-assemble the program and replace the file, since the assembler is less error-prone than a human programmer.

Using PATCH

1. Type PATCH and a carriage return. PATCH will respond with the prompt

   file name?

2. Type the name of the file which you wish to patch. The default extension is ABS.

3. PATCH will search for the file on the disk and, if the file is found, will ask

   address?

4. Type the octal address of the patch. If that address is within the address space used by the program, PATCH will respond

   <address> = <data>/

5. Type the new octal data which is to replace the contents of that address. PATCH will automatically increment the address and display the next address's contents.

6. Type the change for that address (if there is one) or carriage return (if the data is correct).

7. Continue until all addresses in this block are patched. Then type CTL-D (control D). PATCH will respond

   address?

8. If there is another block of addresses to correct, repeat the procedure, starting at step 4. Otherwise, type CTL-D again. PATCH will write the file back to the disk with your corrections made. When the file is written, PATCH will again ask

   file name?

9. If there is another file to patch, repeat the procedure at step 2. Otherwise, type CTL-D again to exit.

Patches made to a program do not take effect until the file is re-written. Therefore, it is not possible to list the changed file. Also, CTL-C aborts the patch process and returns you to the file name? question. Several error conditions are possible, but the error messages for those conditions are self-explanatory.

But alas, PATCH won't allow you to patch system files. :JB:

EOF

ET-3400 AND TINY BASIC

The ET-3400 microprocessor trainer can be expanded upon by adding to it the ETA-3400 accessory box. The accessory box not only provides additional hardware but also provides the user with an additional software language, TINY BASIC.

BASIC is a computer language which is quite easy to learn, and for this reason it is used by many hobbyist rather than assembly language. The BASIC statements provide mathematical operations, decision making, and I/O operations which the user would otherwise have to code in lengthy and possibly hard to follow assembly language statements.

TINY BASIC is a subset of the regular BASIC language. It only does whole number arithmetic, has no alphabetic string capability, and does not allow arrays. Figure 1 lists the statements included in TINY BASIC. As can be seen, there are the necessary statements for doing arithmetic, making decisions, and doing I/O. This subset of regular BASIC is quite powerful in the ETA-3400 environment.

TINY BASIC also contains two very useful functions. The RND function returns a positive pseudo-random integer between zero and one less than the argument passed to it within parentheses. If a subroutine is written in TINY BASIC, then you use the GOSUB and RETURN statements to call and return from the subroutine. But what if the subroutine you wish to call is not writen in TINY BASIC, but is instead written in machine code. Maybe you want to use a subroutine in the ET-3400 monitor to display something on the 7-segment displays. The other function that TINY BASIC has will allow you to call any machine code subroutine. It is called the USR function.

VECTORED TO PAGE 21

--EDIT

Dear Mr. Blake:

I am enclosing a lot of words on how I connected two mainframes together. I hope that it will be of use to some other member. The ground plane cable was obtained by a friend who helped in this modification. It is fairly expensive but it is available and probably better than any other alternative. At least it works.

Sorry that I don't have pictures of the motherboard modification, but I feel that it is self-explanatory to anyone capable of building the kit in the first place. The reason for the modification was that I needed the extra memory, but couldn't find a buyer for the 8K boards and didn't want to scrap them. The 16K boards weren't available at that time anyway. Know of anyone who wants to buy a new CPU and Control board?

EXPANDING YOUR H8

R. E. CLARK, M.D.
MEMPHIS, TEXAS 79245
806-259-2232

As probably many enthusiasts have discovered, the H8 is an excellent starting point. Most of us have had a lot of fun and have been able to develop some routines with 8K or 16K of memory with little difficulty. As our systems have grown, we have all felt the pinch of too little memory or restricted program space. Heath first designed our system with what they considered totally adequate motherboard space. After the addition of a disk system with disk BASIC and increasingly complex programming, many of us have been faced with the lack of sufficient space. Those who have 32K of memory in 8K boards do not want to discard them to purchase the newer 16K boards. An expansion board has not been forthcoming from Heath and is not in the works as far as I have been able to determine. This is written to offer one solution which I have had the fortune to be able to manage to get to work well without too much investment or involvement. We are now running 56K on our system with two H8-5 serial boards and one H8-4 serial board. A parallel board has been used but is out of the line at this time.

With the disk system, 32K of Heath memory and two serial boards, my system was completely filled. An occasional program with the Extended Disk BASIC would give me a memory overflow error and I would lose a portion of vital files unless I could add to the system. Heath was contacted about the problem and the additional motherboard space needed, but to no avail. A man working at the Dallas Heath Retail center was contacted about the possibility of connecting two mainframes together. He had no knowledge of this having been done. A friend with a degree in electronics with a specialty in computers was contacted when he was at home for a vacation and we put a scope on the motherboard. We did find spike potentials on the address and data lines especially when loading from a cassette. He suggested shielding any connecting lines and terminating the slave mainframe on the data and address lines.

A second H8 was purchased from Heath and was carefully assembled. All traces with the exception of the ground traces were cut at location 1 on the motherboard. A grounded tinned copper bus was extended from 0 to 49 at this position. The 25 pin plugs were left out at P2 and all lines except 2, 47, 48, and 49 were connected through 680 ohm 1/4 w resistors to the ground bus. Four connector shells with enough spring clips were obtained from the parts department at Heath. Five feet of 50 conductor flat gray laminated ground plane cable with drain wire was obtained from the friend. This is Allied stock number 943-1635 and costs about $3 a foot if it can be found. It is only available from Allied in 100 foot rolls. It is tedious to separate the ground plane which is a mesh of fine copper laminated to the bottom of the flat cable, but it is worth it. The wire is not color coded and extreme care must be used in attaching the spring clips and the plugs. By experimentation, we found that it was not necessary or wise to connect the pins 2, 47, 48, and 49. Pins 0 and 1 must be connected and the drain wire must also be connected at each plug. Position 10 on each motherboard was connected with this cable.

Two Godbout 12K boards were obtained and assembled. These were tested first in the main H8 and found to be working well. They were then placed in the slave mainframe and a memory test with 64K as the high limit (377 377) was performed and left running for a couple of days with no failure. As word processing was my prime concern the H9 was sold and a ADM-3A kit was purchased and assembled. The lower case option was obtained from a chip compnay in California for $14 and the additional RAM was purchased for $12.50. This furnished me with a 24 line terminal with upper and lower case at about the least money output possible. I had tried the mods on the H9 with poor success and it wouldn't work without a definite flicker and jitter above 600 Baud anyway. The ADM-3A will function with no difficulty with the H8-5 at 9600 Baud.

REMOVE R152 ON THE H8-5 :JB:

To complete the system a NEC Spinwriter with panel was obtained. The NEC in serial configuration did not supply a CTS signal but it did have a full buffer signal. The H8-5 was modified with a n-p-n transistor switch grounding pin 17 of IC 124 on the H8-5 after cutting the ground trace to this pin. When the full buffer signal drops to zero the CTS of the IC 124 is raised above ground effecting an interrupt. When the buffer is sufficiently empty (about 60 bytes) the signal is positive 5 v and the transistor conducts, grounding pin 17 of the USART and allowing transfer of data. As pin 2 on P102 was not being used, it was pressed into service to serve as the input of this signal to the H8-5. This allows the NEC to run at 1200 Baud with very little interruption in output.

The slave mainframe does not use the Control circuit board or the CPU board. It is now serving as the housing for a H8-2 parallel board not currently used and for a H8-4 board driving a modem for communication over the phone line at 300 Baud to my office computer and to a Northstar Horizon at my home. The only complaint I have is that the slave mainframe had the holes in the front of the panel for the keyboard which I could only cover with tape. With the current units I have as good a wordprocessor and as effective a general purpose computer as any I have seen for three or four times the money. I am limited for disk storage, but in anticipation of Heath and their computer inquiry, I hope someday to see a reasonable 7.5M-10M hard disk added to their line with S-50 bus compatibility. I feel that this would probably be the ultimate hobby (?) system.

EOF

USING A H14 WITH A H8-5 IS OK.

Chesney Twombly
15 Storer Street
Kennebunk, ME 04043

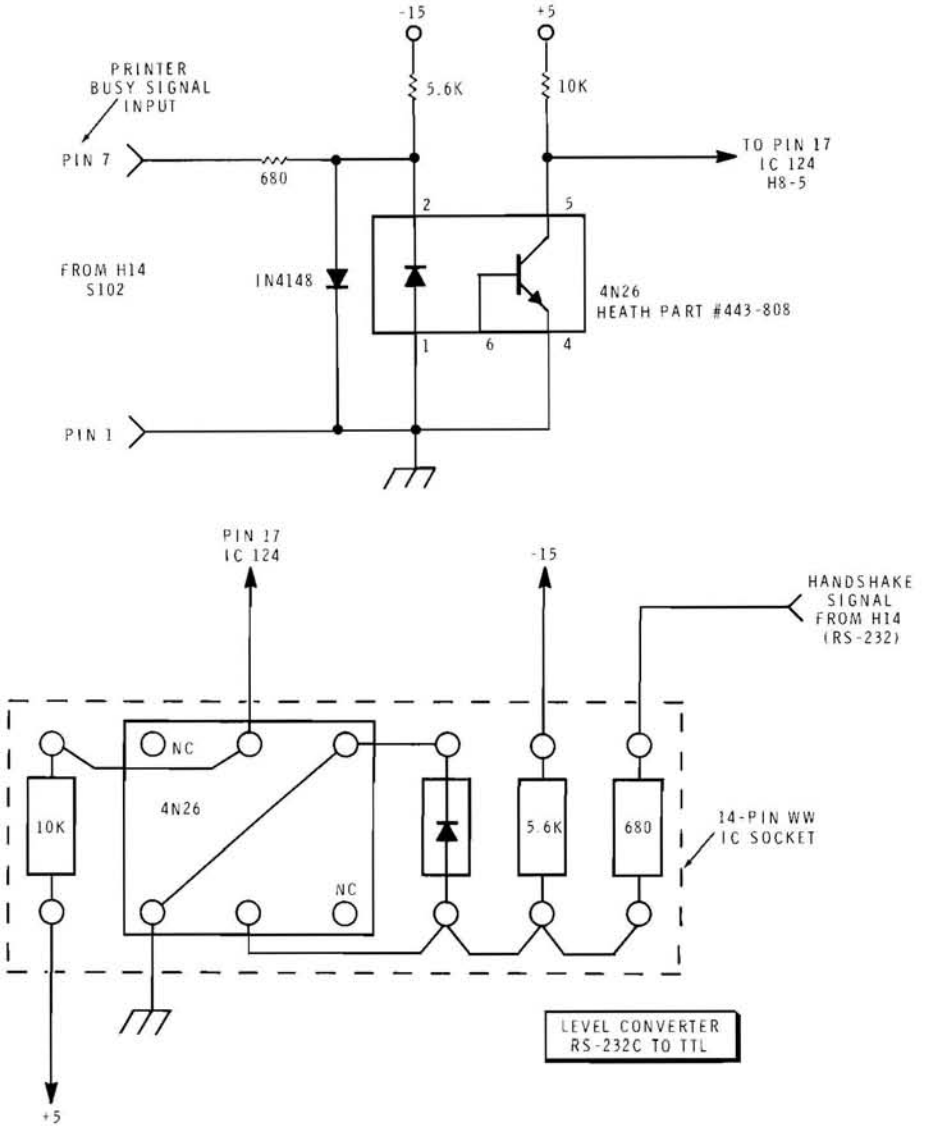
Here is some good news for H14 owner's and prospective buyers who would like to use it with the H8-5 Serial I/O and Cassette Interface. All it takes to do it is a simple outboard adapter to change the H14 'handshake' signal from RS-232C to TTL level. IC124 pin 17, is lifted from ground and connected to the output of the level converter. It will cost about $2.00 for parts. No need to get a H8-4 Multiport board.

The schematic shows everything you need to know. No construction details are given — the prototype was built on a 14-pin, wire wrap type IC socket, which holds all the parts and is mounted on the back side of the H8-5 board by soldering one of the socket's grounded pins to the eyelet hole in the ground foil near IC124 pin 17. Note that after isolating pin 17 from ground, a jumper must be added to restore the ground connection to pin 4.

Only three wires between H8 and H14 are needed. They can be permanently connected, if desired, since the printer when turned off has no effect on H8 operation. When hard copy is wanted, the H14 is powered up and its 'on line' button pushed down. Anything appearing on the H9 Video Terminal, will be printed on the H14. Nothing could be simpler. There is one restriction, however. The video display will be inhibited whenever the 'on line' button is in the out position. The reason for this effect is explained on page 10 of the H14 Operations manual.

The assembly manual has errors in the last two steps on page 76 and in the first step on page 77. This is in the "Initial Tests" section and is certain to cause inconvenience and unnecessary checking. The same information is given correctly in the Operation manual, pages 26 and 27.

A word of caution when connecting the H14 to the H8-5. Baud rates must be the same in both units. Mechanical damage to the printer could result if baud rates disagree. Be careful. A replacement print head is listed at $133.00 on the Heath parts list. I was lucky — fuse F101 blew and shut down the printing short of disaster.



Here's a simple modification to the H14 Line Printer that costs nothing. The path of the ribbon is almost horizontal. So the print head is using only a small protion of the width of the ribbon. This causes the print darkness to fade after a short time. A more ideal ribbon path would be sloping so that the print head strikes the ribbon on the top of the ribbon on one side and the bottom of the ribbon on the other.

It is easy to improve the situation by swapping around the shim washers used with the ribbon guide spools. I increased the ribbon tilt by moving one washer from the right front guide to the left rear guide and by moving both washers from the bottom of the right rear guide to the top. This should increase ribbon life and print uniformity and you can still turn the ribbon over and use the other half.

Bill Phillips
6 Monterey Circle
Ormand Beach, FL 32074

MODIFICATION OF AN H11-5 SERIAL INTERFACE FOR 19.2 K BAUD OPERATION

Grand J. Munsey
909 Kennard Way
Sunnyvale, Ca. 94087

INTRODUCTION

Many CRT terminals will run at 19.2 K baud. This speed allows extremely fast screen writing capability for such applications as screen oriented editors. The H11-5 is designed to run at baud rates up to 9600 baud. This document describes a simple modification which allows the H11-5 to run at 19.2 K baud without giving up the ability to run at other speeds.

THEORY

IC 27 (Fairchild 7402, Heath 443-793) is used to produce a crystal controlled clock which is 16 times faster than the required baud rate. The proper rate is selected by setting jumpers FR0 through FR3. By setting FR0 through FR3 to zero the frequency selected is input from pin 15 of IC27. This option was provided to allow users of the IC to produce custom baud rates in addition to the standard rates normally provided by the chip. The proper clock frequency for 19.2 K baud operation is provided by IC 27 at pin 3. Thus by connecting pin 3 to pin 15 a "custom" clock is available which runs at 19.2 K baud.

MODIFICATION

1. Make sure the H11 computer is powered off.
2. Remove the H11-5 serial interface.
3. Carefully remove IC 27 (Heath part number 443-793) and place it on a piece of conductive foam. This integrated circuit can be damaged by static electricity so please follow the instructions supplied by Heath when handling this part.
4. Turn the printed circuit board over so that the foil side shows.
5. Use a small piece of bare wire (a portion of 1/4 watt resistor lead will do) to connect pins 3 and 15 of IC 27 together. Be careful to place the jumper so that it does not touch any PC traces or other IC pins.
6. The baud rate selection jumpers FR0 through FR3 should be set to all zeros (all jumpers installed) for 19.2 K baud operation.
7. Turn the board over and re-install IC 27 in its socket. Be careful to insert the IC so that pin 1 is in the right place.
8. Install the interface in the H11 computer and re-connect the terminal cables.
9. Set the terminal to run at 19.2 K baud.
10. Use micro ODT to test the interface.

EOF

A/D CONVERTER

By: Don Moore

The analog to digital converter presented in this article is simple, yet versatile.

The wiring is not critical and the circuit can quickly be built up on the Heathkit H8-7 breadboard.

The heart of the converter is the National Semiconductor 16 channel, 8 bit A/D converter designated ADC0816. The 16 channel multiplexer can directly access any one of 16 single-ended analog signals, and operates on a single 5 V supply. Referrring to Figure 1, the 74LS155 and the 74LS00 are used as address decoders. In this case the ports are 0 to 16, but other port addresses are available at the unused pins of the 74LS155 (see table 1).

The 74LS240 is necessary to drive the data bus and provide signal inversion to satisfy the negative true bus of the H8.

The four lower address lines need not be inverted, so the channels are in reverse order (i.e., channel 0 is really channel 15, 1 is 14, etc.) but this does not affect the operation.

The 74C04 Hex inverter is used as a clock at approximately 500 kHz. This is not critical as the ADC0816 will run from 10 kHz to 1200 kHz, so it could be run off the H8 system clock if desired.

The ADC0816 is designed as a complete data acquisition system (DAS) for rationmatic conversion systems, the physical variable being measured is expressed as a percentage of full scale which is not necessarily related to an absolute standard. A good example of a ratiomatic transducer is a potentiometer. (See Figure 2.)

A program listing is provided for display of all channels and one for a 5 volt voltmeter in listings 1 and 2.

PARTS LIST

IC1	ADC0816	IC5	74C04
IC2	74LS240	IC6	LM340T-5
IC3	74LS155	R1	39 kΩ
IC4	74LS00	C1	120 pF

Listing 1

```
10 FOR I = 0 TO 15: OUT I, 0: PRINT PIN (I);: NEXT: END
```

Listing 2

```
5 REM USES CHANNEL 0 AS PROBE FOR 5 V VOLTMETER
10 OUT 0, 0
20 PRINT INT (PIN (0)/5.1 + .5)/10; " "
30 OUT 250, 13
40 GO TO 10
```

VECTORED TO PAGE 31

BUGGIN' HUG

I am sure many HUG members are duffers like myself. I for one would like to see the REMark articles written in English instead of computer jargon loaded with about 50% buzz words. The CPM article is a horrible example – what in hell are you talking about?

W. B. Grandjean

Dear HUG,

I appreciated the article on CPM, and I have a few more comments, both good and bad, on the subject. My own system is an H8 with a humble 12K of ram and cassette mass storage, but at work I have access to some Intel MDS's with both ISIS (Intel's own) and CPM disk operating systems. These two systems are even more different than HDOS and CPM, yet we have two programs that run under CPM, called FROMISIS and TOISIS that allow transfer from one system to the other, and I use both systems to experiment with programs that will eventually be used on my H8. I don't know where the programs came from, but most of our CPM software is from the user's group.

This brings me to the one bad thing that I have to say about CPM. Many of the programs from the user's group have bugs. For example, there was an editor that resembled the ISIS editor, which I really like, but unfortunately, it had the nasty habit of over-writing other disk files. This is not to say that all of the user's group software is buggy. In fact, most of it is terrific. But some of it has bugs. Caveat emptor.

One really nice thing about CPM that was not mentioned in the article is BASIC-E. It is not part of the CPM package, but most versions of CPM come with it thrown in at little or no cost, because it is public domain software. BASIC-E is a very nice full featured BASIC that has one important difference from most other BASIC's. It is not an interpreter, but rather a compiler-interpreter. BASIC-E programs are prepared using an editor, as in assembly language, and then compiled with the BASIC-E compiler. What is produced is not directly executable object code, but an intermediate code that must be run using a program called the "run-time monitor". The whole thing sounds complicated, but is much easier to do than to explain. The disadvantages of BASIC-E are that programs are a little harder to prepare and debug, and more ram is required than with a comparable interpreter. The big advantage is that programs run faster in BASIC-E than in most other BASIC's.

There are a lot of other nice things about CPM not mentioned in the article, and I am glad to know that Heath is joining the "family".

Patrick Swayne
290 Springdale
Sebastopol, CA. 95472

Since I purchased my H8 system last summer, my son has taken a tremendous interest in computer science. At the age of 10 he has a long way to go, but I am sure the experiences with our modern 'technology' will help him greatly in years to come.

We were present at the Heath Group meeting last year when you spoke about personal computers and the Heath system. Because of that I purchased a system and my son and I have been 'learning' ever since . . .

My son's success in learning and using computer systems has affected his relationship with friends. He even has his 8 year old sister practicing her weekly spelling words on the terminal.

Kevin E. Foley

Just received issue #6 of REMark, and as always found it very informative and interesting. I talked to you a couple of times via phone concerning a problem I was having in getting my integral data system IP-125 printer to work with the H8. Well as you can see, it is now working with the H8-5 serial board. Some of your readers may be interested. Memory location 040.367Q on all the new printer software needs to be changed from 302 to 312. This change lets the H8 accept an inverted (low) instead of high clear to send signal from the printer buffer.

Hugh R. Carrington
Amateur Radio Wb4TDY
3689 Huckleberry St.
Memphis, TN 38116

I am writing to once again make an announcement in your August 1979 issue, of a seminar program here at Virginia Tech. Dr. Jonathan Titus, Dr. Paul Field, Dr. Christopher Titus and I are directing these workshops.

WORKSHOPS: Two expanded workshops on 8080/8085/Z80 Microcomputer Design, Microcomputer Interfacing, Software Design and Digital Electronics are being given by the editors of the popular Blacksburg books. Participants have the option of retaining the equipment used in these courses. Dates are October 1 to 6, 1979. For more information contact Dr. Linda Leffel, C.E.C., VPI and SU, Blacksburg, VA 24061 (703-961-5241).

This effort on your part to bring these programs to the attention of readers is greatly appreciated by the Virginia Polytechnic Institute and State University Extension Division and the course directors.

The H9 screen erase works OK up to 600 baud, but is erratic at higher speeds. The reason seems to be that the erase pulse at these higher speeds sometimes occurs during an extended TPU cycle, and is gone by the time a screen refresh cycle occurs. The cure is to increase the timing resistor to 33 K ohms which stretches the erase pulse to about 20 ms. If a still longer pulse is needed, increase the size of the capacitor. The erase command is then (assuming the terminal at the usual port): OUT 250,5:PAUSE 10. This works on my H9 to 4800 baud. My H9 refuses to run at 9600, but I don't care since I don't like the blinking display above 600, so don't run any faster.

William C. Richter
1001-140 Evelyn Ter. E.
Sunnyvale, CA. 94086

Thanks for your REM #5 article on HT11/H27. It was a welcome sight. Having received my H27 in early December 1978, I was aware of the existence of CBASIC on my system disk, and having experimented with it, was vaguely aware of some of its advantages over BASIC.

I have enclosed a listing of the Resequence Program given as an example in Chapter 9 of my Heath HT11 BASIC USERS' GUIDE. It has been modified to handle ON-GOTO and ON-GOSUB statements. I have also included a provision for specifying a last input line. This eliminates the need to always renumber to the end of a program. Anyone who has not yet tried this program is missing out on a powerful and versatile tool worthy of attention.

(See Page 22)

Roland L. Penny
4505 Junction Dr.
El Paso, TX 79924

Greetings:

1. You asked for information regarding local HUG groups:

CINCINNATI

HUG-26 (started with the impetus of store #26) meets every second Tuesday at the Cincinnati Heath Electronic Center. We have a wide range of sophistication, of interests, and of equipment, and would like to include anyone interested in Heath computers, whether they have their own yet, or not. We'll send a copy of our newsletter "I/O PORT" to anyone who calls Don Skiff, 351-6830.

2. You asked for feedback on the new HUG Text Editor (I just figured our where "BWEDIT" came from: Thank you, Barry Watzman!

We judge it to be software that suits the do-it-yourself inclinations of Heath users, and we really get off on doing things with it (how can we experiment with object code?) — such as:

Routine GCMD (enclosed) prints, through a parallel port or H8-5 serial port, any part of the text buffer, exactly as it would be displayed on the CRT. It retains CNTRL-C and CNTRL-S, but doesn't go through an HDOS output buffer. In fact, it is exactly the same as the TCMD routine, substituting the printer port for the terminal. Unlike EDIT, it does not remove anything from the buffer as it prints.

Routine HCMD is a simple "halt" or PAUSE command that can be part of a macro, to stop and wait for the operator to hit RETURN — useful in printing or paging through the buffer (e.g. "B12TH#<PH>$$"). The H-8 horn alerts the operator, and the front panel displays "PAUSE-ing".

Routines XCMD and YCMD dump text to the cassette port, and load from it. The tape code isn't readable through TED-8 or tape BASIC, only through this editor. But if you need tape backup files, this is easy to do. And this is called automatically when the ecitor discovers "NO MORE DISK SPACE", to give you a chance to save your file that won't fit on disk. Later, you can put another disk in, and transfer the file from tape.

We're working on a justification routine (JCMD), and place to try our hand at block moves, and line identification, too.

So you see what a boon source code is? The basic file access and I/O routines give you a leg up, so you can experiment with assembly language programming without having to know all the complexities. Also, programming for text manipulation seems easier than number crunching, for beginners. It's a good way to become familiar with the 8080 instruction set.

Now, is there a way we can get source code for the I/O routines in the tape software? We'd like to try modifying BWEDIT to run from cassette.

And thanks for the interviews on CP/M — they help us to evaluate, and that's hard to do in this field when nobody tells us anything. We're aware of the problem of proprietary software protection, but those of us who are trying to learn to program have a rough time adapting magazine programs to the Heath architecture.

Cheers.

Mike Morrison
Don Skiff
HUG-26
Cincinnati, OH

```
        TITLE   'HUGED-II EDITOR WITH PRINTER ROUTINE'
*
           *** BY MORRISON & SKIFF, CINCINNATI ***
*
*THIS VERSION HAS BEEN MODIFIED TO PRINT THROUGH I/O PORT
*374, USED WITH A PARALLEL INTERFACE AND SELECTRIC PRINTER.
*IT DOES NOT USE HDOS OUTPUT BUFFER, AND DOES NOT DELETE THE
*TEXT IT PRINTS; USE COMMAND "G" TO PRINT, EXACTLY AS "T"
*IS USED TO TYPE ON CONSOLE.
*
*IT WILL ALSO PERMIT DUMPING OF BUFFER TO CASSETTE, WITHOUT
*DISTURBING BUFFER.  THIS ROUTINE ALSO IS CALLED AUTOMATICALLY
*IN THE EVENT OF INSUFFICIENT SPACE ON OUTPUT DISK.
*TYPE "X" TO DUMP, TYPE "Y" TO LOAD
*
PORT    EQU     374Q            ;SAME AS AT:
START   EQU     040000A         ;DUMP ROUTINE FOR PAM-8
ABUSS   EQU     040024A         ;SEE PAM-8 PAGE 21
DUMP    EQU     002002A         ;TAPE FILES NOT USABLE
LOAD    EQU     001267A         ;BY OTHER SOFTWARE
ALARM   EQU     002136A
$MOVE   EQU     030252A
SUPCAS  DB      12Q,'** SET UP CASSETTE **',212Q
CASERR  DB      12Q,'** CASSETTE ERROR **',212Q
$DSPMOD EQU     040007A
.MFLAG  EQU     040010A
PAUSE   DB      255,152,144,131,164,140,123,57,32
FPLEDS  EQU     040013A
*
*
********** PART OF WRITEF ROUTINE **********
        RNC                     ;DONE IF NO ERROR
        LXI     H,NSPMSG        ;SCREAM ABOUT ERROR
        SCALL   .PRINT          ;
        MVI     A,2             ;CLOSE FILE (SAVE WHAT WE CAN)
        SCALL   .CLOSE          ;DO IT
*
*SAVE THE FILE BY DUMPING TO CASSETTE
*
        CALL    XCMD            ;LAST CHANCE
*
*FINAL EXIT BACK TO HDOS
*FIRST RESET HDOS CONSOLE TYPE BYTE
*
```

```
BYEBYE  LDA     CONVAL          ;GET OLD CONSOLE VALUES
        MOV     B,A             ;IN REG B
        MVI     C,10H           ;SET UP TO RESTORE THOSE WHICH WE RESET
        MVI     A,I.CONTY       ;
        SCALL   .CONSL          ;SO WE GO BACK AS WE CAME IN
*****************************************

******** PART OF MAIN ROUTINE -- SEE GINIT *********
        LXI     B,SIGNON        ;PRINT SIGNON MESSAGE
        LXI     D,ILGMSG-SIGNON-1
        CALL    MESSBC
        CALL    GINIT           ; INITIALIZE PORT
        CALL    GINIT           ;HIT IT AGAIN
MAIN2   LXI     H,0FFFFH        ;ASK FOR MAX MEMORY
        SCALL   .SETTP          ;FIND MAX MEMORY SIZE
        SHLD    MAXMEM          ;
        SCALL   .SETTP          ;NOW GET IT FROM HDOS
*****************************************
*
*PRINTER OUTPUT ROUTINES - SAME AS "TYPE" COMMAND,
*EXCEPT SENDING CHARACTER TO "PORT" INSTEAD OF CONSOLE.
*RETAINS CONSOLE  CONTROL THROUGH CNTRL C AND CNTRL S
*NOTE THAT PORT WAS CONFIGURED DURING 'MAIN' ROUTINE,
*RIGHT AFTER SIGNON
*
GCMD    CALL    SKPLIN
        XCHG
        LHLD    TXTPTR
        MOV     A,L
        SUB     E
        MOV     L,A
        MOV     A,H
        SBB     D
        MOV     H,A
        XCHG
        JNC     GCMD1
        LHLD    TXTPTR
        MOV     A,D
        CMA
        MOV     D,A
        MOV     A,E
        CMA
        MOV     E,A
        INX     D
GCMD1   MOV     B,H
        MOV     C,L
        MOV     A,D
        ORA     E
        DCX     D
        JNZ     GESSBC          ;WAS MESSBC
        RET
*
GESSBC  MOV     H,B
        MOV     L,C
GESSHL  MOV     C,M
        INX     H
        CALL    GCHAR           ;WAS PCHAR
        CALL    BRKCHK
        MOV     A,D
        ORA     E
        RZ
        DCX     D
        JMP     GESSHL
*
GCHAR   PUSH    D
        PUSH    H
        MOV     A,C
        SUI     9
        JNZ     GCHAR1
        MVI     C,020H
GCHAR1  SUI     1
        SBB     A
        MOV     B,A
        LDA     LINPOS
        ANI     7
        CMA
        ADI     8
        ANA     B
        MOV     B,A
        MOV     A,C
        SUI     0AH
        JNZ     GCHAR2
        STA     LINPOS
```

```
GCHAR2  PUSH    B
        CALL    PRTOUT          ;WAS CONOUT
        POP     B
        LXI     H,LINPOS
        MOV     A,C
        CPI     020H
        JNC     GCHAR4
        CPI     'G'-40H
        JZ      GCHAR5
        CPI     0AH
        JZ      GCHAR5
        CPI     'H'-40H
        JNZ     GCHAR3
        DCR     M
        JMP     GCHAR5
GCHAR3  INR     M
GCHAR4  INR     M
GCHAR5  DCR     B
        JP      GCHAR2
        POP     H
        POP     D
        RET
*
PRTOUT  MOV     A,C
        CPI     0AH             ;IF NEWLINE, MAKE CR
        CZ      CRCNV           ;CONVERSION
PRTOUT1 CALL    GOUT            ;DIRECT TO PORT(NO BUFFER)
        CALL    CSTS            ;STILL NEED CONTROL
        ORA     A
        RZ
        CALL    CONIN
        CPI     'C'-40H
        JZ      BREAK
        CPI     'S'-40H
        RNZ
        CALL    CONIN
        RET
*
CRCNV   MVI     C,0DH           ;CR CHARACTER
        CALL    GOUT            ;PRINT THE CR
        MVI     C,0             ;PADDING
        CALL    GOUT
        RET
*
GOUT    IN      PORT+1          ;IS PORT READY?
        ANI     1
        JZ      GOUT            ;NOT YET
        MOV     A,C             ;GET CHARACTER AGAIN
        OUT     PORT
        RET
*
GINIT   MVI     A,100Q          ;RESET PORT USART
        OUT     PORT+1
        MVI     A,116Q          ;CONFIGURE PORT:MODE WORD
        OUT     PORT+1
        MVI     A,005Q          ;COMMAND WORD
        OUT     PORT+1
        RET                     ;ALL SET TO PRINT
```

```
*
*GINIT ROUTINE IS CALLED DURING INITIALIZATION OF THE EDITOR,
*TWICE, BECAUSE USART IS RESET ON POWER UP, AND RESETTING
*IT WITHOUT CONFIGURING IT THROWS IT OUT OF SYNCH.  THIS PROCESS
*ENSURES IT IS CONFIGURED WHETHER THE EDITOR IS BEING CALLED FOR
*THE FIRST TIME, OR SUBSEQUENTLY.
*
* HCMD HALTS PROCESSING, WAITS FOR 'RETURN', LIGHTS FRONT PANEL,
* SOUNDS THE HORN.  IT IS CALLED BY CASSETTE LOAD/DUMP ROUTINES,
* OR CAN BE USED IN MACRO COMMANDS
*
HCMD    LDA     .MFLAG          ;TURN OFF DISPLAY UPDATE SO THAT
        ORI     00000010B       ;"PAUSEING" CAN BE WRITTEN IN THE FPLEDS
        STA     .MFLAG
        LXI     B,9             ;WRITE FRONT-PANEL LEDS
        LXI     D,PAUSE
        LXI     H,FPLEDS
        CALL    $MOVE
        CALL    ALARM
HCMD1   CALL    CONIN
        CPI     10
        JNZ     HCMD1
        CALL    CRLF
        LDA     .MFLAG          ;TURN DISPLAY UPDATE BACK ON
        ANI     11111101B
        STA     .MFLAG
        RET
*
* ROUTINE TO DUMP ENTIRE BUFFER TO CASSETTE.  TAPES ARE NOT
* COMPATIBLE WITH TED-8 OR TAPE BASIC, TXTCOM OR BASCOM.
* USE YCMD TO LOAD >>>INTO EMPTY BUFFER<<<
*
XCMD    CALL    CASETE          ;GO DO CASSETTE SET-UP AND PAUSE
        LXI     H,TXTTOP        ;START OF TOP-OF-BUFFER ADDR DUMP
        SHLD    START
        LXI     H,TXTTOP+1      ;END OF TOP-OF-BUFFER ADDR DUMP
        SHLD    ABUSS
        CALL    DUMP            ;CALL PAM-8 DUMP ROUTINE
        LXI     H,TXTBUF        ;START OF TEXT-BUFFER DUMP
        SHLD    START
        LHLD    TXTTOP          ;END OF TEXT-BUFFER DUMP
        SHLD    ABUSS
        CALL    DUMP            ;CALL PAM-8 DUMP ROUTINE
XYEND   LXI     H,$DSPMOD       ;SET FRONT-PANEL DISPLAY MODE TO
        MVI     M,2             ;REGISTER READ
        RET
*
YCMD    CALL    CASETE          ;GO SET-UP CASSETTE AND PAUSE
        LXI     H,YCMDERR       ;CASSETTE LOAD ERROR ADDR
        CALL    LOAD            ;CALL PAM-8 LOAD ROUTINE TWICE
        CALL    LOAD
        JMP     XYEND
YCMDERR LXI     H,CASERR        ;CASSETTE ERROR MESSAGE
        SCALL   .PRINT          ;PRINT IT
        JMP     XYEND
*
CASETE  LXI     H,SUPCAS        ;SET-UP CASSETTE MESSAGE
        SCALL   .PRINT          ;PRINT IT
        CALL    HCMD            ;PAUSE
        LXI     H,$DSPMOD       ;SET-UP FRONT-PANEL FOR
        MVI     M,0             ;MEMORY DISPLAY
        RET
*
```

MEETINGS and CLUB NOTICES

Seattle WA

pacific Northwest H8 owners club. . . Contact Marty Lindal at 283-0806 or Gary Hawthorne at the Heath Electronic Center at 682-2172.

Corpus Christi TX

Member Peter Tewes needs some help with his H8 system. Anyone around that area that could give Peter a hand? Contact him at 4445 Hannigan Dr. Corpus Christi 78413.

Milwaukee WI

Anyone interested in forming a H8 users' group should contact Marvin Olson at 9040 N. Lake Dr. Milwaukee, WI 53217 or phone him at 352-3346.

Kailua HI

Contact Gerry Cramm at 2545a Lawrence Place Kailua HI 96734.

Virginia Beach VA

Contact Jim Egerton at (804) 464-9487 or 460-0997.

Indianapolis IN

Contact Howard at store #34 for meetings times and dates.

Incidentally Jon Hoerbel et al visited the Heath plant in June and we enjoyed meeting and chating with the group. If you plan on being in the area, please let us know and we will arrange for the 25¢ tour.

Hialeah FL

The local HUG group meets the 3rd Tuesday of every month at 7:00 PM at the Heath Electronic Center. Contact Ray for further details.

Buffalo NY

Contact John Hodge at 716-662-7122 for the details of the Buffalo area users' group meetings.

Danders MA

HUG Northshore meets every 2nd Wedensday at 7 PM at the Hilltech Bldg. at 88 Holten St. (3rd floor) Danders, MA. 01923. You can get a free copy of the newsletter by writing Perry Miller at PO Box 112 Danders, MA.

Detroit MI

The Metro Detroit Area Heathkit H8 Computer Users Group meets in various places once monthly. . . contact Bob Mathias at 313-465-0068 for details.

VECTORED FROM PAGE 13

To demonstrate how much easier it would be to write a program in TINY BASIC than in machine code, Figure 4 is a TINY BASIC program which performs a simple guessing game between human and machine. Best of all I didn't have to hand assemble this program. I just typed it in on my terminal, typed the RUN command, and TINY BASIC did the rest.

The software reference manual for the accessory box contains an appendix with several articles which demonstrate the use of the TINY BASIC statements. Since TINY BASIC is a subset, these articles also show alternate ways to provide for some of the regular BASIC statements which are not part of the subset.

As an example of an alternate method for providing a statement found in regular BASIC, let us look at the FOR and NEXT statements. In regular BASIC the FOR statement is used for executing a section of the program a number of times. The syntax for this statement is FOR Var=Exp1 TO Exp2. The Var is initialized to the value of Exp1. The section of the program up to the NEXT statement is executed. When the NEXT statement is encountered, Var is incremented by one. If the value of Var is not greater than the value of Exp2, then the section of the program is executed again. Figure 2 is a regular BASIC program that prints HELLO ten times. Figure 3 is a program written in TINY BASIC that does the same thing.

STATEMENT	DESCRIPTION
REM TEXT	THE REMARK IS A NONEXECUTABLE STATEMENT USED ONLY FOR COMMENTARY.
VAR=EXPRESSION	THIS INSTRUCTION ASSIGNS THE VALUE OF THE EXPRESSION TO THE VARIABLE.
INPUT VAR	THIS INSTRUCTION ALLOWS YOU TO READ DATA FROM THE KEYBOARD AND ASSIGN VALUES TO THE VARIABLES.
PRINT VAR	THE VALUE OF THE VARIABLE IS PRINTED ON THE CONSOLE TERMINAL.
GOTO NNN	THE PROGRAM IS UNCONDITIONALLY TRANSFERED TO THE STATEMENT NUMBER NNN.
GOSUB NNN	A SUBROUTINE CALL IS EXECUTED TO STATEMENT NNN.
RETURN	RETURN FROM A SUBROUTINE.
IF EXP1 REL EXP2 THEN STMT	IF THE TEST "EXP1 REL EXP2" IS TRUE, THE STATEMENT AFTER THE "THEN" IS EXECUTED.
RUN	THIS INSTRUCTION STARTS PROGRAM EXECUTION.
LIST	THIS INSTRUCTION LISTS THE PROGRAM ON THE CONSOLE TERMINAL.
CLEAR	THIS INSTRUCTION ERASES THE CURRENT PROGRAM FROM MEMORY.
BYE	THIS INSTRUCTION EXITS TINY BASIC AND RETURNS TO THE ETA-3400 MONITOR.
SAVE	THIS INSTRUCTION SAVES THE PROGRAM ON CASSETTE TAPE.
LOAD	THIS INSTRUCTION RETRIEVES A PREVIOUSLY SAVED PROGRAM FROM CASSETTE TAPE.

Figure 1

```
10 FOR I=1 TO 10
20 PRINT "HELLO"
30 NEXT I
40 END
```

Figure 2

```
10 I=1
20 PRINT "HELLO"
30 I=I+1
40 IF I<=10 THEN GOTO 20
50 END
```

Figure 3

```
010 REM GUESS MY HEX NUMBER GAME.
020 PRINT "I AM THINKING OF A HEX NUMBER BETWEEN 0 AND F"
030 PRINT "ENTER YOUR GUESS ON THE ET-3400 KEYBOARD"
040 PRINT "IF YOU ARE HIGH OR LOW, THEN I WILL DISPLAY"
050 PRINT "'HI' OR 'LO' RESPECTIVELY ON THE ET-3400 DISPLAY"
060 PRINT "IF YOU ARE CORRECT, THEN I WILL RING YOUR"
070 PRINT "CONSOLE TERMINAL BELL"
080 PRINT
090 PRINT "GOOD-LUCK HUMAN"
100 REM USE RND FUNCTION TO SELECT NUMBER TO BE GUESSED.
110 A=RND(16)
120 REM LET HUMAN GUESS.  USE ET-3400 MONITOR ROUTINE INCH.
130 B=USR(-524,0,0)
140 REM= RESET ET-3400 DISPLAY VIA ET-3400 MONITOR ROUTINE REDIS.
150 C=USR(-836)
160 REM IF GUESS IS HIGH, DISPLAY HI VIA OUTCH.
170 IF A>=B THEN GOTO 210
180 C=USR(-454,0,55)
190 C=USR(-454,0,48)
200 GOTO 130
210 REM IF GUESS IS LOW, DISPLAY LO VIA OUTCH.
220 IF A=B THEN GOTO 260
230 C=USR(-454,0,14)
240 C=USR(-454,0,126)
250 GOTO 130
260 REM HUMAN GUESSED CORRECT.
270 C=USR(-454,0,0)
280 C=USR(-454,0,0)
290 C=USR(7177,0,7)
300 PRINT
310 PRINT "CORRECT-  -- TRY ANOTHER"
320 GOTO 80
```

Figure 4

BASIC IDEAS

This program permits formatting of dumps from the H8 Disk Files to a printer not having any type of forms control. It will print each page with the file name, current date, and page number, and allows the user to specify the number of lines desired per page.

Allan H. Moser

LISTDUMP.BAS

```
00010 D$="":FOR I=8383 TO 8391:D$=D$+CHR$(PEEK(I)):NEXT
00020 PRINT "     PROGRAM 'LISTDUMP'     11-JAN-79"
00030 PRINT
00040 PRINT "THIS ROUNTINE WILL OUTPUT A DISK FILE TO A PRINTER WITH"
00050 PRINT "A SELECTED NUMBER OF LINES PER PAGE.  TO SET THE NUMBER"
00060 PRINT "OF LINES, MODIFY LINE NUMBER 150 BY SETTING Y= TO THE"
00070 PRINT "NUMBER OF LINES DESIRED PER PAGE.  EACH PAGE WILL SHOW THE"
00080 PRINT "FILE NAME, CURRENT DATE AND PAGE NUMBER."
00090 PRINT
00100 PRINT "THE ROUTINE ASSUMES YOU ARE USING THE ALTERNATE TERMINAL DRIVER"
00110 PRINT "AT: FOR YOUR PRINTER.":PRINT
00120 LINE INPUT "ENTER FILE FILE NAME TO BE PRINTED (DEV:FNAME.EXT) ";F$
00130 OPEN F$ FOR READ AS FILE #1
00140 OPEN "AT:" FOR WRITE AS FILE #2
00150 Y=60
00160 C=1
00170 PRINT #2,TAB(42);F$;TAB(60);D$;TAB(70);"PAGE ";C:PRINT #2,
00180 FOR X=1 TO Y
00190 J=CIN(1)
00200 IF J<=0 GOTO 270
00210 LINE INPUT #1,;S$
00220 S$=CHR$(J)+S$
00230 PRINT #2,S$
00240 NEXT X
00250 PRINT #2,:PRINT #2,
00260 GOTO 170
00270 CLOSE #2
00280 CLOSE #1:CLOSE #2
00290 END
```

```
10 REM ******************** RENUM.BAS **********************
20 REM
30 REM This program will renumber HT-11 BASIC program lines in a new
40 REM sequence.  In order to use RENUM your BASIC program must be stored
50 REM on disk.  This program will use that disk file as its source.  It
60 REM will generate a new file on the disk (which may have a different name)
70 REM which will contain your  renumbered BASIC program. (Note: if the same
80 REM file name is used for output, your original file with original numbers
90 REM will be replaced by the new one.)  All program statements involving
100 REM branches, such as GOTO or GOSUB, are updated to point to the new line
110 REM numbers, even those outside the range of lines being renumbered.
120 REM
130 REM If the output file will have the same name as the input file, merely
140 REM press "RETURN" in response to the question "NEW FILE NAME?".  If either
150 REM the Input or Output file is located on other than the system disk,
160 REM include the appropriate prefix, DX0: or DX1: (do not use quotation
170 REM marks in giving file names).
180 REM
190 REM If a "0" is entered for the LAST INPUT LINE then the end of the
200 REM program is assumed.  If a "0" is entered for the FIRST INPUT LINE,
210 REM then the beginning of the program is assumed.  If a "0" is entered
220 REM for the INTERVAL SIZE, then an interval size of "10" is assumed.
230 REM If "0" is entered for FIRST OUTPUT LINE, then "10" is assumed.
240 REM
250 DIM L(500),M(500),K$(2)
260 READ D
270 DATA 500
280 READ K$(0),K$(1),K$(2)
290 DATA 'GO TO ','THEN ','GOSUB '
300 PRINT 'RESEQUENCE'
310 PRINT 'OLD FILE NAME:';
320 INPUT P$
330 PRINT 'NEW FILE NAME:';
340 INPUT Q$
350 PRINT 'FIRST INPUT LINE:';\INPUT L0
```

```
360 PRINT 'LAST INPUT LINE:';\INPUT L3
370 PRINT 'FIRST OUTPUT LINE:';\INPUT L1
380 PRINT 'INTERVAL SIZE:';\INPUT I1
390 IF L3<>0 THEN 400 \L3=65532
400 IF Q$<>'' THEN 410 \Q$=P$
410 P$=P$&'.BAS'
420 Q$=Q$&'.BAS'
430 IF L0<>0 THEN 440 \L0=1
440 IF L1<>0 THEN 450 \L1=10
450 IF I1<>0 THEN 460 \I1=10
460 OPEN P$ AS FILE #1
470 C=-1
480 IF END #1 THEN 600
490 INPUT #1:L$
500 L2=L2+1
510 T=POS(L$,' ',1)
520 S$=SEG$(L$,1,T-1)
530 S=VAL(S$)
540 IF S<L0 THEN 480
550 IF S>L3 THEN 480
560 C=C+1
570 IF C>D THEN 1070
580 L(C)=S
590 GO TO 480
600 S=INT(L1)
610 FOR I=0 TO C
620 M(I)=S
630 IF S>65530 THEN 1080
640 S=S+I1
650 NEXT I
660 RESTORE #1
670 OPEN Q$ FOR OUTPUT AS FILE #2
680 FOR I=1 TO L2
690 INPUT #1:L$
700 C2=POS(L$,' ',1)-1
710 C1=1
720 GOSUB 950
730 FOR J=0 TO 2
740 C1=1
750 C1=POS(L$,K$(J),C1)
760 IF C1=0 THEN 900
770 C1=C1+LEN(K$(J))
780 C2=POS(L$,' ',C1)-1
790 E=POS(L$,'\',C1)
800 IF E<>0 THEN 810 \E=256
810 Q1=POS(L$,"'",C1)
820 Q2=POS(L$,'"',C1)
830 IF C2<>0 THEN 840 \C2=E-1
840 IF (E-Q1)*Q1>0 THEN 750
850 IF (E-Q2)*Q2>0 THEN 750
860 GOSUB 950
870 IF SEG$(L$,C2+2,C2+2)<>',' THEN 750
880 C1=C2+3
890 GO TO 780
900 NEXT J
910 PRINT #2:L$
920 PRINT L$
930 NEXT I
940 PRINT '--->DONE<---'\END
950 S$=SEG$(L$,C1,C2)
960 S=VAL(S$)
970 IF S>=L0 THEN 980 \RETURN
980 FOR K=0 TO C
990 IF L(K)=S THEN 1020
1000 NEXT K
1010 RETURN
1020 L1$=SEG$(L$,1,C1-1)
1030 L3$=SEG$(L$,C2+1,256)
1040 L2$=STR$(M(K))
1050 L$=L1$&L2$&L3$
1060 RETURN
1070 PRINT 'TOO MANY LINES'\STOP
1080 PRINT 'LINE NO. TOO BIG'\STOP
1090 END
```

Roland L. Penny
4504 Junction Drive
El Paso, Texas 79924

```
00010 REM           BASIC PROGRAM TO ILLUSTRATE POKING COMMANDS INTO
00020 REM           THE HDOS TYPE AHEAD BUFFER.
00030 REM           IF USED AS A SUBROUTINE ENTER AT LINE 130 WITH M$
00040 REM           CONTAINING THE COMMAND(S) TO BE POKED INTO THE
00050 REM           BUFFER.  EACH COMMAND MUST BE TERMINATED WITH A
00060 REM           CHR$(10) NEW LINE CHARACTER.  ALSO TERMINATE THE
00070 REM           ROUTINE WITH A "RETURN" STATEMENT.
00080 REM           RUNNING THE PROGRAM BELOW CAUSES IT TO
00090 REM           CONTINUALLY LIST ITSELF AND THEN RUN ITSELF.
00100 REM           VARIABLES USED ARE M$, Z, Z1, Z2, Z3, Z4, Z5 AND X.
00110 M$="LIST"+CHR$(10)+"RUN"+CHR$(10)
00120 REM           Z=ADDRESS OF START OF HDOS
00130 Z=PEEK(8400)+256*(PEEK(8401))
00140 REM           Z1=ADDRESS OF LINE COUNTER
00150 Z1=Z+2025
00160 REM           Z2=ADDRESS OF POINTER TO LOW BYTE OF TAIL OF QUEUE
00170 Z2=Z+2029
00180 REM           Z3=ADDRESS OF POINTER TO LOW BYTE OF HEAD OF QUEUE
00190 Z3=Z+2031
00200 REM           Z4=ADDRESS OF POINTER TO LOW BYTE OF START OF BUFFER
00210 Z4=Z+2033
00220 REM           Z5=ADDRESS OF POINTER TO END OF BUFFER+1
00230 Z5=Z+2035
00240 REM           MAKE SURE M$ IS TERMINATED BY A NEW LINE CHARACTER.
00250 REM           IF M$ CONTAINS SEVERAL SEPERATE COMMANDS IT IS THE
00260 REM           RESPONSIBILITY OF THE PROGRAM PASSING M$ TO THIS
00270 REM           ROUTINE TO SEE THAT EACH IS TERMINATED BY A NEW LINE.
00280 IF RIGHT$(M$,1)<>CHR$(10) THEN M$=M$+CHR$(10)
00290 REM           HERE'S WHERE THE STRING IS PUT INTO THE QUEUE
00300 FOR X=1 TO LEN(M$)
00310 REM           STORE CHARACTER IN QUEUE
00320 POKE (PEEK(Z2)+256*(PEEK(Z2+1))),(ASC(MID$(M$,X,1)))
00330 REM           IF CHARACTER IS A NEW LINE CHARACTER INCREMENT LINE COUNTER
00340 IF MID$(M$,X,1)=CHR$(10) THEN POKE Z1,(PEEK(Z1)+1)
00350 REM           INCREMENT TAIL POINTER, IF THIS CAUSES IT TO POINT PAST THE
00360 REM           END OF THE BUFFER MAKE IT POINT TO THE HEAD OF THE BUFFER
00370 POKE Z2,(PEEK(Z2)+1)
00380 IF PEEK(Z2)=PEEK(Z5) THEN POKE Z2,PEEK(Z4)
00390 REM           CHECK IF QUEUE IS FULL
00400 IF PEEK(Z2)=PEEK(Z3) THEN PRINT "QUEUE FULL!":END
00410 REM           GET NEXT CHARACTER
00420 NEXT X
00430 REM           ALL DONE
00440 END
```

The **BASIC IDEA** by Sam Cox seems to be one of the better ideas HUG has published. I find myself referring to it whenever a BASIC problem develops.

One of the most useful ideas is the money formatter routine. It does, however, have one drawback and that is that values above 9,999.99 revert to engineering notation.

An easier and shorter method is one adapted from an article by Dr. Marc I Leavy in the June issue of KILOBAUD/MICRO-COMPUTING starting on Page 54.

The following is a listing of the process used to generate right justified columns of figures:

```
1000 REM A= ENTRY A$=EXIT
1010 A1= -(INT(A)-A)
1020 A$= STR$(A1)
1030 IF A1<.10 THEN A$= "0.10"
1040 IF LEN(A$)= 4 THEN 1070
1050 IF LEN(A$)> 4 THEN A$= LEFT$(A$,4)
1060 IF LEN(A$)< 4 THEN A$= A$+"0"
1070 A$= RIGHT$(A$,3)
1080 A$= STR$(INT(A))+A$
1090 PRINT"RIGHT JUSTIFIED NUMBER";TAB(40-
     LEN(A$));A$
```

Line 1030 sets the minimum for A1. In other words, all dollar values below 10¢.

The above method will allow dollar values up to 99,999.99 before reverting to engineering notation. If a leading dollar sign is required change line 1090 to the following:

```
1090 PRINT"RIGHT JUSTIFIED NUMBER";TAB(33)"$";
     TAB(40-LEN(A$));A$
```

The TAB values determine where the first digit of the $ or number is to be printed therefore they can be set to any practical value between 1 and 70.

Lines 1000 through 1080 can be used as a sub-routine while line 1090 would be a RETURN. The print out is still accomplished by the information contained in the listing (line 1090).

Keep up the good work.

Larry T. Wier
1068 149Pl. S.E.
Bellevue, WA 98007

Here is something that may be of interest to other HUG members. In REMark Issue 3 (Page 17) there is a description of how to use GET and PUT under program control by using a series of POKE's to set up command mode instructions. The following routine will simplify this process:

```
1000 FOR Q = 1 TO LEN(Q$)
1010 Q1$ = MID$(Q$,Q,1)
1020 IF Q1$ = "\" THEN Q1 = 13: GOTO 1050
1030 IF Q1$ = "↑" THEN Q1 = 34: GOTO 1050
1040 Q1 = ASC(Q1$)
1050 POKE 8301 + Q, Q1
1060 NEXT Q
1070 POKE 8301, LEN(Q$)
1080 RETURN
```

To use the routine, let Q$ = the string of instructions we wish to execute in command mode. "\" will generate a carriage return and "↑" will generate a double quote. For example, to simulate the GET routine on Page 17 (lines 65000-65060) we merely write:

```
Q$="UN\GEZ$\YCON\":GOSUB 1000:STOP
```

Now that there is a simple way to force the computer into command mode, there are some other interesting things that can be done. While in command mode we can add or delete program lines. That is, a BASIC program can be self-modifying. Here are two useful applications.

(1) One of the problems in using GET is that all program variables in use before using GET are lost; i.e. set to zero or loaded with some other value from tape. Suppose we wish to retain a variable "X" so it will have the same value after GET as it had before. We place in the beginning of the program a dummy line: 100X = 0. Now, when we wish to GET a new record from tape we write:

```
Q$="100 X=" + STR$(X) + "\UN\GETT\YCON\":GOSUB
1000:STOP
```

When the preceding line is executed, command mode is entered, line 100 is replaced with a line saying "X = whatever-the-current-value-of-X-is", and GET is executed. After CONTINUE the program jumps to the start of the program, line 100 is executed, and X will be set back to its previous value.

(2) The second application concerns a common problem in many BASIC programs. Most programs have some initializing routine, commonly in the form of a series of user instructions that must be printed out at the start of the program. On a large program with limited memory, one must make a trade-off between giving adequate instructions and leaving enough memory to execute the program. Here is a simple solution:

Suppose lines 10 through 90 are a series of print instructions to be executed one time at the start of the program. For line 100 we write the following:

```
Q$="DEL10,100\RU\":GOSUB 1000:STOP
```

When the preceding line is executed, command mode is entered, lines 10 through 100 are deleted and the program is immediately reRUN, this time with both the print instructions and the delete instructions missing. One could of course also delete subroutine 1000-1080 or any REMarks.

There are unlimited possibilities of self-modifying programs. One must only keep in mind that if program lines are to be added or deleted which precede the instruction STOP (causing the command mode to be entered), then one must use RUN rather than CONTINUE as the final command mode instruction. Otherwise the BASIC interpreter will try to continue at an invalid address, the program statements having been shifted in memory. However, after GET, the program will jump to the first program instruction in any case and CONTINUE must be used to retain the variables just loaded from tape.

Paul Doudna

There is frequent need to pull data out of any array in random sequence, particularly for games. This method random "cuts" any **even numbered** array, S$(F), of F "cards" at C, finds the midpoint, M, to form two equal "stacks", and one-on-one "riffles" the "stacks" together, arithmetically. It then repeats the whole performance, resulting in a computed "random two-riffle shuffle", without actually moving the data.

Before the game starts, provide for the variables:

```
50 DIM S$(F),C(2),M(2):FOR A=1TO F:READ S$(A):NEXT A
60 C=RND(-PEEK(8219)):REM - SEED FROM TICCNT
70 FOR A=1TO 2:C(A)=INT(F*RND(1))+1:REM - RANDOM
   CUTS
80 M(A)=C(A)+F/2:IF M(A)>F THEN M(A)=M(A)-F:REM -
   MIDPOINTS
90 NEXT A
```

During the game, start each turn, S, (counted up or down,) by calling this subroutine:

```
300 D=S:FOR A=1TO 2
310 IF INT(D/2)=D/2THEN E=2:C=M(A):GOTO 330:REM -
    FOR EVEN TURNS
320 E=1:C=C(A):REM - FOR ODD TURNS
330 FOR B=E TO D STEP 2:C=C+:IF C>F THEN C=1:REM -
    RIFFLE
340 NEXT B:D=C:NEXT A:RETURN :REM - D IS THE DRAW
```

The player at turn S will "draw card D", S$(D), just as though the data had been literally cut, shuffled and dealt.

If you want to see how effectively this shuffler will work on the alphabet, add these lines to the above and watch it:

```
10 CLEAR :F=26:FOR X=1TO 8:IF X>1GOTO 70
200 FOR S=1TO F:GOSUB 300:PRINT S$(D);:NEXT S
210 PRINT " C1 =";C(1);" M1 =";M(1);" C2 =";C(2);" M2
    =";M(2)
220 NEXT X:END
500 DATA "A","B","C","D","E","F","G",
510 DATA "H","I","J","K","L","M","N",
520 DATA "O","P","Q","R","S","T","U",
530 DATA "V","W","X","Y","Z"
```

Jim Tennant

```
        MODIFICATION FOR BOOTUP PROGRAM
        CAUSES BASIC TO BE LOADED UPON COMPLETION
        OF BOOTSTRAP PROGRAM (CHAIN).


        ;CHUCK SADOIAN
        ;PO BOX 112
        ;DINUBA, CALIF
        ;93618

CHANGES TO BOOTUP.SYS:

1660   167             ;PATCH TO ADDED PROGRAMMING
1662   13474

4267   30460           ;CHANGE VERSION NUMBER
10070  30460           ;CHANGE VERSION NUMBER

ADDED PROGRAMMING:

15360  1002            ;BNE 15364 (COMPLETE DISPLACED INSTRUCTIONS)
15362  167             ;JMP 1664  (RETURN TO PROGRAM IF ERROR)
15364  164276
15366  12700           ;THIS ROUTINE TRANSFERS THE FILE NAME
15370  500             ;'SY BASIC SAV' TO THE CHAIN AREA,
15372  12720           ;LOCATIONS 500-507
15374  75250           ;NOTE: FILE NAME IN RAD50 FORM
15376  12720           ;75250='SY '
15400  6273            ;6273='BAS'
15402  12720
15404  34270           ;34270='IC '
15406  12720
15410  73376           ;73376='SAV'
15412  12700           ;SET UP EMT CODE FOR .CHAIN
15414  4000            ;EMT CODE FOR .CHAIN IS 4000, CHAN #0
15416  104374          ;.CHAIN TO BASIC
15420  END OF ADDED PROGRAMMING

THE BEST WAY TO MODIFY THE BOOT PROGRAM IS TO BRING THE FILE INTO
MEMORY WITH A 'GET' COMMAND, MODIFY THE PROGRAM USING ODT,
AND THEN SAVE THE PROGRAM USING THE FOLLOWING:

        SAV SY:BOOTUP.SYS 1000-15420

ALSO, IT IS NECESSARY TO CHANGE WORD #44 IN THE BASIC PROGRAM
YOU ARE CHAINING TO.  THIS IS EASILY ACCOMPLISHED BY USING THE
PATCH PROGRAM.  CHANGE #44 TO 400.  THIS WILL SET BIT #8
OF THE JOB STATUS WORD TO INDICATE TO THE MONITOR THAT THIS PROGRAM
HAS BEEN CHAINED TO.  THIS WILL ASSURE LOCATIONS 500-776 ARE
PROPERLY LOADED.  THIS IS NECESSARY BECAUSE BASIC USES
LOCATIONS BELOW THE NORMAL STARTING ADDRESS OF 1000(8).
```

ULTIMATE NAME INPUTTER

By James A. Tennant
Ketchikan, AK

Inputting names for CAI programs and games is easy if you 'line input' them seperately, but I have found that many newcomers, particularly children, resent this formality and lose their spontaneity. (They are demeaned by being systematically introduced to a machine). This inputting routine is lengthy, but it is worth using when you want to minimize inhibitions and instill confidence.

Here is what you get for your fifty lines:

1. no required entry format (allowing 'ands')
2. eight-name limit
3. individual re-entry prompts for overlength names
4. auto-dropout of three-time entry 'goof-offs'
5. alphabetized, discretely stored names
6. a recallable 'comma, comma, and' name string

If you want to see the results of the organizer, add the last six lines for a printout and try it.

```
                    ULTIMATE NAME INPUTTER
                                          BY JAMES A. TENNANT

5Ø CLEAR :DIM P$(8),N$(8)
6Ø PRINT :PRINT :PRINT "ARE YOU READY TO BEGIN";
7Ø LINE INPUT "?  ";Q$:IF LEFT$(Q$,1)<>"Y"THEN END
8Ø PRINT
9Ø PRINT "PLEASE TELL ME WHO YOU ARE.  (YOU MAY ENTER UP TO EIGHT NAMES.)"
1ØØ LINE INPUT "?  ";N$:IF ASC(LEFT$(N$,1))<65GOTO 1ØØ
11Ø PRINT :Z=Ø:L=Ø:W=Ø:FOR E=1TO LEN(N$)+1
12Ø IF ASC(MID$(N$,E,1))>9Ø OR ASC(MID$(N$,E,1))<65 GOTO 15Ø
13Ø L=L+1:IF L=1THEN K=E
14Ø GOTO 31Ø
15Ø IF L=ØGOTO 31Ø
16Ø IF MID$(N$,K,L)="AND"GOTO 3ØØ
17Ø Z=Z+1:IF Z>8THEN PRINT "TOO MANY NAMES!  ONCE AGAIN, ":GOTO 9Ø
18Ø P$(Z)=MID$(N$,K,L):IF LEN(P$(Z))<9GOTO 3ØØ
19Ø PRINT "SORRY, ";P$(Z);" BUT YOUR NAME IS TOO LONG.":A=Ø
2ØØ A=A+1:LINE INPUT "WHAT SHORTER NAME DO YOU USE?  ";A$
21Ø FOR D=1TO LEN(A$)
22Ø IF ASC(MID$(A$,D,1))>9ØOR ASC(MID$(A$,D,1))<65GOTO 25Ø
23Ø NEXT D
24Ø IF D<1ØGOTO 29Ø
25Ø IF A=3GOTO 27Ø
26Ø PRINT "USE ONLY LETTERS - NO MORE THAN EIGHT.  NOW,":GOTO 2ØØ
27Ø IF Z>1THEN Z=Z-1:PRINT :GOTO 3ØØ
28Ø PRINT "I HAVE NO TIME FOR FOOLISHNESS.":STOP
29Ø PRINT "OK, ";P$(Z);", I'LL CALL YOU ";A$;".":P$(Z)=A$:PRINT
3ØØ L=Ø
31Ø NEXT E
32Ø IF Z=1THEN N$(1)=P$(1):GOTO 47Ø
33Ø PRINT "DO WE AGREE THAT THERE ARE";Z;
34Ø LINE INPUT "OF YOU?  ";A$
35Ø IF LEFT$(A$,1)<>"Y"THEN PRINT "THEN, ";:GOTO 9Ø
36Ø FOR C=1TO Z
38Ø FOR A=1TO Z
39Ø N$=P$(A)
4ØØ FOR B=1TO Z
41Ø IF N$>P$(B)THEN NEXT A
42Ø NEXT B
43Ø N$(C)=N$:P$(A)="[":NEXT C
44Ø N$=N$(1):IF Z=2GOTO 46Ø
45Ø FOR G=2TO Z-1:N$=N$+", "+N$(G):NEXT G
46Ø N$=N$+" AND "+N$(Z):PRINT
47Ø P$=N$(1):O=Z
175Ø PRINT :PRINT :PRINT "ALRIGHT, ";P$;:IF O=1GOTO 179Ø
176Ø PRINT MID$(N$,LEN(P$)+1,LEN(N$)-LEN(N$(O))-4-LEN(P$));
177Ø IF LEN(N$)>4ØTHEN PRINT
178Ø PRINT TAB(5)RIGHT$(N$,LEN(N$(O))+4);
179Ø PRINT ", LET'S GET STARTED!"

18ØØ REM - FOLLOWING LINES ADDED FOR DEMONSTRATION, ONLY
181Ø PRINT :D=Ø:FOR A=1TO Z
182Ø PRINT TAB(B)"N$(";MID$(STR$(A),2,1);") = ";N$(A);
183Ø B=B+2Ø:IF B=8ØTHEN PRINT :B=Ø
184Ø NEXT A:IF INT(Z/4)<>Z/4THEN PRINT
185Ø PRINT :PRINT "N$ = ";N$:PAUSE 3ØØØ:GOTO 6Ø

                                          JAT
```

H17 TRACK SECTOR ACCESS

B. Watzman

From time to time there have been inquiries as to how to access a diskette at the track/sector level. However, there are no entry points in the H17 ROM (public or otherwise) that allow this unless the diskette is a valid HDOS diskette and the volume number is known in advance. One of the features designed into the H17 is that the address of every major routine in the ROM is stored in a jump table in RAM at 040130 and all inter-routine calls and jumps are via this jump table rather than directly. This makes possible selective modification of the H17 ROM, which, in turn, will permit track/sector access. The only problem is that any attempt to return to HDOS without a cold boot may result in damage to currently mounted diskettes! With that precaution in mind, let's look at a procedure for track/sector access of the H17 system.

The normal read routine (R.READ) is called via a jump at 040147 however, it is expecting a block number. . A volume number and certain other file parameters and will not run correctly where a file is not being accessed. To correct for this situation, patch the jump table so that the Seek Track Routine is called by the read routine instead of the decode track/sector (from block) routine. Specifically:

```
LXI     H,R.SDT
SHLD    R.DTS+1
```

Where R.SDT = 040166 and R.DTS = 040163. Then store the desired track (0-39) at R.TT and the desired sector (0-9) at R.TS. Next we have to keep the H17 ROM from getting upset because he doesn't know the volume number (part of the sector header) in advance. This can be accomplished by replacing the Locate Proper Sector routine with a similar routine which dosen't bother to check for the volume ID. Specifically:

```
LXI     H,OURLPS
SHLD    R.LPS+1
```

Where R.LPS = 040177

```
        TITLE   'TRACK SECTOR ACCESS ROUTINE'
R.ERRT  EQU     040232A         ; ERROR HANDLER
R.STZ   EQU     004213A         ; SEEK TRACK ZERO
R.SDT   EQU     040166A         ; SEEK DESIRED TRACK
R.TT    EQU     040240A         ; TRACK
R.TS    EQU     040241A         ; SECTOR
R.STS   EQU     040210A         ; SKIP THIS SECTOR
R.WSC   EQU     040221A         ; WAIT SYNC CHARACTER
R.RDB   EQU     040202A         ; READ DATA BYTE
R.LPSA  EQU     040116A         ; NUMBER OF RETRYS FOR R.LPS
R.DLYHS EQU     040244A         ; HEAD SETTLE TIME DELAY
*
* REPLACEMENT FOR R.LPS - LOCATE PROLPER SECTOR ROUTINE
*
* ESSENTIALLY THE SAME ROUTINE THAT IS IN HDOS BUT DOESN'T CHECK
* VOLUME ID AT ALL.
*
LPS0    CALL    R.STS           ; SKIP THIS SECTOR
OURLPS  LDA     R.LPSA          ; (A) - # OF TRYS FOR THIS SECTOR
        MOV     B,A
        LDA     R.DLYHS
        ANA     A
        JNZ     LPS0            ; WAIT FOR HEAD TO SETTLE
*
LPS1    DI                      ; DISABLE INTERUPTS
        CALL    R.WSC           ; WAIT SYNC CHARACTER
        JC      LPS3            ; NONE
        CALL    R.RDB           ; READ VOLUME ID
        LXI     H,R.TT
        CALL    R.RDB
        CMP     M               ; SEE IF PROPER TRACK
        JNZ     LPS5            ; WRONG TRACK
        INX     H
        CALL    R.RDB           ; READ DATA BYTE
        CMP     M
        JNZ     LPS2            ; WRONG SECTOR
*
* GOT RIGHT SECTOR. READ CHECKSUM
*
        MOV     H,D
        CALL    R.RDB
        CMP     H
        RZ                      ; ALL OK
        MVI     L,270Q          ; HEADER CHECKSUM ERROR CODE
LPS1.5  MVI     H,40Q           ; (HL) = ERROR BYTE ADDRESS
        CALL    R.ERRT          ;GET ERROR HANDLER
*
* WRONG SECTOR OR BAD DATA. TRY AGAIN
*
LPS2    CALL    R.STS           ; SKIP THIS SECTOR
        DCR     B
        JNZ     LPS1            ; TRY AGAIN
        STC                     ; ENOUGH TRYS
        RET                     ; GET OUT
*
LPS3    MVI     L,266Q          ; HEADER SYNC ERROR
        JMP     LPS1.5
*
* NOT ON RIGHT TRACK
*
LPS5    EI
        CALL    R.STZ           ; FIND TRACK ZERO
        CALL    R.SDT           ; NOW FIND DESIRED TRACK
        MVI     L,272Q          ; BAD TRACK NUMBER
        JMP     LPS1.5          ; RECORD ERROR AND RETRY OPERATION
        END     OURLPS
```

To select the desired drive; store binary 0 or 1 at AIO.UNI (041.061). Note that this must be updated before every read or write since the code does not 'remember' the last drive selected.

We can now do a read by calling R.READ at 040.147 with the number of consecutive sectors to be read in REG (B) . . . (C) must be 0! And the memory buffer address in (DE). A carry flag set on return indicates an error.

To perform a write; call R.WRITE at 040.155 with the same register usage. The dangers inherent in tinkering with the disk system cannot be underestimated, however this information is necessary at times.

EOF

```
        TITLE   'FIND FILE NAME - REGARDLESS OF WHICH DISKETTE IT'S ON'
        XTEXT   ASCII
        XTEXT   HOSDEF
EC.FNF  EQU     14Q
EC.VPM  EQU     41Q
        ORG     42200A
$MOVE   EQU     30252A
FIND    EQU     *
        XRA     A               ; MAKE SURE WE GET OVERLAY 0
        SCALL   .LOADO
        JC      ERROR
        MVI     A,1
        SCALL   .LOADO          ; NOW, GET THE OTHER OVERLAY
        JC      ERROR           ; NO USE
        MVI     A,-1
        SCALL   .CLEAR          ; CLOSE THE .LINK CHANNEL
        MVI     A,CTLC          ; PROCESS ^C
        LXI     H,CNTRL.C
        SCALL   .CTLC
        LXI     H,0
        DAD     SP
*
DIR0    MOV     A,M             ; MUST PLACE THE FILE NAME ON THE COMMAND LINE
        INX     H
        CPI     ' '             ; SEE IF SPACE
        JZ      DIR0            ; IT WAS
        CPI     TAB             ; IS IT A TAB
        JZ      DIR0            ; NO GOOD
        DCX     H
        PUSH    H
DIR1    MOV     A,M             ; GOT CHARACTER
        CPI     ':'             ; SEE IF HE SPECIFIED A DEVICE (MUST DO!)
        JZ      DIR2            ; HE DID GOOD.
*
        INX     H
        ANA     A
        JNZ     DIR1            ; NOT TO END OF LINE YET
        JMP     DIR3            ; OK
*
DIR2    POP     H
        PUSH    H

        LXI     B,3
        LXI     D,DIRC
        XCHG
        CALL    $MOVE
        LXI     H,DIRC          ; TRY MOUNTING A DISK ANYWAY.  MIGHT BE EMPTY
        SCALL   .MONMS          ; BUT DON' SAY ANYTHING ABOUT IT
        JNC     DIR4.5          ; OK
        CPI     EC.VPM          ; WE DO HAVE A DISK IN THE HOLE
        JNZ     ERROR
        JMP     DIR4.5          ; FIND  FILE
DIR3    LXI     H,DIRC
        SCALL   .DMNMS          ; GET ANOTHER DISKETTE
        JC      ERROR
        LXI     H,DIRF
        SCALL   .PRINT          ; ASK USER FOR ANOTHER DISK
DIR4    SCALL   .SCIN           ; WAIT FOR HIM TO RESPOND
        JC      DIR4
        SCALL   .CLRCO
        LXI     H,DIRC
        SCALL   .MONMS          ; GET ON DISK.. BUT DON'T SAY ANYTHING
        JC      ERROR
DIR4.5  EQU     *
        POP     H
        PUSH    H
        LXI     D,DIRD
        MVI     A,1
        SCALL   .OPENR
        JNC     DIR5
        CPI     EC.FNF
        JZ      DIR3            ; FILE NOT FOUND
*
ERROR   MVI     H,NL
        SCALL   .ERROR          ; LET HDOS LOOK UP ERROR MESSAGE
        JMP     EXIT
*
DIR5    POP     H
        MVI     A,1
        SCALL   .CLEAR
        LXI     H,DIRE
        SCALL   .PRINT          ; FOUND IT! TELL ABOUT IT
EXIT    XRA     A
        SCALL   .EXIT           ; JOB DONE.
CNTRL.C LXI     H,CNTA
        SCALL   .PRINT
        LXI     H,DIRC
        SCALL   .MONMS
        JMP     EXIT
*
CNTA    DB      '^','C'+200Q
DIRC    DB      'SY0:',0
DIRD    DB      'SY0',0,0,0
DIRE    DB      NL,BELL,'FILE FOUND',ENL
DIRF    DB      NL,BELL,'CAN''T FIND IT HERE.. GIMME ANOTHER DISKETTE',':'+200Q
        END     FIND
```

This program will work only with VER 1.5 of HDOS —

New SYScalls that should be added to your HDOS.ACM file are:

.LOAD0 = 11Q
.MOUNT = 200Q
.DMOUNT = 201Q
.NOMNS = 202Q
.DMNMS = 203Q
.OVL0 = 0
.OVL1 = 1

DO NOT interchange VER 1 and VER 1.5 system files, PIP and SYSCMD.SYS!

:JB:

```
.TITLE  FORMAT - INIT H27 DISKETTES TO IBM 3740 FORMAT

;       FORMAT uses the formatting capability of the H27 to initialize
;       a diskette in DRIVE # 1 to the IBM 3740 format

        .SBTTL  Program parameters
        .ENABLE LC
        .enabl  ama
        .nlist  TTM,BEX
        .MCALL  .REGDEF,.PRINT,.TTYIN
        .MCALL  .EXIT,..V2..
        .REGDEF
        ..v2..

;ASCII Characters
LF      =       12
CR      =       15
; H27 Commands

FC.HOS  =       11                      ; HOS Escape
FC.DV1  =       20                      ; Drive 1 select
HF.FMT  =       140                     ; FORMAT

; H 27 Status Bits

FS.DON  =       40                      ; Done Bit
FS.TR   =       200                     ; Data Transfer Bit

; H 27 Registers

RXCS    =       177170                  ; Command and Status
RXDB    =       177172                  ; Data (in and out)

        .Page
.Sbttl  Main

; Program begins execution at PC = 1000 when assembled
; using HT 11. Thus, the label 'FORMAT' is at location
; 1000 Octal

FORMAT: .PRINT  #FMTA                   ; Print instruction Message
        JSR     PC,INPUT                ; Get Response
        BITB    #CR,R1                  ; Was it a Carriage Return ?
        BEQ     FMT4                    ; No.
        MOV     #FC.HOS+FC.DV1,@#RXCS   ; Drive '1' HOS Command
FMT1:   BITB    #FS.TR,@#RXCS           ; Wait for TR
        BEQ     FMT1                    ; If no TR yet
        MOVB    #HF.FMT,@#RXDB          ; Format Command
 FMT2:  BITB    #FS.DON,@#RXCS          ; Wait for Done Bit
        BEQ     FMT2                    ; If no DONE bit
        .PRINT  #FMTB
        JSR     PC,INPUT                ; Get response
        CMPB    #'Y,R1                  ; Was it Yes?
        BNE     FMT3                    ; If no
        BR      FORMAT                  ; Else, do again
FMT3:   .EXIT                           ; Return to monitor
FMT4:   .PRINT  #FMTC                   ; Print INPUT error message
        BR      FORMAT                  ; Try again

FMTA:   .ASCII  <CR><LF>/This routine will only format a diskette inserted/
        .ASCII  <CR><LF>/in Drive 1/
        .ASCII  <CR><LF><LF>
        .ASCII  <CR><LF>
        .ASCII  /       1) Insert disk with write enable tab into/
        .ASCII  <CR><LF>/          drive '1' and close the door./<CR><LF>
        .ASCII  <LF>/   2) Set the H27 'FORMAT' switch to DRIVE 1/<CR><LF>
        .ASCII  <CR><LF>/       3) Hit 'RETurn'/
        .BYTE   200
FMTB:   .ASCII  <CR><LF>/       DO YOU WISH TO FORMAT ANOTHER DISKETTE?- - > /
        .BYTE   200
FMTC:   .ASCIZ  <CR><LF>/       INVALID ENTRY... Try again!/
        .EVEN
        .PAGE
        .SBTTL  Input Character
;
;
INPUT:  .TTYIN  R1                      ; Save first character in R1
1$:     .TTYIN                          ; Gobble up rest of characters: .TTYIN
        CMPB    #LF,R0                  ; Did last character = LF
        BNE     1$                      ; No get new
        RTS     PC
        .END    FORMAT
```

NOTE: When I finished assembly of this program, naturally, Iwanted to try it. Right? It cost me about half of the manuscript for this magazine plus the program itself. Enough said? :JB:

SOFTWARE CONTROL FOR 5 LEVEL PRINTERS

By: *Adam L. Keller, W9EF*
354 Southwood Drive
Michigan City, IN 46360

Using Heath Extended BASIC 'USR' function with the software/hardware package described by Howard L. Nurse in June, 1978 issue of KILOBAUD magazine.

About a year ago, I was intrigued by the article in the June, 1978 issue of KILOBAUD magazine by Howard Nurse, describing his use of a 5 level baudot TTY machine with the H8 Heath Computer via an H8-2 PIO board and external UART. In fact, I was intrigued enough to purchase a model 28 ro printer and H8-2 PIO board and try the set-up for myself. I found that the printer would produce excellent results, but that the use of the printer all the time slowed down the whole system to 110 baud. When writing programs, this speed was just not fast enough, and when executing programs the printer was not needed all of the time. The following short article will describe the use of the basis 'USR' function to control the printer. The function can be used in either 'program' or 'command' mode, so can be used at will during the writing or execution of programs.

The following software changes will permit the 'USR' function to turn on the printer (if it was off), or to turn off the printer (if it was on). Each time the 'USR' function is called, the printer will reverse states. It can be called as any valid 'USR' function, such as 'LET A (E) USR (A)', or 'PRINT USR (A)', or any other 'USR' use.

The following software changes need to be made, in conjunction with Mr. Nurse's article:

1. Make the changes as described in Mr. Nurse's Table #3.

2. In the Heath 'BASIC' manual, under "Entry Points to Utility Routines" (Appendix D) find the address for the routine called 'USRFCN'. In version 10.02.01, this address is 106.106. In version 10.01.01, it is 103.163. Other versions will have different addresses, determine from your software manual what yours is.

   In Mr. Nurse's Table #4, add the following memory changes to his 5 changes;

   At 'USRFCN' (106.106 for 10.02.01) enter 320 at 'USRFCN' (+) 1 (106.107) enter 136

   These two bytes make up the beginning address of the software routine described in Figure 1. If you put the Figure 1 routine as a diffenent location, enter that address at 'USRFCN' and 'USRFCN' (+) 1, lo byte first. The Figure 1 routine should be entered in memory just above Mr. Nurse's driver program, wherever you decide to put that.

3. Enter the Figure 1 program in memory just above Mr. Nurse's driver routine. Using 16K of RAM, I used the memory space starting at 136.320, where his driver ends at 136.314.

4. Make a dump of the memory from the beginning of Mr. Nurse's ASCII/Baudot driver, to the end of the program entered in (3.) above. I dumped this memory block on tape immediately following my configured BASIC, so that I do two consecutive 'loads' and have the whole system running with a minimum of effort.

Now, when the 'GO' function is given after loading BASIC and the whole TTY routine, the printer will be running. Any valid 'USR' function will cause the printer to stop running and the system will run at full terminal speed. Another 'USR' will activate the printer again, etc.

Your printer has now become a valuable tool, not just a source of irritating noise!!!!

Have fun!!! (ES GL)

(P.S. I found that my particular printer generated a considerable amount of electrical noise, and had to add a .0022 UFD capacitor from the H8-2 PIO connector, pin 10, to ground to eliminate flase 'data taken' signals. Try that change if you are having trouble with your 28 printing 'gibberish' no matter what setting you have on the new 'baud rate' potentiometer.)

A.K.

```
                                (X)
136.320                 BEGIN   ORG     136320A  START ROUTINE
106.106                 USR     EQU     106106A  'USRFCN', 10.02.01
040.111                 CRT     EQU     040111A  CRT ONLY
136.027                 PRNTER  EQU     136027A  CRT & PRINTER BOTH
040.365                 DRIVER  EQU     040365A  DRIVER JMP ADDRESS
                                (X)
                                (X)
136.320  345           START   PUSH    H        SAVE H,L
136.321  041 111 040           LXI     H,CRT    TO RUN CRT ONLY!
136.324  076 331               MVI     A,331Q   'USRFCN' NXT 'USR'
136.326  303 337 136           JMP     DOIT     AND DO IT!
136.331  345           #2PASS  PUSH    H        SAVE H,L
136.332  041 027 136           LXI     H,PRNTER RUN PRINTER & CRT
136.335  076 320               MVI     A,320Q   'USRFCN' NXT 'USR'
136.337  062 106 106   DOIT    STA     USR      CHANGE JUMP ADRESS
136.342  042 365 040           SHLD    DRIVER   CHANGE DRIVER JMP
136.345  341                   POP     H        RESTORE H,L
136.346  311                   RET              RETURN TO BASIC
136.347                 END     START
```

BITS AND NIBBLES

SOFTWARE:

Even though you have a new software catalog, there are more 'goodies' already available . . . NOW.

SMALL BUSINESS PACKAGE — prepared by Nick Niamo of St. Louis, here is a package that has been a long time coming for the H8. Nick worked almost a year on developing this system and it has been in use in his business as well as some others for some time. Therefore, it should be pretty reliable. The system:

1. Prints statements.
2. Prints mailing labels for all accounts or a specific zip code.
3. Provides account aging reports.
4. Provides Balance reporting for all accounts or any specific account.
5. Prepares Invoices and Credit memos.
6. Outputs a Journal, Ledger and Profit/Loss statement.
7. Outputs monthly sales totals and compares with any month from past year/s.

This system requires 24K memory and a dual drive floppy. Order HUG P/N 885-1041 $50. Includes two diskettes and written documentation.

MORSE 8 — Send and receive Morse Code on your H8. Morse 8 is intended to facilitate communication by Morse Code over a wide range of code speeds, dot/dash ratios, signal strengths, and noise conditions. Many commonly used abbreviations (CQ, DE, call sign etc.) are generated by a single key stroke. Package includes schematic of hardware interface and user manual. Available in source code and object code on cassette only. HUG P/N 885-1027 — $11.

RTTY COMMUNICATIONS PROCESSOR — As described in Issue #6 of REMark, this complete package is now available on cassette. HUG P/N 885-1028 $11.

VECTORED FROM PAGE 16

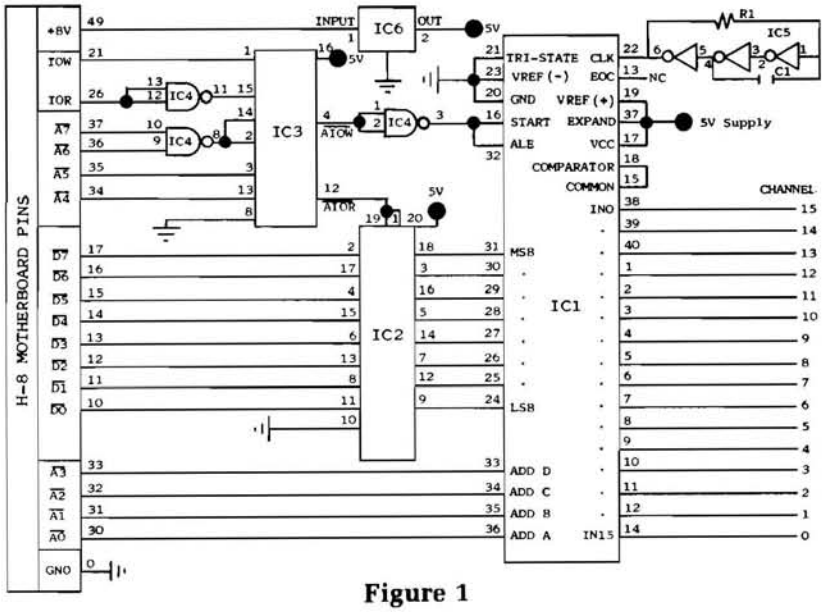


Figure 1

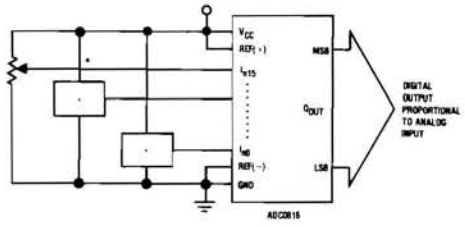


Figure 2

ADDR	PIN NO	
	$\overline{AIOR}$	$\overline{AIOW}$
Ø-15	12	4
16-31	11	5
32-47	1Ø	6
48-63	9	7
74LS155		

Changing your address? Be sure and let us know since the software catalog and REMark are mailed bulk rate and it is not forwarded or returned.

CUT ALONG THIS LINE

HUG MEMBERSHIP RENEWAL FORM

You can determine your expiration date by examining the last six digits of your ID number — example: 780202 indicates your membership began 02/02/78 and expires one year from then.

IS THE INFORMATION ON THE REVERSE SIDE CORRECT? IF NOT FILL IN BELOW

Name _____

Address _____

City-State _____

Zip _____

REMEMBER — ENCLOSE CHECK OR MONEY ORDER

CHECK THE APPROPIATE BOX AND RETURN TO HUG

NEW MEMBERSHIP? FEE IS:

RENEWAL RATES			NEW
US DOMESTIC	$11 ☐		$14 ☐
CANADA	$13 ☐	US FUNDS	$16 ☐
INTERNAT'L*	$18 ☐	US FUNDS	$24 ☐

* Membership in England, France, Germany, Belgium, Holland, Sweden and Switzerland is aquired through the local distributor at the prevailing rate.

THE
BACK PAGE —

NEW INTERFACE CARDS FOR THE H8?

We hope to be able to offer new interfacing ideas for the H8 before the end of the year. Here is how that might happen. We know that several, perhaps many H8 users have developed their own unique interfacing cards that do any number of tasks, such as playing music, monitoring temperatures in large buildings and many other things. If you are such a person, we have a plan to modestly compensate you for your circuit and then distribute a 'mini-manual' to the rest of the HUG members so they may do something similar. If that sounds interesting, write HUG for details.

CONTEST #6 WINNERS

Robert Mathias of the Detroit area won Contest #6 with his version of H8 RUNOFF, a text formatter.

John Boesel of Evanston Il submitted several programs for HT11 Software. Congratulations to John and Bob.

H8 MODEM COMMUNICATION LINK

We have just acquired a super program that allows H8/H17/H8-4 users to exchange gossip and disk files over the telephone with full CRC etc. — P/N 885-1043 — $21.00.

HUG PRODUCT PRICE LIST

PART NUMBER	DESCRIPTION		PRICE
885-1100	'HUG' Tee Shirt	Small	$ 4.50
885-1101		Medium	$ 4.50
885-1102		Large	$ 4.50
	Keep your volumes neat with a:		
266-945	Cassette Holder (holds 2 cassettes)		$ 2.45
885-4	and a 'HUG' binder		$ 5.75
885-1008	Volume I Documentation		$ 9.00
885-1009	Tape I — H8 Cassette Tape		$ 7.00
885-1010	Adventure — H8 Disk		$10.00
885-1012	Tape II — H8 (BASIC) Cassette		$ 9.00
885-1013	Volume II Documentation		$12.00
885-1014	Tape II — H8 (Assembly) Cassette		$ 9.00
885-1015	Volume III Documentation		$12.00
885-1018	HDOS Programming Guide		$ 5.00
885-1019	HDOS Device Driver — H8 Disk		$10.00
885-1022	HDOS Editor — H8 Disk		$15.00
885-1023	RTTY Communication Processor — H8 Disk & Documentation		$22.00
885-1024	Disk I — H8 Software		$18.00
885-1025	Runoff Word Processor — H8 Disk		$35.00
885-1026	Tape III — H8 Cassette Tape Financial & Amateur Packages		$ 9.00
885-1027	Morse 8 — H8 Cassette & Documentation		$14.00
885-1028	RTTY Communications Processor — H8 Cassette & Documentation		$11.00
885-1029	Disk II — H8 Software 'Games 1'		$18.00
885-1030	Disk III — H8 Software 'Games 2'		$18.00
885-1031	Disk IV — H8 'Music' 2 Disks		$23.00
885-1032	Disk V — H8 Misc. Software		$18.00
885-1033	Disk I — HT11 Misc. Software		$19.00
885-1034	Character Editor — H8 Cassette Tape & Documentation		$11.00
885-1035	Co-resident Editor/Assembler — H8 Cassette Tape & Documentation		$11.00
885-1036	Tape IV — H8 Misc. Software*		$ 9.00
885-1037	Volume IV Documentation*		$12.00
885-1038	WISE — H8 Disk Software		$18.00
885-1039	WISE — H8 Cassette Tape		$ 9.00
885-1040	PILOT — H8 Cassette Tape & Documentation		$11.00
885-1041	Small Business Package (2 disks & DOC.)		$50.00

*Available September

Heath Users' Group
Hilltop Road
St. Joseph MI 49085

BULK RATE
U.S. Postage
PAID
Heath Users' Group

POSTMASTER: If undeliverable, please do not return.

885-2007