J. SCHWARTZ (1820)

PPC East Coast Conference

Rockville, Maryland
March 28, 1981

Sponsored by:

PPC Washington Chapter
Bill Kolb, Chapter Co-ordinator

A PPC Conference is a periodic gathering of PPC members, manufacturers' engineering personal, members of the user community and those interested in personal computing. Computing is intended to mean numerical computation (though not limited to) rather than data processing.

Reprints are available from the Washington Chapter for a limited time.
Contact: Bill Kolb (265)

HUMAN FACTORS IN PROGRAMMING

or

PROGRAMMING FOR HUMANS ON THE HP-41C

In the past, with the HP-65 and HP-67, better programming has been thought of as a way to make programs shorter and faster. With the 41C, Hewlett Packard has given us something more, the alpha-numeric display and, with a good supply of modules, additional program memory, so that we can begin to add touches to our programs to accomodate human factors.

What are human factors? By human factors I mean those features in a program that assist the user in using the program. In most cases, these are not related to the algorithims used in the calculations (although in some cases they could be), but are related more to the interactions between the calulator and user in, for example, the methods used for entry of information, selection of program functions, and presentation of results.

Unlike our earlier approach to better programming, the human factors approach that I am addressing is not another way of saving bytes of program memory or reducing execution time. On the contrary, the addition of human factors to programs will increase the byte count and may slow execution time. I believe, however, that the human factors approach should be thought of, not as replacing our former concept of better programming, but as extending it. The emphasis on efficiency is still important, especially in finding ways to compensate for the program space and time devoted to human factors and to intergrate human factors into the programs more efficiently.

Why would anyone want to expend valuable program memory of which, despite memory modules, there is still not enough, on "human factors?" We would do it to make the program easier to use and reduce the possibility of error, to make the program more self-documenting in that the user can look more at the calculator and less at the instruction book for guidance in using the program, and to make the calculator more fun to use. Although these things may not seem necessary to someone writing a program and using it immediately, they become more important if the program is to be shared with others, or if writer is expects to pick it up and use it again six months later.

To give one example, let us consider the Vaunted Ratio Routine that has appeared on the outside of the brown envelopes that Richard uses to send materials to PPC members. For those of you unfamiliar with this routine, it uses a single algorithm to solve for the unknown variable in the equation A/B=C/D when the values for the three known variables are entered in their appropriate positions and a zero is entered in the place of the unknown. From time to time, the PPC Journal has published various ingenious ways to reduce the number of program steps required for that routine. But, at the same time, user operation has become a more critical factor in successfully using the program. For example, the user may be required to enter the appropriate values in the stack in a precise order; to prepeat the calculation with a change in one known variable, all four values may have to be entered again; etc. As I view human factors programming, it would allow the user to enter the known variables in any order, would not require him to make any entry for the unknown variable, and would enable him to make repeat calculations after keying in only those values that are different from those used in the previous calculation.

I have included, at the end of this paper, a humber of relatively short programs (inluding the Vaunted Ratio Routine) to illustrate some of the human factors considerations that I discuss. I make no claims that the algorithms used for the calculations are most efficient, and I have no doubt that they could be improved by many PPC members. I hope, too, that other PPC members will find and contribute more efficient ways to implement the human factors features.

The Starting Point: The Alpha Display

The thing that, more than any other single factor, opens to door to adding human factors in programming, is the alpha register. With it, we are able to instruct our 41C's to speak to us, to ask us for information, to tell us what to do, to tell us what it's doing, and to explain its answers.
do.

To start with the most obvious, we can, and I am sure that all of us havealready done so, use the alpha display to identify output, that is, the resuts of our calcuations. This, of course is done very simply by inserting a program line with an alpha name for the result immediately following the calcualtion, recalling

the contents of the X register into the alpha display, and then using an alpha view (AVIEW) or prompt. thus a program to calculate temperature might end with the following steps:

```
"TEMP="
ARCL X
AVIEW
END.
```

We can also label input. When for instance, you enter values for a compuation to be done, if you store the values in the desisnated registers, or even if the program does it, all you see in the display is the value that has been stored. Why not have the calculator show you not only the value, but also the register or the label under which it has been stored. Thus, when entering 91 as the number of days needed for a calculation, instead of seeing simply "91" in the display, we could generatre a display that reads "DAYS: 91". Most of the programs that I have included as illustrations have this feature.

In some cases we might want to show both the input and the output in the same display. For example, in a program that converts Farenheit temerature to Celsius, the display might include both the starting and ending ponts, with labels on each: "98.6F=37.0C".

I have found it helpful in programs involving calulations (as opposed to housekeeping routines such as size finders) to make sure that when there is an alpha display of some numeric value, the value displayed is in fact the value in the X register. Although this does not have to be done, it saves wear and tear on the user, who can prodceed to perform calculations on the number he sees without having to first clear the display to find out what is in the X register.

I have also found it helpful to include in any alpha display, some indication to the user that it is an alpha display so that he knows he is not looking directly as the X register.

In some cases, it is useful to have an alpha display while the program is running, either to tell what the calculator is doing to or to explain what the results will be, especially if the result will be so long is so long that it will not fit comfortably in

the display without scrolling.

Try to avoid scrolling displays if possible. Fortunately, the longer displays can be accomodated when necessary, but displays that fit in the window after coming to rest are most easily used. This will, of course, require some attention to the selection of suitable abbreviations of labels.

I have found that it may be better, when a lengthy calculation is underway, to let the goose fly instead of retaining an alpha display. This should keep the user from thinking that that the program has stopped because the display has not moved lately. Of course, displays which show what the calculator is doing, such as "IM THINKING", would not have this problem.

I have also found it very useful to use headings in my programs to let me know what the program is. Usually, I will put a heading in a program right after its alpha label and insert a prompt right after the heading. When the program is called, the title heading appears in the display. Sometimes the heading is the name of the program, but sometimes not. It may, for example, be a display of what the program does, such as A/B=C/D for the ratio routine, or N I PV PMTFV for the fincancial functions program.

When there is such a heading in a program and its global label is assigned to a key, it may be called with a single keystroke and it will show its heading in the display but will not execute until you instruct it to. I find this a useful feature when I am working with several programs in the calculator (which is almost all the time).

If a program has this kind of heading, it can be recorded on a card with flag 11 set so that, when the program is read into the caculator from the card, the heading of the program will pop into the display, reassuring you that the correct program has been loaded and that it has been loaded properly.

These are just some of the things that you can do with aplha displays. The use of aplpha has become so routine for me that when I pick up my 33E, I find it disconcerting at first when a calculation result appears in the display but is not labeled.

Use of Local Labels

One thing we should not forget in programming is the use of the local labels provided by HP. These can make the execution of the program quite easy. We have fifteen local labels and that is quite good. Whenever we have a small number of data items to store, we can use the local labels to control input, and design the program to use random input. Thus, in making subsequesnt calculations, we need replace only those items which are changing, and we can run the program again. We can avoid the annoyance of having to go through a whole series of prompted inputs just to re-enter data that has not changed.

When using the local labels, we should remember that the register numbers are coordinated to keys A through J. Thus, we can use RCL A or STO B and we are really instructing the calculator to RCL 01 and STO 02, etc. The moral of this story is that, if we have five items to store and if we are planning to use local labels A through E to do it, by using registers 01 through 05 (rather than 00 through 04) we gain the advantage of having the STO and RCL functions correspond with the local labels used.

There are, however, some hardware drawbacks in using local labels.

Usually, not all of the local lablel keys will be easily available. For example, keys 21 (X<>Y) and 22 (RDN) I like to use when even when I am running programs. Anyone doing a trig program might want to have the trig functions always available. Other functions of the local label keys may be required at different times and preclude their assignment in a program.

There is a lack of space to indicate the functions assigned to local label keys. There is alost no room on the overlays, and even the use of the magnetic card in thecard holder on the card reader has drawbacks. First, it is not close to the keys, and second, there is not much room on card for more than one key assignment. For example, how would you show on a card assignments for local labels E, e, and J?

Also, HP has not given the program writer the ability to include an initialization routine in his program that will clear previous key assignments from key which will be used as local labels. (Do I hear a

voice in the background whispering "synthetic programming?")

Those Wonderful Flags! (22 and 23)

Among the very useful programming tools HP has given us, are Flags 22 and 23. With them we are able to use our local label keys as double function keys, depending on whether an alpha or numeric entry has been made. We can, for instance use them to either store/recall or store/compute.

The store/compute function is rather straightforward and is used in the standard 41C aplications pack by HP in its financial program. The store/reall function is somewhat more involved but works just fine. The are several ways of doing it. Two are illutrated in the attached programs to calculate Treasury Bill Yields and Depreciation.

The standard I use for deciding whether to make the key a STO/RCL or STO/COMP is to examine whether the input and output items are the same. If they are, as in the ratio routine or the finance routine, I use STO/COMP is appropriate. In a case where they are different, as in the T-Bill or depreciation programs, I use STO/COMP.

Printer Compatability

One of the luxuries of life is the possession of a printer for your 41C. I have one now, but acquired it only recently, after I had developed some extensive programs. I couldn't wait, however, to run them with the printer to see the results printed out in all their glory. I found, much to my dismay, that there was no glory. When the printer was in manual mode, many of the displays I wanted printed were not printed. when it was in normal mode, it printed the displays I wanted, but also a lot of things that I didn't want. Well, as they say in the experimental airplane business, back to the drawing board.

My human factors standard: to develop programs that can be keyed into a program by someone not having a printer, that will produce the desired printout with the printer in manual mode, and provide proper execution when no printer is available or when the printer is attached but turned off. I have managed to do this, but the printer is still new to me, and I am hoping to

do it more efficiently as I learn more about the printer. Therefore, I will leave the discussion of this subject for another time.

Of course, I have not even gotten started on special printer programs for plotting, special characters, etc., which seems to me to a complete field by itself.

A couple of observations, anyway. In spite of the attempted explanation in the Printer Owner's Manual, which I found thoroughly confusing, I found Flag 21 to be somewhat helpful. Also, I found myself handicapped by the lack of a flag to test whether the printer is on.

Winding Up

There is, of course, lots, lots more to the subject of human factors programming. I haven't even started the subject of programs that use serial data entry through indirect addressing, different subroutines for input and output items, determination of when to store input, when to store intermediate results, when to rely solely on the stack, suppressing prompts, etc. etc. But I can't go on here forever. Perhaps at the next conference, or in a Journal article...

But there is nothing to stop anyone writing a program from determining and incorporating human factors even if they haven't been discussed. The principal thing required is to envision how the program will be used. Will it be used only by you or shared with others? If only you, will you ever want to use it again? If others, will they be experts in the field for which program is written? Will they be experts in the use of the 41C? MIght they be persons who are not 41C owners? Will it be a chore to run the program many times? Should input data be stored for future program runs? Would be better to provide serial entry of data, or random entry, or both? You get the idea.

Is it all worth the trouble?

In the end, of course, the answer to that question is up to each of you as a writer and user of your own softwear. For me, the answer has been a definite yes. It makes my programs easier for me to use and more fun to write and use. It lets me spread the enjoyment because others persons, including some who don't own a

41C, run the programs with confidence. I am pleased to report that one person, the owner of a TI-59, decided to buy a 41C after working with some of my programs. (Who knows? Perhaps human factors even have something to do with commerical factors.)

* * *

The Vaunted Ratio Routine
(Arranged for the 41C)

```
01 ¶LBL "RTO"        19 ¶LBL 01          37  /
02  "  A/B=C/D"      20  X<>Y            38  RCL 02
03  GTO 03           21  STO IND Y        39  RCL 03
04 ¶LBL A            22  FC?C 22          40  *
05  "A"              23  GTO 06          41  RCL 01
06  1                24  "†: "           42  RCL 04
07  GTO 01           25  GTO 02          43  +
08 ¶LBL B            26 ¶LBL 06          44  /
09  "B"              27  CLX             45  +
10  2                28  STO IND Y       46  STO IND 00
11  GTO 01           29  X<>Y            47  "†="
12 ¶LBL C            30  STO 00          48 ¶LBL 02
13  "C"              31  RCL 01          49  ARCL X
14  3                32  RCL 04          50 ¶LBL 03
15  GTO 01           33  *               51  AVIEW
16 ¶LBL D            34  RCL 02          52  END
17  "D"              35  RCL 03           -- AAV -
18  4                36  +
```

Note: The symbol ¶ is used to identify labels.
The symbol † is used to represent "APPEND".

To use the program:

The calculator must be in user mode.

1. Key in the value for each known value for three variables of the equation: A/B=C/D, and after keying in each value, push the key with the leter corresponding to the respective variable.

2. Without making pressing a numeric key, press the lettered key corresponding to the variable whose value is to be determined.

3. To perform successive calculations, key in the value for any know variable that is different from that used in the original calculation, and perform step 2.

Mantissa View Routine

This is my entry in the Mantissa Sweepstakes. I like this routine because: (1) it leaves the stack and LASTX intact, (2) it tells me what's going on while it's doing the job, (3) the mantissa display includes an "M" which identifies it and lets me know that I looking at an alpha display, and (4) it works.

```
01 ¶LBL "M"          11  SCI 9          21  ASHF
02  "  MANTISSA"     12  ARCL 00        22  AVIEW
03  AVIEW            13  STO d          23  CLX
04  CLA              14  "†    "        24  RCL 00
05  X<0?             15  ASTO X         25  END
06  X=0?             16  ASHF           -- AAA --
07  " "              17  ARCL X
08  STO 00           18  ASTO X
09  CLX              19  ASHF
10  RCL d            20  ARCL X
```

Note: The symbol ¶ is used to identify labels.
The symbol † is used to represent "APPEND".
Line 14 is APPEND and four spaces.

To use the program:

1. Execute "M".

* * *

New postal rates

```
01 ¶LBL "POST"       11 ¶LBL 00         21  X>0?
02  "  POSTAGE"      12  X<>Y           22  +
03 ¶LBL 01           13  CLA            23  FIX 2
04  PROMPT           14  FIX 0          24  ARCL X
05 ¶LBL A            15  ARCL X         25  GTO 01
06  INT              16  "† OZ=$"       26  END
07  LAST X           17  .17            -- AAV --
08  FRC              18  *
09  X>0?             19  .01
10  ISG Y            20  X<>Y
```

To use the program:

1. Key in the weight, in ounces, of first class mail to be sent.

2. Press R/S or A.

T-Bill Yield Routine

```
01 ¶LBL "YLD"        24  STO IND Z        47  "CASH="
02  "  T-BILLS"      25  "†: "            48  GTO 10
03  CF 22            26  ARCL X           49 ¶LBL 06
04  GTO 12           27 ¶LBL 12           50  1 E 2
05 ¶LBL A            28  CF 02           51  X<>Y
06  "DAYS"           29  GTO 11           52  STO Z
07  1                30 ¶LBL D            53  -
08  FIX 0            31  "PRICE, YIELD=" 54  CLX
09  GTO 08           32  AVIEW            55  ARCL X
10 ¶LBL B            33  SF 02            56  /
11  "FACE"           34 ¶LBL E            57  365 E2
12  2                35  RCL 02           58  *
13  FIX 0            36  RCL 03           59  RCL 01
14  GTO 08           37  RCL 01           60  /
15 ¶LBL C            38  +                61  "†, "
16  "DISCNT"         39  360              62  FIX 2
17  3                40  /                63 ¶LBL 10
18  FIX 3            41  FIX 3            64  ARCL X
19 ¶LBL 08           42  RND              65 ¶LBL 11
20  X<>Y             43  FS?C 02          66  AVIEW
21  RCL IND Y        44  GTO 06           67  END
22  FS?C 22          45  %                -- AAV --
23  X<>Y             46  FIX 2
```

Note: The symbol ¶ is used to identify labels.
The symbol † is used to represent "APPEND".

To Use the Program:

1. Key in the number of days to maturity, face value of the bill, and the discount quote and after each value is keyed in , press keys A, B, and C, respectively.

2. To determine the price and cupon equivalent yield, press key D. 3. To determine the amount of cash returned by the Treasury after a tender of the full face amount, Press key E.

4. Re-enter, with keys, A, B, and C, as appropriate, any new values with which the program is to be run. Do not re-enter values which are not changed. Continue with step 2 or 3.

Keys A. B, and C have a STO/RCL funtion depending on whether they are pressed following numeric entries.

Sum-of-the-Years-Digits Depreciation

```
01  ¶LBL "DEP"
02  "DEPRECIATION"
03  CF 22
04  FIX 0
05  GTO 11
06 ¶LBL A
07  "LIFE"
08  1
09  GTO 09
10 ¶LBL B
11  "DEP YR"
12  2
13  GTO 09
14 ¶LBL C
15  "BASIS"
16  3
17  GTO 09
18 ¶LBL D
19  RCL 01
20  ENTER
21  X↑2
22  +
23  2
24  /
25  RCL 01
26  1
27  +
28  RCL 02
29  -
30  X<>Y
31  /
32  RCL 03
33  *
34  RND
35  "DEP YR"
36  ARCL 02
37  "†="
38  GTO 10
39 ¶LBL 09
40  X<>Y
41  FC?C 22
42  X<> IND Y
43  STO IND Y
44  "†: "
45 ¶LBL 10
46  ARCL X
47 ¶LBL 11
48  AVIEW
49  END
-- AAV --
```

Note: The symbol ¶ is used to identify labels.
The symbol † is used to represent "APPEND".

To usethe program:

1. Key in the following values, and pressing the letter key indicated after each numeric entry:

   A useful life of the asset to be depreciated
   B the year (1, 2, 3, etc.) for which depreciation is to computed
   C the basis (dollar amount) of the asset

2. To determine the amount of depreciation for the year indicated, press Key D.

3. Re-enter any different values for which you wish to make the computation. Values which do not change do not have to be re-entered. Continue with Step 2.

Keys A, B, and C have a STO/RCL function depending on whether they are pressed following numeric entries.

PPC

The Unit Management System

I. Overview

A. Example of the Need for Unit Management -- Gas Viscosity
   1. Tc in R or K from Tables
   2. Pc in PSI, ATM, or KPA from Tables
   3. T in F or C from Lab/Field Measurements
   4. P in PSI, ATM, or KPA from Lab/Field Measurements
   5. Gas Viscosity in CP, LBM/FT*S, PA*S, KG/M*S

B. How Units Have Been Managed by Earlier Software
   1. Burden of Dimensional Consistency Placed on User
   2. Mandatory Set of Units
   3. Optional (Metric) Set of Units
   4. Any Dimensionally Consistent Set of Units

C. How Units Are Managed in HP-41 Petroleum Fluids Pac
   1. For Inputting Variables
   2. For Outputting Results

D. Further Details
   1. Optional Unit Systems
      a. English
      b. SI
      c. Other
   2. No Halt on Output
   3. Error Checking
      a. On Input
      b. On Output

E. Significance of the "Unit Management System" Name

II. How Unit Management Works -- the I/O Subroutines

A. General Description
   1. Prompt the User
   2. Manage the Units
   3. Store the Variables
   4. Format Printed Output

B. How to Call the I/O Subroutines
   1. IN (Input)
      a. Pointer to Storage Location
      b. Name of Input Variable
   2. INU (Input With Units)
      a. Pointer to Storage Location
      b. Name of Input Variable
      c. Units of Input Variable
   3. INK (Input With Units Known)
      a. Pointer to Storage Location
      b. Name of Input Variable
      c. Units of Input Variable
      d. Flags
   4. OUT (Output)
      a. Output Value
      b. Name of Output Value
   5. OUTU (Output With Units)
      a. Output Value
      b. Name of Output Value
      c. Units of Output Value
   6. OUTK (Output With Units Known)
      a. Output Value
      b. Name of Output Value
      c. Units of Output Value
      d. Flags

C. What Operations Are Performed by the I/O Subroutines
   1. Initialize Registers and Flags
   2. Build Prompt
   3. Set Up Halt Conditions and Halt
   4. Check for Numeric or ALPHA Input
   5. Store Result
      a. Convert to Proper Units
      b. Check for Conversion Errors
   6. Print and/or Display Value
   7. Set Up Return Conditions and Return

D. What Is Returned by the I/O Subroutines
   1. Input or Output Value
   2. Input or Output Units

III. Basic Functions -- CON and INCON

A. General Description
   1. Syntax of Unit Equation
   2. Conversion To/From SI
      a. Unit Name Table
      b. Conversion Factor Table
   3. Example -- ACRE*FT/DAY-GAL/MIN
      a. ACRE-M2 = 4.046856422 X 10^3

b. FT-M = 3.048 X 10^-1
c. DAY-S = 8.64 X 10^4
d. GAL-M3 = 3.785411784 X 10^-3
e. MIN-S = 6.0 X 10^1
4. Allowable Base Units
5. Exponents
6. Error Conditions
7. Short Form Unit Equation

B. How Dimensional Consistency Is Checked
1. Use of Checksums
2. Example -- ACRE*FT/DAY-GAL/MIN
a. ACRE = 138
b. FT = 69
c. DAY = 161
d. GAL = 207
e. MIN = 161
3. Checksum Selection
a. LN of Consecutive Primes
b. 1 Checksum for Each Type of Unit With a Single Mnemonic
c. Derived Unit Checksums Calculated from Base Unit Checksums
4. Base Unit Checksums
a. Length = 69 (FT)
b. Mass = 110 (KG)
c. Time = 161 (S)
d. Temperature = 240 (F)
e. Matter = 283 (MOL)
5. Derived Unit Checksums
a. Area = 138 (ACRE)
b. Volume = 207 (GAL)
c. Amount of Substance = 393 (SCF)
d. Pressure = 743 (PSI)
e. Power = 789 (KW)
f. Force = 881 (N)
g. Dynamic Viscosity = 904 (CP)
h. Density = 927 (API)
i. Energy = 950 (BTU)
j. Kinematic Viscosity = 1001 (CST)

C. Technical Details
1. 1400 Words Total
2. Speed
3. Processing Loop -- 500 words
a. Read Each Unit
b. Check for Invalid Characters
c. Check If Unit Exists
d. Process Special Cases
e. Multiply or Divide Each Conv. Factor N Times (Exponent N)
f. Add or Subtract Each Checksum N Times (Exponent N)
g. Repeat
h. Check Final Checksum

i. Return
4. Organization of Unit Name Table
   a. Alphabetical
   b. Address of Conversion Factor Table -- 1 word per unit
   c. Check First Character, Then Other Characters
5. Organization of Conversion Factor Table -- 900 words
   a. Last 1K of Any ROM
   b. Unit Name -- 1 word per character
   c. Checksum -- 1 word
   d. Conversion Factor -- 1 word per digit
   e. Exponent Order
   e. Repeated Entries
6. Error Conditions
7. System Scratch Usage

D. How Special Cases Are Handled (Temperature, API)

E. Use of CON and INCON in I/O Subroutines

IV. Application of Unit Management to the HP-41 Petroleum Fluids Pac

A. Program Structure
   1. Naming Conventions
   2. Two-Block Structure
      a. Control Block
      b. Computation Block
   3. Data Structure

B. Control Block
   1. CON and INCON Used in I/O Subroutines
   2. I/O Subroutines Used in Input and Output Subroutines
   3. Input and Output Subroutines Used in Control Block
      a. Naming Conventions
   4. General Purpose Subroutines
      a. TITLE
      b. Y/N?

C. Computation Block
   1. Naming Conventions
   2. Calling Conventions
      a. Use of Stack
      b. Use of Storage Registers
      c. Use of Flags
      d. Units at Call
   3. Return Conventions
      a. Values in Stack
      b. Units on Return

D. Analogy With the PPC (Routines) ROM
   1. Programs Built From Library of Subroutines

E. Weaknesses of the Unit Management System
   1. Speed of I/O Subroutines
   2. User's Units Not Stored
   3. Uppercase Only
   4. Checksums

V. Evolution of the Unit Management System

A. Machine Design
   1. FCON
   2. BCON
   3. 42 Allowed Units
   4. 3 Special Cases (F, C, R)

B. Thermal and Transport Science
   1. SI-
   2. -SI
   3. 55 Allowed Units
   4. 3 Special Cases (F, C, R)
   5. I/O Subroutines
      a. INPUT
      b. OUTPUT
      c. User Prompted for Units

C. Petroleum Fluids
   1. CON
   2. INCON
   3. 82 Allowed Units
   4. 4 Special Cases (F, C, R, API)
   5. Error Message
   6. I/O Subroutines
      a. IN, INU, INK
      b. OUT, OUTU, OUTK
      c. Unit Management Always Used Automatically

VI. Future of the Unit Management System

A. Upper and Lowercase

B. Use of Multipliers

C. User-Defined Units and Checksums
   1. Allow Expansion
   2. Allow Unit Redefinition (nanometer vs. nautical mile)

D. Multiple Spellings of Same Unit

E. Additional Units and Checksums

1. Electricity and Magnetism
2. Light
3. Angular Measure
4. Currency

F. New Checksum Scheme

G. Selective Unit Catalog

N O T E S

The Unit Management System

I. Overview of the Unit Management System in the 41 Petroleum Fluids Pac

A. Example of the Need for Unit Management -- Gas Viscosity
   1. Tc in R or K from Tables
   2. Pc in PSI, ATM, or KPA from Tables
   3. T in F or C from Lab/Field Measurements
   4. P in PSI, ATM, or KPA from Lab/Field Measurements
   5. Gas Viscosity in CP, LBM/FT*S, PA*S, KG/M*S

B. How Units Have Been Managed by Earlier Software
   1. Burden of Dimensional Consistency Placed on User
   2. Mandatory Set of Units
   3. Optional (Metric) Set of Units
   4. Any Dimensionally Consistent Set of Units

C. How Units Are Managed Now
   1. For Inputting Variables
   2. For Outputting Results
   3. Error Checking
      a. On Input
      b. On Output

D. Further Details
   1. Default Unit Systems
      a. English
      b. SI
      c. Other
   2. Optional Halt on Output
   3. 12-Character Limit

E. Program Structure
   1. Naming Conventions
   2. Two-Block Structure
      a. Control Block
      b. Computation Block
      c. Analogy With PPC (Routines) ROM
   3. Data Structure
      a. Variables Preserved
      b. Variables Stored in English Units

F. Computation Block
   1. Naming Conventions
   2. Calling Conventions
      a. Use of Stack
      b. Use of Storage Registers
      c. Use of Flags
      d. Units at Call
      e. Called With XEQ
   3. Return Conventions
      a. Values in Stack
      b. Units on Return

G. Control Block
   1. General Purpose Subroutines
      a. TITLE
      b. Y/N?
   2. Input and Output Subroutines
      a. Naming Conventions
   3. I/O Subroutines Used in Input and Output Subroutines
   4. Called With XROM

H. Significance of the "Unit Management System" Name

II. How Unit Management Works -- the I/O Subroutines

A. General Description
   1. Prompt the User
   2. Manage the Units If Any
   3. Store the Variables
   4. Format Printed Output

B. How to Call the I/O Subroutines
   1. IN (Input)
      a. Pointer to Storage Register
      b. Name of Input Variable
   2. INU (Input With Units)
      a. Pointer to Storage Register
      b. Name of Input Variable
      c. Units of Input Variable
   3. INK (Input With Units Known)
      a. Pointer to Storage Register
      b. Name of Input Variable
      c. Units of Input Variable
      d. Flags
   4. OUT (Output)
      a. Output Value
      b. Name of Output Value
   5. OUTU (Output With Units)
      a. Output Value
      b. Name of Output Value

1. Use of Checksums
   a. Add or Subtract for Speed
   b. Should Total to Zero
2. Example -- ACRE*FT/DAY-GAL/MIN
   a. ACRE -- 138
   b. FT -- 69
   c. DAY -- 161
   d. GAL -- 207
   e. MIN -- 161
3. Checksum Selection
   a. LN of Consecutive Primes
   b. 1 Checksum for Each Type of Unit With a Single Mnemonic
   c. Derived Unit Checksums Calculated from Base Unit Checksums
4. Base Unit Checksums
   a. Length = 69 (FT)
   b. Mass = 110 (KG)
   c. Time = 161 (S)
   d. Temperature = 240 (F)
   e. Matter = 283 (MOL)
5. Derived Unit Checksums
   a. Area = 138 (ACRE)
   b. Volume = 207 (GAL)
   c. Amount of Substance = 393 (SCF)
   d. Pressure = 743 (PSI)
   e. Power = 789 (KW)
   f. Force = 881 (N)
   g. Dynamic Viscosity = 904 (CP)
   h. Density = 927 (API)
   i. Energy = 950 (BTU)
   j. Kinematic Viscosity = 1001 (CST)

C. Technical Details
1. 1400 Words Total
2. Speed
3. Processing Loop -- 500 words
   a. Read Each Unit
   b. Check for Invalid Characters
   c. Check If Unit Exists
   d. Process Special Cases
   e. Multiply or Divide Each Conv. Factor (N Times for Exponent N)
   f. Add or Subtract Each Checksum (N Times for Exponent N)
   g. Repeat for Entire Unit Equation
   h. Check Final Checksum
   i. Return
4. Organization of Unit Name Table -- 85 words
   a. 3rd 1K of Any 4K ROM
   b. Alphabetical Order
   c. Address of Conversion Factor -- 1 word per unit
   d. Check First Character, Then Other Characters
5. Organization of Conversion Factor Table -- 900 words
   a. 4th 1K of Any 4K ROM

b. Exponent Order
c. Unit Name -- 1 word per character
d. Checksum -- 1 word
e. Conversion Factor -- 1 word per digit
f. Repeated Entries

D. Limitations of the Unit Management System
1. Speed of I/O Subroutines
2. Uppercase Only
a. Cannot Distinguish Between mW and MW
b. Choice by Needs of Petroleum Industry
3. Accuracy
a. 10-Digit Conversion Factors
b. Exponent Wraparound
c. Single-Digit Exponent
4. Number of Basic Units
5. Checksum Scheme
6. User's Units Not Always Saved
a. Space Restrictions

IV. Evolution of the Unit Management System

A. Machine Design
1. FCON
2. BCON
3. 42 Allowed Units
4. 3 Special Cases (F, C, R)
5. Speed
a. Best Case -- 1X
b. Worst Case -- 1X

B. Thermal and Transport Science
1. SI-
2. -SI
3. 55 Allowed Units
4. 3 Special Cases (F, C, R)
5. Speed
a. Best Case -- 1X
b. Worst Case -- 0.8X
6. I/O Subroutines
a. INPUT
b. OUTPUT
c. User Prompted for Units

C. Petroleum Fluids
1. CON
2. INCON
3. 82 Allowed Units
4. 4 Special Cases (F, C, R, API)
5. Speed

a. Best Case -- 0.9X
b. Worst Case -- 3.3X

6. I/O Subroutines
   a. IN, INU, INK
   b. OUT, OUTU, OUTK
   c. Unit Management Always Used Automatically
7. New Error Message

V. Future of the Unit Management System

A. Upper and Lowercase
   1. Allow True Unit Mnemonics (like Pa, ft)

B. Use of Multipliers
   1. Allow mW and MW Without Repeated Entries
   2. Speed Increase With Reduced Table Length

C. Additional Units
   1. Electricity and Magnetism
   2. Light
   3. Angular Measure
   4. Bases?
   5. No Repeated Entries

D. Additional Checksums
   1. Electricity and Magnetism
   2. Light
   3. Angular Measure
   4. Currency
   5. Bases?
   6. Wild Cards

E. User-Defined Units and Checksums
   1. Allow Expansion of Conversion Factor Table
   2. Allow Customization
   3. Allow Unit Redefinition (nanometer vs. nautical mile)

F. Better Accuracy
   1. Use Full Internal Precision

G. New Checksum Scheme
   1. Keep Coefficients of Base Unit Components

H. Selective Unit Catalog

I. Always Save User's Units
   1. Do Conversion in Calculation Subroutine

UNIT REQUIREMENTS OF GAS VISCOSITY CALCULATION

VARIABLE	TYPICAL UNITS	SOURCE
Pseudocritical Temperature (Tc)	R K	Tables
Pseudocritical Pressure (Pc)	PSI ATM KPA	Tables
Temperature (T)	F C	Measurements
Pressure (P)	PSI ATM KPA	Measurements

INPUT AND OUTPUT UNITS IN 41 PETROLEUM FLUIDS PAC

P=?

103 [ALPHA] ATM [R/S]

P=103.0000 ATM

UG, CP?

LBM/FT*S [R/S]

UG=1.3275E-5 LBM/FT*S

UG=1.3275E-5

P=?

[←] *1513.6827*

[ALPHA] *PSI*

*UG, LBM/FT*S?*

INPUT AND OUTPUT UNITS
ERROR HANDLING

P=?

103 [ALPHA] GAL [R/S]

GAL?

UG, CP?

ACRE [R/S]

UG, ACRE?

TWO-BLOCK STRUCTURE

CONTROL BLOCK -- ALL USER INTERACTION

CALLS TO GENERAL PURPOSE SUBROUTINES

CALLS TO INPUT AND I/O SUBROUTINES

CALLS TO COMPUTATION BLOCK

COMPUTATION BLOCK -- ALL CALCULATIONS

CALCULATION SUBROUTINES

CALLS TO CALCULATION SUBROUTINES

GENERAL DESCRIPTION OF I/O SUBROUTINES

PROMPT THE USER FOR INPUT OR OUTPUT

MANAGE THE UNITS IF ANY

STORE THE VARIABLES

FORMAT PRINTED OUTPUT

I/O SUBROUTINES AT CALL

IN (Input)

ALPHA: Variable Name

R00: Pointer

INU (Input With Units)

ALPHA: Variable Name

R00: Pointer

R01 & R02: English Units

Y & Z: SI Units

INK (Input With Units Known)

ALPHA: Variable Name

R00: Pointer

R01 & R02: English Units

Y & Z: SI Units

F08: Set On 1st Pass Only

I/O SUBROUTINES AT CALL

OUT (Output)

ALPHA: Variable Name

X: Output Value

OUTU (Output With Units)

ALPHA: Variable Name

X: Output Value

R01 & R02: English Units

Y & Z: SI Units

OUTK (Output With Units Known)

ALPHA: Variable Name

X: Output Value

R01 & R02: English Units

Y & Z: SI Units

F08: Set On 1st Pass Only

OPERATIONS PERFORMED BY I/O SUBROUTINES

INITIALIZE REGISTERS AND FLAGS

BUILD PROMPT

SET UP HALT CONDITIONS AND HALT

CHECK FOR NUMERIC OR ALPHA INPUT

STORE RESULT

Convert to English Units

Check for Conversion Errors

PRINT AND/OR DISPLAY VALUE

SET UP RETURN CONDITIONS AND RETURN

I/O SUBROUTINES UPON RETURN

X: INPUT OR OUTPUT VALUE
IN ENGLISH UNITS

Y & Z: UNITS ENTERED BY USER

SYNTAX OF UNIT EQUATION

UNIT#1–UNIT#2

GAL–BBL

*KW*HR*

FT/S

FT/S2

*HR*FT2*F/BTU*IN*

CONVERSION EXAMPLE

20 ACRE*FT/DAY = ?? GAL/MIN

ACRE–M2	20	→	80937	(*)
FT–M	80937	→	24670	(*)
DAY–S	24670	→	3 E–01	(/)
GAL–M3	3 E–01	→	75	(/)
MIN–S	75	→	4526	(*)

ALLOWABLE BASIC UNITS

ACRE	GALUK	LBM	PA
API	HP	M	PSF
ATM	HR	MBAR	PSI
BAR	IN	MCF	R
BBL	INHG	MD	S
BCF	INH2O	MG	SCF
BTU	J	MI	SCM
C	K	MIN	SCMZ
CAL	KCAL	MJ	SPGR
CM	KG	ML	ST
CP	KGF	MM	T
CST	KIP	MMCF	THERM
D	KJ	MMHG	TON
DAY	KM	MN	TONUK
DYNE	KMOL	MO	TORR
ERG	KPA	MOL	UM
F	KSI	MPA	W
FT	KT	MT	YD
FTH2O	KW	MW	YR
G	L	N	
GAL	LBF	P	

ERROR CONDITIONS

ALPHA DATA IN X

INVALID CHARACTER IN UNIT EQUATION

(not *,/,−,1−9, or A−Z)

ZERO EXPONENT ON UNIT

(e.g., FT0)

MULTIPLE DASHES IN UNIT EQUATION

MULTIPLE "/" IN UNIT EQUATION

DIMENSIONAL INCONSISTENCY

UNIT NOT ALLOWED

API WITH EITHER "*" OR "/"

API WITH 0 OR −131.5 IN X

DIMENSIONAL CONSISTENCY EXAMPLE

20 ACRE*FT/DAY = ?? GAL/MIN

ACRE	0	→	138	(+)
FT	138	→	207	(+)
DAY	207	→	46	(−)
GAL	46	→	863	(−)
MIN	863	→	0	(+)

CHECKSUMS USED IN 41 PETROLEUM FLUIDS PAC

UNIT TYPE	SUM	EX.
LENGTH	69	FT
MASS	110	KG
TIME	161	DAY
TEMPERATURE	240	F
MATTER	283	MOL
AREA	138	ACRE
VOLUME	207	GAL
AMOUNT OF SUBSTANCE	393	SCF
PRESSURE	743	PSI
POWER	789	KW
FORCE	881	N
DYNAMIC VISCOSITY	904	CP
DENSITY	927	API
ENERGY	950	BTU
KINEMATIC VISCOSITY	1001	CST

PROCESSING LOOP FOR CON AND INCON

READ EACH UNIT

CHECK FOR INVALID CHARACTERS

CHECK IF UNIT EXISTS

PROCESS SPECIAL CASES

MULTIPLY OR DIVIDE EACH CONV. FACTOR
(N times for exponent N)

ADD OR SUBTRACT EACH CHECKSUM
(N times for exponent N)

REPEAT FOR ENTIRE UNIT EQUATION

CHECK FINAL CHECKSUM

RETURN

ORGANIZATION OF TABLES

UNIT NAME TABLE (85 words)

3RD 1K OF ANY 4K ROM

ALPHABETICAL ORDER

ADDRESS OF CONV.FACTOR (1 word per unit)

CHECK 1ST CHARACTER, THEN OTHER CHARS.

CONVERSION FACTOR TABLE (900 words)

4TH 1K OF ANY 4K ROM

EXPONENT ORDER

UNIT NAME (1 word per character)

CHECKSUM (1 word)

CONVERSION FACTOR (1 word per digit)

REPEATED ENTRIES

LIMITATIONS OF THE UNIT MANAGEMENT SYSTEM

SPEED OF I/O SUBROUTINES

UPPERCASE ONLY

ACCURACY

NUMBER OF BASIC UNITS

CHECKSUM SCHEME

USER'S UNITS NOT ALWAYS SAVED

EVOLUTION OF UNIT MANAGEMENT SYSTEM

PAC NAME	FCN. NAME	NO. UNITS	SPEC. CASE	SPEED BEST	SPEED WORST	I/O SUBS
MACH. DES.	FCON BCON	42	F,C,R	1X	1X	NONE
T & T SCI.	SI– –SI	55	F,C,R	1X	0.8X	INPUT OUTPUT
PET. FLUIDS	CON INCON	82	F,C,R API	0.9X	3.3X	IN INU INK OUT OUTU OUTK

(INCLUDES NEW ERROR MESSAGE)

FUTURE OF THE UNIT MANAGEMENT SYSTEM

UPPER AND LOWERCASE

USE OF MULTIPLIERS

ADDITIONAL UNITS

ADDITIONAL CHECKSUMS

USER-DEFINED UNITS AND CHECKSUMS

BETTER ACCURACY

NEW CHECKSUM SCHEME

SELECTIVE UNIT CATALOG

ALWAYS SAVE USER'S UNITS

STATE-OF-THE-ART IN SOFTWARE

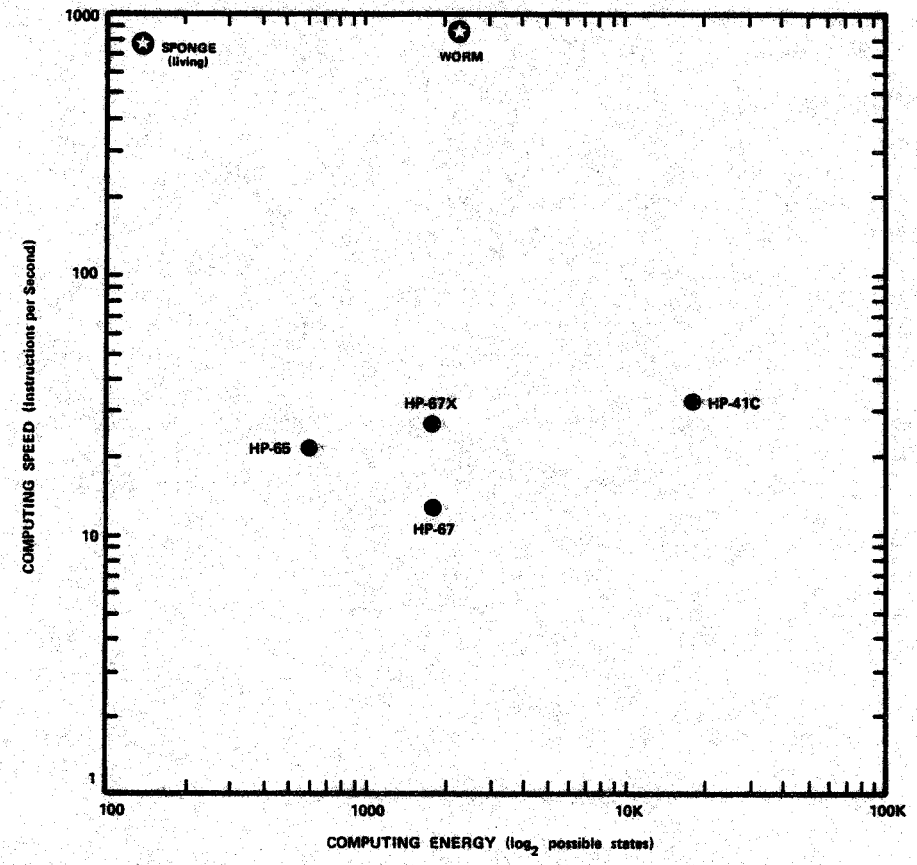
This paper deals with the state-of-the-art in hand-held computer software. The first part is primarily a quantitative assessment of the subject while the second part deals more with qualitative issues.

The first hand-held programmable calculator was introduced by HP in 1974 for $795. It was followed two years later by the HP-67/97 at $450. For many of us, these machines were our introduction to software and the world of computer programming. With the advent of the HP-41C and subsequent price reductions, the cost and convenience of hand-held computers fell within a range acceptable to a vast number of potential users. It's instructive for us to look back at the programs that were developed for these machines to gain some perspective for where we stand with today's software.

Let's begin by looking at the relative power of these now classic calculators. Figure 1 compares them on the basis of speed and capacity. Speed is measured in terms of the average number of instructions processed per second. Capacity is based on the number of possible programs or states the machine could ever be in. This is estimated from the program memory size and the size of the instruction set. If each of the 100 steps in the HP-65 could contain one of 64 possible instructions, there would be 4×10^{180} different programs. This number is more conveniently expressed as a logarithm to the base two. The compute energy thus calculated is about 600 bits. The corresponding compute energy of the HP-67 is 1.8K bits and the HP-41C is an impressive 17.8K bits! For simplicity, we have ignored the permutations possible by considering the data registers as well. The increase in speed from the HP-65 to the HP-41C, however, has been marginal. The HP-67 was in fact slower than its predecessor. A few PPCers, bold enough to challenge the supreme order of things, were able to double the speed of their HP-67's and thereby obtain their rightful advance in calculator state-of-the-art.

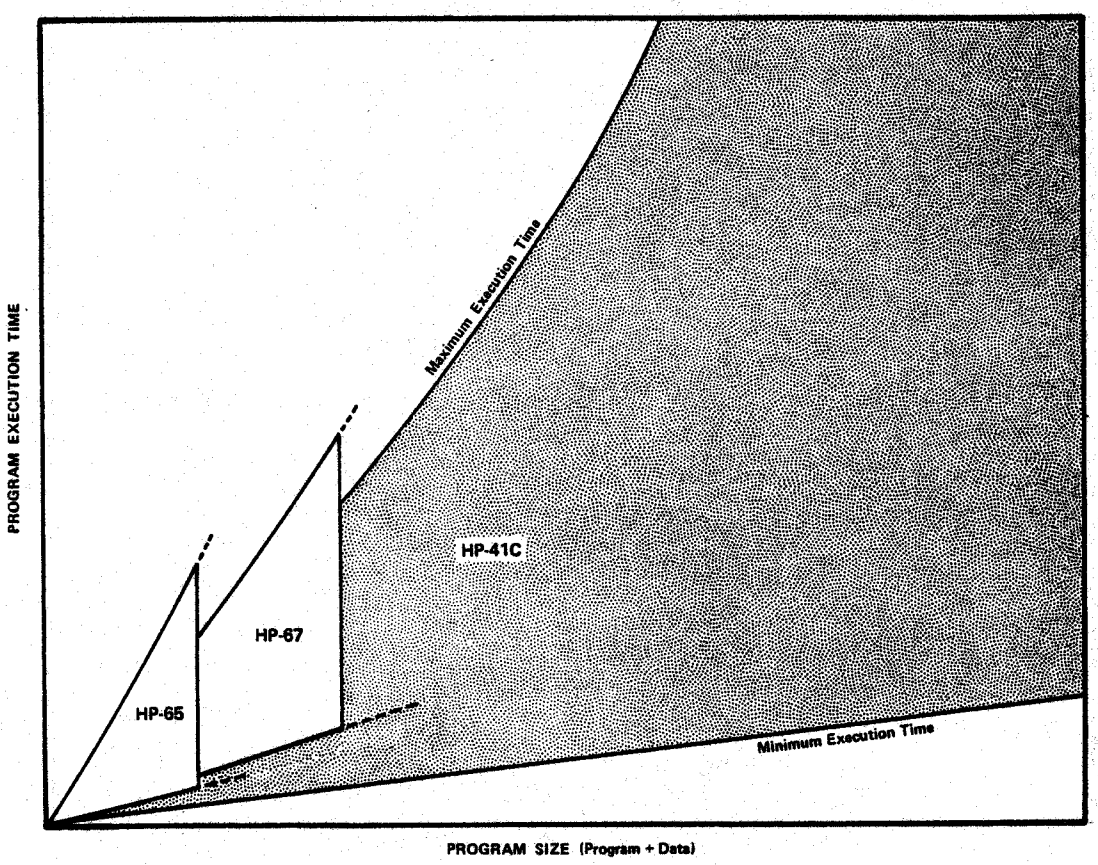
It's fair to point out that some of the "lowest" life-forms have far greater computing power than our marvelous, but nevertheless limited, machines. It could be argued, however, that such life-forms are not nearly as "friendly" as the HP-41C.

The limitations in speed and memory for the first hand-held calculators resulted in a practical definition of good software: better programming became synonymous with fast and short. The limitations of this definition represented by each generation of calculator are illustrated in Figure 2. All of the useful programs for a particular machine can be characterized by the respective areas for each. At the lower limit is the maximum speed of execution of the simplest instruction; at the upper limit is the time needed for iterative loops and built-in functions such as N! and cosine.



MILESTONES IN HAND-HELD COMPUTING POWER

Figure 1



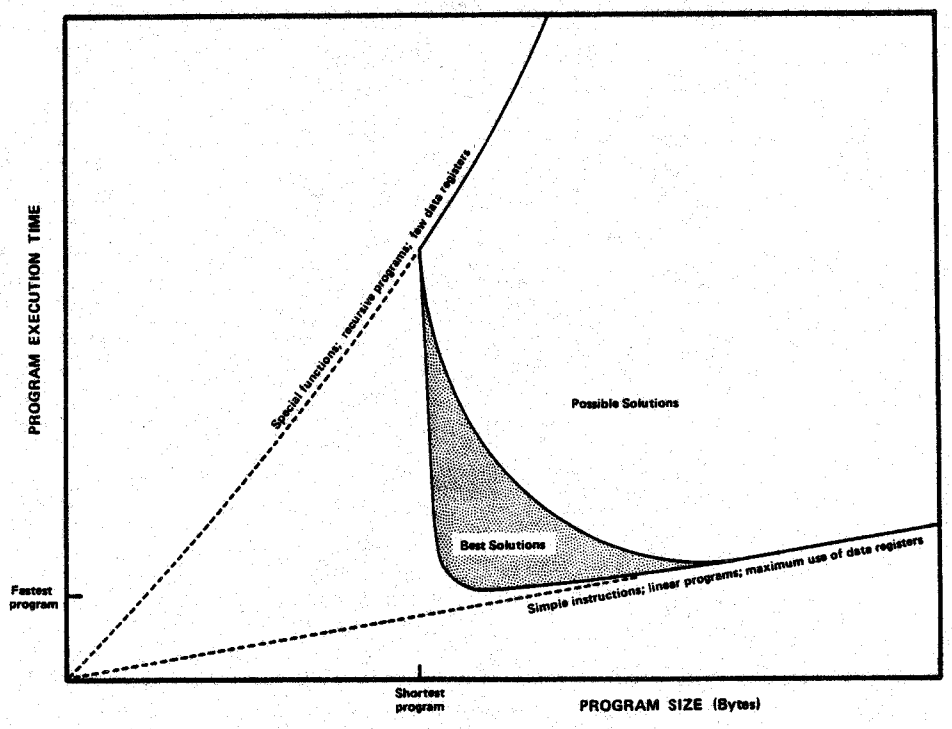
PROGRAM CHARACTERISTICS OF THREE GENERATIONS OF CALCULATORS

Figure 2

Figure 3 can be considered a magnified view of Figure 2 for a particular calculator and program. The generalized curve was obtained by examining a number of different programs designed to do the same thing. Examples of two such routines are presented in Table 1. These data were found in various issues of the PPC Journal and may be considered state-of-the-art samples.

Table 1. Comparison of Program Size and Execution Times for Comparable HP-41C Programs

SIZE Finder		Calendar Routine	
Bytes	Speed (sec.)	Bytes	Speed (sec.)
25	86	343	115
31	4.1	367	390
34	4.0	373	139
35	14	1397	74
46	22		
55	1.5		

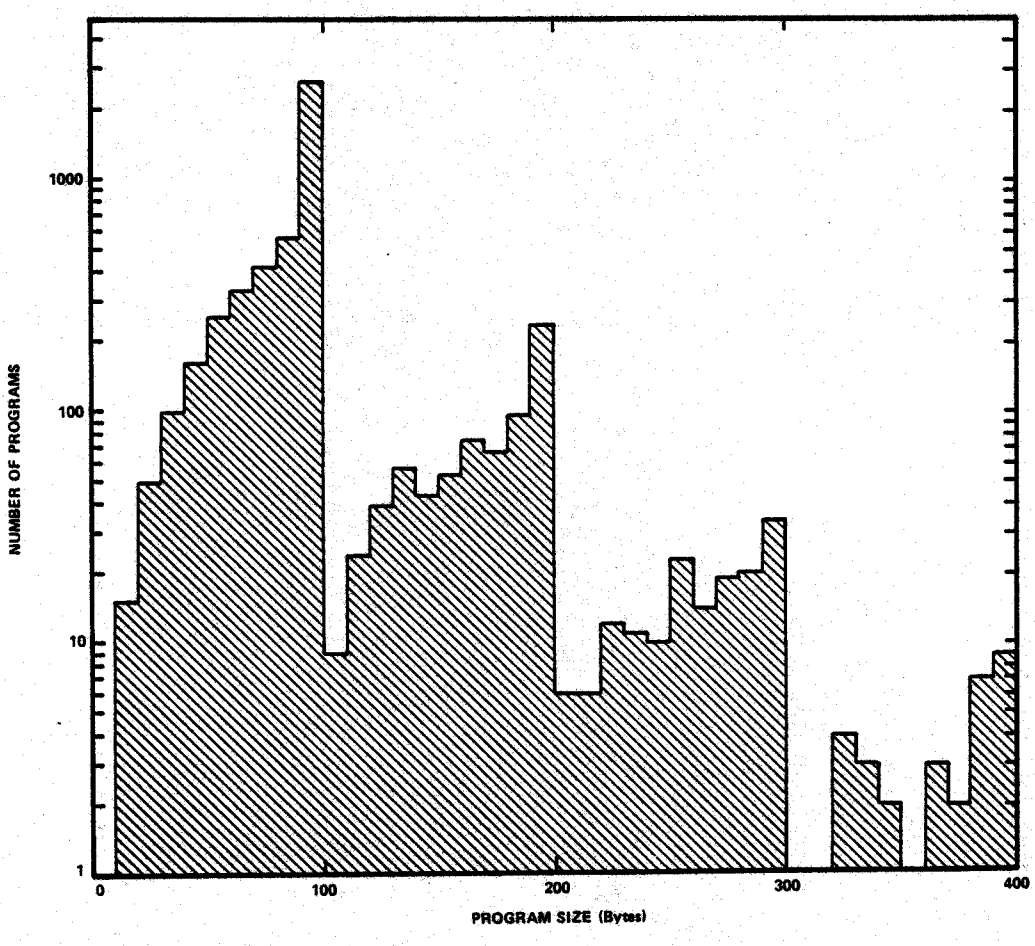


EXECUTION TIME VERSUS PROGRAM SIZE

Figure 3

The operative definition of good software in these examples spans the gamut from very short and fairly fast to very fast and fairly short. Many other solutions are obviously possible, but the "best" programs fall on the leading edge of a hyperbola-like function.

One of the early constraints on "good" software was the necessity to get everything into the calculator's memory. A glance at the distribution of program sizes in the HP-65 Users' Library (Figure 4) hints at the enormous efforts expended to achieve this objective. You can be sure that most of the programs that ended up in the 91 to 100 step category started out in the over 100 category. Many ingenious tricks and unsupported features were evoked which often resulted in programs that were impossible to decipher. In many cases, the programmer made compromises that also excluded desirable features. Failing this, the programmer would chain two or more programs together, passing the unfinished computations on to the next part of the program. It is interesting to note that while the machine's 100 step limitation did not deter all programmers, over 83% of the programs in the HP-65 Library were 100 steps or less. It's particularly striking that 49% of these programs were between 91 and 100 steps! Ninety-six percent of the programs were 200 steps or less.



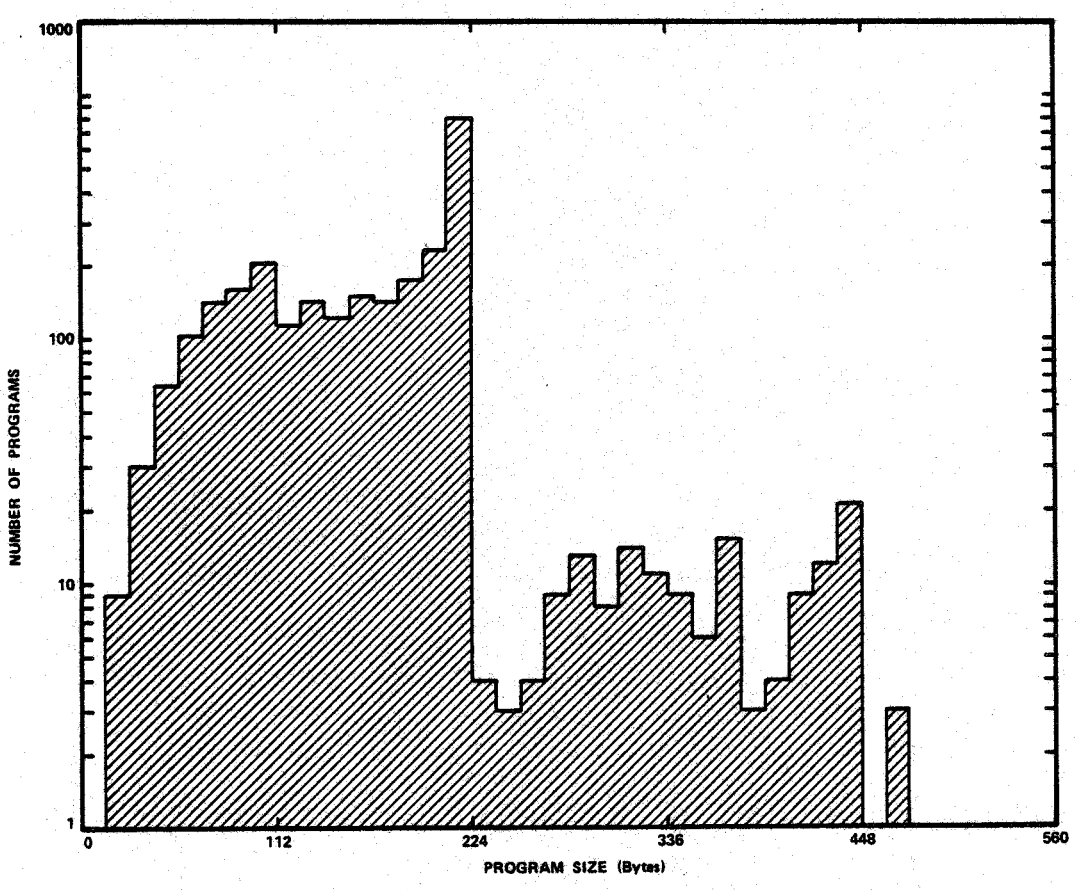
DISTRIBUTION OF PROGRAM SIZES IN HP-65 USERS' LIBRARY

Figure 4

The message to HP was clear: give us more memory --- 224 steps, give or take a few, should be adequate for most purposes. HP obliged with the HP-67/97. One result of this improvement is reflected in Figure 5. Only 6% of the programs submitted to the HP-67/97 Users' Library actually exceeded program memory and virtually none required more than 448 steps. (The most interesting extremes were a 2,055 step program and a 4 step program.) With larger programs, the speed limitations of the HP-67 became more apparent. Better software not only meant doing everything in 224 steps but doing it in a reasonably short time. More emphasis was placed on programming techniques that would shorten execution time as well as steps. This was the theme of "Better Programming on the HP-67/97." Apparently most programmers using such techniques found it possible to provide most of the desired features within the confines of available memory.

It wasn't apparent at this point, however, whether increasing memory size once again would stimulate longer programs and greater sophistication

or if 224 steps was actually a good stopping place. After all, there's a limit to what you would use a hand-held calculator for; a truly big program such as one to play checkers requires a big computer, right? A related issue is whether or not all of the useful programs have been written already. We'll digress a moment and attempt a partial answer to this last point and come back to it again in the second part of this paper.

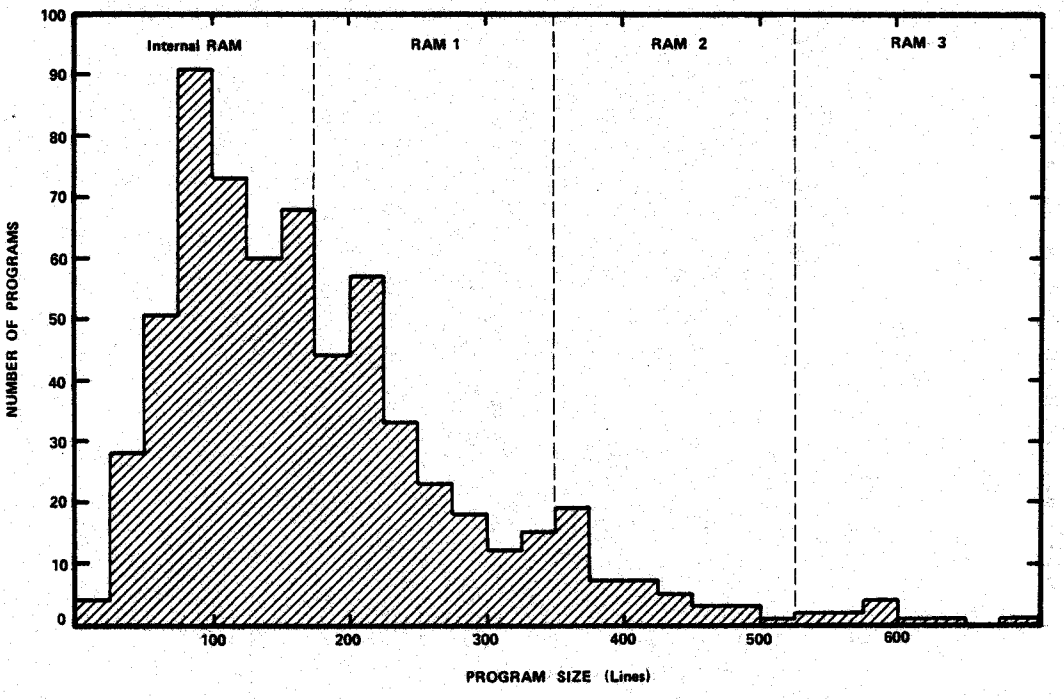


DISTRIBUTION OF PROGRAM SIZES IN HP-67/97 USERS' LIBRARY

Figure 5

Recall the earlier discussion of compute energy and all of the possible programs. Most of these would be gibberish although some would undoubtedly reveal ingenious answers if we only knew the question. A more realistic estimate of the plausible program count can be obtained by assuming any instruction is likely to be followed by one of two possible alternatives. The number of such programs is about 600,000 for the HP-65, 13,000,000 for the HP-67 and 1,300,000,000 for the HP-41C. Since the combined libraries for all three calculators hardly exceeds 10,000 entries, this should encourage anyone wondering if there are any programs left to write.

Sadly, the HP-67/97 Library has yet to reach the size of the former HP-65 Users' Library in spite of its greater capability and lower price. Undaunted, HP produced another calculator with far greater capabilities than either of its predecessors. It should come as no great surprise then that the potential of the HP-41C is hardly being tapped (see Figure 6). This is partly because the same old programs are still being written in the same old ways.



DISTRIBUTION OF PROGRAM SIZES IN HP-41C USERS' LIBRARY

Figure 6

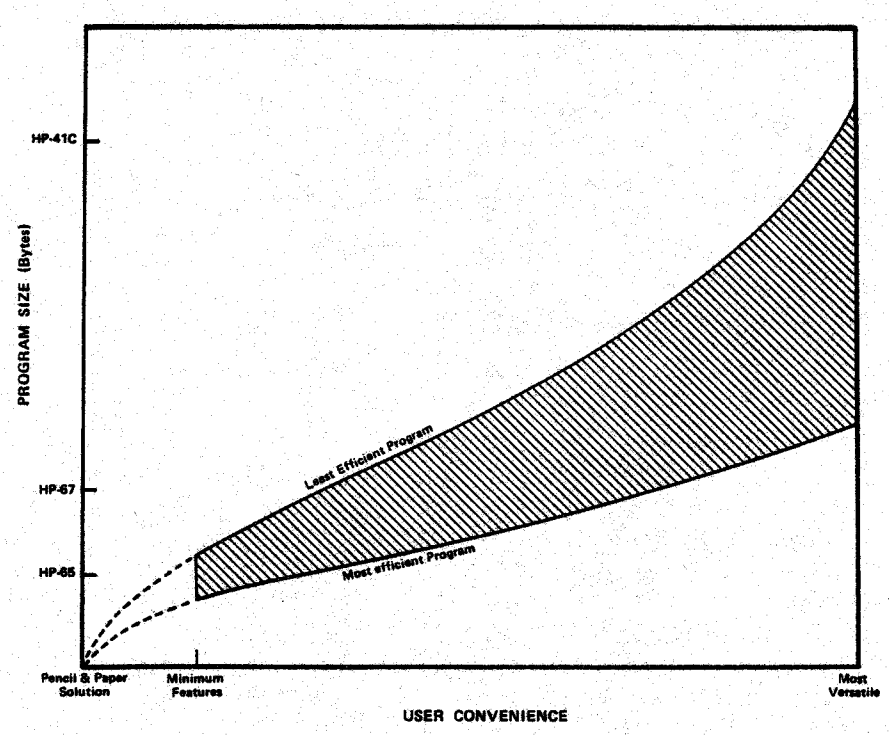
Program size on the HP-41C is somewhat misleading since the number of bytes per line is variable. A good rule-of-thumb, however, is 2.1 bytes per line. Unlike previous machines, there are no compelling reasons to terminate a program within a certain number of steps. The availability of plug-in memory modules or RAM constitutes the major practical barrier. We can estimate the average number of lines in each memory module by assuming the number of data registers required is roughly proportional to program size. It would thus appear that perhaps 60% of the programs in the current HP-41C Library require no external memory. About 90% of the programs can be run with only one external memory module.

The feature of the HP-41C that sets it apart from the rest, however, is its full alpha capability. The alphanumeric display makes it possible for the first time to read program code, to insert prompts and instructions in the program, and to identify each output. Memory limitations have essentially been eliminated leaving only execution speed as a possible consider-

ation. We believe these developments suggest a new philosophy in designing software for hand-held computers. First and foremost in any program for general use is a human factors approach from input to output. Following closely behind is complete documentation. More often than not, other users will want to tailor your program to their specific needs. The difficulty in doing this is inversely proportional to the thoroughness of your documentation. The educational value of good documentation should not be underestimated either.

The ultimate objective of all state-of-the-art software should be user convenience. In Figure 7, it is clear that earlier machines were limited by their memory size and numeric abilities. This obstacle no longer dominates our programming efforts. The challenge now is to provide all the conveniences in as efficient a program as possible. HP has made the calculator friendly but it's up to the users to make the software friendly.

W.M.KOLB (265)



PROGRAM SIZE VERSUS CONVENIENCE

Figure 7

EVALUATING CALCULATOR SOFTWARE

Along with the explosive development of the handheld programmable calculator has come a corresponding flood of "software", i.e., the programs written for the calculators. The thousands of programs in, for instance, the HP User's Library is only the tip of the iceberg, considering that these programs are just the few that their authors took the trouble to submit to the Library. The problem I want to address in this paper is whether it is useful or possible to establish a set of objective criteria for evaluating software. To be specific, how do we decide if a program is "good" or "bad" or somewhere in-between?

The utility of such evaluation should not really be in question. A calculator user most likely will have a definite use for only a tiny fraction of available programs; he must be able to distinguish good software from bad as well as be able to determine which programs are applicable to his needs. If he plans to purchase software, the evaluation becomes especially crucial--nobody wants to spend hard-earned bucks on junk programs. The seller also need to be able to evaluate his own product. Sale of a few bad programs could very well end his software-sales career, regardless of the quality of his subsequent efforts.

The problem, then, is not "why?", but "how?". The first test a program must pass is obvious: does it correctly compute and output a result based upon user-supplied input? But given the myriad of possible routes between input and output, the criteria necessary to elaborate upon this crude quality test are much less clear-cut. Let us consider three general areas of software performance:

1. Memory use. Does the program fit into the available calculator memory? For the HP-41C, this problem is of much less importance than for the smaller memory machines. But regardless of the memory size, it is always desirable to minimize the length of a program--for small machines, the length of the program, i.e., whether it will fit in the memory at all, is often the dominant consideration; whereas for a 41C, the question is more often whether the program will fit in along with the umpteen other programs the user wants in the active library. In brief: the shorter, the better.

2. Execution speed. Sure, watching the goose fly around, or the LED's blink, is lots of fun, but it grows old in a hurry. Given two programs that accomplish the same task, the one that executes faster is clearly preferable, all other things being equal.

3. User-Calculator interaction. This might also be called "convenience of use." Here I am primarily considering input and output convenience, a fuzzy concept that is much harder to quantify than the first two criteria. For input, the best programs will prompt the user in some logical way for data entry, so that he needs to consult program instructions as little as possible. The HP-41C

has a great advantage in this regard over previous machines in its capablility of providing word prompts. Furthermore, a program should require only one entry of each set of data. If possible, the program should check the keyed-in data for error, and a straightforward error-recovery routine should be provided. On output, the program should display results for the convenience of the user, again in such a way as to minimize his need for written explanation of the output seqence. Computed results should be stored for simple retrieval even after the normal output sequence is complete.

So all we have to do to write a good program is write one with minimum memory use, maximum execution speed, and optimum user-calculator interaction, right? The answer is an unequivocal maybe. The problem is that the three criteria are incompatible. For example, a prime method of reducing the length of a program is to use indirect addressing of registers in loops, rather than a single long sequence of direct register calls. But indirect register calls are much slower than direct, and when you fold in the delays associated with GTO's and label searches, your beautiful ingenious 100-byte tour-de-force may take four times as long to run as the el crudo 400-byte dinosaur it replaces. Also, all those wonderful prompts, error checks, output drawings, etc., have a voracious appetite for execution time, not to mention their cost in program length.

In the face of the fundamental incompatibility of the three criteria described so far, we are reduced to falling back on the simple practical test: does the program do its job? But we can elaborate the question by recognizing that "doing the job" includes a lot more than just whether the output numbers are correct. We must consider whether a program has an appropriate mixture of the considerations embodied in the three criteria discussed so far. We must take into account the amount of time spent in producing the program and in trying to perfect it. Finally, we must look hard at how the program is to be used, and by whom. The remainder of this paper is an attempt to provide a list of questions a programmer should ask about his program, to help him decide whether his program is a "hit" or a "miss." I will conclude with some examples of what I consider good programming.

A Programmer's Self-Test

1. Length.
   A. Is the program as short as possible?
   B. Would shortening the program require an unreasonable investment in programming time?

2. Execution time.
   A. Does the program run in an acceptable amount of time?
   B. Is there a minimum of "slow" functions, such as SIN or LOG?
   C. Are all those GTO's and XEQ's, not to mention the fancy indirect calls, really necessary?
   D. Does the program sacrifice user input convenience for faster calculator execution? If so, remember that it's

total execution time, which includes user input and output, that counts.

3. Programming time.
   A. Is this program going to be used more than once?
   B. How much programming time is worthwhile to perfect a program?
   C. Why is this program being written at all?

4. Accuracy.
   A. Are the calculated results obtained with sufficient accuracy?
   B. Does the program waste time working for 10-digit accuracy when the inputs are only good to a couple of significant figures?
   C. Does the program handle properly malignant data, i.e., does it try to divide by (near-) zero, or subtract small numbers from large, etc.?

5. Input.
   A. Does the user need elaborate instructions, or is the input self-explanatory?
   B. Is the input format convenient?
   C. Are inputs properly prompted?
   D. Does the program test input for reasonability?
   E. Is each set of data input only once?
   F. Can one input be changed without re-entering the rest?
   G. Is straightforward error correction provided?

6. Output.
   A. Are outputs self-explanatory, with appropriate units?
   B. Is there a choice between output media?
   C. Are preliminary results output without requiring user intervention?
   D. Are outputs presented in a logical order?
   E. Can individual results be retrieved following execution without having to run the whole program again?
   F. Are outputs stored in locations suitable for use with other programs?

7. Program structure.
   A. Is the program organized into autonomous blocks suitable for use by other programs where appropriate?
   B. Can the entire program be called as a subroutine without I/O halts?
   C. Are inputs and outputs clustered together, or does the user need to babysit the program throughout its execution?
   D. Is the flow of execution straightforward and easy to decipher?

8. Documentation.
   A. Is the documentation prepared at a level appropriate to the intended user?
   B. Will I be able to understand this program from its documentation if I come back to it after 6 months?
   C. Does the documentation provide the fundamental formulae and algorithms?
   D. Are the input/output units explained?
   E. Is the program self-documenting?

F. Are sample problems and solutions provided?
G. Is there a list of references?
H. Based on the documentation provided, who could operate this program (Einstein, professor of computer science, engineer, businessman, a disinterested wife, my 5-year-old son, the family dog, a guppy, an AOS-user)?

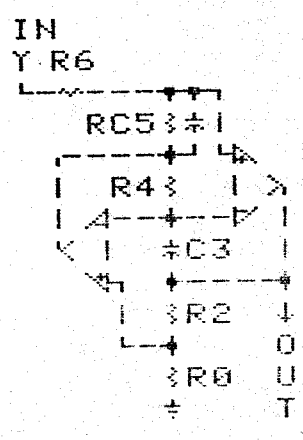
9. Sophistication.
   A. Does this program take advantage of the full capabilities of the machine for which it's written?
   B. Would this program work just as well on a lesser calculator?
   C. Does the sophistication of the program, i.e., use of special functions, indirect addressing, subroutines, synthetics, etc., actually improve the overall performance of the program?

10. Final Analysis.
   A. When all is said and done, am I better off with this program than I would have been if I had obtained the result by any other method?
   B. Would you buy a used program from this man.

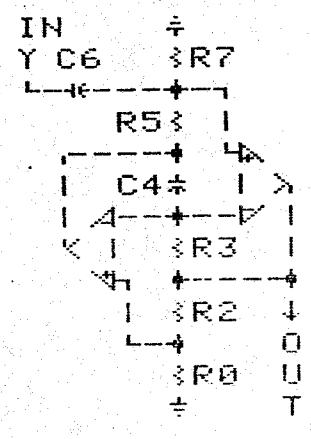
Some Examples of Good Programming

1. Clever output: Leland van Allen's circuit designer. The printout results of this program speak for themselves. User inputs the filter type, and the filter frequency, and the 41C proceeds to design and draw the filter circuit:

```
LOWPASS:
FP=1,400.00 HZ
Q= 2.14

R0=1000.00 Ω
R2=1000.00 Ω
C3=0.01 μF
R4=11368.21 Ω
R5=24282.50 Ω
C5=0.01 μF
R6=11368.21 Ω

AT 1,200.00 HZ:
dB= 12.38
DEG= -56.53
```




```
HIGHPASS:
FP=786.14 HZ
Q= 3.33

R0=1000.00 Ω
R2=1000.00 Ω
R3=20245.11 Ω
C4=0.01 μF
R5=20245.11 Ω
C6=0.01 μF
R7=67416.23 Ω

AT 1,200.00 HZ:
dB= 10.40
DEG= 19.02
```



2. Convenient input. This fraction-arithmetic program, which I wrote for use in my shop, is distinguished by the ease of entry of fractions. The user enters a fraction just as he would say it, with the decimal point as the "and": "one and two-thirds" is entered simply as 1.23; 2-27/64 is entered as 2.2764; 9-1/32 is entered as 9.132. The user isn't slowed down by having to remember a special input format--the calculator does the work.

3. Mr. Five-by-Five--programming ingenuity at its best. This gem by Hal Brown (362) and Earl Robinson (1153) is still to me the best program I have ever seen, given the limitations of the HP-67. The brilliance of being able to invert a 5x5 matrix in place, in a machine with only 26 data registers, is a supreme achievement in program compactness and speed (we are quite willing to overlook the necessary clumsiness of data entry/output).

WILLIAM C. WICKES

Input and output formats:

A. The nominal format for fractions is I.NNNDDD represents $I^{NNN/DDD}$
(Example 1.022047 ⇒ $1^{22/47}$)

B. All fractions are output in this manner.
For convenience and speed of entry, however, if the denominator DDD ≤ 99, the fraction may be entered without intervening zeros, i.e. $A^{bb/cc}$ can be entered as A.bbcc, $A^{b/cc}$ as A.bcc, and $A^{b/c}$ as A.bc. Examples: $1^{2/3}$ → 1.23 ; $1^{22/47}$ → 1.2247 ; $1^{1/16}$ → 1.116 etc.

C. If all numbers entered have a non-zero fractional part, then further simplification is provided by the "all fraction" mode. In this mode numbers which are fractions only: NNN/DDD (I = 0) may be entered without bothering with the decimal point. bb/cc → bbcc ; b/cc → bcc ; b/c → bc. Examples: $^{13}/_{64}$ → 1364 ; $^{1}/_{4}$ → 14. Numbers with non-zero integer parts are still entered as in B

D. For numbers with denominators ≥ 100, the considerations in B and C above still apply except for numbers of the form (A) b/ccc", which if entered as (A.)bccc" would be interpreted as (A) bc/cc". Such numbers should be entered as (A).0bcc'c" or (A).00bcc'c"

E. Numbers with numerators with more digits than the denominators should be entered in the nominal format only. Example: $^{199}/_{64}$ → 1.199004

	KEY ENTRY	KEY CODE	COMMENTS
.	LBL A	31 25 11	x → Decimal
	a	32 22 11	Find num. and denom.
	RCL A	34 11	
	FRAC	32 83	
	x=0?	31 51	
	GTO 6	22 06	
	RCL 3	34 03	
	x	71	numerator
	INT	31 83	
010	LST x	35 82	
	FRAC	32 83	
	RCL 4	34 04	
	x	71	denominator
	÷	81	decimal fraction
	RCL A	34 11	
	INT	31 83	Restore integer
	+	61	
	DSP 3	25 03	
3	LBL 3	31 25 03	
020	PO?	35 71 00	Restore sign
	CHS	42	
	CF0	35 61 00	
	RTN	35 22	
6	LBL 6	31 25 06	
	RCL A	34 11	Short cut for integer x
	GTO 3	22 03	
B	LBL B	31 25 12	x → Fraction
	x<0	31 71	
	SF0	35 51 00	Flag 0 for x<0
030	ABS	35 64	
	STO A	33 11	
	FRAC	32 83	
	RCL D	34 14	
	x=0?	31 51	
	GSB 7	31 22 07	Look up default parameters
	STO 2	33 02	Starting denominator
	x	71	
	DSP 0	25 00	
	RND	31 24	Nearest integer numerator
040	DSP 6	25 06	
	x=0	31 51	
	GTO 9	22 09	integer short cut
	STO 1	33 01	
0	LBL 0	31 25 00	
	RCL E	34 15	"Base"
	RCL 1	34 01	
	x≠y	35 52	
	÷	81	
	FRAC	32 83	
050	x≠0?	31 61	
	GTO 1	22 01	Numerator is not multiple of Base
	RCL 2	34 02	
	RCL E	34 15	
	÷	81	
	FRAC	32 83	
	x≠0?	31 61	

STEP	KEY ENTRY	KEY CODE	COMMENTS
	GTO 1	22 01	Denom. is not mult. of Base
	RCL E	34 15	
	STO + 1	33 81 01	Reduce num. & denom.
060	STO + 2	33 81 02	
	GTO 0	22 00	Try again
9	LBL 9	31 25 09	
	RCL A	34 11	Integer short cut
	GTO 3	22 03	
1	LBL 1	31 25 01	
	RCL A	34 11	Output in nominal
	INT	31 83	format
	RCL 1	34 01	
	EEX	43	
070	3	03	
	÷	81	numerator
	+	61	
	RCL 2	34 02	
	EEX	43	
	6	06	
	÷	81	denominator
	+	61	
	GTO 3	22 03	
C	LBL C	31 25 13	$F_1 + F_2$
080	STO B	33 12	
	x≠y	35 52	
	A	31 22 11	F_2 → decimal
	STO C	33 13	
	RCL B	34 12	
	A	71 22 11	F_1 → dec
	RCL C	34 13	
	x≠y	35 52	
	F2?	35 71 02	
	CHS	42	Subtract
090	+	61	$F_1 \pm F_2$
	GTO B	22 12	dec → frac
D	LBL D	31 25 14	Subtract
	SF2	31 51 02	
	GTO C	22 13	
a	LBL a	32 25 11	Find numerator and denominator
	x<0	31 71	
	SF0	35 51 00	SF0 if x<0
	ABS	35 64	
	STO A	33 11	
100	FRAC	32 83	
	x≠0?	31 61	
	GTO 4	22 04	
	F1?	35 71 01	
	RTN	35 22	return if "mixed" mode
	LST x	35 82	
	ENT↑	41	Move fraction
	LOG	51 53	to follow decimal
	INT	31 83	point
	1	01	
110	+	61	
	10^x	32 53	
	÷	91	

	KEY ENTRY	KEY CODE	COMMENTS
	STO A	33 11	
4	LBL 4	31 25 04	
	6	06	
	ST x	35 33	
	RCL A	34 11	
	EEX	43	
	6	06	
120	x	71	
	ABS	35 64	Put x in LST x
8	LBL 8	31 25 08	
	LST x	35 82	
	1	01	
	0	00	
	÷	81	
	FRAC	32 83	
	x≠0?	31 61	Look for non-zero digit
	GTO 5	22 05	
130	DSZ	31 33	
	GTO 8	22 08	
5	LBL 5	31 25 05	Put num. "position"
	RCL 3	35 34	in R_3, denom.
	2	02	"position" in R_0
	÷	81	
	ST I	35 33	
	INT	31 83	
	10^x	32 53	
	STO 3	33 03	
140	RCL	35 34	
	.	83	
	5	05	
	+	61	
	INT	31 83	
	10^x	32 53	
	STO 4	33 04	
	RTN	35 22	
E	LBL E	31 25 15	Store min. frac.
	STO D	33 14	
150	RCL E	34 15	
	x≠0?	31 61	if $R_E = 0$, set
	RTN	35 22	default $R_E = 2$
	2	02	
	STO E	33 15	
	RCL D	34 14	
	RTN	35 22	
e	LBL e	32 25 15	Store Base
	STO E	33 15	
	RTN	35 22	
160 7	LBL 7	31 25 07	Set default parameters
	CLX	44	
	2	02	
	STO E	33 15	
	CLX	44	
	6	06	
	4	04	
	STO D	33 14	
	RTN	35 22	

STEP	KEY ENTRY	KEY CODE	COMMENTS
b	LBL b	32 25 12	Select mode
170	DSP 0	25 00	
	F1?	35 71 01	
	GTO b	22 34 12	
	SF1	35 51 01	
	0	00	0 for "mixed"
	RTN	35 22	
b'	LBL b	32 25 12	
	CF1	35 61 01	
	1	01	1 for "all frac"
	RTN	35 22	
180 c	LBL c	32 25 13	$F_1 \div F_2$
	STO B	33 12	
	x≠y	35 52	
	A	31 22 11	F_1 → decimal
	STO C	33 13	
	RCL B	34 12	
	A	31 22 11	F_2 → decimal
	RCL C	34 13	
	x≠y	35 52	
	F2?	35 71 02	
190	1/x	35 62	Divide if F_2
	x	71	
	GTO B	22 12	$F_1 \frac{x}{÷} F_2$
d	LBL d	32 25 14	Divide
	SF2	35 51 02	
	GTO c	22 51 15	
200			
210			
220			

REGISTERS

0	1 Numerator	2 Denominator	3 Num. Location	4 Denom. Location	5	6	7	8	9
S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
A x	B F_2	C F_1	D Minimum Fraction	E Base	I used				

LABELS

A 001 → Rcl	B 027 → Frac	C 079 +	D 092 −	E 105 Min frac
a 695 find num + denom	b 169, 176 mode switch	c 180 ×	d 198 ÷	e 157 Base
0 44	1 65	2	3 016	4 118
5 132	6 24	7 100	8 122	9 082

FLAGS

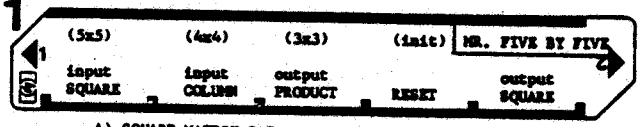
0	x<0
1	mode
2	user
3	

SET STATUS

FLAGS		TRIG	DISP
	ON OFF		
0	☐ ☒	DEG ☐	FIX ☒
1	☒ ☐	GRAD ☐	SCI ☐
2	☐ ☒	RAD ☐	ENG ☐
3	☐ ☒		n 0

1. MR FIVE BY FIVE — BY: Hal Brown (362)
2. A matrix manipulation facility. — BY: and Earle Robinson (1153)

1



67

A) SQUARE MATRIX DATA INPUT

a) If from data card -- read card.
b) Using data input routine
1. Clear registers -- key d (f,D)
2. Enter values in left-to-right, row by row order. Key "A" after each. If a 4x4 matrix enter a zero at end of each row; if a 3x3 enter two zeros at end of rows.
3. If desired, data can be stored directly with STO N's where N represents register numbers arranged as follows:

5x5 or 4x4	0	1	2	3	4	primaries
	5	6	7	8	9	
3x3	0	1	2	3	4	secondaries
	5	6	7	8	9	
	A	B	C	D	E	

INSTRUCTIONS
See text on page 6.

B) SQUARE MATRIX DATA OUTPUT (Use to output inverse after a D & I run, or for a check on data after it is input)
1. Reset counter -- key D
2. Key E successively to display matrix elements in same order entered (see Ab2 above). If a 4x4 case ignore one value at end of each row; if a 3x3 ignore two values.

C) COMPUTE DETERMINANT AND INVERSE
1. Input data -- see instruction A
2. For 5x5 key a (f,A), for 4x4 key b (f,B), for 3x3 key c (f,C).
3. Optional. If end-of-run signal is desired, while program is running enter signal card in slot for automatic read.
4. In most cases the run will stop in about 3 min. 40 sec. (signaled by sound of card read if signal option is used). The determinant is displayed and inverse elements are stored in registers.
5. If the matrix input has zero determinant the inverse does not exist (by definition). This is signified by repeating long-pause displays of one diagonal position number (an integer from 0 through 4). Stop run by depressing any key.*
6. In rare cases the matrix cannot be handled by the program as entered. This is indicated by repeating long-pause displays of the same sequence of more than one diagonal position number. Stop run by depressing any key*, reformat the matrix (try exchanging one or more rows or columns) and reenter for another run.

*If the signal option is used, remove the signal card before halting the run in step 5 or 6.

D) MULTIPLY SQUARE MATRIX BY COLUMN MATRIX
1. Put square matrix in registers, either via Instruction A or as a result of a Determinant and Inverse run.
2. Reset counter -- key D
3. Enter column matrix values one at a time in order from the top -- key B after each.
4. Reset counter -- key D.
5. Output elements of product column matrix one at a time with successive keyings of C (Steps 4 & 5 can be repeated to review values).

EXAMPLE USE OF PROGRAM: SOLUTION OF SIMULTANEOUS LINEAR EQUATIONS

Problem: solve the following equations simultaneously

$$3v + w - 5x + y - z = -15$$
$$-2v + 3w + x - 2y + 3z = 20$$
$$v - 3w + x + y - 2z = -12$$
$$4v + w - 5x - 2y + z = -12$$
$$2v + 2w + 3x - 4y + 3z = 15$$

$$\begin{matrix} 3 & 1 & -5 & 1 & -1 \\ -2 & 3 & 1 & -2 & 3 \\ 1 & -3 & 1 & 1 & -2 \\ 4 & 1 & -5 & -2 & 1 \\ 2 & 2 & 3 & -4 & 3 \end{matrix}$$

1. Use Instruction A,b to enter the matrix: _____
2. Optional. Use Instruction B to check stored data.
3. Run D & I (key f,A for 5x5). At run-stop read determinant, 14.000 (not used in simultaneous equation solution). Matrix inverse is now stored in registers (Can be read out if desired, via Instruction B. If this is done key D to reset counter before proceeding with next step).
4. Enter column matrix (values to right of equal signs) -- keying B after each.
5. Reset counter -- key D
6. Output values of unknowns one at a time with successive keyings of C: v = -2, w = 3, x = 2, y = -1 and z = 1.

Set Display: FIX, DSP 3, Angle Mode: DEG

STEP	KEY ENTRY 1	KEY CODE
1	LBL 6	31 25 06
	GSB 5	31 22 05
	LBL 5	31 25 05
	GSB 7	31 22 07
5	LBL 4	31 25 04
	GSB 7	31 22 07
	LBL 3	31 25 03
	GSB 7	31 22 07
	LBL 2	31 25 02
10	GSB 7	31 22 07
	ISZ	31 34
	RTN	35 22
	LBL 7	31 25 07
	R↓	35 53
15	STO x i	33 71 24
	RCL I	35 34
	5	05
	/	81
	INT	31 83
20	R↓	35 54
	FRAC	32 83
	+	61
	5	05
	x	71
25	↓	41
	X - I	35 24
	-	51
	R↑	35 54
	LST X	35 82
30	R↓	35 53
	-	51
	RCL i	34 24
	X - Y	35 52
	STO I	35 33
35	CLX	44
	RCL i	34 24
	R↑	35 54
	STO I	35 33
	CLX	44
40	LST X	35 82
	R↓	35 53
	x	71
	STO + i	33 61 24
	R↓	35 53
45	X - Y	35 52
	X - I	35 24
	RCL i	34 24
	X - Y	35 52
	STO I	35 33
50	R↓	35 53
	STO / i	33 81 24
	ISZ	31 34
	RTN	35 22
	LBL e	32 25 15
55	6	06
	.	83
	2	02
	X - Y	35 52
	x	71
60	STO I	35 33
	RCL i	34 24
	X = 0	31 51
	GTO 6	22 06
	R↑	35 54
65	CHS	42
	STO i	33 24
	CLX	44
	LST X	35 82
	STO I	35 33
70	CLX	44
	1	01
	CHS	42
	GSB 7	31 22 07
	CLX	44
75	STO I	35 33
	RTN	35 22
	LBL B	31 25 12
	GSB 7	31 22 07
	X - I	35 24
80	2	02
	4	04
	-	51
	X - I	35 24
	RTN	35 22
85	LBL 7	31 25 07
	GSB 5	31 22 05
	GSB 5	31 22 05
	GSB 5	31 22 05
	GSB 5	31 22 05
90	LBL 5	31 25 05
	STO x i	33 71 24
	LBL 9	31 25 09
	X - I	35 24
	5	05
95	+	61
	X - I	35 24
	RTN	35 22
	LBL 6	31 25 06
	SF 3	35 51 03
100	R↑	35 54

STEP	KEY ENTRY 2	KEY CODE
1	LST X	35 82
	STO I	35 33
	- X -	31 84
	X = 0	31 51
5	R↓	35 53
	GTO i	22 24
	LBL A	31 25 11
	STO i	33 24
	LBL E	31 25 15
10	RCL i	34 24
	ISZ	31 34
	RTN	35 22
	LBL C	31 25 13
	RCL i	34 24
15	ISZ	31 34
	GSB 6	31 22 06
	GSB 6	31 22 06
	GSB 6	31 22 06
	LBL 6	31 25 06
20	RCL i	34 24
	+	61
	ISZ	31 34
	RTN	35 22
	LBL d	32 25 14
25	CL RG	31 43
	P - S	31 42
	CL RG	31 43
	LBL D	31 25 14
	DEG	35 41
30	0	00
	STO I	35 33
	SF 1	35 51 01
	RTN	35 22
	LBL a	32 25 11
35	RAD	35 42
	LBL b	32 25 12
	CF 1	35 61 01
	LBL c	32 25 13
	CF 0	35 61 00
40	CF 2	35 61 02
	CF 3	35 61 03
	DSP 0	23 00
	1	01
	LBL 0	31 25 00
45	4	04
	COS	31 63
	X ≥ 0	31 81
	GTO 4	22 04
	LST X	35 82
50	e	32 22 15
	GSB 6	31 22 06
	GSB 6	31 22 06
	DEG	35 41
	LBL 4	31 25 04
55	F? 1	35 71 01
	GTO 3	22 03
	3	03
	e	32 22 15
	GSB 4	31 22 04
60	GSB 6	31 22 06
	GSB 7	31 22 07
	GSB 9	31 22 09
	GSB 4	31 22 04
	GSB 7	31 22 07
65	SF 1	35 51 01
	LBL 3	31 25 03
	F? 2	35 71 02
	GTO 0	22 00
	2	02
70	e	32 22 15
	GSB 3	31 22 03
	GSB 5	31 22 05
	GSB 3	31 22 03
	DSZ	31 33
75	GSB 9	31 22 09
	GSB 3	31 22 03
	GSB 5	31 22 05
	GSB 3	31 22 03
	LBL 0	31 25 00
80	SF 2	35 51 02
	LBL 2	31 25 02
	F? 0	35 71 00
	GTO 1	22 01
	1	01
85	e	32 22 15
	GSB 2	31 22 02
	GSB 4	31 22 04
	DSZ	31 33
	GSB 9	31 22 09
90	GSB 2	31 22 02
	GSB 6	31 22 06
	GSB 4	31 22 04
	SF 0	35 51 00
	LBL 1	31 25 01
95	π	35 73
	FRAC	32 83
	RND	31 24
	X ≠ 0	31 61
	GTO 1	22 01
100	e	32 22 15

201	GSB 9	31 22 09
	ISZ	31 34
	GSB 6	31 22 06
	GSB 6	31 22 06
205	DSP 3	23 03
	↓	41
	LBL 1	31 25 01
	2	02
	4	04
210	STO I	35 33
	R↓	35 54

	F? 3	35 71 03
	GTO 0	22 00
	LBL 8	31 25 08
215	STO / i	33 81 24
	DSZ	31 33
	GTO 8	22 08
	STO / 0	33 81 00
	MERGE	32 41
220	PAUSE	35 72
	R/S	84

PPC CUSTOM ROM HOUSEKEEPING ROUTINES

A major contribution the PPC ROM makes to calculator functions is the "block" concept of operating on a block of registers. The "block" concept is not entirely new. The HP-67 REG instruction displayed the machines registers in sequence from lowest to highest. The ROM routine Block View, performs a similar function in the "PPC Block Mode" by displaying or printing non-zero registers.

"PPC BLOCK MODE"

Before I describe the many block operations possible with the PPC ROM let me define a "block". A Block of registers is a contiguous 'run' of registers usually defined by bbb.eee. Where the 'b' value is the first register of the block and the 'e' is the register number of the end of the block. The block operations were designed -in the time allowed- to allow the "full power of ISG". The complete block definition is bbb.eeeii of the same format as the ISG (or DSG) instruction.

An example of using the block view, BV, routine mentioned above would be to list the even numbered non-zero registers from R20-R43. The RAM program would look like this:

```
20.04302
XEQ ▼ BV
```

If R20 contained a non-zero value it would be printed. An example is shown below. The first column shows some test data. The second column shows the same data except R28 and R36 were cleared. An added feature of this routine is a fast, short, tic that is heard each time a register is analyzed. The third column uses a block instruction of 20.04303 in fix 4 mode on the same data as the second column.

```
20: 20.02
22: 2,222.00
24: 2,424.00
26: 2,626.00
28: 2,828.00
30: 3,030.00
32: 3,232.00
34: 3,434.00
36: 3,636.00
38: 3,838.00
40: 4,040.00
42: 4,242.00
```

```
20: 20.02
22: 2,222.00
24: 2,424.00
26: 2,626.00
30: 3,030.00
32: 3,232.00
34: 3,434.00
38: 3,838.00
40: 4,040.00
42: 4,242.00
```

```
20: 20.0200
23: 2,323.0000
26: 2,626.0000
29: 2,929.0000
32: 3,232.0000
35: 3,535.0000
38: 3,838.0000
41: 4,141.0000
```

FIGURE 1. Examples of BV routine executed to illustrate the bbb.eeeii Block concept of viewing or printing a block of registers.

The Block routines perform such operations as sort, randomize, sum, view, move, rotate, increment, and exchange, to name a few.

These are shown in Table 1.

TABLE 1.- Major Block Routines

Block Clear, BC	Block Rotate[1], BR	Small Array Sort[1], S2
Block Exchange, BE	Block View, BV	Large Array Sort[1], S3
Block Increment, BI	Block Extremes, BX	Selection without replacement,[2] SE
Block Move[1], BM	Block Statistics, BΣ	

1 - These routines do not utilize the full power of ISG/DSE. S2 and S3 are not fully tested.

2 - This is a label change, previous published label was SW.

The capabilities of the PPC Block Routines are impressive, but not all inclusive of the types of operations possible. A block plus was ommitted because it could be implemented using BΣ. The reader could add several other block operations to the list. A brief minimal operating instructions for the routines of Table 1 follows.

BC - Block Clear bbb.eeeii in X, XEQ BC, stores 0's in the defined block.

BE - Block Exchange The smallest block bbb.eeeii is placed in Y, the largest, bbb.eee in X and the two blocks are exchanged first register of block 1 with first of block 2, second of block 1 with second of block 2, etc. The HP-67/97 P⇄S is obtained with .009 ↑ 10 XEQ BE.

BI - Block Increment This powerful routine is useful for creating data bases and testing data management programs. Data input is: block definition, ENTER, Start value, ENTER, Increment value (±), XEQ BI.

Note: 1) If the increment value is zero a block fill function is implemented.
2) If the start value is zero a block clear function is implemented.

BM - Block Move Input is: First register of source block, ENTER, first register of destination block, ENTER, number of registers to move, XEQ BM.

BR - Block Rotate Rotates a block of registers up +, or down −, one register, i.e., shift right, or left. Input is not conventional: First register of block, ENTER ± number of registers in block, XEQ BR.

BV - Block View Block is viewed with bbb.eeeii in X, XEQ BV. Only non-zero registers are viewed. Format RNNN: NNNNNNN, set flag 10 to stop at each register - does not stop on zero values. To pause at each register set flag 9.

BX - Block Extremes Define block with bbb.eeeii, XEQ BX. If flag 10 is set the absolute values are processed. Output is:
M = Address of Maximum value
N = Address of Minimum value
X = Minimum value
Y - Maximum value

BΣ - Block Statistics runs faster than you would think. Two blocks are used as X, Y inputs for Σ+. The "Y" values are defined as bbb.eee in Y, the X values as bbb.eeeii in X. The ii portion of X is used for loop control. ΣREG must be defined by the user.

S2,S3 - Sort Arrays S2 is for arrays of registers less than 32, S3 for others. NO ALPHA. Sorts lowest value to highest compared to E99.
S2 - Use RØØ and up. Input bbb.eee, XEQ S2
S3 - Use RØ3 and up. Input bbb.eee, XEQ S3

NOTE:
1. Data arrays with 33 or more equal values won't sort properly. S3 uses S2 which operates on 32 registers at a time.
2. DO NOT STOP S3 while sorting.
3. Test: 3.103↑103↑ -1, XEQ BI
3.103, XEQ S3
Sorts 100 registers in less than 3 minutes
4. Order of array, sorted, inverse order, random, won't affect sort time by more than ≈ 5-15%.

SE - Selection Without Replacement This unique routine is simple (only after it's finished!), short, and very useful. RØ6 is register number of first register of 'block'. RØ7 is number of consecutive registers in block. Pre-load these registers! XEQ SE with RN pointer in X.

Notes:
1. Each time SE is called, RØ7 contains items available for selection. User must test for zero, or lower limit.
2. Elements that have been drawn are separated from those yet to be drawn. All elements are saved.
3. Cycle RØ7 = N to RØ7 = Ø once to randomize a block of data.

The block group of ROM instructions may illustrated by an example of their use.

A deck of 52 cards are used for a game. For convenience of study the cards are identified 1 thru 52.

1. Clear data space for deck. Place deck starting at R11. 11.062,XEQ BC

2. Create deck in order.
11.062, ENTER, 1, ENTER .1, XEQ BI.

2. View Deck. 11.062, XEQ BV

3. Verify statistics of deck. ΣREG, 04; 11.062, ENTER, XEQ BΣ
   XEQ MEAN gives 26.5; XEQ SDEV gives 15.15

5. Place top card on "bottom" of deck. 11 ENTER 52, XEQ BR.

6. Shuffle Deck. PI, STO 00; 11 STO, 06; 52 STO 07.
   Execute the following loop to complete the shuffle.

```
LBL S
CLX          RN pointer value "zero" (RN scratch, seed)
XEQ SE       Routine selects a card
RCL 07       Check number of items (cards) left.
x ≠ 0?
GTO S        Repeat if items remain
STOP
```

7. View deck to check. 11.062, XEQ BV

8. Check for highest and lowest values. 11.062, XEQ BX

9. Sort deck back to 1 to 51 order. 11.062, XEQ S3

10. View deck as in 7. above.

11. Exchange part of the deck, R00 with R11, etc.
    0.01, ENTER, 11.021, XEQ BE

12. Move remaining part of deck down. 22 ENTER, 11, ENTER, 41, XEQ BM

The example above exercised the ROM Block routines using a deck of cards. A very powerful routine, SE, will be discussed in detail if time permits.

The block concept is only simply illustrated by the PPC ROM routines. I believe that this concept could be fine tuned and become standard instructions for future PPC's. There is considerable room for improvement in developing a uniform block definition format. While there are conceptual difficulties all block routines could be implemented with the full power of ISG/DSE. The block move offers an excellent challenge in this reguard.

The Block routines are only a small part of the PPC ROM. They were selected to illustrate some of the new PPC concepts that PPC members have contributed to their ROM.

Richard Nelson

PPC ROM Peripheral Routines

Many HP41C users who have ordered the club ROM posess the HP82143A printer and/ or the 82153A wand as well. For this reason, a small section of the ROM has been allocated for routines and programs which enhance the power of the printer and provide diagnostic aid in the use of the wand.

The routines are as follows:

1. CP Column Print Formatting
2. HS High Resolution Histogram
3. HA Histogram with Axis
4. LG Print PPC Logo
5. MP Multiple Function Low Res. Plotting
6. HP Multiple Function High Res. Plotting
7. BA Barcode Analyzer

Briefly describing each routine: CP formats printed output in columnar fashion, allowing mixed numeric and ALPHA columns of information across the entire 24 position printed line. While ALPHA columns are 'ACA'ed' by the user's RAM program, CP places the numeric columns in the print buffer so that decimal points remain in fixed positions. If overflow to SCI or ENG occurs, then the numeric entries are managed in order to maintain right justification. The routine behaves correctly regardless of the state of flag 29 (presence or absence of numeric commas).

Routine HS generates high resolution (up to 168 column) bar charts from scaled inputs between 0 and 1. The user may choose which standard printer character is to be used to fill the bars. In addition, the partial character is also specified by the user. Bars are merely loaded into the print buffer, not PRBUF'ed, so other information may also be added to printouts if desired.

HA draws histogram bar charts similar to HS, however, this routine is fully compatible with the printer's internal routine REGPLOT. Not only does the user specify the fill character, but he may also position an axis in any column across the printer paper. If the value to be plotted is larger than the axis value, then the histogram bar is drawn up from the axis to the value. If the value falls below the axis, then the bar is drawn up from the value to the axis. Plot values do not have to be scaled.

The LG routine adds a PPC logo to the current contents of the print buffer. The logo occupies 21 buffer positions, leaving 22 more for additional information. The utilization of synthetic text lines in this routine saves a significant number of ROM bytes over the more traditional ACCOL or BLDSPEC techniques of creating user-defined symbols.

Routines MP and HP both operate from the same 600+ byte routine which allows the printer to plot simultaneous multiple functions in either low or high resolution. Low resolution is similar to PRPLOT, using one plot point per printed line, and high resolution works like the old PLOTHR in PPC CJ, V7N1, with 7 plotted points per printed line. Up to 9 functions may be plotted in low resolution and up to 4 may be done in high resolution. These limits are primarily based on the size of the print buffer.

Four different symbols are used in HP and 7 predefined and 2 user defined symbols are used in MP, with order and frequency of choice completely controlled by the user. The standard header information and Y-axis may either be used or alternately supressed or replaced by RAM-produced versions. Many more features are available in this program, which hopefully will be discussed in the ROM presentation.

BA is a program which analyzes any type of HP41C barcode, a row at a time. After XEQ BA, the machine prompts for a barcode row to be scanned. The following information is then printed: Barcode type (0-15) and the type name, if applicable; byte-by-byte contents in decimal, hex and binary; the equivalent printer character for each barcode byte; and finally the computed 8-bit checksum if the barcode was not of the paper keyboard variety(<= 3 bytes long).

This software is merely a cross section of the vast amount of material related to peripherals which has been generated by PPC members in the past 18-20 months. There is indeed enough to support a separate single- or dual-density ROM exclusively devoted to Peripheral Routines alone. Perhaps when new external devices for the 41C become available, such a project will be justified.

Jake Schwartz (1820)

N O T E S

Multiple function plotting
low res / high res.
(MP/HP)
SIZE 035, 040

USER FUNCTIONS MUST LEAVE IN FIX 0

X = # functions plotted
R_{00} = Ymin
R_{01} = Ymax
R_{02} = Plot width (columns)
R_{04} = Name of Y-axis plotting routine in RAM, if used
R_{08} = Xmin
R_{09} = Xmax
R_{10} = Xincrement
R_{12} = Symbol/Function map
R_{15} = FCN #1 MP or HP
R_{16} = FCN #2 MP or HP
R_{17} = FCN #3 MP or HP
R_{18} = FCN #4 MP or HP
R_{19} = FCN #5 MP
R_{20} = FCN #6 MP
R_{21} = FCN #7 MP
R_{22} = FCN #8 MP
R_{23} = FCN #9 MP
R_{33} = USER ACCOL SYMBOL #8 if USED → ".000" to ".127" (MP only)
R_{34} = USER ACCOL SYMBOL #9 if USED ".000" to ".127" (MP only)

HP :	1 to 4 functions
MP :	1 to 9 functions

F10 = set if MP value plotting like REGPLOT

F09 = set if RAM Y axis routine is in R_{04}

F08 = internally used

F07 = set if user wishes to skip header

F06:	clear	set	set	clear
F05:	clear	clear	set	set
	plots clipped at edge	points disappear at edge	mirror plotting	wraparound plotting

F04 = set if remapping functions to symbols via R_{12}.

Special Instructions:

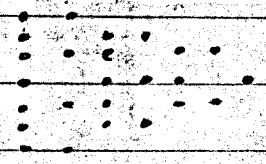
If F04 clear, R_{12} = .123456789. To reverse symbol usage for 4 functions, for example, SF04 and STORE .4321 in R_{12}. To use symbol #3 exclusively for, say, 5 functions, SF04, 3.33333 STO 12; etc.

To use symbols #8 & #9, store $\frac{\text{ACCOL \#}}{1000}$ (up to .127) in R_{33}, R_{34} and set R_{12} accordingly (SF04)

To do value plotting, store values in R_{15}-R_{23} and SF 10 and XEQ MP.

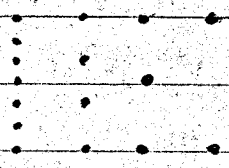
To plot X-axes as well as functions, mix axis values with function names in R_{15}-R_{23}. Program interprets values as constants. Ex: plot SINE, COSINE and axes using 127 ACCOL at -1, 0, +1: R_{15}="SINE", R_{16}="COSINE", R_{17}=-1.0, R_{18}=0.0, R_{19}=1.0, SF04, R_{12}=.12111, 5 XEQ "MP".

Standard Symbols for MP:



1 2 3 4 5 6 7 plus 2 user defined ACCOL symbols

Standard Symbols for HP:



totalling 16 buffer positions.
Any combo may be chosen so buffer is not exceeded (18 dots is safe max)

1 2 3 4

To add X axis labelling, reduce plot width by (# label chars x 7) columns and ACX the X label in the first function to be plotted before returning X value. Ex: 3-digit labeling = 168-21=147 col. width in R_{02}, first fcn:

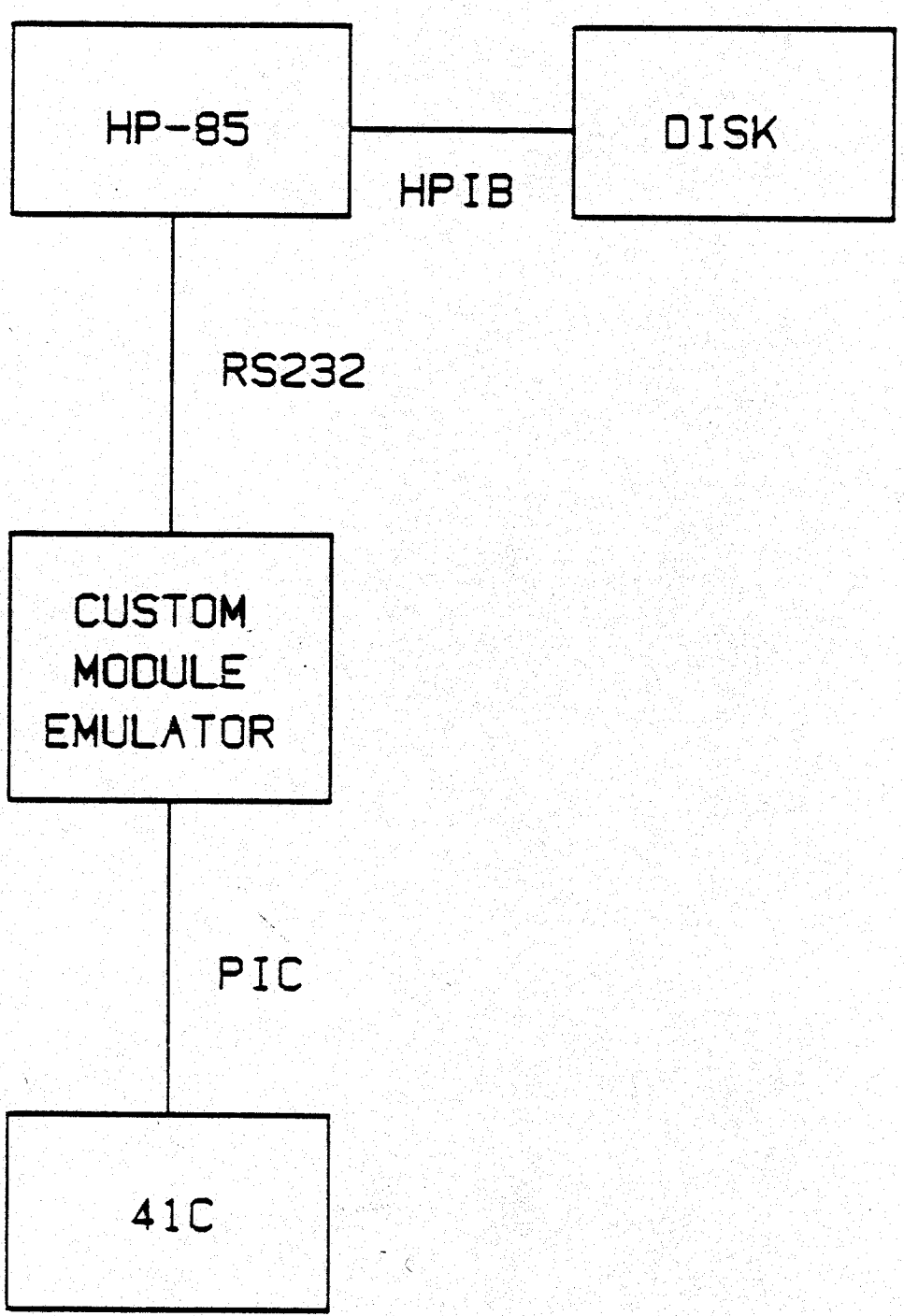
```
LBL __
FIX 0
RND
ACX
 ⋮
RTN
```

HP Custom Services

- PLAN, DEVELOP, DEBUG 41C SOFTWARE
- RENT SDS - BUILD AND DEBUG ROM
- SEND ROM IMAGE TO HP
- 20 WEEKS TO MAKE MODULES

PR: 3/25/81

Hardware Components of SDS



PR: 3/25/81

Software Functions of SDS

- UPLOAD 41C PROGRAMS

- DESCRIBE ROM IMAGE(S)

- BUILD ROM IMAGE(S)

- DOWNLOAD IMAGE(S) TO CUSTOM MODULE EMULATOR

PR: 3/25/81

41C Microcode Capability

- EXECUTABLE FROM 41C USER LANGUAGE PROGRAMS AND KEYBOARD

- EXECUTABLE ONLY VIA OTHER MICROCODE ROUTINES

- EXECUTABLE AT CERTAIN TIMES DURING 41C OPERATION

- HP SUPPLIED

PR: 3/25/81

The so called Hand Held Computer, HHC, as represented by the Sharp PC-1211 offers the User a wide range of BASIC TM language instructions and increased accuracy math functions. This shirt pocket computer seems capable of eventually replacing the top of the line PPC's such as the HP-41C and future TI-88.

Active members of the PPC User Community view the Sharp machine and others soon to be available from other Japanese manufacturers, as a strong indicator that the HHC could obsolete the PPC as we know it. The Japanese have recognized the special historical demands and expectations of calculator users and have chosen to 'leapfrog' the top of the line PPC with the HHC. BASIC TM doesn't require the support and attention to detail that AOS TM or RPN calculators currently provide.

Should the User community expect U.S. Manufacturers to continue developement of PPC's or will the next generation machines be computers rather than calculators. This panel will address this question.

Panel Leader and keeper of the peace

Bill Kolb

Panel Members pro-calculator

Maurice Swinnen, Editor TI-PPC Notes
Richard Nelson, Editor PPC Calculator Journal
Frank Blachly, TI-59, HP-41C User

Panel Member pro-computer

Jake Schwartz, HP-41C User, column editor.

The goals of this discussion are several:

a. Express the differences (perceived by the current calculator user community) between calculator and computer.

b. Survey the conference attendees as to their feelings on what they think they need at this time.

c. Compile the above findings into a formal document that will be made available to interested parties.

d. Express a future goal for manufacturers to be sensitive to when examining future machine designs. Would we buy a next generation calculator if made available?

e. Determine how many calculator users are now computer users.

The Handheld Computer's Time is Now

In the past few years, over a half million personal computers were purchased for use in the home and on the job. Both business and scientific applications software have been made available at an increasing rate to fill the large need in industry; and likewise home management, financial, educational and game/entertainment software has pervaded the home marketplace. The demands on the personal computer, as we know it, are increasing for both data processing and alphanumeric information processing. At the same time, techology is bringing us more things in our small desktop units , and existing computational capabilities in smaller packages.

Advancement from the rum mode pocket calculators and language translators of the 70's to the sophisticated PPC's, pocket memopads and early handheld computers of the 80's indicates a potential trend toward full fledged general purpose computers which are briefcase sized and DC powered. Full QWERTY keyboards are a must, but key spacing will decrease somewhat, and one-key-one-function philosophies should become common. With the increasing availability of 64Kbit Random Access Memory chips, soon in CMOS an well as in NMOS, eight chip 64Kbyte pseudononvolatile memories would become trivial.

Interchangable ROM operating systems, much like those available in some of the desktop machines of today, will permit multiple higher level language capability as well as assembly and a 'PPC mode', which would allow pocket calculator emulation with microcomputer speed. Imagine running BASIC, then switching a mode switch to 'PPC', where one now has a lightning fast HP41C-like language with full programmability and 64Kbytes of RAM to share between programs and data.

It is known that at least a half dozen companies are currently designing flat, dot matrix displays using either liquid crystals or electroluminescent pixels, and another company is about to offer a flat CRT. It is not out of the question to envision a device whose keyboard closes under a lid whose inside surface is a flat display, with perhaps a 15 line by 50 character capability, including full graphics.

Interfacing is a definite necessity, and a multitude of interface protocols should be handled by the portable computer. It is even possible to consider wireless, infrared transmission of information from main unit to nearby peripherals, as well as standard communication techniques for other devices not within an arm's grasp. This computer would be ideal as a remote data gatherer, which could later comunicate to a larger number cruncher back at a centrally located office. Terminal emulation is another possibility along these lines.

For simple mathematical tasks, the pocket programmable calculator is perfectly adequate, but many of our needs exceed this limited realm. This is exemplified by the sophisticated programs in the PPC Club ROM, which will greatly enhance the HP41C capability, but will not deliver answers at microcomputer speed. For those who use micro's daily, it should be apparent how much more convenient it would be if this much power could be available in a carry along package.

Jake Schwartz (1820)

Good Afternoon, Ladies and gentlemen. I am Maurice Swinnen. I am the founder and present publisher and editor of the TI PPC NOTES. We feel that we are in direct competition with the highly sanitized newsletter edited by TI itself, the PPX newsletter, although that one has improved lately, probably do to the pressure exerted by us. Good old free enterprise at its best.

I am in good standing with TI though, witness I have a contract with them to teach two-day and five-day seminars on TI-59 programming. But as the editor of the TI PPC NOTES I am completely independent from TI.

The point that I would like to make this afternoon is as follows:

If we will have the next generation hand-held programmable, be it HP, TI or Japanese made, it should have the following capabilities:

2. High level language capability, preferrably some form of BASIC, capable of accessing the calculator portion and the modules.
1. Calculator language, capable of accessing the modules and using parts of them as subroutines in a program in user-memory.
3. Constant-memory user-programmable modules. No card reader needed but nice to have in addition to the software modules.

We have already seen the trend towards BASIC in hand-held programmables: TRS-80 and Sharp. But BASIC is not as easy to use when doing computation in comparison to using a true calculator language. On the other hand, BASIC is ideal for control functions, accessing RS 232 I/O ports, etc, while a true calculator language does not lend itself so easily to these functions.

We have already a good example of a calculator that has also BASIC capabilties, the HP 9825. But, unfortunately, it is not portable or hand-held. And, by the way, it is my favored where I work, because it uses the AOS system as opposed to RPN. They cleverly disguised the equal key, though, by marking it "execute." It is a marvelous machine, lots of memory available, built in casette record-playback, built-in thermal printer, plug-in ROMS, I/O ports for external control. Its price, however, is not within the reach of a not too well heeled amateur like me.

Another, more modest approach of a calculator-computer combination is the TI-99/4 home computer. When it powers up it gives you the choice of selecting either BASIC, plug-in ROM or Equation calculator. The latter is rather easy to use, but has it limitations: You enter the equation as you would write it down, with (forgive me the dirty word) parenthesis where needed to circumvent the normal rules of the AOS system. You next enter the value of each variable and press the ENTER key. The result is flashed on the screen. But if I wanted to change one of the variables by a small amount "delta" I would have to add it through the keyboard, after which I will get a new result. It is, however, not possible, as with my TI-59, to do that operation automatically and obtain a series of results, so that I may observe a certain trend in them.

For that it should be possible for the BASIC user-program to access the equation calculator. But the designers did not include that capability. Nor did they include the capability of the BASIC program to access the plug-in ROMs.

That is exactly what I would like to see in a future hand-held: BASIC program, capable of accessing all of the calculator functions and the plug-in ROMs. Calculator capable of accessing the plug-in ROMs, as is now done in the HP-41C and the TI-59.

An added feature I would like to see (in Spanish they say SONAR NO CUESTA NADA, or dreams are cheap) is a couple of RS-232 I/O ports, in order to be able to talk to the world: telephone modems, line printers, speech synthesizers, disk or casette recorders, thermal printers, x-y plotters.

This capability is no idle gimmick, though. I know of surveyors who would give their right arm if they had a hand-held that enabled them to collect data in the field, then go to the nearest telephone booth and transmit the data to the home office. In lots of less developed countries there is a real need for an inexpensive data terminal. A hand-held programmable would fill the bill rather nicely, if it were equipped with I/O ports.

One of the drawbacks to hand-held computers is the small one-line display. But we see already larger plasma displays being developed which are capable of displaying several lines simultaneously. In the very near future these displays will be available as TV screens, graphic displays. Toshiba recently demonstrated an experimental TV with a screen size of 3.1 by 4.1 cm. liquid-crystal screen.

So, to sum it all up: HP, TI, or any other aspirant manufacturer, please do not try to sell us a duplicate of the capabilities we already possess in our hand-held programmable calculators by just adding a few gimmicks to the newer model. Nor give us a hand-held purely Basic machine. My home computer will beat it by a mile.

But a combination machine, where one capabilty accesses the other will sell for sure, because it will open a host of new possibilities. Add a few I/O ports to it and I will respond by dipping in my pocket book.

I thank you for your attention.

PPC EAST COAST CONFERENCE SCHEDULE

ROCKVILLE, MARYLAND

MARCH 28, 1981

8:30	AM	Registration	SESSION
9:00	AM	Introductions	
9:15	AM	Human Factors in Programming - Tony Vertuno (4089)	#1
10:30	AM	The Unit Management System used in HP's PAC's - Eric Vogel, Hewlett-Packard Application Engineer	#2
11:30	AM	State of the Art in Software - Bill Kolb (265) and Bill Wickes (3735)	#3
12:30	PM	LUNCH	
2:00	PM	PPC ROM Overview and Demonstration - Richard Nelson (1) and Jack Schwartz (1820)	#4
3:00	PM	Hewlett-Packard Custom Services (SDS), Pam Raby, CVD Lab.	#5
4:00	PM	Panel Discussion addressing the question: "Do we need a PPC beyond the TI-88 and HP-41C?"	#6
		Panel members are: Maurice Swinnen, Editor TI-PPC Notes Richard Nelson, Editor PPC Calculator Journal Frank Blachly, TI-59, HP-41C User Jake Schwartz, HP-41C User Bill Kolb, will coordinate and keep the peace	
		NOTE: Audience participation is vital to this session. An official 'scribe' will keep notes (and recordings) and record the 'opinion' of conference attendees. A formal "position paper" will be prepared for circulation to interested parties and manufacturers.	
4:55	PM	Vote for best speaker	
5:00	PM	DINNER	
7:00	PM	Open - Ended to Sunday morning. Informal exchange, the "real" conference.	