Programming the Personal Computer

Wherein are revealed the functions of the keys, how problems are solved, and a bit of what goes on inside.

by R. Kent Stockwell

THE HP-65 CALCULATOR uses the same reverse Polish keyboard language, the same four-register operational stack, and the same architecture as its predecessors, the HP-35,[1] the HP-45, and the HP-80.[2] It also has two important features that are new to hand-held calculators. One is its greatly expanded function set, and the other is programmability, complete with conditional and unconditional branching, user-definable functions, and magnetic-card program storage.

Function Set

Thirteen HP-65 keys are for data entry. These are the digits 0 to 9, the decimal point, **CHS** (change sign), and **EEX** (enter exponent). Numbers may be entered with or without a power-of-ten exponent.

Keyed-in digits set the value of the X register, which is also the display, in the four-register operational stack.* The **CLx** (clear x) key allows corrections. Any other key except **SST** and **R/S** terminates entry of a number.

The four arithmetic functions (+, −, ×, ÷) operate on x and y, the contents of the X and Y registers. Operands are loaded into the stack with the **ENTER↑** key; they may then be operated upon by the function keys. Operations execute immediately and results appear in X.

Thirty-three other functions derive from using three prefix keys (**f**, **f** ^{1}, **g**) to condition eleven suffix keys (digits 0-9 and decimal point). The two gold-colored prefix keys, labeled **f** and **f**$^{-1}$, access the functions printed in gold above the suffix keys and the inverses or complements of these functions. The blue prefix key, **g**, accesses the functions printed in blue on the angled lower side of the suffix keys. (The no-prefix meanings of the suffix keys appear on their top faces.) All of these functions execute immediately, operating on x, or x and y, or the entire operational stack. Thus, for example, the key sequence **f** **4** obtains sin x in the display, **f**$^{-1}$ **4** obtains $\sin^{-1}$ x, and **g** **4** obtains 1/x.

Computations requiring more data storage than is provided by the operational stack may use any of nine data storage registers. For example, pressing **STO** **4** stores x into register four, leaving x unchanged. Pressing **RCL** **4** recalls r_4 to X, leaving r_4 unchanged. Arithmetic accumulation to any storage register is accomplished by inserting the desired operation key between **STO** and the digit key that addresses the register. Thus the key sequence **STO** <arithmetic operator><digit n> gives r_n<arithmetic operator>x in register R_n and leaves x in the display.

The user can change the display format as required by the particular problem. The key sequence **DSP** <digit n> rounds the display value to n digits after the decimal point in scientific notation,* while **DSP** **.** <digit n> results in an absolute display rounded to n digits following the decimal point. For example, 12.366 gives 1.24 01 in **DSP** **2** mode and 12.37 in **DSP** **.** **2** mode. Display rounding does not affect internal values.

All functions involving angles, that is, sin, cos, tan, R→P (rectangular to polar conversion), → D.MS (conversion to degrees, minutes, seconds), and the inverses of these functions accept arguments or produce results in degrees, radians, or grads, set by the key sequence **g** **ENTER↑** or **g** **CHS** or **g** **EEX**, respectively. These settings remain in effect until changed.

On the theory that users should be able to correct key-sequence errors with minimal effort, any prefix key overrides any previous prefix key, and the sequence **f** **ENTER↑** clears any prefix keys. Thus, for example, the key sequence **STO** **+** **f** **g** **g** **4** gives 1/x,

---

*Capital letters are names of registers and lower-case letters are register contents.

*One digit to the left of the decimal point with power-of-ten exponent, e.g., 2.54 × 10^{12}.

while g f **ENTER↑** 4 gives the value 4 in the display.

By now it must be clear how key conditioning with color-coded keys and legends has been used to provide access to many functions with a limited number of keys on a small keyboard. Although another level of conditioning would further expand the function set (e.g., f g 4 or f^{-1} g 4 or g f 4 could possess functional meanings), this would greatly increase keyboard complexity, keyboard busyness, and internal control programming. For these reasons, most of the key conditioning remains at the one-prefix level.

HP-65 functions are listed on page 14. Fig. 1 shows an example of a problem solution.

Problem:

Evaluate $V_B - \frac{kT}{q}\ln\left(\frac{I_D}{I_S} + 1\right) - RI_D$

for $V_B = 8$ volts, $kT/q = 0.026$ volts, $I_D = 6 \times 10^{-3}$ amperes, $I_S = 10^{-10}$ amperes, $R = 1200$ ohms

Solution:

	Display	Stack Registers		
Keystrokes	X	Y	Z	T
8	8.			
ENTER↑	8.00×10^0	8		
.026	.026	8		
ENTER↑	2.60×10^{-2}	.026	8	
.006	.006	.026	8	
ENTER↑	6.00×10^{-3}	.006	.026	8
EEX 10 CHS	10^{-10}	.006	.026	8
÷	6.00×10^7	.026	8	8
1	1	6×10^7	.026	8
+	6.00×10^7	.026	8	8
f ln	1.79×10^1	.026	8	8
×	4.66×10^{-1}	8	8	8
−	7.53×10^{-1}	8	8	8
1200	1200	7.53×10^{-1}	8	8
ENTER↑	1.20×10^3	1200	7.53×10^{-1}	8
.006	.006	1200	7.53×10^{-1}	8
×	7.20×10^0	7.53×10^{-1}	8	8
−	3.34×10^{-1}	8	8	8

Calculator in DSP 2 Mode

Fig. 1. *An example of HP-65 use as a scientific calculator.*

Programming

All operations described so far apply when the switch in the upper right-hand corner of the HP-65 keyboard is in the RUN position. When this switch is in the W/PRGM position, the keystrokes are stored in the 100-step program memory instead of being executed. Twenty-five frequently used two-keystroke sequences merge into a single memory step; thus the program memory may actually contain more than 100 keystrokes.

STO 1
STO 2
STO 3
STO 4
STO 5
STO 6
STO 7
STO 8

RCL 1
RCL 2
RCL 3
RCL 4
RCL 5
RCL 6
RCL 7
RCL 8

g R↓
g R↑
g x⇄y
g LSTx
g NOP
g x≠y
g x≤y
g x=y
g x>y

Fig. 2. *User programs may have as many as 100 steps. These twenty-five keystroke sequences merge into a single step. Thus programs may contain more than 100 keystrokes.*

The memory itself contains no absolute addresses. Instead, it is a circulating shift register organized into six-bit words. One word is a marker that denotes the boundary between the beginning and the end of the memory. Another word is a pointer which denotes the last step executed in run mode, and the last step filled in program mode. As a program runs, this pointer is moved down through memory. Branching is accomplished by moving the pointer to the location of the destination label. User-defined function calls are implemented by leaving the main pointer at the call and activating a second pointer at the function location (see Fig. 3). When the return to the calling location occurs, the second pointer is deactivated and the first pointer reactivated. Neither the marker nor the pointers subtract from the 100 user steps.

Programs may contain three types of tests to allow conditional execution of all operations. These are x-y comparisons (x≠y, x≤y, x=y, x>y), four flag tests

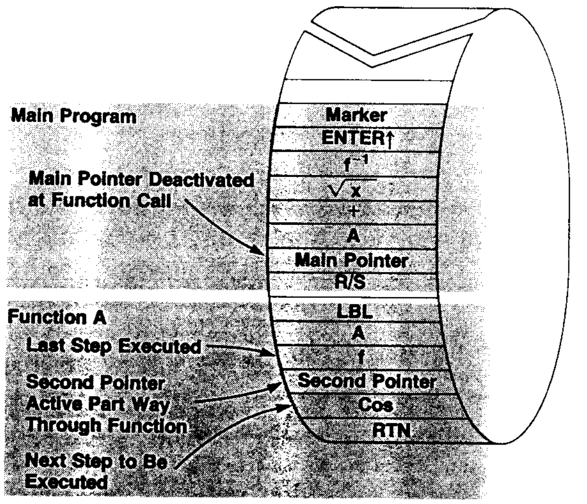
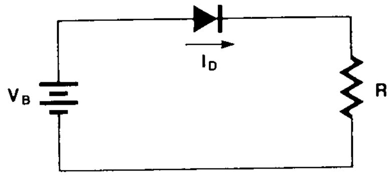


Fig. 3. *The program memory circulates continuously, its beginning and end denoted by a marker. The main pointer moves as programs are entered or executed. A second pointer is activated when a user-defined function is called.*

Problem:

Find the diode current I_D in the circuit shown. Also find its sensitivity with respect to V_B and R, i.e., $\partial I_D/\partial V_B$ and $\partial I_D/\partial R$.



Equations:

$$V_B = \frac{kT}{q}\ln\left(\frac{I_D}{I_S} + 1\right) + RI_D$$

$$\frac{\partial I_D}{\partial V_B} = \left[\frac{kT}{q}\left(\frac{1}{I_D + I_S}\right) + R\right]^{-1}$$

$$\frac{\partial I_D}{\partial R} = -I_D\left[\frac{kT}{q}\left(\frac{1}{I_D + I_S}\right) + R\right]^{-1}$$

I_S = diode saturation current in amperes
R = resistor value in ohms
V_B = battery voltage in volts
kT/q = thermal voltage in volts

Algorithm:

For Newton-Raphson iteration,

$$I_D(n + 1) = I_D(n) - \frac{f\,[I_D(n)]}{f'[I_D(n)]}$$

where $I_D(n)$ = nth guess
$f[I_D(n)]$ = function evaluated for nth guess
$f'[I_D(n)]$ = first derivative of function, evaluated for nth guess
$I_D(n + 1)$ = (n + 1)st guess

Let $$f(I_D) = V_B - \frac{kT}{q}\ln\left(\frac{I_D}{I_S} - 1\right) - RI_D$$

Then $$f'(I_D) = -\left[\frac{kT}{q}\left(\frac{1}{I_D + I_S}\right) - R\right]$$

Specify convergence criterion: if $|I_D(n + 1) - I_D(n)| < C$ the algorithm halts.

Program halts after ten iterations. The user may then start ten more iterations.

Example:

$I_S = 10^{-10}$ A
R = 1.2 kΩ
V_B = 8 V
kT/q = 0.026 V
C = 10^{-9} A

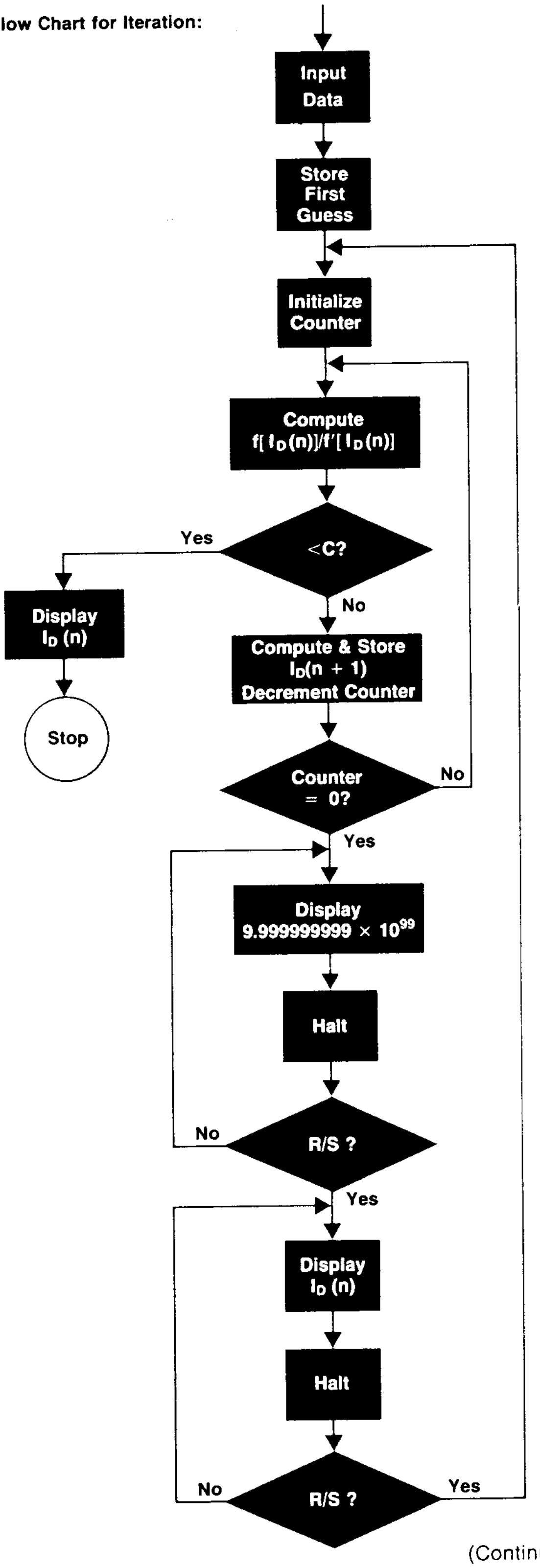
Load card and follow user instructions.

Results:

I_D = 6.278 A
$\partial I_D/\partial V_B$ = 0.8305 mA/V
$\partial I_D/\partial R$ = −5.213 μA/Ω

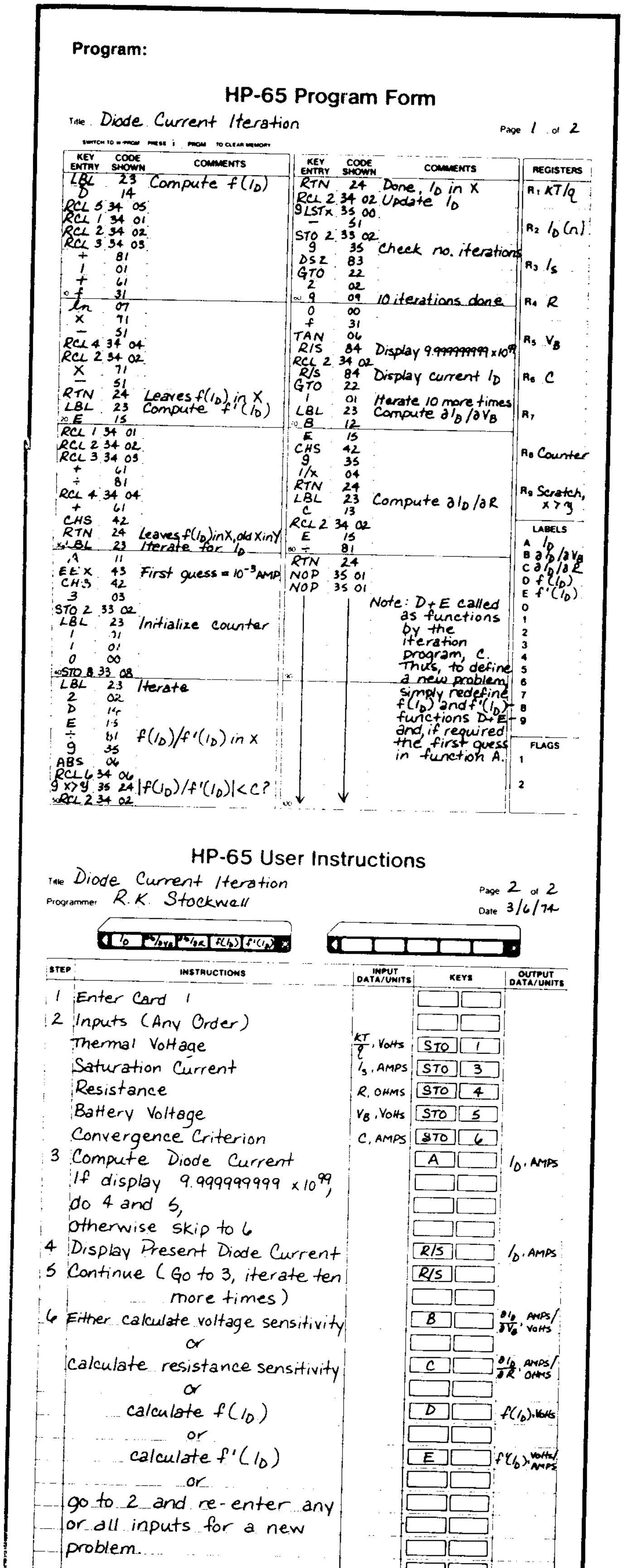
Time required to compute I_D (step 3): 11 seconds.

Flow Chart for Iteration:



(Continued)

Fig. 4. *An example of HP-65 programming. A common problem in many disciplines is the solution of irreducible equations, such as x = 5 ln x. Finding the answer requires a clever first guess at the solution and, based on the results of the first guess, an even more clever second guess, and so on. The iterative procedure, tedious if done manually, can often be automated. In this example the Newton-Raphson method is used to solve an electrical engineering problem.*



Program:

HP-65 Program Form

Title Diode Current Iteration

Page 1 of 2

SWITCH TO W/PRGM PRESS f PRGM TO CLEAR MEMORY

KEY ENTRY	CODE SHOWN	COMMENTS
LBL	23	Compute $f(I_D)$
D	14	
RCL 5	34 05	
RCL 1	34 01	
RCL 2	34 02	
RCL 3	34 03	
+	81	
1	01	
+	61	
f	31	
ln	07	
×	71	
−	51	
RCL 4	34 04	
RCL 2	34 02	
×	71	
−	51	
RTN	24	Leaves $f(I_D)$ in X
LBL	23	Compute $f'(I_D)$
E	15	
RCL 1	34 01	
RCL 2	34 02	
RCL 3	34 03	
+	61	
÷	81	
RCL 4	34 04	
+	61	
CHS	42	
RTN	24	Leaves $f(I_D)$ in X, old X in Y
LBL	23	Iterate for I_D
A	11	
EEX	43	First guess = 10^{-3} AMP
CHS	42	
3	03	
STO 2	33 02	
LBL	23	Initialize counter
1	31	
1	01	
0	00	
STO 8	33 08	
LBL	23	Iterate
2	02	
D	14	
E	15	
÷	81	$f(I_D)/f'(I_D)$ in X
g	35	
ABS	06	
RCL 6	34 06	
g x>y	35 24	$\lvert f(I_D)/f'(I_D)\rvert < C$?
RCL 2	34 02	

KEY ENTRY	CODE SHOWN	COMMENTS
RTN	24	Done, I_D in X
RCL 2	34 02	Update I_D
g LSTx	35 00	
−	51	
STO 2	33 02	
g	35	Check no. iterations
DSZ	83	
GTO	22	
2	02	
9	09	10 iterations done
0	00	
f	31	
TAN	06	
R/S	84	Display $9.999999999 \times 10^{99}$
RCL 2	34 02	
R/S	84	Display current I_D
GTO	22	
1	01	Iterate 10 more times
LBL	23	Compute $\partial I_D/\partial V_B$
B	12	
E	15	
CHS	42	
g	35	
1/x	04	
RTN	24	
LBL	23	Compute $\partial I_D/\partial R$
C	13	
RCL 2	34 02	
E	15	
÷	81	
RTN	24	
NOP	35 01	
NOP	35 01	

Note: D+E called as functions by the iteration program, C. Thus, to define a new problem simply redefine $f(I_D)$ and $f'(I_D)$ functions D+E and, if required the first guess in function A.

REGISTERS	
R_1	KT/q
R_2	$I_D(n)$
R_3	I_s
R_4	R
R_5	V_B
R_6	C
R_7	
R_8	Counter
R_9	Scratch, x>y

LABELS	
A	I_D
B	$\partial I_D/\partial V_B$
C	$\partial I_D/\partial R$
D	$f(I_D)$
E	$f'(I_D)$
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

FLAGS	
1	
2	

HP-65 User Instructions

Title Diode Current Iteration

Programmer R.K. Stockwell

Page 2 of 2

Date 3/6/74

I_D | $\partial I_D/\partial V_B$ | $\partial I_D/\partial R$ | $f(I_D)$ | $f'(I_D)$

STEP	INSTRUCTIONS	INPUT DATA/UNITS	KEYS	OUTPUT DATA/UNITS
1	Enter Card 1			
2	Inputs (Any Order)			
	Thermal Voltage	$\frac{KT}{q}$, Volts	STO 1	
	Saturation Current	I_s, AMPS	STO 3	
	Resistance	R, OHMS	STO 4	
	Battery Voltage	V_B, Volts	STO 5	
	Convergence Criterion	C, AMPS	STO 6	
3	Compute Diode Current		A	I_D, AMPS
	If display $9.999999999 \times 10^{99}$,			
	do 4 and 5,			
	otherwise skip to 6			
4	Display Present Diode Current		R/S	I_D, AMPS
5	Continue (Go to 3, iterate ten		R/S	
	more times)			
6	Either calculate voltage sensitivity		B	$\frac{\partial I_D}{\partial V_B}$, AMPS/Volts
	or			
	Calculate resistance sensitivity		C	$\frac{\partial I_D}{\partial R}$, AMPS/OHMS
	or			
	calculate $f(I_D)$		D	$f(I_D)$, Volts
	or			
	calculate $f'(I_D)$		E	$f'(I_D)$, Volts/AMPS
	or			
	go to 2 and re-enter any			
	or all inputs for a new			
	problem.			

(there are two flags, each of which may be set or cleared and then tested for set or clear), and decrement and skip if zero (DSZ). Except for DSZ, each test, if false, causes program control to skip the next two memory steps; otherwise, execution continues normally. The DSZ operation decrements data-storage register R_8 by one, using integer arithmetic, and if the result is zero, program control skips the succeeding two steps.

Literal labels with the GO TO function implement branching. Thus LBL<n> is the destination for GTO <n>, where n is a digit or a key **A-E** in the top row.

The HP-65 user may store two types of programs in the program memory. First, he may precede a section of memory containing various functions with LBL <m>, where m is A, B, C, D, or E, and terminate the section with RTN (return). Thereafter, pressing key A, B, C, D, or E in the RUN mode causes that memory section to execute immediately. Any or all of keys A to E may be defined but the sum of memory steps for all functions cannot exceed 100. These user-defined functions behave exactly like the preprogrammed functions described earlier, yet the user may create the functions to fit his special needs.

The user's second option is to precede a block of code with a label definition and terminate it with the R/S (Run/Stop) key. In RUN mode this key stops an executing program; if no program is running, pressing the R/S key starts execution. Pressing GTO<label name>R/S then starts program execution, and the program halts at the R/S in memory. If the program starts at the beginning of memory no label is needed; in RUN mode control can be transferred to the beginning of memory by pressing **RTN**. Programs defined in this way may call any of the functions A through E; the desired key is simply entered into the program definition.

The **SST** (single step) and **DEL** (delete) functions implement debugging and editing. In W/PRGM mode, each depression of SST advances the memory pointer one step and displays each memory step as a two-digit key code. These codes represent digit keys by their values and all other keys by a row-column index of the key position referenced to the upper left-hand corner of the keyboard. For example, the decimal-point key is in the eighth row, third column, so its code is 83. In the RUN mode, each depression of SST advances the memory pointer one step and executes the adjacent memory step.

The key sequence **g CLx** in W/PRGM mode deletes the displayed memory step and moves up the next step to fill the gap. Any keys entered in W/PRGM mode are automatically inserted following the displayed memory step. Thus the replacement operation consists of a delete operation followed by the desired key.

The sequence **f CLx** clears the entire memory.

Programs can be stored on magnetic cards for later use. Cards can be recorded and rerecorded as many times as desired. To protect a recorded program on a card, further recording can be prevented by clipping the notched tab on the upper left corner of the card. Users may write on the card and place it in a slot above the keys **A** through **E**, thereby labeling any specially defined keys.

Fig. 4 shows an example of HP-65 programming.

Firmware

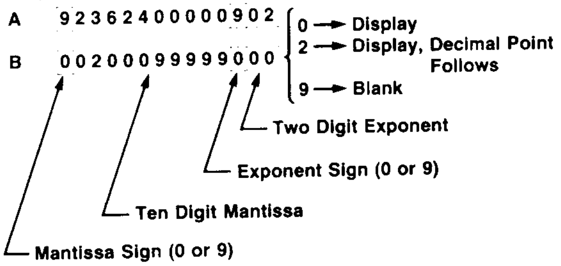
To direct the various computational and control functions of the HP-65, 3072 words of read-only memory (ROM) are used. Each ROM word contains ten bits and constitutes a calculator microinstruction. Microinstructions grouped together in blocks perform the various external functional tasks of the calculator. A task may require one block of words or several blocks woven together. For example, the **CLx** function requires only a few words, while the sin function uses the tan function, which uses the add function, and so on.

Although production of efficient microcode is an iterative process, the first step is the choice or design

HP-65 Normalization Routine

User input form is stored in two processor registers, A and B.

Example: $- 23.624 \times 10^{-2}$



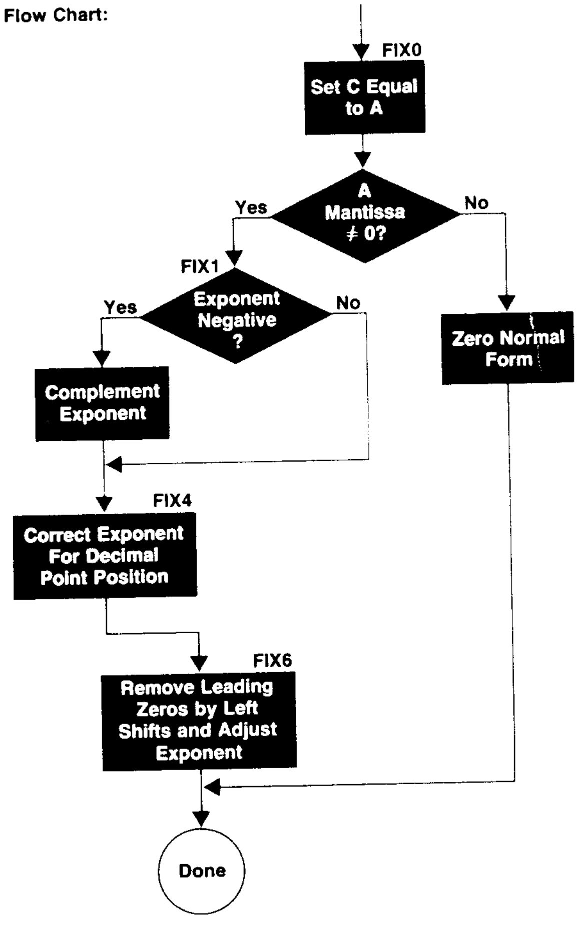
This is converted to internal normal form in a third register, C. For computational efficiency this is mantissa in sign-magnitude form with one digit to the left of an implied decimal point and exponent in ten's complement form (see ref. 4). In the mantissa sign position, 9 represents minus, 0 represents plus. Thus, -23.624×10^{-2} would store internally as

9 2 3 6 2 4 0 0 0 0 0 9 9 9

Program Listing:

Step Number	ROM Address	ROM Code	ROM Subroutine Addresses	Labels	Program Statements
51	L01063:	. 111..		FIX5 :	P - 1 -> P
52	L01064:	1111 1.1.			C + 1 -> C[X]
53	L01065:	.111.11.1	-> L1073		JSB FIX7
		⋮			⋮
56	L01070:	..11 11.1.		FIX3 :	0 -> C[XS]
57	L01071:	11 1..11.		FIX4 :	13 -> P
58	L01072:	.1 11 1.1.			C - 1 -> C[X]
59	L01073	........1.		FIX7 :	IF B[P] = 0
60	L01074	.11 1111	-> L1063		THEN GO TO FIX5
61	L01075:	11 ...11..			12 -> P
62	L01076:	1..11...1.		FIX6 :	IF A[P] >= 1
63	L01077:	.11111 .11	-> L1174		THEN GO TO FIX2
64	L01100:	1 ....11.			SHIFT LEFT A[M]
65	L01101:	.1.11.1.1.			C - 1 -> C[X]
66	L01102:	11111 1	-> L1076		JSB FIX6
		⋮			⋮
102	L01146	111111 1.		FIX1 :	C + 1 -> C[XS]
103	L01147:	. 111...11	-> L1070		IF NO CARRY GO TO FIX3
104	L01150:	. 1.1.1 1.			0 - C -> C[X]
105	L01151:	..111 .1 1	-> L1071		JSB FIX4
		⋮			⋮
119	L01167:	111 1.111.		FIX0 :	A EXCHANGE C[W]
120	L01170:	.11...111.			C -> A[W]
121	L01171:	1..11..11.			IF A[M] >= 1
122	L01172:	.11..11.11	-> L1146		THEN GO TO FIX1
123	L01173:	1.111..11.			0 -> A[M]
124	L01174:	111.1..11.		FIX2 :	A EXCHANGE C[M]

Flow Chart:



Program Listing Notes:

Subscripts, e.g., C + 1 → C[X], refer to various register fields for arithmetic operation (see ref. 1).

S = Mantissa Sign
MS = Mantissa and Sign
M = Mantissa
XS = Exponent Sign
X = Exponent and Sign
P = Pointer
WP = Word up to and Including Pointer, Right to Left

Fig. 5. *An example of the HP-65's internal microprogramming. Even such a seemingly trivial operation as digit entry requires careful design so it seems trivial to the user. Values must be displayed as keyed in, yet be normalized to a standard internal form. This is the normalization routine and the flow chart and ROM listing for it.*

of an algorithm. This may involve such constraints as accuracy, execution speed, microinstructions required, or even available design time. Next, a functional flow chart is drawn to outline the sequence of various operations and any conditional operations. This flow chart is then expanded to sufficient detail that it can be translated to microinstructions and implemented on a calculator simulator. More often than not there are implementation errors to correct; sometimes the entire algorithm is faulty, requiring a new design. When the design is complete, integrated-circuit read-only memories are produced.

Where possible, the HP-65 uses the proven algorithm implementations from the HP-35 and HP-45 (trigonometric, logarithmic, and exponential routines). This saved development time and reduced implementation error probabilities.

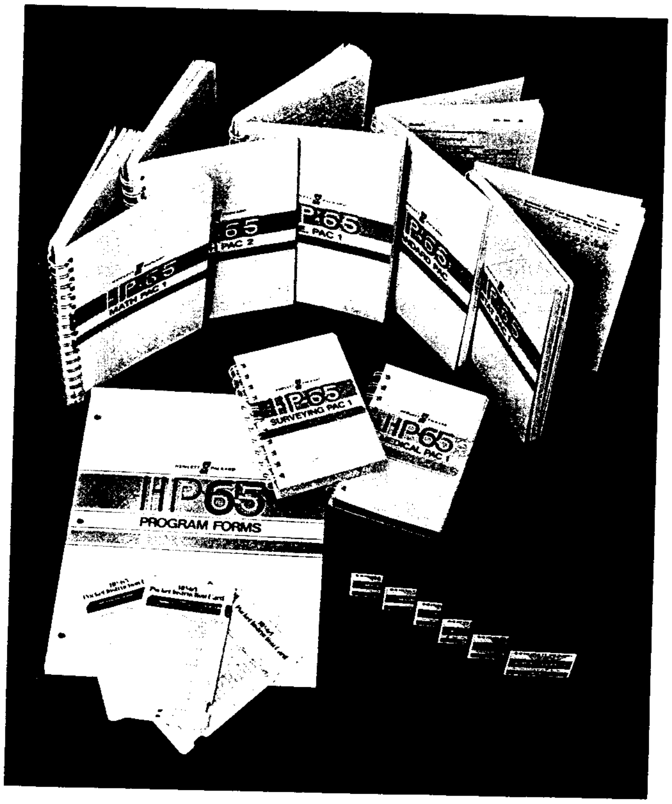
Many HP-65 algorithms would provide interesting descriptions here, but one that demonstrates appreciable complexity is the digit-entry routine. Designing this seemingly trivial function so as to seem trivial to the user required considerable patience and careful thought. Usually, any entry will produce an undesirable result unless the designer specifically accounts for it. Values must be displayed as keyed in, yet they must be normalized to some internal form. The table below lists some of the design constraints on this algorithm.

USER ACTION	DESIRED RESULT
More than ten mantissa digits	Ignore all digits after tenth
First key of new entry	Overwrite existing x if key follows **ENTER↑** or **CLx**; otherwise do automatic **ENTER↑**
Extra digits after **EEX**	Shift exponent left; new digit becomes least significant digit of exponent.
Multiple decimal point	Ignore all decimal points after first
Decimal point after **EEX**	Ignore
Leading zeros keyed in	Accept and display leading zeros, zero normal form.
EEX first key of new entry	Enter one in mantissa; following digits enter exponent.
Decimal point first key of new entry	Display only decimal point; zero normal form.
Digits after decimal point	Continue appending digits; no effect on internal exponent
Digits before decimal point	Continue appending digits, increment internal exponent.
following	Complement exponent sign
Multiple	Complement mantissa sign, or exponent sign if has been pressed.

Such an algorithm was explained in a previous issue.[3] Fig. 5 shows the flow chart and ROM listing for the normalizing routine.

Acknowledgments

Many people of course, contributed ideas to this effort. Particular acknowledgment is due the following: Paul Stoft and Tom Whitney for bringing together the necessary technical resources and people; Dave Cochran, for the trigonometric and exponential routines used in the HP-35, and for help in understanding the HP-35 architecture; Francé Rodé for further explanations of the HP-35 architecture; Peter Dickinson for suggestions and criticisms concerning algorithm implementations, particularly the extension of the HP-35 algorithms; Tom Osborne for helpful advice and suggestions regarding the function set and the external behavior of the HP-65; Homer Russell and Wing Chan for helpful suggestions and criticisms for the function set, and for



patiently keeping up with numerous daily changes; Steve Walther for providing the microinstruction language compiler; Darrel Lauer and Al Inhelder for crystallizing the keyboard layout from a myriad of suggestions; Ed Heinsen and Lynn Tillman for extending the simulation software to accommodate the increased complexity of the HP-65.

References

1. T.M. Whitney, F. Rodé, and C.C. Tung, "The 'Powerful Pocketful': an Electronic Calculator Challenges the Slide Rule", Hewlett-Packard Journal, June 1972.
2. W.L. Crowley and F. Rodé, "A Pocket-Sized Answer Machine for Business and Finance", Hewlett-Packard Journal, May 1973.
3. D.S. Cochran, "Internal Programming of the 9100A Calculator", Hewlett-Packard Journal, September 1968.
4. M.M. Mano, "Computer Logic Design", Prentice-Hall, 1972, chapter 1.

R. Kent Stockwell

Kent Stockwell joined HP four years ago. As a member of HP Laboratories for most of that period, he's done program development, modeling, and numerical analysis for computer-aided circuit design and, more recently, the firmware development for the HP-65. Kent studied electrical engineering at Massachusetts Institute of Technology, graduating in 1970 with SB and SM degrees. A native of Kalamazoo, Michigan, he now lives in Palo Alto, California, where he's currently remodeling his house and putting his woodworking skills to good use. He also plays trombone and baritone horn, and enjoys backpacking in the mountains of California and Colorado.

APPENDIX

HP-65 Programmable Pocket Calculator Functions and Operations

Arithmetic
- add
- subtract
- multiply
- divide

Logarithmic
- natural logarithm (base *e*)
- natural antilogarithm (base *e*)
- common logarithm (base 10)
- common antilogarithm (base 10)

Trigonometric
- set operating mode (degrees, radians, or grads)
- sine
- arc sine
- cosine
- arc cosine
- tangent
- arc tangent
- add or subtract degrees/minutes/seconds
- convert angle from degrees, radians, or grads to degrees/minutes/seconds and vice versa
- convert polar coordinates to rectangular coordinates and vice versa

Exponential
- square
- square root
- raising a number to a power (y^x)
- reciprocal (can be used with y^x function to extract nth roots)

Other Preprogrammed Functions and Operations
- extract integer or decimal portion of a number
- factorial
- recall value of π to 10 significant digits
- convert decimal-base integers to or from octal-base integers
- "roll down" or "roll up" numbers in operational stack
- clear display
- clear operational stack
- clear all nine addressable memory registers
- recall last input argument from separate "last-x" storage register
- store or recall numbers from any of the nine addressable memory registers
- register arithmetic
- display formatting

Program Structure and Edit Functions
- clear program memory
- user-definable keys (A-E)
- label
- go-to
- return
- run/stop
- no-operation
- set flag 1
- test flag 1
- set flag 2
- test flag 2
- $x = y$
- $x \neq y$
- $x \leq y$
- $x > y$
- decrement and skip on zero
- delete program step
- single-step