Notes on the back story of this letter:

This *18-page* letter is one of the longest I sent to ***Richard Nelson***, which I began by thanking him for including in the *V7N4* issue of the *PPC Calculator Journal* two programs of the many I had submitted for publication, even if one of them was the wrong choice. I also praised the quality of said issue, and in particular ***Bill Wickes'*** outstanding contributions and his amazing kindness towards newbies (like myself) who asked him for help in getting copies of his programs on mag cards.

The letter also included a treasure trove of additional new materials submitted for publication in the *PPC Journal*, among them: (1) an updated version of my ***Black Box*** programs featuring extra functionalities, (2) my solution to a *Computer Challenge* issued by ***John Kennedy***, (3) the shortest & fastest random number generator routine for the *HP-67/97/41C* (perfectly adequate for games), (4) several inputs for the *NOP* and *Feedback* columns, (5) two inputs for the *ROM Progress* column, including my ***Display Test*** synthetic routine, (6) a full 3-page article "***STO b, RCL b or The Universal Byte-Jumpe***r" showing how to use just those two synthetic instruction to easily create any synthetic function, text or key assignment, how to unrestrictedly jump to anywhere in memory, and how to break **`PRIVATE`**, with full documentation and examples

Also included were two programs: (7) a 5-page, complex *HP-67/97* program to design ***Elliptic Lowpass Filters*** (220+112 steps, two cards), fully documented with graphs and examples, and (8) an *HP-25/25C/33* program to numerically solve a given 2^{nd}-order differential equation of the form $y'' = f(x,y)$ using a 5^{th}-order method while taking just 37 program steps.

I ended the letter with a few comments about some tentative *PPC ROM* contents and, most important, I offered myself to support a new regular column about *Numerical Analysis*, enthusiastically detailing my ideas about its proposed layout, contents, and goals, and asking ***Mr. Nelson*** to drop me a note to discuss it further, should he be interested.

Well, he probably wasn't because I never got any note or letter from him, not on that subject, not ever, which obviously caused my enthusiasm to quickly fade away. As for the many materials I sent in this letter, some were published in full, some were censored (the **`STO/RCL b`** article), and the ***Display Test*** routine was included in the *PPC ROM*, the only one of the many I sent but I guessed I could consider myself lucky, though I very much resented the utterly ignored heartfelt column offer.

Last but not least, photocopying all 18 pages and sending them by insured mail costed me a little fortune, which I really couldn't afford at the time without making some sacrifices. Which I did.

Valentin Albillo, 21-12-2021

Richard Nelson
Editor, PPC Journal
2541 W. Camden Place
Santa Ana, CA 92704
U. S. A.

Valentin Albillo (4747)
Padre Rubio, 61- 2ºC
Madrid 29
SPAIN

Dear Richard :

Thank you very much for the inclusion into V7 N4 (May issue) of two of my programs, "LIN. EQ. GAUSS-SEIDEL" , and "NEW BLACK-BOX PROGRAMS". It was quite exciting to see my name in printed characters, you know. However, I was amazed to see the "LIN. EQ. GAUSS-SEIDEL" published instead of the "LINEAR EQUATIONS - GAUSSIAN ELIMINATION". I suppose you believed the latter to be redundant with previously submitted programs which also used gaussian elimination. That's not so. My program solves up to 30x30 systems, whereas previous programs reach only 15x15 or 16x16. Hope you'll eventually publish it, as well as all the others submitted.

Wonderful issue, the MAY issue. I think that Bill Wickes should receive a special mention or some other kind of honor, for he works so hardly and so well. See his BB programs, his pionering onto status registers access, his Byte-jumper article. He should be a model for all active PPC members. Besides, he is equally gentle towards beginners. I wrote him a letter several months ago, asking for information and the BB programs. I am sure that almost thousands of other members must have written him asking for the same thing. Yet, he was so kind as to send me the BB programs, photocopies of several of his articles, and a hand-written letter with explanations and comments. Really admirable ! .

Here included are several contributions:

- a revision & improvement of my BB programs published in V7 N4 P28-29 . Notice this revision, because it resumes in only 160 bytes of program memory the full power of CODE, DECODE, and SYNTHETIC KEY ASSIGNMENTS programs. The owners of basic machines will be delighted with this one. The techniques are useful for everybody , however. Also, resembles like B2.

- A solution to the Computer Challenge which John Kennedy presented in V7 N4 P 12 (MAY iss) It is easily seen why John posed out this challenge: A(4,2)=2.0035 E19728 approximately which is far beyond the range of any PPC or computer. See my solution included herein .

- Several inputs for the columns ROUTINES, NOP , FEEDBACK , etc

- Two inputs for the ROM PROGRESS column. One is a comment about microcode in the ROM. If it is feasible, it must be included. The other is a test display routine to be included in the PPC Custom ROM: it fully test all segments & functions of the display, is only 112 bytes long, requires no bugs, uses no register, and leaves the whole status of the calculator , (flags, registers, etc) undisturbed, except for T,LAST X , and ALPHA, which are lost. A magnetic card is included with the routine stored onto it. (A single side)

- An article about the couple of functions STO b, RCL b. It shows how those functions may be used to achieve everything: (1) any synthetic text (2) any synthetic function (3) any synthetic key assignment (4) jumping to everywhere in memory instantly without constraints , and (5) how to break PRIVATE. All that is required to do all these things is STO b, RCL b. So STO b, RCL b are almost universal tools. You can delete the part about breaking PRIVATE, if you believe it convenient. Notice the improved method over my last letter on it.

- A 67/97 program about a very difficult technical subject: ELLIPTIC LOWPASS FILTER DESIGN. This program should delight all members who are electrical engineers or the like. As any filter may be normalized to a lowpass type, the program is quite general. No magnetic card is included, as I cannot record it (I haven't access to 67 or 97).

- a 25/25C/33 program to numerically solve any given 2º order differential equation of the form $y'' = f(x,y)$. 12 steps remain to define f. A 5-th order method is used.

That's all. Hope you will like some of it.

Now, some ideas: Do you think a good idea to include some kind of game into the ROM ? Of course, the ROM should contain only general purpose routines, but maybe a short game will fill a real need: Many times, when one is tired, one appreciates a challenging game to be present into the calculator to have some fun: It could be useful too for demo purposes. I would like to know what the membership thinks about this idea.

Finally, I will offer myself to support a column in the PPC CJ about Numerical Analysis, similar to the column that Craig Pearce supported on BASIC programming (THE MICRO SCENE). My column will discuss in an orderly manner the best numerical algorithms to be implemented in PPCs, ranging from matrix manipulations (inversion, eigenvalues, linear systems) to almost every topic (diff, eq. , systems, boundary value problems, curve fitting, least squares, interpolations, root finding, integrals, derivatives, special functions, summations, Fourier analysis, .. etc, etc, etc). The column will be structured like

this: -presentation of the subject
-discussion of the methods available to solve the problem
-the selected method, its advantages over the others. (Many times, there will be several optimum methods,depending in amount of memory, speed required, precision,...)
-its implementation in a specific model of PPC (25,34c,67,41c)
-a real-life application

This is only a sketch of what can be accomplished. The whole column will scarcely exceed one page if there are space constraints imposed. Otherwise, I may develop the subject in the specified amount of space. I have enough time to compromise myself to the task, and (I think) my knowledge about Numerical Analysis and my experience programming HP PPCs (25,34,67,41) is rather high (excuse my being unmodest). The idea for this column appeared when I realized that most optimum methods for solving problems were not known to most programmers. Also, most books about Numerical Analysis told the user about a great deal of methods, without ever pointing what the optimum should be.

Nothing more to say, excuse the length of this letter. If you agree with my supporting the Numerical Analysis column, drop me a note and we can discuss some procedures specific to the task, so I can set the task at once.

Best regards ;

P.S . Keep the stamps. They are worth a little fortune ! (about 4 US dollars)

-this is a revision of the programs published in V7N4P28-29 under the title "BLACK-BOX - DECODE & INSERT" , written by myself. This new revision, cuts 3 bytes of the previous programs, and add to their CODE , DECODE capabilities, the KEY ASSIGNMETS OF SYNTHETIC FUNCTIONS capability at the cost of 29 bytes of program memory. Thus, the whole set of programs "U","R",and the new add , "W" , will allow the user of a very basic machine to decode, insert, create any 7-byte NNN, and assign almost automatically any instruction to any key, using only 131 + 29 = 160 bytes of program memory.

NEW BLACK BOX REVISED & IMPROVED

Hey, you RAMless user ! Do you liked the "NEW BLACK BOX PROGRAMS" published in V7N4P28-29 (May issue) ? There is a good new for you herein. Now you can assign also any function to any key almost automatically (only HEX-OCTAL conversion required) using an add-on to the programs "U","R".

The new "program" (only 29 bytes) called "W" , will store any NNN configuration (created by you,using "R") into any absolute register address. For instance, if you store something into register OC0,OC1, ... you are creating assignments. Also into program memory, data registers, thus performing an Bug-2 like operation. Are you interested ? Notice that using "W", you can (among other things) create any assignment, for instance, any synthetic function may be assigned to any key, very similarly to the program SYNTHETIC KEY ASSIGNMENTS in V7N3P3.

- First, line 02 "A" of the Black Box programs in V7N4P28-29 may be changed to 02 CLA , thus saving 1 byte. Similarly, the line 34 "A" may be deleted, saving 2 extra bytes. The total is 134-3 = 131 bytes for "U", "R".
- Now, add the new program "W". After line 75 RCL 0 of the BB programs, load RTN. (this is, line 76 RTN). Now, key in this program

```
77 LBL"W"     84 X()c
78 DEC        85 RCL Z
79 192        86 STO IND Z
80 -          87 X()Y
81 ENTER      88 X()c
82 "天■μ¯■"  89 .END.
83 ASTO X
```

29 bytes

total U,R,W =160 bytes

-line 82 is a synthetic text. It is F50169CC0CBF. It may be created using STO b, RCL b, the byte-jumper, "R", etc. This line changes contents of Rc temporarily, to allow the NNN created by you to be stored wherever desired using simply STO IND Z. Once this is performed, the previous contents of Rc are restored.

HOW TO USE "W"
- Place the NNN you want to store in Y
- Place the absolute address of the register in which the NNN should be stored, in X (the address should be in OCTAL . For instance, to store something into OC0 (the first assignment register), the address is HEX(OC0) = OCT(300), so place 300 in X.
- XEQ "W" → ignore the display. Your NNN will be stored wherever desired. No register, flag, etc, except the stack and ALPHA , will be disturbed. Rc remains unchanged.

EXAMPLE:We are going to create 2 assignments: ENG IND a to the key -24, and ASTO b to the key 72.

-we must create a synthetic assignment register: the first byte is always F0. Now, ENG IND a is 9E FB. The key -24 is 3A. ASTO b is 9A 7C .The key 72 is 17. So, the NNN to be created is F09EFB3A9A7C17. We use "R" to create it as follows:

-first HEX(F09EFB3A9A7C17) must be translated to OCTAL. use the HEX-OCTAL conversion table (see V7N4P29) to get: F0 = 360

9EF = 4757 , B3A = 5472 , 9A7 = 4647
C17 = 6027

-so that HEX(F0/9EFB3A/9A7C17) = OCT(360/47575472/46476027)

-now, using R, we build the NNN in 3 stages, using n=1,4,7:

1 ENTER 360-ENTER 0 , XEQ"R" → -0.0000
4 ENTER 47575472 RCL Z , XEQ"R" → -1.0561
7 ENTER 46476027 RCL Z , XEQ"R" → -0.1056 "

-the NNN is in the display. Now, store it into the first = assignment register. Its HEX address is HEX(000) , which is OCT(300). Previously, you must assign anything to the keys -24 and 72, to set the appropiate flags.

Press : ASN SIN f COS (assigns SIN to -24)
ASN SIN 1 (assigns SIN to 72)

-now, store the NNN (which should be still in X) into 300

300 XEQ"W" → 天圓μ圖

-the work is done. To see the assignments, press GTO.., switch to PRGM, set USER mode, and press thefCOS key (-24 key) . 01 ENG IND a must appear. Now press the 1 key (72 key). ASTO b will be loaded into memory.

-to create additional assignments, create the NNN required, then store it into 301, etc.

VALENTIN AIBILLO (4747)

- This is an answer for the Computer Challenge proposed by John Kennedy (918) in the column "BITS & PIECES" , in - V7N4P12.

- -

In V7N4P12 , John Kennedy (918) proposes the following :

given $A(o,k) = k + 1$ (I)
$A(j,o) = A(j-1,1)$ (II)
$A(j,k) = A(j-1,A(j,k-1))$, find $A(4,2)$ (III)

John asks : " Program any PPC or the HP-85 to compute A(4,2)"

here is my answer: - I am sure that John would agree with me that my own brain is the best PPC I own, so I've made a brain program to compute A(4,2) as follows:

(a) lets find $A(1,k)$: using (III) , we have
$A(1,k) = A(o,A(1,k-1)) =$ using (I) $=$
$= 1 + A(1,k-1)$

and if k=0 , $A(1,o) =$ using (II) $= A(o,1) =$ using (I) $= 2$

so we have the difference equation (not differential!)

$A(1,k) = 1 + A(1,k-1)$, $A(1,o) = 2$ (the initial condition)

whose solution is $A(1,k) = k+2$ (IV)

(b) let's find $A(2,k)$: using (III) , we get
$A(2,k) = A(1,A(2,k-1)) =$ using (IV) $=$
$= 2 + A(2,k-1)$

and if k=o , $A(2,o) =$ using (II) $= A(1,1) =$ using (IV) $= 3$

so we get the difference equation

$A(2,k) = 2 + A(2;k-1)$, $A(2,o) = 3$

whose solution is $A(2,k) = 3 + 2k$ (V)

(c) now, $A(3,k)$: using (III) ,
$A(3,k) = A(2,A(3,k-1)) =$ using (V) $=$
$= 3 + 2A(3,k-1)$

if k=o , $A(3,o) =$ using (II) $= A(2,1) =$ using (V) $= 5$

and now, the difference equation is

$A(3,k) = 3 + 2A(3,k-1)$, $A(3,o) = 5$

whose unique solution is : $A(3,k) = 2^{k+3} - 3$ (VI)

(d) the final stage, $A(4,2)$

using (III), $A(4,2) = A(3,A(4,1))$
$A(4,1) = A(3,A(4,o))$
using(II) , $A(4,o) = A(3,1) =$ using (VI) $= 13$

so, $A(4,1) = A(3,13) =$ using (VI) $= 2^{16}-3 = 65533$

and finally, $A(4,2) = A(3,65533) = 2^{65536} - 3$

this is, $A(4,2) = 2^{2^{16}} - 3 = 2^{65536} - 3$

which is approximately 2.0035 E19728

Would someone try A(5,2) ?!

VALENTIN ALBILLO (4747)

- here are some inputs to your column "Routines"

- -

Here is the shortest, fastest randonnumber generator for programmable calculators :

LBL any
RCL any
R-D
FRAC
STO any
RTN

-it may be used for 41c & 67/97, and all the calculators with a built-in R-D conversion.

-It may be used as a subroutine, or be in the body of the program : specially useful for games, or simulations.(If desired, another R-d could be added.)

-the previous seed must be stored in the selected register, and cannot be 0. The RNG generates numbers between 0 and 1 uniformly. The seed can be any integer, or decimal, except PI or its multiples. The method used is the following:

$$u_{i+1} = \text{FRAC}(57.2957795 \times u_i),$$

which is very similar to the popular

$$u_{i+1} = \text{FRAC}(997 \times u_i)$$

but is 3 bytes (steps) shorter (about 42% shorter) and faster. It will be very useful to those people who need 2 6 3 steps of program memory for its program to fit.Here below is reproduced a short statistics about the frequencies resulting in a trial simulation using seed=1 .

number	0	1	2	3	4	5	6	7	8	9	total
freq	104	123	103	92	84	101	104	90	95	104	1000

$\overline{x} = 4.4$, $s = 2.9$ (theory: $\overline{x} = 4.5$, $s=3.0$)

I have tested the method for cycle length and repetitions, but chosing seed 0.1 I didn't find the period after 30000 generations. Maybe someone with a fast computer would like to test this method for uniformity and randomness.

VALENTIN ALBILLO (4747)

- -

-this are some inputs to your column "NCP"

- -

Reference : INDIRECT CONTROL by WM Kolb - V7N4P15-17

The answer to question 14 is incomplete : the display will show NONEXISTENT if there is no program in program memory labeled "HMS+" . Otherwise, the program - HMS+ (user's program, not mainframe function) will be executed.

This should be pointed out to avoid the user thinking that programs labeled the same as mainframe functions cannot be executed indirectly.Also, answer to question 17 should be R02=2.002 instead of R03=2.002

VALENTIN ALBILLO (4747)

- -

Reference : SOLVING INTIGER TRIANGLES by RB Smith -V7N4P30

The given sample run shows an error: given the number 4, program outputs the "NO TRI(S" message, while it is evident that a triangle with sides 3,4,5 is the proper answer: if a=4, b=3, c=5, the triangle is right, isn't it ?

Also, apart from numbers 0,1,2, all other result in a given triangle existing, so the "NO TRI(S" message is useless. The program has some bugs in it, anyway.

Line 27, GTO a may be deleted, as LBL a is line 28.

VALENTIN ALBILLO (4747)

- -

-this is an input for FEEDBACK

- -

Reference : FREEDOM FROM BUGS by WC Wickes - V7N2P30

Wickes says : "... I have yet to see a real -

practical programming advantage to being able to control flags 30-55, except perhaps for the capability of determining display and trig status under program control..."

Well, I will give a very practical application of B3 (indirect control over system flags 30-55) :

-most owners of Application Pacs (ROMs) agree that the programs in the ROM are almost useless as subroutines. For instance, Marlin R. Gillette says in V7N1P4-5 that the user cannot supress intermediate results when using the MATH 1A rom (the case of no printer) and must continually press R/S to get through a program ; this is true. The programs in the MATH 1A ROM are only slightly useful to those owners with a printer. The matrix program stops after every output. The complex functions always display imaginary & real parts, thus halting program execution, so they can't be called as subroutines of a main program.

This is due to the fact that most MATH 1A programs set flag 21. If flag 21 is set and no printer is present, flag 55 is cleared, so AVIEW , VIEW , etc, stop program execution. There is no way to avoid this, as the SF 21 in the ROM cannot be deleted. However, if the user sets flag 55 , (the printer existence flag) then flag 21 set will not stop program execution when VIEW is performed. This allows most complex calculus subroutines to be used in that fashion by user's programs: the R & I parts will be displayed, but the program will not halt. This applies also to the Differential Equations program, which now may run without R/S being needed after each point is displayed.

To resume: if you want to use MATH 1A ROM programs as subroutines, and you do not have a printer, you must set flag 55 (using B3 or the routine in V7N4P23) prior to calling the ROM program as a subroutine.

VALENTIN AIBILLO (4747)

- another input for FEEDBACK

Reference : HP-41C BUS INTERFACING by Jim de ARRAS
See V7N3P20-21-22

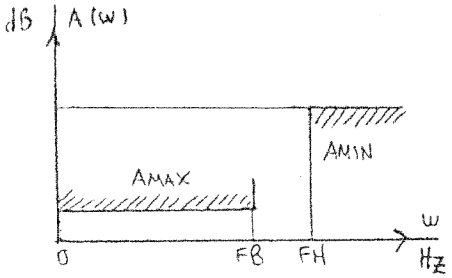
Jim asked for feedback from its article, to "...ensure my descriptions are understood by most members" Well, here is the feedback. Please, submit the second part of your article as soon as possible. It is fascinating , to say the least, and can help quite a lot to interface external devices to the 41c , a tape-recorder mass storage device, for instance, and external RAM beyond 319 registers. By the way, Jim, how do you get that information ? Are you the son of some HP tycoon ? Best regards.

VALENTIN AIBILLO (4747)

67 - ELLIPTIC LOWPASS FILTER DESIGN

Here presented is a technical program that - Bruce K. Murdock will surely like. Bruce is the author of "Handbook of Electronic Design and Analysis Procedures - using Programmable Calculators", a real bible for the advanced student of Electronics. I hope this program will catch the attention of Bruce and of all those people with electronical interest.

This program finds the minimum degree N of an elliptic lowpass filter that meets the standard requirements A_{max}, A_{min}, FB, FH . If the degree is less than 31, it also finds and stores the zeros and poles of attenuation, and can find the loss at any frequency either manually or automatically, chosing between linear or logarithmic sweep. The loss is given in dB , and the frequencies in Hz. As an additional convenience for engineers designing elliptic filters, the elliptic sine sn(u,k) and the complete elliptic integral of the first kind , are available as keyboard routines. All routines are fast, and accuracy is 8 decimals or better.



The program is intended as a help when designing elliptic lowpass filters. If they are highpass, bandpass, bandstop types, they must be previouly transformed into a normalized lowpass. The standard requirements are:

A_{min} = minimum stopband loss
A_{max} = maximum passband loss
FB = upper passband edge
FH = upper stopband edge

(Program is 224+112 steps long, so it fits onto 1½ magnetic cards)

Algorithms : consider the elliptic integral of the 1st kind, U(phi,k)

$$U(phi,k) = \int_0^{PN} dx/SQRT(1-k^2\sin^2 x)$$

if phi=PI/2, the integral is called "complete" :

K(k) = U(PI/2,k), and the complementary elliptic integral is K'(k) = K(k'), where k' = SQRT(1-k²)
-consider also the elliptic sine, sn(u,k) = sin (phi)
-then, we compute X_L = FH/FB , e=10^(.1A_{max})-1

$$L = SQRT((10^{(.1A_{min})}-1)/(10^{(.1A_{max})}-1))$$

-and so, the minimum degree N necessary to meet the requirements is:

$$N = INT((K(1/X_L).K'(1/L))/(K'(1/X_L).K(1/L)))+1$$

-once the necessary degree is found, the zeros & poles of attenuation may be computed and stored (if N(=30)
-in normalized form:

for i=1,2,...INT(N/2) ; N even → $x'_{zi} = sn((\frac{(2i-1)K(1/X_L)}{N},1)$

N odd → $x'_{zi} = sn((\frac{2i.K(1/X_L)}{N},\frac{1}{X_L})$

and the normalized poles of attenuation are:

$$x'_{pi} = X_L/x'_{zi}$$

both, zeros and poles are denormalized by: $x_{zi} = x'_{zi}.FB$, $x_{pi} = x'_{pi}.FB$

-the loss at any frequency w is :

$A(w) = 10 \log (1+e.R_n^2)$, where $C = 1/R_n(FB)$ and

$$R_n(w,L) = C.w^{1-NI} . \prod_{i=1}^{[N/2]} (w^2 - x_{zi}^2)/(w^2 - x_{pi}^2)$$

-routines for computation of $K(k) = \int_0^{\pi/2} dx/\mathrm{SQRT}(1-k^2\sin^2 x)$ and its complementary $K(k')$ and the elliptic sine $sn(u,k)$, are available from the keyboard without disturbing the stored zeros & poles. The following algorithms are used to compute $K(k)$ and $sn(u,k)$:

K(k) : INPUT k, $phi_o = PI/2$, $p_o = 1$, $x_o = \mathrm{SQRT}(1-k^2)$

LOOP :

$$x_{i+1} = 2/(1+\sin(\cos^{-1} x_i) - 1$$

$$p_{i+1} = p_i(1 + x_{i+1})$$

$$phi_{i+1} = \tfrac{1}{2}(phi_i + \sin^{-1}(\sin(\cos^{-1}x_i).\sin(phi_i)))$$

if $x_{i+1} = 0$, EXIT LOOP

THEN, $K(k) = p_{i+1} \cdot \mathrm{IN}(\mathrm{TAN}(\mathrm{TAN}^{-1}1 + \tfrac{1}{2}phi_{i+1}))$

sn(u,k); first, sn(u/8,k) is computed using a Taylor series expansion. Then the duplication fromula

$$sn\ 2u = (2sn\ u\ .\ \mathrm{SQRT}(1-sn^2u).\ \mathrm{SQRT}(1-k^2sn^2u))/(1-k^2sn^4u)$$

is applied 3 times to yield sn(u,k)

- this procedure is very fast (11 seconds) and accurate (about 9 decimals).

HOW TO USE : both cards work in radians mode

-card 1 does not depend on card 2 to perform its functions. In fact, card 2 is only for evaluate the loss once the zeros,poles (and order) have been computed.

-to find the minimum necessary degree N: load card 1 ,

A_{min},ENTER,A_{max},ENTER,FH,ENTER,FB, A → N

-it takes about 87 seconds (on the average)

-destroys previous zeros & poles stored in R8,R9

-to find zeros & poles of attenuation : once the degree has been computed:

press B → (1) → x_{z1} → x_{p1}
→ (2) → x_{z2} → x_{p2}
... ...
→ (m) → x_{zm} → x_{pm}

where m = INT(N/2) . This has to be done once N has been computed. It takes about 6 seconds per degree. All zeros, x_{zi}, are stored in R_1 thru R_{15}. Poles need not be stored, as they are function of the zeros. Maximum degree N is 30.

-to review zeros & poles : press C → (1) → x_{z1} → x_{p1} ,etc.

-to compute K(k): input k, press D : k , D → K(k)

-to compute K(k'): input k, press fD: k ,fD → K(k') = K'(k)

-it takes from 16 to 32 sec. 22 on the average. k should be between 10^{-4} and 1, not including. K(1) tends to infinite and K(0) tends to PI/2. Accuracy is 8 decimals or better. Its use does not alter stored zeros.

-to compute sn(u,k) : store k : k, fE → k
input u : u, E → sn(u,k)

-it takes 11 seconds. k must be positive. Accuracy is 8 decimals or better. It does not alter stored zeros.

-to compute the loss at any frequency : once the degree and zeros & poles have been computed using card 1. Now , load card 2. There are several options:

LBL A : computes the loss (dB) for a given w (Hz). Simply, input w : w , press A → A(w)

LBL B : sets linear sweep : B → 0.0000

LBL C : sets logarithmic sweep : C → 1.0000

LBL D : starts sweep of frequencies:
enter initial freq: w_o ENTER
enter increment : inc D → (w_o) → A(w_o) →
→ (w_1) → A(w_1) → ...

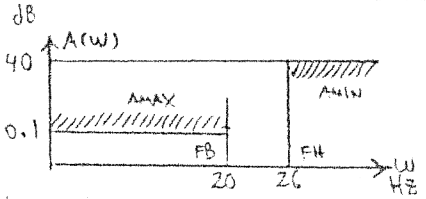
to stop the sweep, press R/S after A(w) is printed

remember that: linear : $w_{i+1} = w_i + inc$
logart : $w_{i+1} = w_i \cdot inc$

-linear sweep is selected when card 2 is loaded. Remember, stop sweep just after A(w) has been displayed (printed).

LBL E on card 2 performs exactly as LBL C on card 1 : review of zeros and poles. It is included for convenience, to avoid reloading card 1 for a simple revision.

EXAMPLE : Find the degree , poles, zeros, and plot the loss against the frequency, for an elliptic lowpass filter that is to meet the following requirements:



A_{max} = 0.1 dB
A_{min} = 40 dB
FB = 20 Hz
FH = 26 Hz

Degree: load card 1 ; 40 ENTER 0.1 ENTER 26 ENTER 20 A → 6.00 so, the minimum necessary degree to meet the requirements is 6, an elliptic lowpass filter of order 6.

loss as a function of frequency : We will make a plot of the loss as a function of the frequency. To make this plot, we need to know:

(1) zeros & poles : press B → (1) → 6.295011340 → 82.60509346
→ (2) → 15.62227766 → 33.28579938
→ (3) → 19.56561718 → 26.57723471
→ 0.00 zeros poles

(2) frequencies of maximum passband loss : these are

$f_{ei} = FB.sn(2iK/N)$, i=0,1,2,3 . Proceed as follows:

DSP 4 ; 0 E → 0.0000 (f_{e0})
RCL 0 ; 2, x, 6 , / , E , 20 , x → 11.6623 (f_{e1})
RCL 0 ; 4, x, 6 , / , E , 20 , x → 18.1792 (f_{e2})
RCL 0 , E , 20 , x → 20.0000 (f_{e3})

(3) frequencies of minimum stopband loss : these are

$f_{Ei} = FB.FH/f_{ei}$, so $f_{E1} = 44.5881$, $f_{E2} = 28.6041$
$f_{E3} = 26.0000$, f_{E0} = infinite

now, to compute the loss, load card 2 . Let's test if our poles,zeros,etc, are correct:

the zeros : 6.2950 A → 0.0000 dB ; f_{ei} : 0 A → 0.1000
15.6223 A → 0.0000 dB 11.6623 A → 0.1000
19.5656 A → 0.0000 dB 18.1792 A → 0.1000
20 A → 0.1000

etc, the same with the f_{Ei} → 46.8540 dB

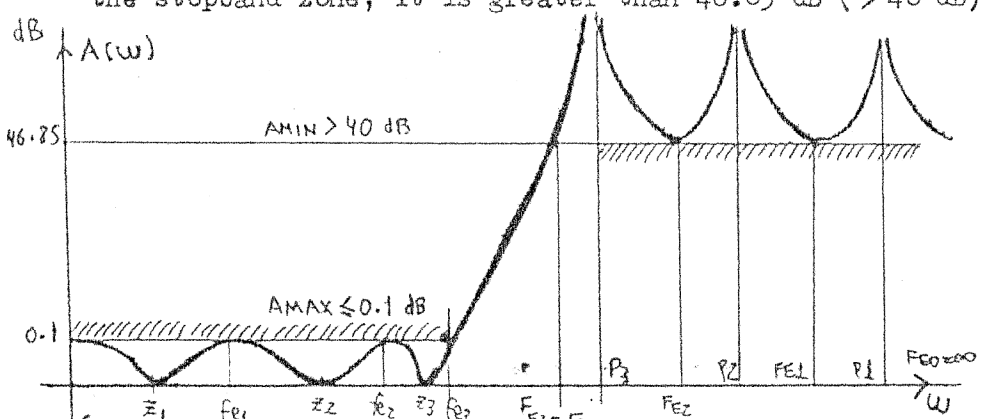
finally, a general sweep.Select linear sweep, starting with w_o = 0 Hz, and proceed with an increment of 2 Hz:

B → 0.0000 , 0 ENTER 2 D → (0.0000) → 0.1000
→ (2.0000) → 0.0783 → etc.

we form the table:

w	0	2	4	6	8	10	12	14
A(w)	0.1000	0.0783	0.0311	0.0006	0.0200	0.0748	0.0987	0.0434

etc, up to w = 36 Hz or so. It will be seen that in the passband zone, the loss does not exceed 0.1000, while in the stopband zone, it is greater than 46.85 dB (> 40 dB)



001	LBL A	31	25	11	068	/			81	135	$x \not\equiv y$		32	61
002	STO D		33	14	069	STO×8	33	71	08	136	GTO 0		22	00
003	X( ) Y		35	52	070	1			01	137	CLX			44
004	/			81	071	−			51	138	DSP 2		23	02
005	STO E		33	15	072	STO 9		33	09	139	RTN		35	22
006	Rdown		35	53	073	$X \not\equiv 0$		31	61	140	LBL e	32	25	15
007	GSB 8	31	22	08	074	GTO 3		22	03	141	STO E		33	15
008	STO A		33	11	075	RCL 7		34	07	142	RTN		35	22
009	X( )Y		35	52	076	2			02	143	LBL E	31	25	15
010	GSB 8	31	22	08	077	/			81	144	P( )S		31	42
011	/			81	078	1			01	145	SF 3	35	51	03
012	SQRT		31	54	079	TAN−1		32	64	146	SF 2	35	51	02
013	GSB 9	31	22	09	080	+			61	147	8			08
014	STO C		33	13	081	TAN		31	64	148	/			81
015	RCL E		34	15	082	LN		31	52	149	RCL E		34	15
016	GSB 9	31	22	09	083	RCL 8		34	08	150	x^2		32	54
017	RCL C		34	13	084	x			71	151	STO 9		33	09
018	/			81	085	P( )S		31	42	152	STO 7		33	07
019	1			01	086	RTN		35	22	153	STO 6		33	06
020	+			61	087	LBL B	31	25	12	154	1			01
021	INT		31	83	088	CF 0	35	61	00	155	+			61
022	STO C		33	13	089	RCL C		34	13	156	STO 8		33	08
023	RTN		35	22	090	2			02	157	STO×6	33	71	06
024	LBL 8	31	25	08	091	/			81	158	x^2 −		32	54
025	.			83	092	ENTER			41	159	RCL 7		34	07
026	1			01	093	INT		31	83	160	1			01
027	x			71	094	STO B		33	12	161	2			02
028	10^x		32	53	095	ST I		35	33	162	x			71
029	1			01	096	X=Y		32	51	163	+ −			61
030	−			51	097	SF 0	35	51	00	164	STO 7		33	07
031	RTN		35	22	098	LBL 6	31	25	06	165	RCL 8		34	08
032	LBL 9	31	25	09	099	RC I		35	34	166	x			71
033	STO 9		33	09	100	2			02	167	RCL 6		34	06
034	GSB D	31	22	14	101	x			71	168	5			05
035	STO 8		33	08	102	0			00	169	N!		35	81
036	STO 0		33	00	103	F? 0	35	71	00	170	STO 6		33	06
037	RCL 9		34	09	104	e^x		32	52	171	x			71
038	GSB d	32	22	14	105	−			51	172	+			61
039	STO/8	33	81	08	106	RCL C		34	13	173	x			71
040	RCL 8		34	08	107	/			81	174	7			07
041	RTN		35	22	108	RCL 0		34	00	175	N!		35	81
042	LBL D	31	25	14	109	x			71	176	/			81
043	P( )S		31	42	110	GSB E	31	22	15	177	x			71
044	SIN^{-1}		32	62	111	RCL D		34	14	178	RCL 7		34	07
045	COS		31	63	112	x			71	179	RCL 6		34	06
046	P( )S		31	42	113	STO(i)		33	24	180	/			81
047	LBL d	32	25	14	114	DSZ		31	33	181	−			51
048	P( )S		31	42	115	GTO 6		22	06	182	x			71
049	STO 9		33	09	116	LBL C	31	25	13	183	x			71
050	1			01	117	0			00	184	RCL 8		34	08
051	STO 8		33	08	118	ST I		35	33	185	6			06
052	SIN^{-1}		32	62	119	LBL 0	31	25	00	186	/			81
053	STO 7		33	07	120	ISZ		31	34	187	+			61
054	RCL 9		34	09	121	RC I		35	34	188	x			71
055	LBL 3	31	25	03	122	DSP 0		23	00	189	x			71
056	COS−1		32	63	123	PAUSE		35	72	190	1			01
057	SIN		31	62	124	DSP 9		23	09	191	−			51
058	ST I		35	33	125	RCL D		34	14	192	x			71
059	ISZ		31	34	126	x^2		32	54	193	CHS			42
060	RCL 7		34	07	127	RCL(i)		34	24	194	LBL 4	31	25	04
061	SIN		31	62	128	−x−		31	84	195	STO 6		33	06
062	x			71	129	RCL E		34	15	196	STO×6	33	71	06
063	SIN−1		32	62	130	x			71	197	2			02
064	STO+7	33	61	07	131	/			81	198	x			71
065	2 −			02	132	−x−		31	84	199	1			01
066	STO/7	33	81	07	133	RCL B		34	12	200	RCL 6		34	06
067	RC I		35	34	134	RC I		35	34	201	GSB 7	31	22	07

```
202 GSB 7   31 22 07   209 P()S        31 42 216 RCL 9      34 09
203 -               51 210 RTN        35 22 217 RCL 6      34 06
204 /               81 211 LBL 7   31 25 07 218 STOx6   33 71 06
205 F? 2    35 71 02 212 -                51 219 x            71
206 GTO 4      22 04 213 SQRT         31 54 220 RTN        35 22
207 F? 3    35 71 03 214 x                71
208 GTO 4      22 04 215 1                01
```

Initial status : FIX / DSP 2 / RAD / CF 0,1,2,3
don't forget to set RAD mode !

Registers : 0 = aux , 1 = x_{z1} , 2 = x_{z2} , ... , S5 = x_{z15}
S6 = aux ,S7 = aux ,S8 = aux , S9 = aux
A = e , B = used, C = N , D = FB , E = $\frac{1}{X_L}$
I = index

ELLIPTIC LOWPASS FILTER DESIGN - by V. Albillo -CARD 1

1 | k → K(k) STO k | 2

[A] →DEGREE | →ZER&POL | REVIEW | k → K(k) | u → sn(u,k)

CARD 2

```
001 LBL A   31 25 11   039 -              51 077 LBL 5   31 25 05
002 DSP 4      23 04   040 x              71 078 PAUSE      35 72
003 F? 1    35 71 01   041 RCL B        34 12 079 STO 8      33 08
004 GTO 0      22 00   042 RCL D        34 14 080 P()S       31 42
005 LBL 3   31 25 03   043 x²           32 54 081 GSB A   31 22 11
006 GSB 1   31 22 01   044 RCL(i)       34 24 082 -x-        31 84
007 RCL 0      34 00   045 /              81 083 0            00
008 x               71 046 RCL E        34 15 084 P()S       31 42
009 1               01 047 /              81 085 RCL 8      34 08
010 +               61 048 x²           32 54 086 RCL 9      34 09
011 LOG        31 53   049 -              51 087 F? 0    35 71 00
012 1               01 050 /              81 088 x            71
013 0               00 051 DSZ          31 33 089 +            61
014 x               71 052 GTO 2        22 02 090 GTO 05       22 05
015 RTN        35 22   053 x²           32 54 091 LBL E   31 25 05
016 LBL 0   31 25 00   054 RTN            35 22 092 0            00
017 CF 1    35 61 01   055 LBL 4   31 25 04 093 ST I          35 33
018 RCL A      34 11   056 CF 2    35 61 02 094 LBL 6   31 25 06
019 STO 0      33 00   057 RCL C        34 13 095 ISZ          31 34
020 Rdown      35 53   058 2              02 096 RC I          35 34
021 STO A      33 11   059 /              81 097 DSP 0         23 00
022 RCL D      34 14   060 ENTER          41 098 PAUSE         35 72
023 GSB 1   31 22 01   061 INT          31 83 099 DSP 9         23 09
024 STO/0   33 81 00   062 X=Y          32 51 100 RCL D         34 14
025 RCL A      34 11   063 SF 2    35 51 02 101 x²           32 54
026 GTO 3      22 03   064 RTN            35 22 102 RCL(i)      34 24
027 LBL 1   31 25 01   065 LBL B   31 25 12 103 -x-          31 84
028 x²          32 54   066 CF 0    35 61 00 104 RCL E       34 15
029 STO B      33 12   067 0              00 105 x            71
030 GSB 4   31 22 04   068 RTN            35 22 106 /            81
031 ST I       35 33   069 LBL C   31 25 13 107 -x-          31 84
032 RCL B      34 12   070 SF 0    35 51 00 108 GSB 4   31 22 04
033 F? 2    35 71 02   071 1              01 109 RC I          35 34
034 1               01 072 RTN          35 22 110 X≠Y          32 61
035 LBL 2   31 25 02   073 LBL D   31 25 14 111 GTO 6         22 06
036 RCL B      34 12   074 P()S         31 42 112 CLX             44
037 RCL(i)     34 24   075 STO 9        33 09
038 x²          32 54   076 X()Y          35 52
```

Initial status : FIX / DSP 4 / RAD / CF 0, SF 1, CF 2, CF 3

VALENTIN ALBILLO (4747)

ELLIPTIC LOWPASS FILTER DESIGN - by V. Albillo - CARD 2

1 | AUTOMATIC SWEEP | 2

[A] w → LOSS | LINEAR | LOGARITHM | w_0 ↑ INC | REVIEW

- this are some inputs to your column "ROM PROGRESS"

- -

Is microcode possible in the PPC Custom ROM ? This has not been told to the membership. I think it is not possible, but a note on the subject would be in order. If microcode is possible, would it be convenient ? Microcode level is beyond the possibility of most PPC members. Would HP have to write it ? What would be the cost ? If it is too high, it could be impractical. However, could the PPC ROM contain a mixed user language-microcode routines ? this is , some conventional language routines and some microcode level ones. This is possible: the Printer ROM has microcode routines (REGPLOT for instance) together with conventional routines (PRPLOT for instance). What would be the cost, in that case ? It is quite clear that microcode has a clear advantage over conventional programming: it is 1000 times faster, at least. It is also more efficient in program memory useage.

I think that the ROM Project Committee will do well throwing a little light about all this. Meanwhile, here are some suggestions about routines to be implemented in microcode level :

- recall arithmetic: it will shorten greatly programs (lines, not bytes) and would help to save bytes & time
- ASCII conversions
- full string comparisons : X(Y? , X)=Y? for instance
- programmable SIZE
- linear regression (a la HP-34c) : I will never understand why HP did include routines like CCT, DEC , CLST , etc, into the 41c mainframe, and forgot linear regression.
- XROM generation : to program XROM functions without the peripheral being present
- and of course, program lenght (bytes),SIZE finder, etc.

VALENTIN ALBILLO (4747)

- -

- Reading the ROM ideas suggested in "old" V6 N8 P15, I notice the request for a Display Test routine, defined as "turns on all segments of the display for test purposes".

Here presented is a routine which completely test the display, by turning on all of its segments & annunciators, and exercises left & right scrolling and other display functions such as AVIEW, CLD . This routine, called TD (test display) should be considered for its inclusion in the PPC Custom ROM . It uses no registers, restores the status of all flags after completion, leaves X,Y,Z unchanged, and it's exactly 112 bytes long, so it fits onto a single side of a mag card. It requires no bugs, only synthetic instructions, uses no subroutine level, and can be used as a subroutine, as it does not set the program mode flag 52.

```
01 LBL"DT"      12 AVIEW      23 AVIEW      34 GTO 00
02 "■乙天乙天|匕"  13 CLX        24 PSE        35 X( ) M
03 RCL M        14 80         25 X( ) L     36 STO d
04 X( ) d       15 LBL 01     26 STO M      37 CLX
05 STO L        16 DSE X      27 CLX        38 CLA
06 ACN          17 GTO 01     28 11         39 CLD
07 "■:■:■:"     18 "SCROLL"   29 STO N      40 RDN
08 ASTO X       19 AVIEW      30 SF 99      41 .END.
09 ARCL X       20 see note   31 LBL 00
10 ARCL X       21 see note   32 SIN
11 ARCL X       22 SF 25      33 DSE N
```

notes : line 02 is F7F80D013C012140

TEXT 7 ■ 乙 天 乙 天 | 匕

line 07 is F6023A023A023A (text (boxst,:,bxst,:,bxst,:))
line 20 is FD2020202020202020202020022C , this is , a line text(11 spaces, box star, comma)
line 21 is "append 11 spaces" (this is, a text line which appends 11 spaces to the contents of the ALPHA register.

How works: lines 02-04 creates the NNN F80D013C012140 which is stored in the flag register d. This causes the indicators for BAT, USER, GRAD, SHIFT, 0,1,2,3,4 to appear in the display. The PRGM annunciator turns itself on by the mere fact of executing the routine. Line 05 saves the status of all flags, to be restored later. line 06 turns on the alpha annunciator. Lines 07-12 display all segments of the display on, except for the comma's tail. This includes everything in the display. Lines 14-17 provide a delay of 8 seconds, plenty of time to see if some segmentfails to turn on. Lines 18-24 scroll a box star and a comma (not tested before) to the left. Lines 25-34 scroll them to the right. lines 35-36 serve to restore the status of all flags as they were before executing this routine. lines 37-40 clear the display and alpha (to avoid garbage), and restore X,Y,Z as they were before calling DT. Only T (and LASTX) are lost.

How to use : load the program.

XEQ"DT" → all segments in the display (except comma tails) are ON for 8 seconds. Then SCROLL appears briefly (while all status annunciators remain ON) , and a box star and a comma begin to scroll from the right to the left. When they reach the left side, they reverse its motion towards the right. As soon as they reach the right side, the display returns to its condition previous to executing the test. Only T and LAST X are lost (as well as ALPHA).No registers or flags are disturbed.

VALENTIN ALBILLO (4747)

- this is an article : it shows how STO b, RCL b may be used to advantageously substitute the byte-jumper function. Several uses of RCL b, STO b are shown: create any line of text, create any synthetic instruction, unconditional jumping to any location, and how to break PRIVATE. This last point may be deleted, at your option, if you don't want to reveal the secret to the membership. I think it is the easiest method possible. Of course, I would prefer that you include this technique as well as the others

xxx STO b , RCL b xxx

or

xxx THE UNIVERSAL BYTE-JUMPER xxx

Last Wickes article , Byte-jumping , was nothing less than a - revelation. Its byte-jumper function finally allowed a bug-free machine the possibility of synthetic programming without a card reader being required. See V7 N4 P26-27-28 for a full description of the byte-jumper function and its generation.

Now, I want to present to the membership another kind of byte-jumper : STO b , RCL b . This is, primary control over the status register b. It will be seen in this article how - those 2 functions allow the user to do everything the byte - jumper is capable to do (except for recalling without normalization directly), plus several other functions , and with greater ease.

All necessary tools to follow the given examples is to have STO b, RCL b assigned to keys for its execution in USER mode. Now, let's go:

(1) Arbitrary text line

- We are going to create an arbitrary text, using standard or non-standard characters as a program line. As an example, we will create the line 06 : text composed of full man, underscore, box star (HEX 02) , ampersand , mu , null (HEX 00)
- first, looking at the HEX TABLE (see V6N5P22-23), write a table with the equivalent instructions for the characters you want :

full man	=	LBL 00
underscore	=	DEC
box star 02	=	LBL 01
ampersand	=	RCL 06
mu	=	LBL 11
null	=	null

- now, go to line 05 of your program. Switch to PRGM mode. press SIN , STO IND 18 . Now introduce any text of 6 characters, say "ABCDEF". BST ,and see 07 STO IND 18 . PACK and switch to RUN mode. Set USER mode and press RCL b . Out of USER mode. Switch PRGM mode, BST , see 06 SIN. Delete the SIN pressing backarrow. PACK . Out of program mode. Set USER mode. Press STO b . Out of USER , switch to PRGM. See 05 ST+ IND N. Now, introduce the instructions you listed before : be careful not to make a mistake:
  press LBL 00 , DEC , LBL 01 , RCL 06 , LBL 11

  the nulls are a little more difficult. If there is a null in your text, insert any 1-byte instruction in its place, then delete it later. To create the null, out of PRGM , ,set USER, STO b , switch to PRGM, see 10 ST+ IND N, mow SST 6 times, see 16 - . Delete that line. Delete the 05 STO IND 18, as the work is finished. SST and see the synthesized line 天_Ⅲ&μ¯ . The line number is arbitrary. Now, delete the 6 lines following the synthesized one, as they are mere garbage (the A,B,...,F). This last part wasn't really neccessary. It was only to show that, if you make an error, you can re-start by simply pressing STO b again ,

-why does it work ? Deleting the SIN and storing back in Rb its previous contents, causes the program pointer to point one byte ahead. This causes the address IND 18 to be considered as a stand-alone, thus becoming ST+ , which in turn, takes the "text 6" bytes as its address, thus becoming IND N, so ST+ IND N is seeing as a whole instruction, and, as the text 6 is not present now, additional characters may be - inserted as normal instructions. Backstepping results in the IND 18 becoming again part of the STO, and the text 6 to be free, thus creating the arbitrary text.

(2) Synthesize any instruction

-let us synthesize an arbitrary instruction as a line in a program. As an example, let us create the line 04 SCI IND d

-go to the line 03 in your program.Press SIN, STO IND 18. We want SCI IND d, so any SCI will do. Press SCI 9. PACK . BST and see STO IND 18. Out of PRGM . Set USER. Press RCL b . Out of USER . Set PRGM . BST and see 04 SIN . Delete the SIN using backarrow. PACK . Out of PRGM. Set USER. Press STO b . Out of USER. Set PRGM . See ST+IND 29. We want SCI IND d, and we already have the SCI. Now, the IND d, - looking in the HEX TABLE; it results that IND d is a text 14, so introduce now any text of 14 characters, say: "ABCDEABCDEABCD", BST and see STO IND 18. The work is done, so delete it using backarrow, SST and see SCI IND d. The line number is arbitrary. The 14 instructions following the SCI IND d are the A,B,...D .
it works as the previous example, deleting the SIN causes the program pointer to jump one byte (not exactly, is the -program which jumps one byte backwards when being packed)

(3) Assign any function to a key

This is done following the guidelines of Wickes in its byte-jumping article in V7N4P26: as working directly on the assingments registers is not practical (a single byte added clears the whole memory), all synthetic assignments are made creating a program to assign the particular keys. This program is just this: a text line which represents the contents of the assignment register we are going to create, RCL M to bring this text to the display, another text which is the future contents of Rc to change the curtain location, then X( )c instructions to store and recall from Rc. The contents of Rc are changed, the byte representation of the assignment register is stored in R00, then Rc is restored to its initial contents. All synthetic text and instructions can be easily created as it was shown in (1) and (2).

(4) Unconditional jump

Whereas the byte-jumper function allowed to skip from 1 to 15 bytes, there are no limits to the number of bytes skipped using STO b. You simply place the address of the desired location into Rb, and the program pointer is instantly at that location. No label searching, no nothing.

```
 28 RCL b      Consider this extract of a program. Lines 28-29
 29 STO M      store the current pointer address onto M. Lines
    ...        176-177 store that address back into b . As soon
    ...        as line 177 is executed, the program is executing
176 RCL M      line 29 again instantly. It does not matter if
177 STO b      lines 177 -176 are in another program. The execu-
- - - - - - -  tion resumes in line 29 of the other program.
```

```
 01 LBL"AB"     01 LBL"PF"      In this other example, though
    ...            ...         AB and PF are separate programs,
 34 RCL b       08 X≠Y?         the execution will pass from li-
 35 STO M       09 RCL M        ne 10 of PF to line 35 of AB in-
    ...         10 STO b        mediately if the test is meet.
 97 GTO"PF"                     If not, it will be transferred to
    ...      some other place, under program control.
 85 END
```

this technique allows great flexibility in the execution of loops. Jumping using STO b is not restricted by the fact of END , alpha labels, independent programs, or the 112 byte restriction of local labels. Execution can be transferred to any place in the 41c memory. Besides, it is instantaneous. This can be a definite advantage.

Consider, for instance, a main program "MAIN" which uses an independent program "AUX" as an auxiliar routine. To execute "AUX" within a loop, the main program must do XEQ "AUX" , thus an alpha label search will be performed within the iterative loop. This can be quite time consuming. However, the program AUX may be structured so that, depending on a flag, which is set by the main program, it stores the address of the beginning of the instructions which are relevant to the main program into an status register, say M (to avoid normalization). Then, after the first execution of AUX, the main program - simply put the contents of M into b , and the jump is instantly performed, thus saving time at very little cost. PACK will give no trouble, as the program MAIN is supposed to run without stop.

5) PRIVATE feature defeat

The instructions STO b, RCL b , prove - useful to break the PRIVATE feature. Any PRIVATE program can be seen, edited, re-recorded (without PRIVATE this time) etc, using simply STO b, RCL b, in an incredibly easy fashion.

To see a PRIVATE program, proceed as follows:

(a) GTO .. , switch to PRGM mode , switch to RUN mode.
(b) load the PRIVATE program . Do nothing but (c),(d),etc
(c) set USER , RCL b , out of USER
(d) GTO ..
(e) set USER , STO b , out of USER
(f) now, set PRGM . You are at the top of the PRIVATE program. You may see it using SST . If you reach the END using SST, the pointer wraps-around, and you will be back at the beginning. If you now attempt to BST, DEL, GTO .nnn,insert, etc, the program will become PRIVATE again, but the insertion, deletion, etc, will be executed.

So, the best warning is : use only SST to get thru a unPRIVATEd program in PRGM mode. If you want to make changes, while in PRGM mode, pass cards thru the card reader. The program will be recorded onto them, but this time it will not be recorded as a PRIVATE program, so, once recorded, delete the PRIVATEd version from memory, load the UNPRIVATED version, and make any desired changes, or print the program, etc.

If you execute BST while seeing a PRIVATE program, or attempt to insert,delete,GTO..,the program will become PRIVATE once more (and your insertion,deletion,... will be performed). To regain access, suposing the contents of Rb are undisturbed in the display, simply go to somewhere in program memory where you can see the display without the message PRIVATE on it (for instance, if you have a program "PEPE" which is not PRIVATE, press GTO "PEPE" in RUN mode).See actually program memory (set PRGM mode) , then out of PRGM , STO b , and set PRGM mode again. You are once more at the top of the PRIVATE program.

That's all, folks ! Hope you'd like it

VALENTIN AIBILLO (4747)

25 - DIFFERENTIAL EQUATIONS : $y'' = f(x,y)$

```
01 RCL 2      30 10^x  -
02 RCL 1      31 STO- 7
03            32 x
... f(x,y)    33 STO+ 6
... GTO 15    34 RCL 2
15 RCL 5      35 STO- 6
16 x     -    36 RCL 4
17 RCL 7      37 STO 2
18 X(0        38 STO+ 6
19 GTO 45     39 STO+ 6
20 X≠0        40 RCL 0
21 GTO 30     41 STO+ 3
22 e^x   -    42 RCL 6
23 STO 7      43 RCL 3
24 Rdown      44 GTO 03
25 STO 6      45 STO- 7
26 RCL 4      46 Rdown
27 RCL 3      47 RCL 6
28 STO 1      48 +
29 GTO 03     49 STO 4
```

registers:

0 h	4 y_k
1 x_{k-1}	5 $h^2/12$
2 y_{k-1}	6 $\hat{y}_{k+1}$
3 x_k	7 flag

Theory : this program solves numerically a 2º order differential equation of the form:

$y'' = f(x,y)$ subject to 2 initial conditions. Notice - that y' does not appear. This is a special case of the more general problem:

$y'' = f(x,y,y')$.

The program uses a 5-th order method: the Numerov's method.

-predict: $\hat{y}_{k+1} = 2y_k - y_{k-1} + \frac{h^2}{12}(10\, f_k + f_{k-1})$

-correct: $y_{k+1} = \hat{y}_{k+1} + \frac{h^2}{12} f_{k+1}$

where $f_n = f(x_n,y_n)$. Two initial values are required, $(x_o,y_o),(x_1,y_1)$, and the spacing is h.

Characteristics: all registers used
-12 steps remain to define f(x,y)
-using h=0.1, about 5 decimal places of accuracy. h is a user's selected increment. A small h means more accuracy and running time. See examples.

HOW TO USE : store constants: h STO 0 (only once)
$h^2/12$ STO 5 (only once)
(this step, only when loading) 0 STO 7 (only once)
x_o STO 1 , x_1 STO 3
y_o STO 2 , y_1 STO 4

-define f(x,y): GTO 02, switch to PRGM, f(x,y) , GTO 15 , switch to RUN , f PRGM

f(x,y) should assume that x is in X and y in Y

-compute successive values: y_2 ,y_3, ...

R/S → y_2 , R/S → y_3 , ... , R/S → y_n , ...

remember, $x_1 = x_o + h$, $x_2 = x_1 + h$, etc, and $y_n = f(x_n)$

after computing each y_k, its corresponding x_k is in R3.

EXAMPLE Solve $y'' = x + y$, $y(0) = 1$, $y'(0) = 1$, using $h = 0.1$, find $y(1)$.

We have $(x_o, y_o) = (0,1)$. We need $y(0.1)$. Using the diff. equation, we have: $y(0) = 1$, $y'(0) = 1$, $y''(0) = 0 + 1 = 1$, $y'''(0) = 1 + y' = 1 + 1 = 2$, $y'''' = y'' = 1$. Using the Taylor series expansion:

$$y(0.1) = 1 + 1(0.1) + \tfrac{1}{2}(0.1)^2 + \tfrac{2}{6}(0.1)^3 + \tfrac{1}{24}(0.1)^4 = 1.10534$$

so, now: $h = 0.1$ STO 0 , $(0.1)^2/12$ STO 5 , 0 STO 7
$x_o = 0$ STO 1 , $y_o = 1$ STO 2 , $x_1 = 0.1$ STO 3
$y_1 = 1.10534$ STO 4 .

-to define f(x,y): GTO 02, switch to PRGM , + , GTO 15 , switch to RUN, f PRGM .

-now, R/S → 1.22274 , R/S → 1.35438 , ... , R/S → 2.89344
we, thus, form the table:

x	0.2	0.3	0.4	0.5	0.6	...	1.0
y	1.22274	1.35438	1.50258	1.66982	1.85877	...	2.89344

-the exact solution is $y = \frac{1}{2}(3e^x - e^{-x}) - x$
so $y(1) = 2.89348$. We've got about 5 places.