Notes on the back story of this letter:

This *7-page* letter is one of the first I sent to ***Richard Nelson***, who was in charge of *PPC* back then. I wrote several others once my *PPC* subscription started for good in *January 1980* (about one per month) but this is the earliest survivor. Sadly, all former ones were either lost or (more likely) I didn't photocopy them because that was quite unaffordable for my meager budget at the time.

Be it as it may, this was *still* an enthusiastic letter though by then, seven months after I sent the first one, very few of my contributions had been published, which I resented.

In this letter I expressed my hope that eventually more of them would be, and included six pages with my newest contributions to the ongoing *PPC ROM Project*, optimized *HP-41C* conversion routines for *HP-67/97* functions in the absence of the very expensive card reader, inputs for the *NOP* and *BIT & PIECES* columns, a nice *Computer Challenge*, and a little article I'd written about *HP-41C Synthetic Programming Techniques*. Regrettably, ***Mr. Nelson*** never wrote back a single word of thanks, *ACK*, comments, or just encouraging me to continue my efforts, not this time, not ever.

Additionally, though all my contributions were quite original when I sent them, by the time they saw publication (the ones that did, that is) they were somewhat outdated and their effect was severely diminished, which is what happens when you find/create something new, try to announce it to the community while it's hot, then your materials spend months gathering dust in a drawer so that when they're eventually published (if at all), they're old hat by then.

Last but not least, at 7-page-weight intercontinental mail rates plus the photocopies' cost, sending and keeping a copy of this letter was *very expensive* to me and I grew weary at the possibility of it being utterly ignored, as most of the previous ones were. Fortunately, this time some of the contents submitted in this letter were indeed published ! Which encouraged me to send much more ...

Valentin Albillo, 30-10-2021

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

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

Dear Richard :

A month has elapsed since my last letter/contribution. I haven't yet received the June issue (today is June 30), it will arrive by July 6, probably, and I can't wait to get my hands on it ! I am sure it will be as good (if not better) as the previous issues, and I hope some of my contributions will be published in it, so I'm really anxious to get it.

Here included are several things:

1) Several routines for the Custom-ROM : a) an improvement to the Block exchange routine by John Kennedy, published in V7N3P5. The improved routine is shorter & faster.
b) flag 55 toggle & block revert routines. The former is essential to all those people without a printer which need to use Application ROM programs as subroutines. The latter, typically useful to speed sort routines.
c) A set of 67/97 conversion routines : these include P()S, CLREG , RCLΣ, FIX, SCI, ENG, ISZ, DSZ, ISZI, DSZI . All of them leave the stack unaltered, use no register, restore all flags, and execution time is 1 second or less. Magnetic card included. The short version of P()S is in the card. The long one is not.

2) Several inputs for NOP
a) a bug in the card reader ROM
b) a possible bug of the PACK function
c) Odd behaviour of editing functions applied to multi alpha-label programs. Notice the slowness of most functions. Notice the 7 minutes required to clear a 1786 bytes program using CLP. This is not a bug, but it can give trouble if memory is increased.

3) 3 inputs to BITS & PIECES : 2 simple routines, which avoid reinventing the wheel, and a Computer challenge. This one is tremendously tricky. Despite the simple and innocent aspect of the equation, its real root rounds to 7.000000000 , to 10 places, so to give the value of x-7, it is necessary to cope with the 10 missing places, so double-precision operations are needed: double precision square root, DP logs , DP addition & multiplication, DP division, and finally, DP PI = 3.141592653589793238 4... . Let's see if some member finds the asked value to 10 places in SCI. The solution, next month.

4) An article about some kind of SYNTHETIC PROGRAMMING TECHNIQUES . The techniques pointed are new to most members, have never been previously mentioned in the Journal explicitly, and allow great savings in program memory , as well as incredible flexibility. They are:
a) any NNN as a line in a program (uses synthetic texts)
b) constants as lines in programs: how to save bytes & time : it shows how to save as many as 6 bytes per constant (even 7 if more than one), and reduce digit entry time by a factor of 5. Uses synthetic texts.
c) Non-stack set up of indexes : shows how to initialize up to 3 indexes for indirect use of up to 3 registers, loading any constants without disturbing the stack. Uses synthetic texts.

That's all . I believe the SYNTHETIC TECHNIQUES to be very useful. I show practical uses of them to save bytes, etc. This article shows how synthetic functions incredibly increase the power of the 41c in most useful applications. Both, beginners and experienced programmers are going to take advantage of those techniques.

Till the next letter. Sincerely

(4747)

- This are some inputs to the column ROM PROGRESS : they are several rotuines to be considered for the PPC Custom ROM

- The following are some routines I have developed for its inclusion in the PPC Custom ROM. They are:

  - an improvement to the Block exchange routine published in V7N3P5, by John Kennedy.
  - 67/97 conversions , featuring P( )S, ISZ, DSZ, ISZI, DSZI, FIX; SCI; ENG; RCLΣ, and CLREG , as required in V6N8P15
  - some general purpose rotuines: flag 55 toggle, revert registers (useful for changing an ascendent sort to a descendent one)

A) Improvement to Block Exchange

```
01 LBL"XC"
02 SIGN
03 LBL 00
04 X( ) IND Y
05 X( ) IND Z
06 X( ) IND Y
07 ST+ Z
08 ST+ Y
09 DSE L
10 GTO 00
11 RTN
```

It exactly meets the same specifications as the previous routine by John Kennedy. Uses no registers, except the stack. It is 11 lines (3 lines shorter), and only 23 bytes (4 bytes less). It is also slightly faster: John Kennedy's routine exchanges 122 registers in 32 sec. This routine does the same in - 30 seconds. It uses no synthetic functions at all, and leaves ALPHA undisturbed. The user's instruc - tions for John's routine apply as well for this - routine. See V7N3P5-6 .

B) Flag 55 toggle

```
01 LBL"55"
02 CLA
03 STO N
04 CLX
05 RCL d
06 STO M
07 "⊢xxxxxx"
08 X( ) M
09 STO d
10 FC?C 15
11 SF 15
12 X( ) d
13 STO M
14 "⊢xx"
15 X( ) N
16 STO d
17 X( ) 0
18 CLA
19 RTN
```

Most HP-41c owners without a printer, who - own an Application Pac (say MATH PAC) know that most programs cannot be used as subroutines, as they set flag 21, and output results using VIEW, or AVIEW. This, in the - absence of a printer, stops program execution. The following routine overcomes this: it sets or clears flag 55 (the printer existence flag) depending on its previous state. If flag 55 is set, VIEW, AVIEW will not stop program execution, thus allowing the ROM routines to be executed as subroutines of your main program.

This routine is 45 bytes long: leaves X,Y,Z,T,L unchanged; does not change status of any flag (except 55), and uses no register. The ALPHA register is used, then cleared. Execution time is less than 1 second. To use, simply XEQ "55" . If flag 55 was clear, - it will be set now ; if it was set, it will be cleared.

C) Block Revert

```
01 LBL"RV"
02 ENTER
03 ENTER
04 FRC
05  E3
06 x
07 ST+ Y
08 X()Y
09 2
10 /
11 INT
12  E3
13 /
14 +
15 X()Y
16 LBL 00
17 X( ) IND Y
18 X( ) IND Z
19 X( ) IND Y
20 ISG Z
21 DSE Y
22 GTO 00
23 RTN
```

It seems that one or several sort routines will be included in the Custom ROM. It is obvious that a sort routine(s) has to do either an ascendent or descendent sort. This can be accomplished using - flags, or some kind of test. However, for large arrays, a test inside a loop is very time consuming on the long run. So I propose that all sorts included must be either ascendent or descendent but not both. The present routine, RV, transforms a descendent sort to ascendent, or viceversa. It reverts any - specified block of consecutive registers, using the standard bbb.eee in X. bbb is the address of - the first register in the block, and eee is the address of the - last register. This number, bbb.eee is returned back to the X register once execution has completed. The routine is 37 bytes - long, uses no register. It is incredibly fast: reverts 100 registers in 13 seconds, 200 registers in 25 seconds. For instance, if 4.008 is in X , after the routine is executed, the contents - of R08 is in R04, R07 is in R05, R06 is in R06, R05 is in R07, and R04 is in R08.

Obviously, Compatibility functions must be included in the Custom ROM, so that 41c owners without a card reader can key in 67/97 programs in a straight-forward way without regard to rewrite the program, or program conversion routines, such as P( )S, etc. However, I think those routines must be microcode level, so that they will be as efficient as they can. Anyway, it is not neccessary to rewrite the microcode. It is already written and included into the card reader ROM, so why not duplicate it in the PPC Custom ROM ? I believe that the price will be reasonable: no need to write microcode, just duplicate existing functions into another ROM.

If this hardware solution is not feasible, here included are some software alternatives, to be considered for its inclusion into the Custom ROM

a) P( )S

To interchange Primary (00 thru 09) with Secondary (10 thru 19) registers.

short version

```
01 LBL"P( )S"
02 see note
03 LBL 00
04 X( ) IND N
05 X( ) IND M
06 X( ) IND N
07 ISG N
08 X( )Y
09 ISG M
10 GTO 00
11 CLA
12 RTN
```

There are 2 possibilities: a short & a longer routines. The short version, here included, is only 32 bytes long. Leaves the whole stack, X,Y, Z,T,L, unaltered. Uses no registers, and disturbs nothing except the ALPHA register, which is used, then cleared. It uses synthetic functions and text.

note: line-02 天-▯---天 is a synthetic text: it is F701001900000001 . It simultaneously places 10.019 in M and 0 in N, to be used as indexes, without disturbing the stack.

Execution time is 3 seconds.

long version

```
01 LBL"P( )S"   09 X( ) 12   17 X( ) 05   25 X( ) 07
02 X( ) 00      10 X( ) 02   18 X( ) 15   26 X( ) 08
03 X( ) 10      11 X( ) 03   19 X( ) 05   27 X( ) 18
04 X( ) 00      12 X( ) 13   20 X( ) 06   28 X( ) 08
05 X( ) 01      13 X( ) 03   21 X( ) 16   29 X( ) 09
06 X( ) 11      14 X( ) 04   22 X( ) 06   30 X( ) 19
07 X( ) 01      15 X( ) 14   23 X( ) 07   31 X( ) 09
08 X( ) 02      16 X( ) 04   24 X( ) 17   32 RTN
```

This version is 69 bytes, which is about twice as long as the short one. However, it has two main advantages: first, it disturbs nothing, not even the ALPHA register. Its operation is exactly as the true P( )S function: the stack remains unaltered, etc,etc. Second, it is 4 times faster than the short version: it executes in 0.8 seconds. Of course, it is twice as long, but this is transparent to the user of the ROM. Besides, P( )S is very frequently used within a loop, so the greatest speed is required. This long version is the one to be preferred for the ROM

b) CLREG

To clear Primary registers: 00-09, 20-25 (0-9, A-I)

```
01 LBL"CLREG"   11 STO 07
02 STO 25       12 STO 08
03 CLX          13 STO 09
04 STO 00       14 STO 20
05 STO 01       15 STO 21
06 STO 02       16 STO 22
07 STO 03       17 STO 23
08 STO 04       18 STO 24
09 STO 05       19 X( ) 25
10 STO 06       20 RTN
```

Following the efficiency guidelines of the long version of P( )S, this routine is 35 bytes long. It disturbs nothing not even the ALPHA register. Leaves the stack X,Y,Z,T,L, unaltered. Uses no register and executes in less than a second. If SIZE 026 or greater is not allocated, gives NONEXISTENT without clearing or disturbing anything.

c) RCLΣ

To recall Σx and Σy simultaneously

```
01 LBL'RCLΣ"    06 RDN
02 X( ) 16      07 RCL 16
03 X( ) 16      08 RCL 14
04 STO L        09 RTN
05 RDN
```

This 20 bytes routine brings Σx to the X register, Σy to the Y register. Z,T remain unaltered. X is copied into LAST X as in the original RCL Σ (67/97) Executes in .5 seconds.

The first two lines make sure that NONEXISTENT appears if R16 is not allocated before the stack is disturbed.

d) FIX , SCI , ENG — To change the mode without changing the selected number of digits.

```
01 LBL"7FIX"  09 RCL d     17 LBL"7SCI"  25 STO M
02 XEQ 02     10 STO M     18 XEQ 02     26 "⊢xx"
03 SF 00      11 "⊢xxxxx"  19 GTO 01     27 X( ) N
04 GTO 01     12 X( ) M    20 LBL"7ENG"  28 STO d
05 LBL 02     13 STO d     21 XEQ 02     29 X( ) 0
06 CLA        14 CF 00     22 SF 01      30 CLA
07 STO N      15 CF 01     23 LBL 01     31 RTN
08 CLX        16 RTN       24 X( ) d
```

This routine is 83 bytes long. It uses no registers, leaves the stack X,Y,Z,T,L, undisturbed, do not change the status of any flag (except the mode flags which select FIX,SCI, or ENG, of course). The ALPHA register is used, then cleared. Executes in 1 second. The labels 7FIX, 7SCI, 7ENG cannot be changed to simply FIX,SCI , ENG, or else the mainframe 41c FIX,SCI,ENG, will not be accesible using XEQ. All users flags are restored: only mode flags 40 & 41 are changed as appropriate.

e) ISZ , DSZ, ISZI , DSZI — To increment, decrement the I-register (R25), or the register indirectly pointed by the contents of the I register (R25). These are the same as the 67/97 functions ISZ, DSZ, ISZ(i) , DSZ(i). The register is first incremented or decremented by 1, then tested. If its value is then between -1 and +1 (not included), it is considered as zero, and the skip condition is activated, as in the 67/97.

```
01 LBL"DSZI"  12 -1          23 FC? 04      34 CHS
02 CF 04      13 GTO 03      24 ST+ IND 25  35 X(Y?
03 GTO 04     14 LBL"ISZ"    25 RDN         36 SF 04
04 LBL"ISZI"  15 SF 04       26 STO N       37 X( ) N
05 CF 04      16 LBL 05      27 RDN         38 X( )Y
06 GTO 05     17 STO M       28 1           39 X( ) M
07 LBL"DSZ"   18 CLX         29 FS? 04      40 CLA
08 SF 04      19 1           30 RCL 25      41 RTN
09 LBL 04     20 LBL 03      31 FC?C 04
10 STO M      21 FS? 04      32 RCL IND 25
11 CLX        22 ST+ 25      33 X(0?
```

The routine is 89 bytes long. It leaves the whole stack, X, Y, Z, T, L, unaltered. It uses no registers. The ALPHA register is used, then cleared. Remember, 1 is added or sustracted of either R25 (I) or the register pointed by R25. The contents of the register are then tested to see if they are between -1 and +1 (limits not included). As it is impossible for the routine to directly skip or not the step following the call , the outcome of the test is given by flag 04 (the 67/97 has only flags 0,1,2,3) : if flag 04 is set, the following step must be skipped. If flag 04 is clear, no skip must take place. In other words, first call the routine (ISZ, DSZ, ..., as desired) , then test flag 04. If set, skip.

```
67/97        41 c
...          ...
...          ...
ISZ          XROM"ISZ"
GTO 1        FC?C 04
SIN          GTO 01
....          SIN
....         .....
```

This example, should make it clear. The only step neccesary apart from the call to ISZ is the FC?C 04 test. Simply insert an FC?C 04 test after the call.

Note: lines 33-34 cannot be shortened to a single ABS. Else, LASTX would be lost.

Final comment : that's all. The routines may be all at the same time in the same program, as there are not conflictive numeric labels among them. Remember, the routines are useful only if microcode is not possible. Otherwise, microcode is the best solution.

VALENTIN ALBILLO (4747)

- this are some inputs for NOP

- -

NOP : -bug in the card reader ROM : the compatibility function 7RCLΣ does not save X in LAST X, whereas the original RCLΣ in the 67/97 does it. My card reader is 1942A00007 .

-"bug" in the 41c ? : Try this: 8 assignments

```
ASN SIN 11   \
ASN COS 12    | this 8 assignments take 4 registers of
ASN TAN 13    | program memory
ASN ASIN 14  >| now, remove the assignments to keys
ASN ACOS 15   | 12, 14, 21, 23
ASN ATAN 21   | Obviously, 4 assignments have been re-
ASN E^X  22   | moved, so 2 registers are free again.
ASN LN   23  /
```

now, PACK. Surprisingly, PACK does not return back the 2 registers. Why ?

- Odd behaviour of programs with multiple alpha labels. Try this: assuming you have 3 modules and a card reader simultaneously, load this program:

```
01 LBL"HP"
02 LBL"?"
03 LBL"?"
 ... ...
356 LBL"?"
357 LBL"?"
358 .END.
```

Use repeated MRG to load the program. The program has in all 357 alpha labels., 1786 bytes. Experimentation shows that:

a) PACK takes 33 seconds
b) GTO.357 takes 6 sec.
c) from line 357, RTN takes 2 sec.
d) CAT 1 takes 2 sec. between labels (beg)
e) GTO "HP" takes 5 seconds
f) any mainframe function not on the keyboard takes 6 seconds to be executed using XEQ (i.e, XEQ X( )Y takes 6 sec)
g) BST from 357 to 356 is instantaneous
   BST from 01 to 357 takes 8 seconds
h) and now, for the final surprise, no printer present, CLP "HP" takes 7 min. 5 seconds to execute, which surely is the record !! A microcode operation that takes 7 min. to execute! (uisng 4 modules, 440 alpha labels are possible, increasing the CLP time).

What does it matter ? Very simple: if the PPC Custom ROM is to have as many as 128 alpha labels, execution time of subroutines which include a call to an alpha label may increase significantly. If the call is within a loop, it may be quite time-consuming. Even mainframe functions will be affected, as all ROM are tested for the alpha label before executing the mainframe function. More interesting, if external RAM is added to the 41c (512 additional registers) , the overcrowding of alpha labels would slow editing speed considerably. Many seconds (or minutes) to execute a PACK, to insert, to BST, not to mention the execution of alpha subroutines in a loop! Certainly the alpha label-END linkage would be very slow.

VALENTIN ALBILLO (4747)

- These are some inputs to ROUTINES or BITS & PIECES

To exchange 2 registers without disturbing the stack (41c)

-- example, to exchange mm & nn : X( ) mm / X( ) nn / X( ) mm } or { X( ) nn / X( ) mm / X( ) nn

only 6 bytes: X,Y,Z,T,L, intact .

To store a number in a register into another, without disturbing the stack (41c)

-- example, to store the contents of mm into nn : { X( ) mm / STO nn / X( ) mm

5 or 6 bytes: X,Y,Z,T,L, intact

VALENTIN ALBILLO (4747)

COMPUTER CHALLENGE : Almost every PPC member knows about 2 or 3 numerical root finders , and has some of then recorded into a magnetic card, etc, waiting for the moment they will be needed. Perhaps you have a mag card, or a MATH PAC ROM, or even a 34C , with its SOLVE key. Well, use any method, but find the answer:

the challenge : given the equation (PI = 3.14....)

$$\sqrt{x(2x+5)} - x + \mathrm{LOG}(x) - \sqrt{x-2} = \mathrm{PI}$$

find the value of x - 7

the value must be given as a 10-digit mantissa with 2-digit exponent, in SCI notation. The mantissa should be exact to ± 1 unit in the last (10) place. Notice the base-10 LOG

This challenge should be very easy to PPC members. Or not ?

(The answer, next month) VALENTIN ALBILLO (4747)

SYNTHETIC PROGRAMMING TECHNIQUES

There are many ways in which synthetic functions can be used to save bytes, registers, or time. Here exposed are several techniques not pointed before, very useful to save space, etc.

1) NNN's in a program

Imagine your program requires the use of some NNN to perform its function. The usual method is to have a status card with the NNN's (up to 5) in the stack. The card is read before running the program. The program is structured so that it makes use of the NNN's in X,Y,...,L.

A better method is to have the NNN as a line in your program, as if it were a normal number. This can be accomplished easily. Simple enter the NNN into the ALPHA register as a synthetic text line, then bring it to the X register using RCL M. This allows any 7-byte NNN to be included as a line in a program (2 lines to be exact, the text and RCL M). Any number of NNN's may be in the program (not just 5). In fact, up to 3 simultaneous NNN's may be in a line. Simply insert a maximum 21-byte synthetic text, then to recall the NNN's use RCL M, RCL N, RCL O. Example: the Byte Counter program requires the NNN F0907C38000000 to be stored into the first assignment register, to create the assignment of RCL b to R/S. This NNN is entered into the X register using

05 "■■■8--- where the text is F5F0907C38000000, created
06 RCL M using STO b, RCL b, or the byte jumper.

2) Normal numbers in a program

The usual method to enter a constant into a program is simply to include it into the program as a line. For instance, to include the constant 365 in the program, simply,

75 365

is included as a line. However, every digit,decimal point,etc is a byte. A constant such as -1.234567809 E-35 takes 16 bytes of program memory if included as a line. Besides, digit entry is rather slow : it takes half a second to execute the whole number.

An improved method is to enter the constant via the ALPHA register. As with NNN, enter a synthetic text line representing your desired constant into the ALPHA register as a line in the program, then RCL M to bring it back to X. As any normal number is 7-byte maximum, the whole constant only takes 7+1+2 = 10 bytes in memory, so 6 bytes are saved ! And the execution time is only 0.1 seconds, 5 times faster. These are great savings, indeed. A typical use of constants as lines of programs are in gaussian methods for numerical integration. Those constants are often multi-digit numbers, so many bytes can be saved using the method. For instance, the program INTEGRAL of Walter Castles (see V7N4P14) includes six 12-byte constants, totalling 72 bytes. Using the synthetic text method, only 60 bytes are needed, and initialization time is reduced from 3 seconds to half a second.

Additional bytes may be saved if up to 3 constants are loaded simultaneously as a text line. The 2 text headers are saved, for the 2 last constants, saving another 2 bytes. For the six constants of W. Castles program, 2 bytes are saved, so 58 bytes (instead of 72) are needed. A very big saving for such a seemingly simple situation ! .

3) Multi-index set up without disturbing the stack

Registers M,N,O , are ideal to be used as index registers for indirect operations. They allow the user to free numeric registers of this kind of work. A typical use is for sorting routines: using M,N,O as indexes, allows all numeric registers to be used for data storage. But there are applications in which the stack itself must not be disturbed, and the registers 00 thru nn (all numeric registers) must remain intact. Then M,N,O can still be used as indexes, but the stack seems necessary to enter the initial values for the indexes into M,N,O . That's not so. A single line of synthetic text can load any values, normal or NNN, into M,N,O, directly, without using the stack.

For example, consider the P( )S function programmed into the 41c. It must not disturb X,Y,Z,T,L, nor any numeric register, so all indexes are carried in M,N. Two indexes are needed, one from 00 thru 09, the other from 10 thru 19. One of the indexes carries also the necessary stop information. For instance, 10.019 must be initially in M, and 0 in N. This is accomplished, without disturbing the stack, in a single line : 02T天-圆---天 . This line is F701001900000001 . The F7 is text 7. The 01001900000001 is the hexadecimal representation of 10.019 . As the ALPHA register is cleared when a text is loaded, N contains 0. So, both indexes are set at once, using only 8 bytes, and the stack is not used. Very simple and useful, isn't it ?

VALENTIN ALBILLO (4747)