Notes on the back story of this letter:

This is the fourth of several letters I sent in quick succession to ***Richard Nelson*** including both comments/questions and new materials created by me for publication in the *PPC Calculator Journal*, and a fifth would follow before my enthusiasm waned for a while.

Being so young at the time and totally used to deal with friendly, receptive *HP* people and enthusiastic calc fans, I still hadn't realized that *Mr. Nelson* wouldn't ever reply to my comments or answer my questions, nor would he comment anything on the materials I copiously sent him, so I began this letter by naively asking him if he had liked my recently submitted ***Othello*** program, and eager to know his opinion about it. Needless to say, he completely ignored me and my questions, again, which added yet another straw to the camel's back.

At any rate, I included several new materials within this *6-page* letter, namely a very complete table of *"ghost" (flag 30 catalogs)* functions, documenting some ***70*** of them, which included their names, the parameters needed for easily generating them using the famed ***John McGechie***'s ***KA*** program, the actual functions executed for all kinds of inputs to their prompts (digits, alpha, **IND**, stack, **IND** stack) and pertinent *notes* detailing the finer points (for instance the use of status register ***Q*** to easily generate non-standard labels) and any caveats, all in all a most useful *TTT (True Treasure Trove)*. It took me a long time of sleuthing and hard effort to compile this table, including many **MEMORY LOST** messages and the occasional locking.

Additionally, I also included two inputs for the *PPC ROM*, consisting in (1) a short and so far the fastest routine to find the address of the first statistic register, as well as the absolute addresses of the statistics block, the curtain (**R000**) and the final **.END.**, thus completely decoding the contents of the ***c*** status register (other than the ***cold start*** constant, ***169***, which didn't need any decoding, just being left alone,) and last but not least, (2) another short routine, this time a 15-line utility to help in loading into program memory any **XROM** instruction *without* the module or peripheral being plugged in. None of them were included in the *PPC ROM*.

Valentin Albillo, 21-12-2021

Richard Nelson
Editor, PPC Journal
2541 W. Camden Place
USA

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

Hello, Richard:

How are you ? Do you liked the OTHELLO program for the 41c ? I guess you have too much work to waste your time playing against the relatively slow 41c, but if you did, what was your opinion about the level of play ?

Here included are some contributions:

a) A table of ghost functions : this one is a much improved, much-detailed, version of the previous one published in V7 N5 P3d . It includes 70 entries, and is as exhaustive as I could make it. Several uses for the ghost functions are included together with examples in the notes. I think it is a good start toward the future "PPC manual on synthetic functions". You would do well encouraging members to further study those functions and the methods to create the-rest that appear on flag 30 catalogs.

b) Two inputs for the Custom ROM : I don't know if there is too late to make new contributions to the Custom ROM. Here included are 2 contributions.

1) a routine to find the address of the 1st statistic register : it is 2.5 times faster than previously submitted routines. It executes in just 4 seconds (vs 10 seconds of faster one). Besides, it also finds the absolute address of the statistics block, the absolute address of the curtain (reg 000), and the absolute address of the final END (.END.) in just 2 seconds, thus providing complete decoding of register c. The routine uses no register, no flag. Magnetic card included.

2) A very simple routine (25 bytes) to make XROM assignments easier. Given the XROM numbers, this routine gives the necessary inputs for the Key Assignments program correctly positioned in the stack, simply add the keycode for the desired key, and call the appropiate entry point in KA. Execution time is half a second. No sinthetic functions used at all.

That's all. If it is too late to include new routines for the ROM, they will be useful as routines as well.

Till the next letter. Sincerely

(4747)

- these are some inputs to the PPC Custom ROM. They are a routine to find the statistics block (Σ find) and the addresses of register 000, and the final END , and an auxiliary routine to compute the inputs for the KA program, given the desired XROM.

Several routines have already been published in this journal, to find the address of the statistic block registers. The fastest I remember is the routine written by Steve Wandzura (4635), in - V7 N5 P12a: it executed in about 10 seconds.

Here included is a routine to locate the start of the statistics block, for those people without a printer, or when - there is no printer available. It is 2.5 times faster than Steve's one: executes in just 4 seconds, uses no register, and does not change the status of any flag. It should be considered for - its inclusion in the PPC Custom ROM.

```
16 LBL"ΣF"   28 LBL 00    40 XEQ 00   52 "⊢----⊀"
17 CLA       29 "⊢12"     41 LASTX    53 RCL M
18 RCL c     30 RCL N     42 ST/ Z    54 FRC
19 STO M     31 CLA       43 x        55 10
20 XEQ 00    32 X( ) M    44 +        56 x
21 RCL c     33 "⊢123456" 45 INT      57 X( ) M      50 lines
22 X( ) M    34 X( ) N    46 RTN      58 INT         106 bytes
23 "⊢1234"   35 "⊢7"      47 LBL 00   59 HMS
24 STO N     36 0         48 STO M    60 16          4 seconds
25 XEQ 00    37 X( ) N    49 RDN      61 x
26 -         38 XEQ 00    50 RCL Z    62 ST+ M
27 RTN       39 X( ) Y    51 X( )Y    63 X( ) M
                                      64 CLA
                                      65 END
```

The reason for the line numbers will be seen in a moment. The - routine is just 50 lines, 106 bytes long. It requires no input from the user. Executes in 4 seconds. Uses the whole stack, but no numbered register. Does not use nor change the status of any flag.

To execute, simply: XEQ "ΣF" → nn

where nn is the address of the first statistic register (this is the last number entered with the ΣREG function).

As an additional bonus, the routine gives also the absolute - address of the first statistic register (in decimal form) and the absolute address of register 000 (the curtain), also in decimal form. Thus, when the routine is executed, the stack is - as follows:

Y : absolute address of the 1st stat. reg.
X : relative address of the 1st stat. reg.
L : absolute address of the 000 register (curtain)

so, executing the routine, you'll have the Σ block address in - the X register. Perform an X( )Y, and you'll have the absolute address of the Σ Block. Execute LAST X , and you'll have the decimal address of the curtain between program and data (000).

This suggests an interesting add-on to the routine: if we can find also the address of the final END, the whole routine would be a kind of decoding of Rc. The following add-on allows that:

```
01 LBL "FE"   09 16
02 CLA        10 x
03 RCL c      11 8
04 X( ) M     12 X/2
05 "⊢12345"   13 X/2
06 STO N      14 MOD
07 XEQ 00     15 RTN
08 LAST X
```

It is 15 lines, 32 bytes, and, together with ΣF , can find the absolute address of the final END in 2 seconds.

Simply , XEQ "FE" → nnn (decimal add)

Both, ΣF and FE, are 50 + 15 = 65 lines, 106 + 32 = 138 bytes , and allow not just the fastest Σ find, but also complete decoding of the contents of Rc, (the address of the curtain, the final END) in seconds. Both can be called as subroutines from another main program (say a curtain mover, or a clear all programs).

EXAMPLE : assume a just master cleared HP-41c, with no RAMS -load programs FE,∑F, then GO TO ..

-to find the address of the statistic block: XEQ "∑F"→ 11.0000

(this is, the address is the default 11) X( )Y→ 250.0000
(the absolute address is 250) LAST X→ 239.0000
(the curtain is at 239) XEQ "FE"→ 218.0000
(the final END is at 218)

-now, let's change a little: XEQ "∑REG" 02
SIZE 009
so, XEQ "∑F " → 2.0000 (all right)
X( )Y → 249.0000 (looks good)
LASTX → 247.0000 (of course, 239 + 017-009)
XEQ "FE" → 226.0000 (218 + 017- 009)

so, the program works. Happy programming.

VALENTIN ALBILLO (4747)

Reading V6 N8 P15c, I noticed the request for a XROM instruction generator, defined as " ... generates desired XROM without the need for the peripheral being available".

Bug 2 is not required, and I think that the best procedure is not to imitate Bug 2, but rather, to use the KA (Key assignment) program, with appropiate inputs.

The bad thing is that you don't know the inputs required to create (assign) a given XROM to a key. Here included is a little routine (25 bytes) which greatly simplifies the work. If this routine is included in the Custom ROM together with the KA program, assignment (and thus, loading into program) of XROM functions is trivial.

```
01 LBL"XR"   09 X( ) Z
02 X( )Y     10 4
03 STO Z     11 MOD
04 4         12 64
05 /         13 x
06 INT       14 +
07 160       15 RTN
08 +
```

This routine is only 15 lines, 25 bytes. You input the XROM numbers, execute the routine, and get the inputs for the KA program, correctly positioned in the - stack.

This is, you desire XROM AA,BB
simply: AA ENTER BB , XEQ "XR" →

and Y,X, will be loaded with the inputs for the KA program. Simply, press the keycode of the key to - which the XROM is to be assigned and call the appropriate routine on KA.

EXAMPLE : we want the following XROMs in a program. We first assign the XROMs: XROM 03,24 , XROM 03,25 , XROM 05,38 , XROM 17,26 , XROM 31,31 , XROM 31,63 , XROM 23,18 , XROM 31,25.

```
03 ENTER 24 , XEQ "XR" → 216 , X( )Y → 160 , so : 160 , 216
03 ENTER 25 , XEQ "XR" → 217 , X( )Y → 160 , so : 160 , 217
05 ENTER 38 , XEQ "XR" → 102 , X( )Y → 161 , so : 161 , 102
17 ENTER 26 , XEQ "XR" →  90 , X( )Y → 164 , so : 164 ,  90
... ... ... ... ... ...      ... ... ...     ... ... ...
31 ENTER 25 , XEQ "XR" → 217 , X( )Y → 167 , so : 167 , 217
```

are the inputs for the KA program, together with the code for the key. Once you have this information, simply:

XEQ "KA" → PAIRS=? , 4 R/S → KEY 1 , 160 ENTER 216 ENTER 11
R/S → KEY 2 , etc.

once assigned, pressing ∑+ in user mode will load an XROM 03,24 in program memory, and so on.

VALENTIN ALBILLO (4747)

- This is an article about the ghost functions whose names appear in the "flag 30 catalogs". It may be included in "FEEDBACK" or "41c NOTES" or "SYNTHETIC PROGRAMMING" or whatever.

After the discovery of the "flag 30 catalogs" (see V7 N4 P26), I attempted to make a list of several of the ghost functions which appeared in those catalogs. My first attempt was published in - V7 N5 P3d, under the "FEEDBACK" column. It was a table describing the real functions of several of the ghost functions. As Richard suggested, flag 30 catalogs are probably listings of functions - yet to be discovered. I agree. The Key Assignments program (see V7 N3 P3, bar code in V7 N5 P53) proves useful to generate many of them, indeed. Other methods to generate the rest are required, however.

Here included is a table, much more complete (over 70+ entries) and detailed than the previous one. The NAME is the name that is seen in the display of the 41c, the printer will - print another version of the name in most cases. The GENERATORS are the inputs for the KA program to assign the function to any key, so that you can generate the function at once. The FUNCTIONS is a list of all known functions performed by the ghost functions for all possible kind of inputs (numbers, alphas, IND, stack).

→ nn stands for 00 thru 99, alpha is any alpha character. ST is stack address (X,Y,Z,T,L). IND is indirect 00 thru 99, and IND ST is indirect X,Y,Z,T,L.

NAME	GENERATORS	FUNCTIONS (and notes)
(blank)	0, 16	Ø (1)
	0,189	GTO 12
	0,193	END
	0,202	END
$T+N IA ___	0, 17	nnn → 1 (also admits EEX nnn) (1)
	0,229	nnn → XEQ local (3) (also EEX nnn)
	0,231	the same as 0,229
	0,233	
	0,235	
__	0, 18	nn,ST,IND,IND ST → 2 (1)
	0, 38	nn → RCL 06 ; alpha → LBL alpha
	0,204	nn,ST,IND,IND ST → END
	0,226	nn,ST,IND,IND ST → XEQ local (3)
μ	0, 19	3 (1)
	0, 47	RCL 15
	0,218	GTO 00
DCABEF-a	0, 20	4 (1)
X	0, 21	5 (1)
	0,242	GTO 00
H_LD	0, 22	6 (1) -
A _	0, 23	alpha → 7 (1)
	0, 33	alpha → RCL 01
	0, 44	alpha → RCL 12
	0,216	alpha → GTO 00
	0,250	alpha → GTO 00
2 __	0, 24	nn,IND,IND ST → 8 (1)
	0, 28	nn,IND,IND ST → CHS (1) (5)
	0, 31	nn,IND,IND ST → W⁻ (spare code)(2)
	0, 40	nn,IND,IND ST → RCL 08
	0, 53	nn,IND,IND ST → STO 05
	0, 63	nn,IND;IND ST → STO 15
	0,188	nn,IND,IND ST → GTO 11
	0,200	nn,IND,IND ST → END
	0,222	nn,IND,IND ST → GTO local (3)
	0,249	00-14 → GTO 00-14 ; 15-99,IND,IND ST (4)
	0,255	00-14 → GTO 00-14 ; 15-99,IND,IND ST (4)
OD	0, 25	9 (1)
	0, 43	RCL 11
	0, 56	STO 08
	0,194	END
	0,237	XEQ 00
	0,251	GTO 00

–	0, 26	• (1) (5)
	0,184	GTO 07
	0,195	END
	0,238	XEQ 00
H	0, 27	nn → E (exp.) ; alpha → LBL alpha (1)
⊏ AHHH	0, 29	GTO⊤ (2)
⊏NQ	0, 30	XEQ⊤ (2)
IEW	0, 34	RCL 02
C _ _	0, 35	nn → RCL 03 ; IND (ST)→ XEQ IND (ST)
		alpha → XEQ alpha
_	0, 36	alpha → RCL 04
	0, 61	alpha → STO 13
	0, 62	alpha → STO 14
	0,185	alpha → GTO 08
	0,223	alpha → GTO 00
	0,225	alpha → XEQ 00
⊤bDCAB⊏⊏¯a	0, 37	RCL 05
	0, 42	RCL 10
	0,179	GTO 02
∅	0, 39	RCL 07
	0, 57	STO 09
	0,190	GTO 13
	0,214	GTO 00
	0,220	GTO 00
	0,230	XEQ 00
	0,234	XEQ 00
	0,236	XEQ 00
⊾Y	0, 41	RCL 09
dS7_μ	0, 45	RCL 13
X∠=Y?	0, 46	RCL 14
ASCI _ _	0, 48	nn → STO 00 ; IND → XEQ IND
		alpha → XEQ alpha
9 _ _	0, 49	nn → STO 01 ; alpha → LBL alpha
REG _ _	0, 50	nn → STO 02 ; ST, IND, IND ST → STO 02
STX	0, 52	STO 04
8 _ _	0, 54	nn, ST, IND, IND ST → STO 06
+	0, 58	STO 10
⊱ SF _	0, 59	alpha → STO 11
↗ _ _	0,160	nn → XROM 00,01 thru XROM 01,35 (6)
μ _ _	0,161	nn → XROM 04,00 thru 05,35
		ST → XROM 05,48 thru 05,52
		IND → XROM 06,00 thru 07,35
		IND ST → XROM 07,48 thru 07,52
∅BEEP _	0,162	alpha → XROM 08,00
∠=∅?	0,163	XROM 12,00
S _	0,164	XROM 16,00
¯a	0,165	XROM 20,00
	0,199	END
T	0,166	XROM 24,00
eG∅BEEP _ _	0,167	nn → XROM 28,00 thru 29,35 ;
		alpha → LBL alpha
SG _	0,176	alpha → RCL 00
⊐⊏⊾	0,177	STO 00
⊁ _ _	0,178	nn, ST, IND, IND ST → GTO 01
W	0,180	GTO 03
	0,182	GTO 05
	0,212	GTO 00
	0,215	GTO 00
	0,240	GTO 00
	0,247	GTO 00
%X∠=∅?	0,181	GTO 04
!Σ + _ _	0,183	nn, ST, IND, IND ST → GTO 06
	0,252	00–14 → GTO 00–14 ; 15–99, ST, IND, ST(4)
e≠ ⊼ _ _	0,186	nn → GTO 09 ; IND (ST) → XEQ IND (ST)
		alpha → XEQ (alpha)
∅∠9 _ _	0,187	nn → GTO 10 ; alpha → LBL alpha
Q _ _	0,191	nn → GTO 14 ; alpha → LBL alpha
DLa ⊼ _ _	0,196	nn → END ; alpha → LBL alpha

⊐Ø	0,197	END
Y	0,198	END
e	0,201	END
⟨ _ _	0,203	nn → END ; IND → GTO IND ; . → GTO .
		alpha → GTO alpha
a _ _	0,205	nn,IND,IND ST → LBL^T (2)
B _ _	0,209	nn,ST,IND,IND ST → GTO local (3)
; _ _	0,210	nn,IND,IND ST → GTO nn, XEQ IND alpha → XEQ alpha
MS	0,211	GTO 00
d ⟨ _ _	0,213	nn,IND,IND ST → GTO local (3)
➢	0,217	GTO 00
⟩ _ _	0,219 0,254	nn,ST,IND,IND ST → GTO local (3) 00-14 → GTO 00-14 ; 15-99,ST,IND, IND ST → (4)
D	0,221	GTO 00
R	0,227	XEQ 00
DCABEL-a _ _	0,228	nn,ST,IND,IND ST → XEQ local (3)
⟩ _ _	0,239	nn,ST,IND,IND ST → XEQ local (3)
Ø	0,241	GTO 00
P	0,243	GTO 00
B1 5Ø	0,244	GTO 00
LST	0,245	GTO 00
AN	0,246	GTO 00
1CLP _ _	0,248	00-14 → GTO 00-14 ; 15-99, ST, IND, IND ST → (4)
R↗	0,253	GTO 00

Notes : (1) Digit entry functions behave diferently in RUN mode and in PRGM mode. In run mode, they enter a single digit (or .,E,etc)into the X register, and digit entry is terminated. In PRGM mode, they seem to enter a single digit, but in fact a text null (F0) is entered as well. Digit entry is not - terminated, so a number of any length, composed of any number - of digits, E, -, may be entered as a line in a program. Simply enter every digit,exp,.,CHS, using "ghost" digit entry functions and every digit will seem to take a line in program memory. It is not so: BST and see all digits at once in a single line.

(2) all functions like GTO^T,XEQ^T,LBL^T, and W^T, take its argument from the Q register (a scratch register). This allows a simple method of creating a LBL,GTO,etc, composed of any characters at all. For instance, to create a LBL^T.:.:.:. , simply (in RUN mode), press (alpha) .:.:.:. (alpha), RCL M, STO Q, switch to PRGM mode, press a _ _ (0,205), fill the prompt with 01, say, and see in the current line: LBL^T.:.:.:.

(3) All locals, including X,Y,Z,T,L, and M,N,...etc,in some cases. Thus you can enter a, say, GTO X (local) in any - line of program memory. GTO X local addresses LBL X (local), whereas GTO^{T}X addresses LBL^{T}X.

(4) For arguments 15-99,etc, a synthetic text line is generated. The length depends on the function. For instance, 1CLP _ _ generates a line of 8 characters. The characters from the 8th are a reflect of the subsequent lines of program.The 2nd character is the number you used to fill the prompt.

(5) It behaves quite oddly: you can create lines with multiple - (minus); when executed, they do behave oddly.

(6) it also generates XROM 02,00 thru 03,35 (IND) and XROM 03,48.thru 03,52 (IND ST)

That's all. Happy programming !

VALENTIN ALBILLO (4747)