* TI PPC NOTES *
****
V5N7, 1980

NEWSLETTER OF THE TI PERSONAL PROGRAMMABLE CALCULATOR CLUB.
9213 Lanham Severn Road, Lanham, Maryland, 20801.

TI PPC NOTES V5N7 P11-12

BRANCHING FROM THE KEY BOARD DURING PROGRAM EXECUTION.- Martin Neef, Zepra-club In-West-Germany-discovered-this-very-useful-trick:-Here follows then a translation of what he writes. I have kept the wording as literal as possible, at the risk of sounding too Germanic once in a while:

«It is very well known that the R/S command will normally start and stop a program in user memory, except when you call a module program from the key board, when the next R/S command will start the module program, rather than the user program. When during program execution a call to a module with PGM xx is encountered, according to the manual, you have to have a SBR call with A through E', or a SBR N or SBR nnn. If you call anything else, the program goes automatically to PGM 00, that is the program in user memory.

One exception to this rule here is the R/S command, which provokes a starting of the module program, beginning at the step the program pointer happens to be at. These R/S calls have to be placed in the program immediately after the program call PGM xx.

By means of a trick it is now possible to avoid that the call to PGM xx would be cancelled by the subsequent commands. That means that the call to a SBR or R/S does not have to follow immediately after the PGM xx call. This also means that an R/S from the key board, during program execution, does not constitute a program interruption anymore of the user program, but it means the start of a module program.

The trick I am alluding to consists of placing immediately after the PGM xx call one of the codes 21, 41 or 51, with 51 being preferred for its lack of side effects. (21 is the code for 2nd, 41=SST and 51=BST, ed)

As long as after this call you have neither a SBR nor a RST, the R/S call from the keyboard, during program execution, will start the module program.

A practical application results from this sequence:

0 STO 00 PGM 01 SBR 098 PGM 01 “51” LBL A (ML module)

After execution of ... PGM 01 “51” ... you may branch to LBL A with a R/S. That means that you can make decisions without stopping the program, which constitutes a rather "friendly" style of programming. This trick would allow you to write a comfortable reaction test program. Note, however, that each call opens up two more SBR levels.

It is clear that this method has so many new possibilities that it would be rather difficult to mention them all. I will leave it to the reader to do some further exploration. Happy searching. (Martin Neef) »

And this is exactly what Richard Snow did, searching. He writes:

«The results can be quite frustrating if you don't know what to expect. Please note the following:

1. Any subroutine call in user memory (SBR A through E') will call the subroutine in the library module.

2. If R/S is held down until the module routine is finished, then the user program will stop.

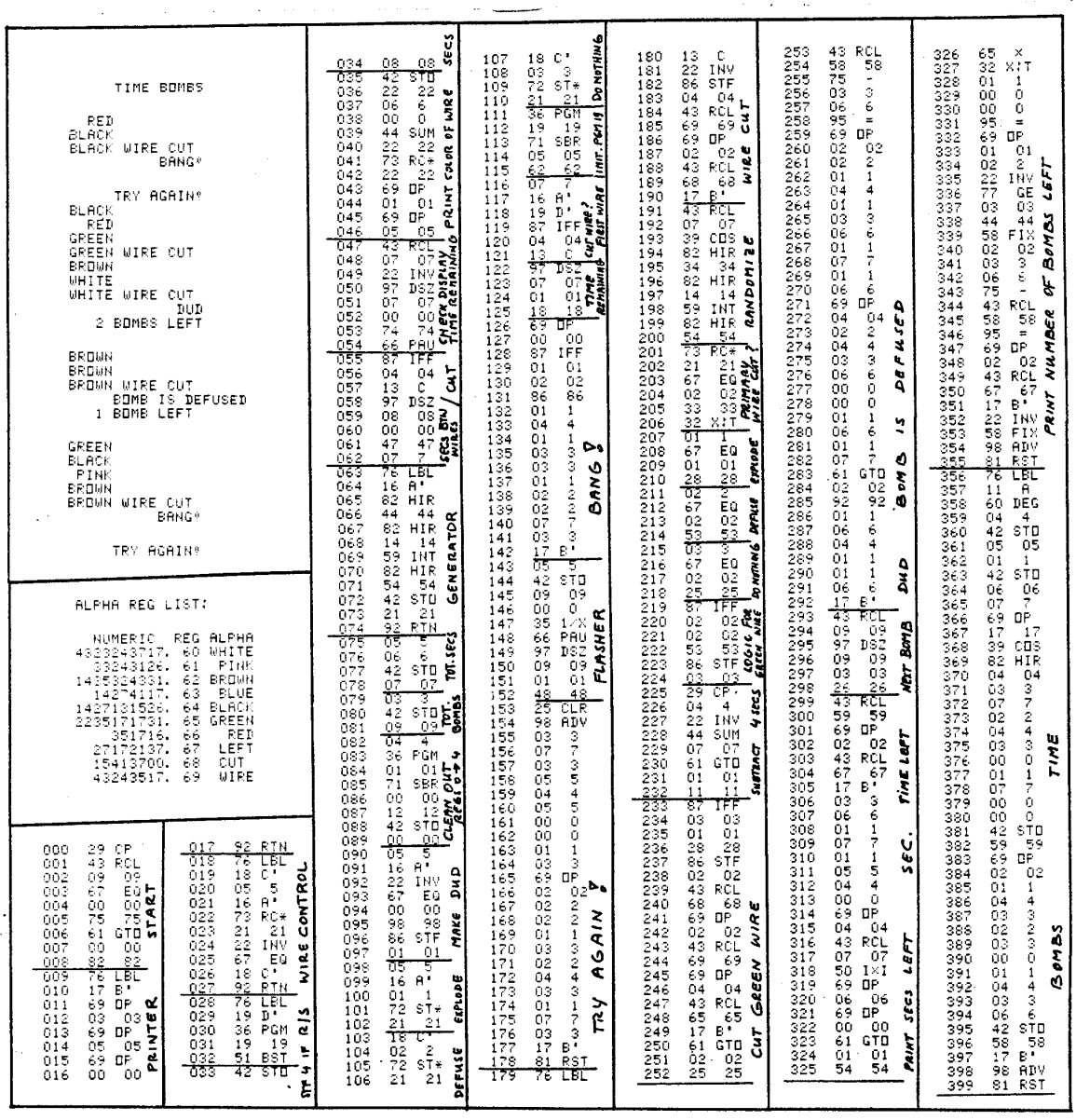
3. A RTN in user memory will cancel the PGM xx “51” call.

4. A R/S in the user program will not stop the program, but will start the module program instead.

As an example, steps 547 through 567 of ML-19 can be used to set a flag. (flags 1, 2, 3, 4, Ed.) A user program can then be made to branch after the R/S key is pressed and the corresponding flag is checked. The problem, however, is that LBLs A' through D' (on steps 547 through 567, Ed) are such short routines; that it is difficult not to stop the user program when R/S is pressed. »

Richard then encloses an enhanced version of his brother's TIME BOMBS game. This program appeared originally as PPX # 918029B. The key board branching trick was added to reduce the possibility of cheating. It should be noted that this program is intended only as a demonstration of the use of the key board branching trick. Press R/S to stop the program. Then change your mind about cutting a particular wire and press R/S again to continue. Unfortunately, if you hold down R/S a little too long, it stops the program for good. Perhaps a longer routine than LBL D' PGM 19 BST (see steps 028-032 of TIME BOMBS) can be found in one of the library modules which will do the job more reliably.

You will find TIME BOMBS somewhere is this issue.




```
000  29 CP
001  43 RCL
002  09  09
003  67  EQ
004  00  00
005  75  75
006  61 GTO
007  00  00
008  82  82
009  76 LBL
010  17 B'
011  69 DP
012  03  03
013  69 DP
014  05  05
015  69 DP
016  00  00
017  92 RTN
018  76 LBL
019  18 C'
020  05  5
021  16 A'
022  73 RC*
023  21  21
024  22 INV
025  67  EQ
026  18 C'
027  92 RTN
028  76 LBL
029  19 D'
030  36 PGM
031  19  19
032  51 DST
033  42 STO
034  08  08
035  42 STO
036  22  22
037  06  6
038  00  0
039  44 SUM
040  22  22
041  73 RC*
042  22  22
043  69 DP
044  01  01
045  69 DP
046  05  05
047  43 RCL
048  07  07
049  22 INV
050  97 DSZ
051  07  07
052  00  00
053  74  74
054  66 PAU
055  87 IFF
056  04  04
057  13  C
058  97 DSZ
059  08  08
060  00  00
061  47  47
062  07  7
063  76 LBL
064  16 A'
065  82 HIR
066  44  44
067  82 HIR
068  14  14
069  59 INT
070  82 HIR
071  54  54
072  42 STO
073  21  21
074  92 RTN
075  06  6
076  06  6
077  42 STO
078  07  07
079  03  3
080  42 STO
081  09  09
082  04  4
083  36 PGM
084  01  01
085  71 SBR
086  00  00
087  13  C
088  42 STO
089  00  00
090  05  5
091  16 A'
092  22 INV
093  67  EQ
094  00  00
095  98  98
096  86 STF
097  01  01
098  05  5
099  16 A'
100  01  1
101  72 ST*
102  21  21
103  18 C'
104  02  2
105  72 ST*
106  21  21
107  18 C'
108  03  3
109  72 ST*
110  21  21
111  36 PGM
112  19  19
113  71 SBR
114  05  05
115  52  52
116  07  7
117  16 A'
118  19 D'
119  87 IFF
120  04  04
121  13  C
122  97 DSZ
123  07  07
124  01  01
125  18  18
126  69 DP
127  00  00
128  87 IFF
129  01  01
130  02  02
131  86  86
132  01  1
133  04  4
134  01  1
135  03  3
136  03  3
137  01  1
138  02  2
139  02  2
140  07  7
141  03  3
142  17 B'
143  05  5
144  42 STO
145  09  09
146  00  0
147  35 1/X
148  66 PAU
149  97 DSZ
150  09  09
151  01  01
152  48  48
153  35 CLR
154  98 ADV
155  03  3
156  07  7
157  03  3
158  05  5
159  04  4
160  05  5
161  00  0
162  00  0
163  01  1
164  03  3
165  69 DP
166  03  03
167  02  2
168  02  2
169  01  1
170  03  3
171  02  2
172  04  4
173  03  3
174  01  1
175  07  7
176  03  3
177  17 B'
178  81 RST
179  76 LBL
180  13  C
181  22 INV
182  86 STF
183  04  04
184  43 RCL
185  69  69
186  69 DP
187  02  02
188  43 RCL
189  68  68
190  17 B'
191  43 RCL
192  07  07
193  39 COS
194  82 HIR
195  34  34
196  82 HIR
197  14  14
198  59 INT
199  82 HIR
200  54  54
201  73 RC*
202  21  21
203  67  EQ
204  02  02
205  33  33
206  32 X!T
207  01  1
208  67  EQ
209  01  01
210  28  28
211  02  2
212  67  EQ
213  02  02
214  53  53
215  03  3
216  67  EQ
217  02  02
218  28  28
219  87 IFF
220  02  02
221  02  02
222  53  53
223  86 STF
224  03  03
225  29 CP
226  04  4
227  22 INV
228  44 SUM
229  07  07
230  61 GTO
231  01  01
232  11  11
233  87 IFF
234  03  03
235  01  01
236  28  28
237  86 STF
238  02  02
239  43 RCL
240  68  68
241  69 DP
242  02  02
243  43 RCL
244  69  69
245  69 DP
246  04  04
247  43 RCL
248  65  65
249  17 B'
250  61 GTO
251  02  02
252  25  25
253  43 RCL
254  58  58
255  75  -
256  03  3
257  06  6
258  95  =
259  69 DP
260  02  02
261  02  2
262  01  1
263  04  4
264  01  1
265  03  3
266  06  6
267  01  1
268  07  7
269  01  1
270  06  6
271  69 DP
272  04  04
273  02  2
274  04  4
275  03  3
276  06  6
277  00  0
278  00  0
279  01  1
280  06  6
281  01  1
282  07  7
283  61 GTO
284  02  02
285  92  92
286  01  1
287  06  6
288  04  4
289  01  1
290  01  1
291  06  6
292  17 B'
293  43 RCL
294  09  09
295  97 DSZ
296  09  09
297  03  03
298  26  26
299  43 RCL
300  59  59
301  69 DP
302  02  02
303  43 RCL
304  67  67
305  17 B'
306  03  3
307  06  6
308  01  1
309  07  7
310  01  1
311  05  5
312  04  4
313  00  0
314  69 DP
315  04  04
316  43 RCL
317  07  07
318  50 IXI
319  69 DP
320  06  06
321  69 DP
322  00  00
323  61 GTO
324  01  01
325  54  54
326  65  X
327  32 X!T
328  01  1
329  00  0
330  00  0
331  95  =
332  69 DP
333  01  01
334  02  2
335  22 INV
336  77 GE
337  03  03
338  44  44
339  58 FIX
340  02  02
341  03  3
342  06  6
343  75  -
344  43 RCL
345  58  58
346  95  =
347  69 DP
348  02  02
349  43 RCL
350  67  67
351  17 B'
352  22 INV
353  58 FIX
354  98 ADV
355  81 RST
356  76 LBL
357  11  A
358  60 DEG
359  04  4
360  42 STO
361  05  05
362  01  1
363  42 STO
364  06  06
365  07  7
366  69 DP
367  17  17
368  39 COS
369  82 HIR
370  04  04
371  03  3
372  07  7
373  02  2
374  04  4
375  03  3
376  00  0
377  01  1
378  07  7
379  00  0
380  00  0
381  42 STO
382  59  59
383  69 DP
384  02  02
385  01  1
386  04  4
387  03  3
388  02  2
389  03  3
390  00  0
391  01  1
392  04  4
393  03  3
394  06  6
395  42 STO
396  58  58
397  17 B'
398  98 ADV
399  81 RST
```

* TI PPC NOTES * V6N1, 1981
****

NEWSLETTER OF THE TI PROGRAMMABLE CALCULATOR CLUB.

9213 Lanham Severn Road, Lanham MD, 20801, USA

TI PPC NOTES v6NIP15

REPARTITIONING THE TI-58C: By now you have heard of the extra 32 steps (480-5ll) or four memories in the TI-58C. Unfortunately these steps are usually accessible only from the keyboard using the STF IND ... sequence. But in the TI-58C, the memory corresponding to steps 480 to 487 is used by the constant memory feature to set the partition, the FIX n display mode, and to check for memory loss at turn-on. By changing the contents of this memory the TI-58C can be repartitioned so that these 32 steps or four memories are directly accessible.

The first 13 digits of this memory store an unscaled In 10. If this number is not present at turn-on, the calculator erases all data and program memory to protect the user from unsuspected memory loss. Instead the user suffers from known memory loss.

The next two digits of the memory (i.e. the first digit of step 480 and the last digit of step 481) set the partition at turn-on. By writing the proper key-codes in these steps, the TI-58C can be partitioned like a TI-59 or entirely new partitions may be generated. First, steps 480-481 must be accessed. To do this, write in program memory LBL A FIX 0 R/S, enter 609.000000609 into the display, and press A. Then press STF IND 7 INV LRN. You should be at step 480. Write |x| RCL in steps 480-481. Then cycle the on-off switch, press Op 16 and you will see a partition of 0.09. This partition means you have no memories and 1200 program steps. Of course there is no hardware beyond step 511- any code you see in these steps seems to be miscellaneous garbage (electrons that got lost in the circuits, perhaps). Now you can repartition to any of the primary partitions in Table 1 by simply going to steps 480-481, entering the proper code and cycling the on-off switch.

However, the most useful partitions are those pseudo-59 partitions in the table. To access them, get into one of the first four primary partitions then simply repartition from the keyboard via Op 17 as you would with a TI-59. These give the TI-58C user an extra 32 steps (480-511) or an extra four memories. For instance, a partition of 479.59 gives 480 program steps and four valid (hard-ware-backed) memories numbered 56 to 59. Memory 59 (or steps 480,481) must be used with care since any change in FIX n or partitioning will alter the contents (but note that the reverse is true only if the on-off switch is cycled) and In 10 must be restored before turn-off or the calculator will be cleared at turn-on. Also, at turn-off, a psuedo-59 partition will revert to the corresponding primary partition.

The last digit of step 480 stores the FIX n display mode. This digit will equal n+2 in a FIX n display for $0 \leq n \leq 7$. In FIX 9 or FIX 8 this digit will be a zero, but in FIX 8 the first digit of step 480 will be increased by one without affecting the partition. This brings up an

important point. A Partition of 0.39 in FIX 9 and a partition of 0.49 in FIX 8 both store code 20 41 in steps 480 and 481, but the calculator goes into the correct partition and FIX n at turn-on. How does the calculator determine which partition and FIX n is correct? I was afraid you'd never ask. Well, in the FIX 8 case the keycode 20 is actually the hexadecimal keycode 1A. (This was determined through other means. None of the hex keycodes ending in A do anything new). Putting a 1 in the last digit of step 480 gives a sort of FIX -1 display. This rounds off the display (but not the display register) to the tens' place and adds a negative sign. Thus, a 5 is displayed as -10 in the last three places of the display and a 2.3 is displayed as -00. In an exponential display. if the most significant digit (MSD) is less than 5, the display shows 0.00; if the MSD is greater than or equal to 5, the display will show a.xx where XX is the exponent +1. .

That's about all there is to repartitioning the TI-58C. Doing something useful with all this is left as an exercise for the reader.

Patrick W. Acosta

(*continues on next page)*

Code in Step .. 480	481	Primary Partition	Memories	Pgm. Steps	Pseudo-59 Partition***
90	41**	239.89*	90	240	--
80	41	159.99*	100	160	--
70	41	79.09*	110	80	--
60	41	0.19*	120	0	--
50	41	0.09	0	1200	--
40	41	0.19	0	1120	--
30	41	0.29	0	1040	--
20	41	0.39	0	960	959.00
10	41	0.49	0	880	879.09
00	41	0.59	0	800	799.19
90	40	0.69	0	720	719.29
80	40	0.79	0	640	639.39
70	40	0.89	0	560	559.49
60	40	479.00	0	480	479.59
50	40	399.09	10	400	399.69
40	40	319.19	20	320	319.79
30	40	239.29	30	240	239.89
20	40	159.39	40	160	159.99
10	40	79.49	50	80	--
00	40	0.59	60	0	--

**Any keycode from 42 to 49 may be substituted for 41 with identical results.
*These partitions allow pseudo-59 partitions to be accessed. via Op 17.
***Pseudo-59 partitions have the usual number of memories and program steps.

An extra four memories in the TI-58C

This material first appeared in the V6N1P15 TI PPC Notes by Partick Acosta.

It turns out that there are an extra four data memories or an extra 32 program steps in the basic TI-58C. Note that this only applies to the TI-58C and NOT the TI-58.

The data memory register 59 corresponding to program steps 480 to 487 is used by the TI-58C's

constant memory feature to set the partition between data and program memories, to set the FIX N display, and to check for memory loss at turn on. By changing the contents of this memory, the TI-58C can be repartitioned so that these 32 steps/4 memories are directly accessible.

The first 13 digits of this memory 59 store an unscaled LN(10) value. If this number is not present at turn on, the calculator erases all data and program memory to protect the user from unsuspected memory loss. Instead, the user suffers from known memory loss!

The next two digits of the memory (the first digit of step 480 and the last digit of step 481) set the partition at turn on. By writing the proper key codes into these steps, the TI-58C can be partitioned like a TI-59 or entirely new partitions can be created.

First, however, steps 480-481 must be accessed. To do this, write in program memory: LBL A FIX 0 R/S. Then enter the number 609.000000609 into the display and press A. (Enter this value by typing 0.000000609 then adding 609 to it. Another example of using the TI-58/59's greater than 10 digit precision). Then press STF IND 7 INV LRN. You should be at step 480. Key in the instructions GRAD and RCL into steps 480-481 in program mode. Press LRN to leave program mode. Then cycle the on-off switch. Press OP 16 and you should see a partition of 159.99, meaning that you have 100 memories and have 160 program steps.

Once this is done, you may repartition the TI-58C for any of the normal TI-59 partitions. For example, a partition of 6 OP 17 provides the display 479.59 and gives the TI-58C 480 program steps and 4 hardwired memories available in addressed as STO 56, STO 57, STO 58 and STO 59.

You must be careful with memory 59 (since that will correspond to step 480 and 481 as part of that memory). Changing the display setting with a FIX command will alter parts of this data memory and LN(10) must be stored there prior to turning off the machine to avoid a memory lost. For that reason, I suggest you leave memory 59 alone!

This would allow you to write a very long program and be able to use 3 memories. The program would otherwise not fit in a TI-58C.

Note: When you turn the machine on and off, any pseudo-TI-59 partitioning scheme will revert back to a "primary" partition. You would need to redo an OP 17 to re-establish the previous partitioning when you turn the machine back on.

Other useful TI-59 partitions would include:

- 6 OP 17 - 480 program steps and 3 memories (56, 57, and 58)
- 7 OP 17 - 400 program steps and 13 memories (0-9, 56, 57, 58)
- 8 OP 17 - 320 program steps and 23 memories (0-19, 56, 57, 58)

And so forth.
You don't really gain anything with partitions less than 6 OP 17.
Enjoy this particular quirk.

T I P P C N O T E S
****

v6N9/10, 1981

Newsletter of
the TI Programmable Calculator Club.

9213 Lanham Severn Road, Lanham MD 20706
or P.O. Box 710, Lanham MD 20706.

Fast Mode - Patrick Acosta. Up to now, the only generally known way of getting into Fast Mode was the PGM 02 SBR 239 method using the ML module. There are two other ways that work:

1. STF IND 7 INV which works only from the keyboard, and
2. a programmable version using a hex keycode STF IND h12.

Both of these methods have certain advantages over the CROM method. All three have in common that they load the calculator's flag register, the CROM method loading the flag register from the following octet, probably after first altering that octet, and the two other methods loading the flag register from the display. This flag register (shown as register 0 in figure 19 of U.S. Patent # 4,153,937) has 16 digits. The digits relevant to fast mode are used as explained below:

User flags	9	8	7	6	5
User flags	4	3	2	1	0
First five digits of flag register	d1	d2	d3	d4	d5

If the digit is.....

0, 4 or 8	Both corresponding flags are reset.
1, 5 or 9	The corresponding bottom flag is set and the corresponding top flag is reset.
2 or 6	The corresponding top flag is set and the corresponding bottom flag is reset.
3 or 7	Both corresponding flags are set.

The ninth through twelfth digits store the address used in any kind of jump (except RST) are encoded as follows:

W X Y	Z
octet	byte

Thus, the address above would normally be 8*(WXY)+Z, except when accessing Fast Mode when the calculator will jump to the address 8*(WXY)+Z+1.

The thirteenth digit is called the "*Program Source Flag*". This digit must be 2, 4 or 6 in order to enter Fast Mode.

Digit sixteen holds the fixed-point display mode. This digit will be zero for an INV FIX display, otherwise it will be n+2 in a FIX-n display.

Now to get into the *STF IND 7 INV* type of Fast Mode, enter a 13-digit number into the display, with the 9th through the 12th digit encoding the address you want the Fast Mode to start at, and the 13th digit always equal to 2, 4 or 6. Then press:
DEG FIX 0 RST STF IND 7 (or any other digit) *INV* (or B, X+T, STO, EE).

For example, pressing .00007032+44444= DEG FIX 0 RST STF lND 7 INV will clear all flags and begin execution in Fast Mode at 8*(070) +3+1 = step 564 in an INV FIX display mode. Or: 2 EE 12 +/- + 3 = RST DEG STF lND 7 INV sets flags 9 and 4 and begins Fast Mode execution at step 001.

A few things you should watch out for:

1. You must be in DEG mode to enter Fast Mode. If necessary, you may later in your Fast Mode program change to RAD or GRAD mode.
2. You may be in FIX 9 mode when entering Fast Mode only if the number in the display shows no fractional part. otherwise you MUST be in FIX 0.
3. An EE or ENG mode is alright as long as rule 2 is also followed.
4. The above procedure does not automatically reload the "command buffer register". So you must press RST (unless the calculator has stopped at the 7th step of an octet) to ensure that this register is loaded immediately with the keycodes you want to execute. Otherwise, the calculator will execute the instructions in the buffer register until reloaded at the end of the octet. (*continues on next page)*
5. You cannot begin execution at step 000.
6. You must have some memory in the current number partition. So, do not use 0 OP 17.
7. Changing the signs of the 13-digit and/or it's exponent will change the FIX n display mode.
8. Using 9 as the byte within the octet (digit 12 of the flag register) sends the calculator to step (WXY)*8+8. (Note to members unfamiliar with computer notation: the asterisk is used to denote "times or multiplication")

As a practical matter, it is most convenient to store the 13-digit number in an available register and recall it when needed. Thus, in program memory you could write: *LBL E FIX 0 DEG RCL NN R/S*. Then from the keyboard you must press E RST (This last step could be omitted if the R/S is at a step congruent to 7 modulo 8) STF IND 7 INV.
For those who think this is still too much key punching, put *LBL E FIX 0 DEG RCL NN STF IND* in your program such that the IND is at the last step of the partition. Then you will only need to press E 7 INV. In that case, just put a CE instruction as the first step of your Fast Mode program, to clear the error condition caused by the aforementioned procedure.

This method does not erase your program or data or change the partitioning, as the CROM method does. But most of the usual rules of Fast Mode still apply.

The hexadecimal method is similar to the keyboard method, with this difference that it can be programmed. Assuming the 13-digit number is stored in register NN, the sequence is *LBL A FIX 0 DEG RCL NN STF IND h12.* Remember that the hex key code

can only be created at the first step of an octet. Since this method also doesn't immediately load the "command buffer register", only some of the next seven keycodes will be executed before the buffer is reloaded. How many and which ones are executed depends on the twelfth digit of the 13-digit number. So, to simplify matters, the instructions in the buffer register, following h12 are used to make the jump to the beginning of the Fast Mode sequence. The sequence then is

LBL A FIX 0 DEG RCL NN STF IND h12 NOP GTO NNIN.

The address now loaded into the flag register is unimportant. You must only be sure that the GTO NNN is allowed to execute completely, which means that digit 12 must be ≤ 7. And, of course, digit 13 must be 2, 4 or 6. Note that the step following h12 is always ignored. Also, any keycode ending in C (2 in display) may be substituted for h12.

The nice thing about this method is, that you may use library programs or statistics and conversion functions, then switch to Fast Mode under program control. Another nice feature, for calculators-alone programs, is that you may call the above LBL A as a subroutine in your Normal Mode program. Then, when your Fast Mode segment arrives at the RTN instruction, you return to Normal Mode at the point from which you called subroutine A. However, with the printer attached, the calculator seems to return in trace mode. (At least it did for Palmer Hansen in one of the programs in which this was tried.) Note that the Fast Mode segment must be the lowest level subroutine. You still cannot call subroutines in Fast Mode.

For the 58C , the STF h12 works just as well, but Palmer Hansen found STF IND h12 necessary on his 59. Also notice that, if you single-step past the STF IND h12, none of the usual methods of stopping Fast Mode will work, except RST.

You can use sequences such as SBR NNN, SBR LBL, X=T LBL (but not GTO LBL) in Fast Mode. But this will cause a return to Normal Mode and if you arrive at a RTN instruction, the calculator begins execution somewhere in the CROM, usually causing a crash.

Other methods of causing a jump in the program counter might also be studied for possible Fast Mode access. For instance, John Mairs' keycode translation in v5n4/5p18. Also, De Mees' quirk in v6n3p6 might be tried with various keycodes in the first octet. My 58C, besides not entering Fast Mode with PGM 02 SBR 239, doesn't have any of the CROM quirks either.

Hexadecimal Keycodes: Each program step in a TI-58/59 occupies one byte of memory and is in a BCD (binary coded decimal) format. This means that each digit (0 to 9) is stored in four bits. But four bits of memory can contain 16 possible "digits". This is the hexadecimal (base 16) system where the "digits" 10 to 15-are represented by the letters A through F respectively. That gives 256 theoretically possible keycodes (00 to FF). It is possible to create 60 of these hex-keycodes on the 58/59.

First you must get into ROM. At turn-on (with the ML module in place) press 3 Op 17 (9 Op 17 for the TI-59) Pgm 12 SBR 444 R/S DMS LRN. Then at a step number in ROM evenly divisible by 8, press Ins and return to RAM by pressing LRN RST. You will find 9 program steps written in user memory (beginning at the step number where you pressed Ins) the first of which will behave strangely. For instance, at step 000, you would see the strange keycode 24 which we will denote h24 to distinguish it from the regular keycode for CE. If previously you had a 01 in RAM step 000, you will get the keycode h25;a 02 in RAM step 000 will give the normal keycode 20. This suggests...

RULE 1: Pressing Ins at a step number nnn evenly divisible by 8 in ROM ,will give the hex keycode (A0-ab)+cd in RAM where ab is the keycode in ROM step nnn and cd is the keycode previously in RAM step nnn but the first digit of the hex keycode will be BCD normalized. For example, with 01 in RAM step 000, pressing INS at ROM step 000 with keycode 82 will put A0-82+01=1F in RAM step 000. The hex digit F is 15 in base 10 and the display will show the hex number 1F as 25 which we will call h25. Note that only the last digit at a step number divisible by 8 can be made hexadecimal. The first digit and the other downloaded keycodes are put into normal BCD form. By writing the proper keycode in RAM and pressing INS the right number of times, any of the 60 possible hex keycodes can be created at any step number divisible by 8--e.g. to create the hex keycode 2B at step 000, write SBR in RAM, then go to step 000 in ROM and press INS three times: 3x(A0-82)+71-CB which is BCD normalized to 2B and displayed as 31. Creating the keycodes you want where you want gets easier with a little practice.

RULE 2: Hex keycodes revert to normal when moved about by editing. Also note that pressing INS in ROM will move your RAM program up a step so if you want to use more than one hex keycode in a program, you must start at the bottom of program memory and work your way up.

RULE 3: All the hex keycodes, except those from h10 to h15, disable any label search. If you want to jump past a hex-keycode in a program, you must either use absolute addressing or mask the hex-code from the label search by making it part of a two step instruction. RCL hxx will allow label searches without crashes or other quirks. Calling a label that exists before an unmasked hex-code is also alright.

None of these hex-codes print out calendars (sorry) but some do useful things.

h04 acts as a super clear. It clears the display, the t register, EE or ENG mode, the SBR return register, error conditions, all HIR registers (see V5N4/5 p13), resets all flags, goes to step 000, and stops execution. This could save some steps in an initialization routine.

h32, h33, h45, h53, h64, h71, h72, and h83 do the same. (This on a 58C. On a 58/59, h04 may clear the program.)

h31, h41, and **h51** act as a normal sine function. Writing INV before any of them

gives a strange function that is probably used internally to calculate trig functions.

h54 will copy the t register into the display if immediately following x:t while preserving pending operations. This is a two step equivalent of copying the display into the t register (V5N6p4). Doing certain things between the x:t and h54 (including CP, strangely enough) may produce other results. (*continues on next page)*

h24 is the strangest hex-code of all. Create it at step 000 by pressing INS once and return to user memory and write 7 program steps in 001 to 007 so that you have: 000:

h24, AB, CD, EF, GH, IJ, KL, MN

If you then press RST R/S, h24 will change the order in which the keycodes are interpreted to:

000: DA, FC, HE, JG, LI, NK, 0L, 00.

This gives the equivalent of 15 program steps packed into 8 steps. SBR 001 will give a 7 step program, and SBR 000 will give an entirely different 8 step program.

All the programmer has to do is write his program so that either way the 8 keycodes are interpreted will be useful. An IQ of 160 and nerves of steel are recommended.

The following illustrates the usual behaviour of hex-codes in multi-step instructions. Recall that h11 is the display of the hex-code 0B which might be written as 00,11 in base 10. This helps explain some of the following behaviour.

RCL h11:: recalls 00, runs SBR A
FIX h11:: goes into FIX 0, runs SBR A.
STF h11:: sets flag 0, runs SBR A.
GTO h11:: runs SBR A, RTNs to step 001.
SBR h11:: runs SBR A, RTNs to SBR 001, RTNs to step following h11.
GTO 0xh11:: SBR A, RTNs to step 10x+1.
SBR 0x h11:: SBR A, RTNs to SBR 10x+1, RTNs to step following h11.
Dsz 01 h11:: if register 01≠0, runs SBR A, RTNs to step 001.
if register 01=0, runs SBR A, RTNs to step following h11.
EQ h11:: if x=t, runs SBR A, RTNS to step 001.
if x≠t, runs SBR A, RTNs to step following h11.

This is the typical kind of behaviour but some multi-step instructions will do completely different things.

With 60 hexadecimal keycodes and all the multi-step combinations, I suspect it will take a while before all the possibilities are understood.

Happy hexadecimal programming,
Patrick Acosta.

* T I P P C N O T E S *
*
****

NEWSLETTER OF v7N1-2, 1982.
THE TI PROGRAMMABLE CALCULATOR CLUB.
9213 Lanham Severn Road
or P.O. Box 710, Lanham MD 20706.

TI PPC NOTES V7N1/2 P11-12

Circular stepping During Listing of the Revealed Firmware - P. Hanson

V5N1p7 and V5N3p6 describe keyboard sequences which permit readout of the firmware which mechanizes the statistics and conversions functions. Neither discussion mentioned a quirk which had been reported in earlier discussions of downloading of the firmware which appeared in V3N10p4 and V3N12p5 of *52 Notes*, namely the "circular stepping" which would occur during an attempt to list the revealed firmware. Patrick Acosta, who does not have a PC-100, suggested that the quirk might be caused by previously unrecognized hexadecimal h22 commands at locations 488 and 504 of the downloaded firmware. My tests confirm Patrick's hypothesis.

The "circular stepping" quirk was originally reported in V3N10p4 of *52 Notes* by Steffen Seitz. He observed that an attempt to continue listing of the firmware past step 487 resulted in a return to step 039 with an abs instruction, the 040, 041, ..., 487 are as before. This circular stepping will continue for as many iterations as you are willing to expend printer paper. In V3N12p5 of *52 Notes* Steve Bepko reported that if the program counter was SSTTd past location 488 then additional printer listing could be obtained up through location 503. At what would have become step 504 the circular stepping to location 039 occurred. In a personal letter Patrick Acosta suggested that the unusual "circular stepping" behaviour might be associated with the code 22 which is seen at 488 and 504 when SSTing rather than listing. He hypothesized that the code at those locations might really be hexadecimal code h22 rather than the normal code 22 (INV). To test the hypothesis I synthesized code h22 at location 016 of an otherwise clear memory with the following sequence (again due to Patrick):

Starting from turn on, or Cms-CP	Clears memory
GTO-016-LRN-SBR-BST-LRN	Puts code 71 in location 016 and returns pointer to 016.
10-Op-17-CLR	Sets partitioning to permit synthesizing hexadecimal codes.
Pgm-12-SBR-999	Flashing 0. in the display
R/S	Flashing 0. 00 in the display
DMS	Flashing 0 in the display
LRN	016 55 in the display

Ins	016 55 in the display
LRN-RST-CLR	0 in the display. Calculator returned to normal mode.
GTO-016-LRN	016 22 in the display confirming that h22 is at location 016 .
SST	017 02 in the display
SST	018 10 in the display
SST	019 38 in the display
SST	020 30 in the display
SST	021 31 in the display
SST	022 30 in the display
SST	023 71 in the display
SST	024 03 in the display
SST	025 00 in the display
If you check the remainder of the user memory you will find zeroes	
RST-List	See listing*(continues on next page)*

Note that "circular stepping" occurs each time location 016 is encountered. Also for this set of conditions the code that had been verified to be at locations 017 through 022 by the SST process appears at locations 000 through 006 with the listing process. The conclusion that h22 code causes circular stepping during listing appears inescapable.

Patrick's conjecture that something is different about the code at locations 488 and 504 is supported by the listing in Table VI of Patent No. 4,153,937. The final character for the constants 0, 13, 14, and 15 all appear as a code C (hexadecimal 12) in that table. The equivalent locations in the downloaded firmware are 384, 488, 496, and 504. Locations 384 and 496 list as code B (keycode 12). In the table in the patent the final character pair for the equivalent locations is 0C. I suspect that the code is really h12, not code B (keycode 12). Locations 488 and 504 in the firmware as downloaded by successive SST commands and readout from the display appear as code INV (keycode 22). In the table in the patent the final character pair for the equivalent locations is 1C, or h22 as Patrick Acosta defines hexadecimal codes.

000 00 0	014 00 0	012 00 0
001 00 0	015 00 0	013 00 0
002 00 0	000 02 2	014 00 0
003 00 0	001 10 E'	015 00 0
004 00 0	002 38 SIN	000 02 2
005 00 0	003 30 TAN	001 10 E'
006 00 0	004 31 LRN	002 38 SIN
007 00 0	005 39 COS	003 30 TAN
008 00 0	006 71 SBR	004 31 LRN
009 00 0	007 00 00	005 39 COS
010 00 0	008 00 00	006 71 SBR
011 00 0	009 00 0	007 00 00
012 00 0	010 00 0	008 00 00
013 00 0	011 00 0	009 00 0

Transparent Fast Mode - Palmer O. Hanson,Jr.

Ever since I became aware of Patrick Acosta's hexadecimal h12 method for entering fast mode under program control I have been searching for a program sequence which would make the use of fast mode "transparent" to the user. That is, once the h12 command has been properly placed with one of Patrick's techniques, then normal mode can be used for program entry from the keyboard, the calculator will automatically enter fast mode under program control, and the calculator will automatically be returned to normal mode at the end of the program sequence.

This automatic return to normal mode has not been available with earlier programs using the h12 technique, particularly if the printer is used (V6N8p4 of *TI PPC Notes*). This limitation was the reason that the fast mode instructions for Patrick's 1 minute 23 second calendar printer, my 12 digit modulo 210 speedy factor finder, and my 13 digit modulo 30 speedy factor finder all called for the use of a RST from the keyboard to return the calculator to normal mode after completion of fast mode calculations.

Implementation of automatic return to normal mode requires two additional program steps. An R/S command (code 91) must be placed at location 000. The fast mode program sequence must be terminated with a RST command (code 81). However, for the RST command to be recognized during fast mode it must be immediately preceded by CLR, 2nd CLR, Pause, Print, the sequence EE-INV-EE or one of the number keys. This is the same rule that applied for an R/S to be recognized during fast mode (item 6 on V6N9/10p19 of *TI PPC Notes*). A fast mode demonstration program was written which not only includes the transparent method of fast mode operation, but also illustrates other h12 fast mode concepts. The instructions for the demonstration program are:

1. Enter the program. An R/S is required at location 000. The remaining instructions are entered from locations 055 through 099. The Stflg 12 sequence at locations 071/072 can be entered by the key sequence 2nd-Stflg-B. The demonstration program is similar to Martin Neef's original counting program (V6N6p4 of *TI PPC Notes*) but with the ability to operate in either normal mode-or in fast mode.

2. Press A to demonstrate normal mode. After about 36 seconds see a 3. in the display. Since the value of pi was called at location 097, the display of a 3. shows that the calculator is in Fix 0 mode as set by the commands at locations 069/070. In normal mode the calculator runs by the Stflg-12-Rad-CLR sequence at locations 071 through 074, sets flag 2 in the process, and Jumps to location 082 when the B command at location 075 1s encountered. you need not worry about filling up the subroutine return register by the use of B rather than GTO-B (see V6N6/7p30 of *TI PPC Notes* for a discussion of that problem) since the subroutine-register will be cleared by the RST at location 099. The STO-16-R/S sequence at locations 076 through 078 is not used in normal mode, but is required to obtain the desired commands after the fast mode initialization process. The pair of Nop instructions at locations 087/088 is required to permit use of the same absolute address for both normal and fast mode for the INV-EQ-089 test at locations 093 through 096.

(*continues on next page)*

3. Initialize for fast mode by synthesizing an h12 command at location 072. The initialisation also changes other commands.

a. Press INV-Fix and see pi in the display.

b. Press 10-Op-17 and see 159.99 in the display. This sets the partitioning required by the initialization process.

c. Press CLR-STO-00-GTO-072. See a zero in the display.
This locates the program pointer for the initialization and clears data register R00.

d. Press Pgm-12-SBR-999 and see a flashing 0. in the display. Do not clear the flashing display at this step or at either of the two following steps.

e. Press R/S and see a flashing 0. 00 in the display. If R00 had not been cleared in step 3.c above there would have been a 100 in R00 left over from the normal mode demonstration. In that case there would have been a 1. 02 in the display at this point and the synthesis of hexadecimal code would not occur. More generally, if R00 contains anything other than a zero at the start of the Pgm-12-SBR-999 sequence then the synthesis of hexadecimal code will not work as planned.

f. Press DMS. See a flashing 0 in the display.

g. Press LRN. See 072 03 in the display.

h. Press Ins. See 072 03 in the display.

i. Press Ins a second time. See 072 03 in the display.

j. Press LRN-RST-CLR and see a zero in the display.

k. If a printer is attached press GTO-072-List. The listing will show altered instructions at locations 073 through 079, two inserted instructions at locations 080 and 081, and the instructions which had been in locations 080 through 099 pushed down two steps to locations 082 through 101. Without a printer press GTO-072-LRN and SST through the program to verify the changes due to initialization. The command at location 072 seems to be unaltered. Either a program listing or a readout in LRN mode shows a 12 both before and after the initialization process. Location 072 was selected for the synthesizing of the h12 command to demonstrate this curious phenomena. The code in ROM as indicated by step 3.g is 03. The double insert sequence at steps 3.h and 3.i causes a two time hexadecimal subtraction of 03 from the 12 in RAM yielding a residual hexadecimal code of 00 at location 072 or RAM. 00 lists and displays as 12, but provides the h12 command (to use Patrick Acosta's notation) which permits fast mode entry.

l. Press A to demonstrate operation in fast mode. After about 22 seconds see a 3. in the display. Again, the 3. indicates that the calculator is in Fix 0 mode.

m. Press GTO-065-LRN-Nop-LRN. The +/- instruction at location 065 has been changed to a Nop. This will change the sign of the initialization constant which must be in the display when the Stflg-h12 sequence is encountered. Press A to demonstrate fast mode with a positive initialization constant. After about 22 seconds see pi in the display. This demonstrates that certain initialization constants will cause the fix mode to change at fast mode entry, in this case to Fix 9. (*continues on next page)*

n. Press GTO-069-LRN-NOP-Nop-Lrn. The Fix 0 instruction which had been at locations 069/070 have been changed to Nop instructions. Press A to demonstrate fast mode without a controlled Fix 0 mode prior to fast mode entry. After about 22 seconds pi will appear in the display. This demonstrates that the Fix 9 mode is permissible at fast mode entry if the initialization constant in the display contains no fractional part--see V6N8p3 of *TI PPC Notes*.

o. Press Fix 2. Then press A. After-about 38 seconds see 3.14 in the display. This illustrates that fast mode entry with the Stflg-h12 sequence cannot be obtained if the fix mode is other than 0 or 9. To avoid potential problems with leftover fix modes it is advisable to include control of the fix mode prior to the fast mode entry sequence.

p. Press 8 to demonstrate normal mode. After about 36 seconds see 3.14 in the display. The display is still controlled by the Fix 2 mode.

q. Press INV-Fix to return to Fix 9 mode. press GTO-100-LRN-Nop-LRN. This changes the Pause instruction which preceded the RST instruction to a NOP instruction. Press A and watch the flashing **[** at the left edge of the display closely. After about 22 seconds the variations in intensity of the flashing **[** will stop, and a dim steady **[** will appear at the left edge of the display. You will be unable to clear this state without turning the calculator off. This demonstrates the necessity that RST be preceded by certain instructions if it is to be recognized during fast mode.

In a subsequent article I will explain the easy way to control the changed instructions in the seven program locations immediately following the 0 modulo 8 locations which must be used for hexadecimal code synthesis. In the meantime, happy hexadecimal programming.

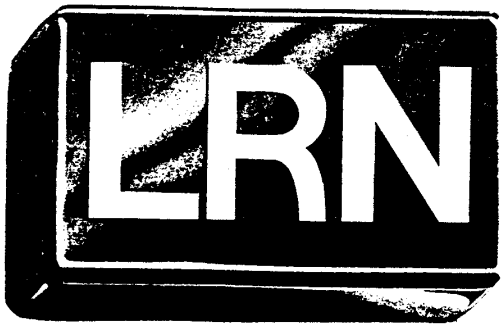
See listings on next page, please.

Before Initialization

000	91	R/S	050	00	0
001	00	0	051	00	0
002	00	0	052	00	0
003	00	0	053	00	0
004	00	0	054	00	0
005	00	0	055	76	LBL
006	00	0	056	11	A
007	00	0	057	02	2
008	00	0	058	85	+
009	00	0	059	02	2
010	00	0	060	52	EE
011	00	0	061	01	1
012	00	0	062	02	2
013	00	0	063	94	+/-
014	00	0	064	95	=
015	00	0	065	94	+/-
016	00	0	066	22	INV
017	00	0	067	52	EE
018	00	0	068	60	DEG
019	00	0	069	58	FIX
020	00	0	070	00	00
021	00	0	071	86	STF
022	00	0	072	12	12
023	00	0	073	70	RAD
024	00	0	074	25	CLR
025	00	0	075	12	B
026	00	0	076	42	STO
027	00	0	077	16	16
028	00	0	078	91	R/S
029	00	0	079	68	NOP
030	00	0	080	76	LBL
031	00	0	081	12	B
032	00	0	082	47	CMS
033	00	0	083	01	1
034	00	0	084	00	0
035	00	0	085	00	0
036	00	0	086	32	X:T
037	00	0	087	68	NOP
038	00	0	088	68	NOP
039	00	0	089	69	DP
040	00	0	090	20	20
041	00	0	091	43	RCL
042	00	0	092	00	00
043	00	0	093	22	INV
044	00	0	094	67	EQ
045	00	0	095	00	00
046	00	0	096	89	89
047	00	0	097	89	π
048	00	0	098	66	PAU
049	00	0	099	81	RST

After Initialization

072	12	B
073	68	NOP
074	68	NOP
075	70	RAD
076	61	GTO
077	00	00
078	84	84
079	91	R/S
080	80	GRD
081	68	NOP
082	76	LBL
083	12	B
084	47	CMS
085	01	1
086	00	0
087	00	0
088	32	X:T
089	68	NOP
090	68	NOP
091	69	DP
092	20	20
093	43	RCL
094	00	00
095	22	INV
096	67	EQ
097	00	00
098	89	89
099	89	π
100	66	PAU
101	81	RST



* TI PPC NOTES *
* *
****

NEWSLETTER OF THE
TI PERSONAL PROGRAMMABLE CALCULATOR
CLUB

9213 Lanham Severn Road
Lanham MD 20706 USA.

TI PPC NOTES v7n7/8p18/19

Creating Hex Keycodes - Patrick Acosta. There are several minor irritations in creating hex-codes via the ROM method. For instance,

(1) Hex-codes can only be created up to step 312.

(2) Different methods of getting into ROM must be used depending on which CROM is installed.

(3) It is not a trivial exercise to determine which keycode to write in RAM and how many times to press Ins to create the hex-code you want.

These problems are alleviated somewhat by a way of implanting hex-codes which doesn't even require you to leave the comfort of user RAM. The general method is to jump to. the first step of an octet without reloading the command buffer register. Then, assuming the command buffer register and the RAM octet contain the proper keycodes, one Ins will give the desired hex-code.

For example, to create the hex-code h12 (= 0C hexadecimal) at step 400, write the following program in an otherwise cleared memory:

000: Lbl B 5 0 9 EE 1 1 +/- + 4 = Deg Fix 0 Nop R/S +/-.

Then from the keyboard press B. This puts the number 4.000000005090 into the display register. Then press STF Ind 7 INV. This loads the above number into the flag register. Since the thirteenth digit is zero, the calculator stays in user memory. The tenth and eleventh digits send the calculator to the fiftieth octet, and the twelfth digit (which holds the byte within the octet), being a 9, sends the calculator to the first step of the octet but without re-loading the command buffer register. Thus, when you press LRN, you will see the keycode 94 at step 400. This keycode is left in the command buffer register from step 017. Now the situation is similar to that when creating hex-codes while in ROM in that the keycode seen in LRN mode is not the keycode actually in that location in RAM. Now, while in LRN mode, press Ins and h12 appears at step 400 (according to the generalized rule: hex-code = A0 minus the keycode apparently at step 400 plus the keycode actually at step 400 using hexadecimal arithmetic for the ones digits and decimal arithmetic for the tens digits, In this example, A0-94+00=0C.) Before running any program, get out of LRN mode and press CLR. Otherwise, the first keycode the calculator encounters will not be executed.

Any other hex-code can be created at step 400 by changing the keycode following the R/S of routine B and the hex-code can be put at any step divisible by eight up to step

872 (392 on the TI-58) by changing the ninth through eleventh digits of the number generated by routine B. Just be sure the R/S does not fall on the last step of an octet. That would leave the command buffer register empty which is no help at all.

This "RAM method" of creating hex-codes enables you to put hex-codes up to step 872 and in fewer keystrokes than the ROM method. Also, since this method is CROM independent, the instructions for hex-initializing a program are always the same no matter which CROM is installed. The only restrictions on partitioning are that the step you want the hex-code at must be in the partition and a 0 Op 17 partition can not be used since STF Ind 7 INV gives an error condition in that case.

The following program is one that I've found useful for a quick look at any desired hex-code. It uses the RAM method and dynamic code generation to implant the hex-code at step 176. The instructions are simple.

(1) Enter the hex-code into the display using the *hnn* notation.
The tens digit can be anything. The ones digit can be 0 through 5, corresponding to the hex digits A through F respectively.

(2) Press A. After two seconds, 4. will appear in the display.

(3) Press STF IND 7 INV LRN. You'll be at step 176.

(4) Press Ins and the hex-code appears immediately.

(5) Get out of LRN mode and press CLR before experimenting.

HEX-KEY CODE CREATOR- P. W. ACOSTA

```
000 76 LBL      016 07  7       032 42 STO
001 11  A       017 04  4       033 07  07
002 94 +/-      018 03  3       034 25 CLR
003 85  +       019 95  =       035 42 STO
004 01  1       020 42 STO      036 97  97
005 00  0       021 99  99      037 09  9
006 69 OP       022 02  2       038 69 OP
007 17  17      023 02  2       039 17  17
008 09  9       024 09  9       040 60 DEG
009 02  2       025 52 EE       041 58 FIX
010 00  0       026 01  1       042 00  0
011 06  6       027 01  1       043 61 GTO
012 93  .       028 94 +/-      044 01  01
013 09  9       029 85  +       045 63  63
014 01  1       030 04  4
015 00  0       031 95  =
```