HP28/48 FAQs (Frequently Asked Questions)

HP 28 stuff

o To find out the version of your HP 28C execute HEX then #A SYSEVAL. It will be 1BB or 1CC.

o The version of your HP 28S is 2BB

HP 28 and 48 stuff

o Flickering display - caused by fluorescent lights.

o Split a matrix in rows by using \Gs-

o Build a matrix from rows by using \Gs+

o Your calculator is very accurate, but not infinite. Don't expect 'SQ(\v/2)' to be exactly 2.

o Mind the values of your symbolic, constant, function evaluation flags.

o The 28 speed nybble has a value 0-f (default 7) which can be changed by software; the 48 is 'hard-wired' to top speed - 2MHz and cannot be changed.

o Transmitting data from the HP28 to the HP48 is done with INPRT which comes with the HP48 cable. INPRT is available from the archives.

HP 48 stuff

o To find out the version type ON-D, Backspace, EVAL. Type ON-C to get back to the real world. Also << #30794h SYSEVAL >>

o Type ON-x as follows: press ON, press x, release ON, release x.

o Flag 19 determines whether 2D makes a vector or a complex number

o MEM invokes garbage collection (memory cleanup). Garbage collection occurs automatically at intervals causing the 48 to pause for a moment. Use less stack to make it faster.

o ON-C gets all memory back, except variables. PICT, last stack, last arg, the stack, etc all use up memory.

o See manual (II) p627 for \Ga and other characters preceded by \.

o Stopping TAYLR gone mad; ON-C halts any operation & doesn't erase variables.

o The pinouts for the serial port are (looking at the it end on, screen facing up) 1-Field Ground 2-Receive 3-Send 4-Signal Ground, and connect to

pins 1, 2, 3, 7 on standard db25 pin RS232 connector, or nothing, 2, 3, on a db9 pin connector.

o Press \pi \->NUM to get pi. Same for e, i, MINR, MAXR.

o PC is DTE, modem is DCE. Use null modem &| gender bender.

o Chip48 is a machine language program which interprets Chip8, a gam language used on small systems in the 1970s. Chip8 games appear as a undecipherable string on the stack, but can be run using the Chip4 interpreter.

Jeremy Smith

GOTOS ON THE HP48SX

It occurred to me that one of the things about RPL that makes it so difficult to learn/program is the "unnatural" programming style that it enforces (to wit: the lack of GOTO).

I think that using system RPL it would not be too difficult to implement at least limited forms of GOTO.

Consider the following HP48 looping structures:

```
s f FOR c ... NEXT
s f FOR c ... i STEP
s f START ... NEXT
s f START ... i STEP
DO ... UNTIL ... END
WHILE ... REPEAT ... END
```

I suggest that:

- a "break" or "leave" keyword would effect a GOTO to the instruction after the associated NEXT, STEP, or END.

- a "continue" keyword would effect a GOTO to the associated NEXT, UNTIL or WHLILE (i.e., the next loop iteration). The STEP form of FOR/START would be difficult as you don't know where the expression to compute the step value begins. A slightly modified syntax could handle this, for example:

```
s f FOR c ... STEP ... END
s f START ... STEP ... END
```

- a "return" keyword would effect a GOTO to the associated \>> character.

By "associated" I mean the <whatever> that ends the current level. For example, if you had:

```
    DO
      ...
      CONTINUE
      ...
      DO
        ...
      UNTIL
        ...
      END
>>  UNTIL
      ...
    END
```

The CONTINUE to effect a GOTO to the UNTIL marked with >>.

It would really be nice if we could do a "pure" GOTO. Tags would come in handy, here. GOTO might work as:

```
    :tag: GOTO
    "tag" GOTO
```

(i.e., the target could be specified directly by a tag on the GOTO or indirectly by a string on level 1.) The target would be identified by a :tag: on the target instruction.

Now in this Bill Wickes was correct: you can get into lots of messy stack/return frame issues. There would probably have to be a restriction such as the :tag: must be at the same level of \<< ... \>> nesting as the GOTO.

Now to turn this over to the Saturn coders... (:-). (Then make and distribute a 48 library with these keywords!)

See you soon,

Craig

From: "Craig A. Finseth" <fin@unet.unet.umn.edu>

And in reply from our Chairman ...

Hi Craig,

Thanks for your GOTO thoughts! I have noted them for further consideration. One thing Bill said at the Open Forum was that GOTO is difficult in RPL because the programs are not at a fixed position in RAM. This struck me as no more than a weak excuse, since GOTOs can be stored as relative jumps, not affected by movement of the program code. After all, that's the way GOTOs are stored in the HP-41 family! Jumping out of a loop (or into one) is more difficult - but the HP48 already deals with that, since an IFERR with a loop inside it can happily deal with closing the loop if an error occurs. The real problem seems to be that of indirect GOTOs, such that a GOTO can jump to another part of the same loop, in which case the loop should not be killed, or it might jump out of the loop, (or even _back_ into a loop, as Fortran used to allow) and in each case the RPL driver must work out what to do. That's why Forth provides only a LEAVE command, which skips to the first word after a loop - one of your suggestions. As you say, with system RPL available, a GTO might be possible - especially if indirect GOTOs are not allowed.

Wlodek Mier-Jedrzejowicz