HP Forum Archive 15

[Return to Index | Top of Index]

How many stack levels?

*Message #1 Posted by **Klaus** on 15 July 2005, 8:15 a.m.*

Has anyone ever calculated the stack levels needed to solve any problem? Since there are some projects to build custom rpn devices (like openrpn), I wondered if they just use 4 (like most HPs), or a infinite number, or if they did some serious calculations? It would be amusing defining the number of stacklevels to solve any problem - one could start with a problem that uses many stacklevels and then add a factor/therm/expression to see if the number of stacklevels grows. I it doesn't grow the stack, then we have the maximum number of stacklevels needed to solve any problem. Writing a compiler I got the impression that you don't need a infinite number of stacklevels, and I wonder if it is true?

Have a nice weekend!

Re: How many stack levels?

*Message #2 Posted by **Namir** on 15 July 2005, 8:54 a.m.,*
in response to message #1 by Klaus

Klaus,

I think that a dynamic stack level works well (with commands that can operate on a specific number of levels). My guess is the folks at HP asked themselves the same question. They came up with the dynamic stack that appeared with the HP-28C.

If you stick with a fixed stack, then I'd say a four-level or five-level stack is good. Use memory registers to store additional info and then pull them into the stack later.

Namir

Re: How many stack levels?

*Message #3 Posted by **John Limpert** on 15 July 2005, 9:02 a.m.,*
in response to message #1 by Klaus

Assuming you have storage registers, or some other type of memory, the problem can be restructured to use no more than N stack levels, where N is the maximum number of arguments for an operation. Otherwise, problems can be constructed that use an arbitrarily large number of stack levels.

Re: How many stack levels?

*Message #4 Posted by **Valentin Albillo** on 15 July 2005, 10:09 a.m.,*
in response to message #1 by Klaus

Hi, Klaus:

Klaus posted:

"Has anyone ever calculated the stack levels needed to solve any problem?"

To solve *any* problem you need an infinite stack. It's easy to concoct a problem which has any desired number of pending operations and operands.

That said, *most* problems can be solved with a fairly low number of stack (i.e., pending operands and operators) registers. Vintage SHARP handhelds had something like 8 stack registers for operands and 16 stack registers for operators, and never at all have I experienced a single "Stack full" or equivalent message.

In my RPN experience, 4 levels + LASTX are adequate for most problems, and perhaps a very convoluted one would require 5 or 6, but that's all, and very infrequently, so I'm happy with a 4-level + LASTX stack and never have felt the need for more. If a 5th number is *ever* needed, which happens once in a blue moon, I'd rather simply STOre it somewhere than have to deal with a 5-level stack at *all* times, whether I need it or not.

4 levels is optimum: you'll solve the vast majority of problems with them, and you can easily and effortlessly hold a mental picture of the stack's contents, what's in each register, and how to roll down, roll up, or exchange them to get what you need where you need it when you need it.

Add more stacks levels, or even an indefinite number of them (i.e., RPL-like) and you lose that.

I'd rather not.

Best regards from V.

Re: How many stack levels?

*Message #5 Posted by **Namir** on 15 July 2005, 12:04 p.m.,*
in response to message #1 by Klaus

I guess how you use the stack also matters. I am talking about the ratio of (values input)/(operations). An efficient way keeps this ratio at 1. An inefficient way would make the ratio higher than 1.

On the topics of memory registers. My question is where is your data coming from? A real time data acquisition process or a block of memory registers to begin with? In case of the latter, you can certainly store intermediate results in memory registers. My own guess is that a four-level stack and a handfull of memory registers (2, 3?) can do the job of evaluating expressions.

I have written several versions of math expression parsers that convert an algebraic expression into a an RPN expression. The RPN expression can then be evaluated as many times as needed. I typically use a limit stack and never had problems in the evaluation.

Namir

Re: How many stack levels?

*Message #6 Posted by **Guillermo** on 15 July 2005, 12:05 p.m.,*
in response to message #1 by Klaus

Until now, I believed that four stack levels was enough for any problem.

But I found a problem that can´t be solved with 4 levels.

My belief of years were destroyed ...

Re: How many stack levels?

*Message #7 Posted by **Klaus** on 15 July 2005, 12:58 p.m.,*
in response to message #6 by Guillermo

Yes, I tried to do a stack algorithm with rules of precedence, but similar to the flwochart in the HP-45 manual. It seems you really need infinite stacklevels:

a+b*c^(e+f*g^(h+i*j^(...))))

My question was just of theoretical nature, no practical use...

Thanks for the answers&inspiration!

Klaus

Re: How many stack levels?

*Message #8 Posted by **Guillermo C** on 15 July 2005, 2:17 p.m.,*
in response to message #7 by Klaus

I was talking about theoretical nature also. Your problem (with finite number of repetitions) can be solved with two levels. I mean if you want to solve it from left to right then you will need infinite number of levels, but if you begin from innermost parenthesis then you need only 2 levels. The problem I found need at least 5 levels no matter where you begin to solve:

((1+1)*(1+2))^3/((1+3)*(1+4))^2+((2+1)*(2+2))^3/((2+3)*(2+4))^2

Re: How many stack levels?

*Message #9 Posted by **Namir** on 15 July 2005, 3:13 p.m.,*
in response to message #8 by Guillermo C

I mentioned in an earlier message that using a few registers can help. The expression:

((1+1)*(1+2))^3/((1+3)*(1+4))^2+((2+1)*(2+2))^3/((2+3)*(2+4))^2

Can use a memory register to store the result of:

((1+1)*(1+2))^3/((1+3)*(1+4))^2+

And then evaluate in the stack:

((2+1)*(2+2))^3/((2+3)*(2+4))^2

which is then added to the contents of the memory register, or the value of the memory register is recalled for a final addition.

Using this approach I can solve an expression like:

((1+1)*(1+2))^3/((1+3)*(1+4))^2 + ((2+1)*(2+2))^3/((2+3)*(2+4))^2 + ((2+7)*(2+5))^3/((2+3)*(3+4))^2 + ((3+1)*(6+2))^3/((7+3)*(2+4))^2 + ...

Re: How many stack levels?

*Message #10 Posted by **Klaus** on 15 July 2005, 4:09 p.m.,*
in response to message #8 by Guillermo C

Yes, I came to exactly the same thoughts after posting my message, and wrote the following text offline, so forgive me repeating some of your ideas:

My equation above takes infinite stacklevels when parsing from left to right. Parsing from right to left needs only a limited number of stacklevels. Parsing from the innermost (correct word?) parenthesis (like the rpn-user does it) will also result in a fixed number of stacklevels for this equation. Perhaps I can solve it like this:

Let A be an equation that needs n stacklevels to be evaluated. Then A+b,A-b,A*b,A/b,A^b , b is a number, can be evaluated by evaluating A (and needing n levels), storing A in the top of Stack, putting b on top of A (needing 2 levels) and executing the operation.

The question is how this will behave with b being another expression.

Re: How many stack levels?

*Message #11 Posted by **htom** on 15 July 2005, 6:22 p.m.,*
in response to message #10 by Klaus

Let A take n levels, and A' be A with different values.

Then A+A' will take n+1 levels; call this expression B.

B+B' will take (n+1)+1 = n + 2 levels.

Eventually you need an infinite stack, or storage external to the stack.

Re: How many stack levels?

*Message #12 Posted by **Guillermo C** on 18 July 2005, 8:47 a.m.,*
in response to message #11 by htom

I not so sure.

Please, could you show me an example needing 10 stack levels? Could you show me an example needing 6 levels?

If they exists I would like to see them!

I am always suposing you only have the stack, no other registers. Also I supose you begin to solve in a smart order, not simply left to right.

Guillermo

Re: How many stack levels?

*Message #13 Posted by **Klaus** on 18 July 2005, 3:04 p.m.,*
in response to message #12 by Guillermo C

Thank you htom for your solution! I will try to show an example:

(1+2) uses 2 levels (1+2)+(3+4) uses 3 levels ((1+2)+(3+4))+((5+6)+(7+8) uses 4 levels (((1+2)+(3+4))+((5+6)+(7+8)))+(((9+10)+(11+12))+((13+14)+(15+16))) uses 5 levels ((((1+2)+...+(15+16))))+((((17+18)+...+(a+b)))) takes 6 levels (((((1+2)+...+(a+b)))))+(((((c+d)+...+(y+z))))) takes 7 levels

By expanding the therm, you can overflow any limited stack.

Re: How many stack levels?

*Message #14 Posted by **Guillermo C** on 19 July 2005, 10:44 a.m.,*
in response to message #13 by Klaus

((1+2)+(3+4))+((5+6)+(7+8))

is equivalent to

1+2+3+4+5+6+7+8

so you only need two stack levels!

The same for the rest of the examples!

What I think interesting to find is the maximum stack levels necessary to solve any problem, if you solve the problem in the smartest way (minimizing stack levels)

(1+2)+(3+4) only needs 2 levels. (1+2)*(3+4) needs 3 levels

Considering this, until now I never found a problem needing more than 5 stack levels...

Re: How many stack levels?

*Message #15 Posted by **htom** on 19 July 2005, 2:50 p.m.,*
in response to message #14 by Guillermo C

That some expressions can be so collapsed does not mean that all of them can be.

```
In the following, the formation


A  B , term  term , factor  factor , or expression  expression
 op        op             op                      op


means (left op right) so that I don't have to type hundreds of ().


a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F
 +   +   +   +   +   +   +   +   +   +   +   +   +   +   +   +
   *       *       *       *       *       *       *       *
       -               -               -               -
               /                               /
                                   ^
```

This evaluation calls for 6 layers of stack. If you substitute it for each of the letters, each letter will call for six layers (that is, it will take six layers to evaluate a, b, c ...), and combining them will call for another five////, for a total of 11// layers.

edit -- opps. will call for another six, total of 12 layers (forgot to evaluate b in counting)

Edited: 20 July 2005, 12:23 a.m.

Re: How many stack levels?

*Message #16 Posted by **Guillermo C** on 20 July 2005, 8:36 a.m.,*
in response to message #15 by htom

I thought that a limited stack could solve any problem, but your explanation shows that I was wrong! Now I see clearly. Thanks!

I had the solution at hand but I didn´t realize. My 5 level problem have the same "binary tree" structure that your explanation shows.

Your way to put the problem helps to show the relation between number of stack levels and number of operators:

2 stak levels can solve any problem up to 3 operands.

3 stak levels can solve any problem up to 7 operands.

4 stak levels can solve any problem up to 15 operands.

5 stak levels can solve any problem up to 31 operands.

n stak levels can solve any problem up to (2^n)-1 operands.

Of course if a problem has more operands but with a different structure could need less stack levels.

Re: How many stack levels?

*Message #17 Posted by **bill** on 18 July 2005, 4:32 p.m.,*
in response to message #6 by Guillermo

Hi Guillermo,

I have run into this problem many times even early on, but it never hurt my belief in RPN....

Regards,

Bill

Re: How many stack levels?

*Message #18 Posted by **Tony** on 15 July 2005, 3:38 p.m.,*
in response to message #1 by Klaus

One situation where I find unlimited stack useful is for book-keeping jobs where lists of amounts need to be added and reconciled against a total. I put all the amounts onto the stack, then use my "STOT" program to get a sub-total. If that looks wrong, I can go back over the stack and edit it, then take another subtotal. Then finally use "TOTAL" to wipe the stack and return the total.

Which fixed-depth RPN calc has the most levels?

*Message #19 Posted by **Katie Wasserman** on 16 July 2005, 12:51 a.m.,*
in response to message #1 by Klaus

The subject states my question.

Not counting LASTx all the HP's except the 9100 and 9810 have 4 levels (I think). The Heathkit OC-1401 has 5 levels, are there others with 5 or more?

Re: Which fixed-depth RPN calc has the most levels?

*Message #20 Posted by **John Limpert** on 16 July 2005, 5:50 a.m.,*
in response to message #19 by Katie Wasserman

It isn't a calculator, but the Intel 8087 math coprocessor has an 8 level stack. It operates in a manner similar to RPN calculators.

Re: How many stack levels?

*Message #21 Posted by **Hugh Evans** on 17 July 2005, 3:11 a.m.,*
in response to message #1 by Klaus

We've never done any calculations on this for OpenRPN, just going with an infinite stack. This is an interesting thread. Rather than focusing on the hypothetical number of stack levels needed for any calculation, what would really be interesting is the number of levels needed for practical (yet extreme) use. My guess it that such an estimate will be suprisingly small number.

Best Regards, HDE

RPN stack levels: My recommendation

*Message #22 Posted by **Karl Schneider** on 17 July 2005, 4:53 p.m.,*
in response to message #21 by Hugh Evans

Hugh --

I would recommend for OpenRPN that an RPN calculator should have a user-settable fixed stack depth, with 2 as the minimum and 9 to 19 as the maximum, and 4 as the default. Commands for changing the stack depth should be short, simple, and programmable, such as:

```
f STACK 9  (9 levels)
f STACK .9 (19 levels)
```

Classical RPN stack functions Exchange, Roll Up, and Roll Down should be provided, along with direct access to stack elements that the 41/42-series models provide. (New identifiers for elements beyond "t" would need to be provided, such as "L5" through "L19").

I agree with Valentin that 4 levels is almost always adequate, but there are situations in which some forethought is necessary to avoid pushing data off the top.

Having up to 19 levels available would allow working with a large set of numbers -- which the dynamically-sized stack of the RPL-based 28/48/49 models allow -- without the numerous and complicated RPL stack operations.

-- KS

Re: RPN stack levels: My recommendation

*Message #23 Posted by **Jonathan Purvis (New Zealand)** on 17 July 2005, 7:45 p.m.,*
in response to message #22 by Karl Schneider

One thing that has always confused me about RPN is why anyone would want **less** than the maximum stack depth? With a dynamically sized stack, you can have as many items on the stack as you can fit in your head. Also, you don't have to worry about how many stack levels a subroutine uses, since it can't overwrite any of your values.

Re: RPN stack levels: My recommendation

*Message #24 Posted by **db (martinez, ca.)** on 18 July 2005, 2:00 a.m.,*
in response to message #23 by Jonathan Purvis (New Zealand)

It's useful to have a finite stack and a replicating "T" register if you sometimes do repetitive arithmetic. An example would be a surveyor checking cut sheets. Any HP, Corvus, Omron, or APF rpn and a couple of Novus/Nat Semi units as well as Katie's fabled Heathkit will operate like that.

Why a limited stack depth?

*Message #25 Posted by **Karl Schneider** on 18 July 2005, 3:37 a.m.,*
in response to message #23 by Jonathan Purvis (New Zealand)

> Quote:
>
> One thing that has always confused me about RPN is why anyone would want less than the maximum stack depth? With a dynamically sized stack, you can have as many items on the stack as you can fit in your head.

It becomes difficult for most people (including myself) to keep track of more than a few values on the stack, and their changing positions as computations are performed.

Four stack elements is adequate for almost any equation, if the user takes a true "pencil and paper"-style approach, working from the inside out, and checking every intermediate result. Having only three stack elements (as did National Semiconductor RPN calc's of the 1970's) can be constraining.

Four elements also gives easy access to every element using only x<>y, Rdown and Rup, even without the direct access that the 41/42 models did. (Compare that with the confusing plethora of stack-mainipulation commands offered by the dynamic-stack 28/48/49 models.)

> Quote:
>
> Also, you don't have to worry about how many stack levels a subroutine uses, since it can't overwrite any of your values.

True, but one way around that is to have all routines take input arguments from memory registers and named variables as much as possible, instead of expecting the arguments to be in certain positions on the stack. This is good programming practice.

-- KS

Re: Why a limited stack depth?

*Message #26 Posted by **.** on 18 July 2005, 4:08 a.m.,*
in response to message #25 by Karl Schneider

> Quote:
>
> > Quote:
> >
> > Also, you don't have to worry about how many stack levels a subroutine uses, since it can't overwrite any of your values.
>
> True, but one way around that is to have all routines take input arguments from memory registers and named variables as much as possible, instead of expecting the arguments to be in certain positions on the stack. This is good programming practice.

How so? It seems to me to be very poor practice in general. Using named registers is like using global variables - something to be discouraged. Such functions are neither recursive nor reentrant.

Personally I only use RPN systems where I can see several stack levels on the screen. I find classic 4-level RPN too much work compared with modern algebraic calculators.

Re: Why a limited stack depth?

*Message #27 Posted by **Eric Smith** on 18 July 2005, 2:14 p.m.,*
in response to message #26 by .

> Quote:
>
> > Quote:
> >
> > one way around that is to have all routines take input arguments from memory registers and named variables as much as possible
>
> It seems to me to be very poor practice in general. Using named registers is like using global variables - something to be discouraged.

That's why RPL has local variables. The arguments to a function are still passed on the stack, but get moved into local variables so that they may be referenced by name. That way you don't have to track how far up the stack they are at any given time during the execution of the function.

The drawback is that they are less efficient than direct stack access. For a Saturn running at a relatively slow clock rate, that can be a problem. For a hypothetical new calculator that runs much faster, it would matter less.

Local variables also make it much easier to understand and modify an RPL function.

Re: Why a limited stack depth?

*Message #28 Posted by **Karl Schneider** on 18 July 2005, 10:12 p.m.,*
in response to message #26 by .

"." --

OK, I'll admit that my statement wasn't very precise, and will clarify it here. Again, I'm referring to RPN programming, not RPL, and especially not recursion-supporting high-level languages (e.g., C++).

The fixed stack, numbered registers, and named variables (HP-42S only) are all global on RPN calculators. There's nothing unsound about placing input arguments to an RPN subroutine on the stack. The risk of error occurs when a program is carrying arguments on the stack that are needed *upon return from* XEQ/GSB, INPUT, or VIEW.

- In the case of XEQ/GSB, the subroutine (or chain of subroutines) must be designed to leave the stack in the expected state upon return to the calling program.
- In the case of INPUT or VIEW, the user might ruin the stack while doing calculations before returning with R/S.

To avoid these problems, it's better for the program to copy the needed arguments to a numbered register or named variable prior to XEQ/GSB, INPUT, or VIEW, then recall them as needed rather than expect them to be there after the "interruption."

-- KS

Re: RPN stack levels: My recommendation

*Message #29 Posted by **Vassilis Prevelakis** on 18 July 2005, 5:36 a.m.,*
in response to message #22 by Karl Schneider

> Quote:
>
> Hugh --
>
> I would recommend for OpenRPN that an RPN calculator should have a user-settable fixed stack depth, with 2 as the minimum and 9 to 19 as the maximum, and 4 as the default.

Having a stack size different from the default (4) would break programs that assume that the stack is 4 deep (e.g. assuming that T replicates its value as the stack drops, that 4 ROLL-UPs or 4 ROLL-DOWNs would bring the stack to its original state, etc)

If you need more stack space (e.g. args, or temporary values) then you may be better off with a sliding register window, whereby you can position your 4 register stack anywhere within a register array, or, perhaps have a stack of stacks (i.e. being able to "push" the 4+1 register stack in a larger stack).

In this way, when you want a fresh stack to perform some calculations or to call a routine without destroying the cointents of the existing stack, you just push it, call your function and then pop it.

**vp

Re: RPN stack levels: My recommendation

*Message #30 Posted by **Karl Schneider** on 18 July 2005, 9:46 p.m.,*
in response to message #29 by Vassilis Prevelakis

Vasssilis posted,

> Quote:
>
> Having a stack size different from the default (4) would break programs that assume that the stack is 4 deep (e.g. assuming that T replicates its value as the stack drops, that 4 ROLL-UPs or 4 ROLL-DOWNs would bring the stack to its original state, etc)

Absolutely right. That's why I said that the "set stack depth" function should be programmable, so that a "STACK 4" instruction would guarantee compatibility of older RPN programs.

> Quote:
>
> If you need more stack space (e.g. args, or temporary values) then you may be better off with a sliding register window, whereby you can position your 4 register stack anywhere within a register array, or, perhaps have a stack of stacks (i.e. being able to "push" the 4+1 register stack in a larger stack).

> In this way, when you want a fresh stack to perform some calculations or to call a routine without destroying the cointents of the existing stack, you just push it, call your function and then pop it.

That's rather complicated, in my estimation. Sounds more like RPL philosophy than RPN. Also, with ample RAM available for low cost, why overwite register space when plenty of free space is probably available?

-- KS

Edited: 18 July 2005, 10:45 p.m.

Re: RPN stack levels: My recommendation

*Message #31 Posted by **John L. Shelton** on 19 July 2005, 1:40 a.m.,*
in response to message #29 by Vassilis Prevelakis

It seems important to differentiate between the issues of why the original RPN calculators had 3, 4, or 5 level stacks from the more modern "what's the right thing to do now." It's almost certain that the decision to have small, limited stacks was a compromise due to engineering cost, rather than a decision that 3, 4, or 5 are the "best" sizes of stack. A desktop calculator with a display of arbitrary size would have been much more expensive.

Once one makes a decision to have a small, fixed size stack, then one can decide whether the top stack variable replicates on stack drop. Was the original goal to have stack drop replication, and let that drive the decision to have a small, fixed-size stack? Probably not. It was probably a consequence, not a driver.

So, for nostalgic reasons, one might design a modern calculator with a fixed size stack. Particularly if compatibility with very old programs is important. But as we have found in many, many areas of computer programming, a fixed limit size to nearly anything winds up being a big headache later. (remember DOS 640kbytes? We could probably fill a screen with thread-headers on just that topic alone.)

In summary: For nostalgia: fixed size. For modern design: unlimited stacks.

Re: RPN stack levels

*Message #32 Posted by **Karl Schneider** on 20 July 2005, 2:14 a.m.,*
in response to message #31 by John L. Shelton

John Shelton posted:

> Quote:
>
> It's almost certain that the decision to have small, limited stacks was a compromise due to engineering cost, rather than a decision that 3, 4, or 5 are the "best" sizes of stack. A desktop calculator with a display of arbitrary size would have been much more expensive.

True, the 4-element stack of the HP-35 was probably a compromise of what was minimally sufficient and what could be done with limited resources in 1972. 4 is still enough for most every practical application, which is one reason why the stack on RPN HP's RPN never got larger (though the 42S should have had a larger stack due to complex-number operations). The 42S, BTW, is the first RPN HP handheld calc with a two-line display, after 14 years of production.

> Quote:
>
> Once one makes a decision to have a small, fixed size stack, then one can decide whether the top stack variable replicates on stack drop. Was the original goal to have stack drop replication, and let that drive the decision to have a small, fixed-size stack? Probably not. It was probably a consequence, not a driver.

Well, with a fixed stack, the choices were to zero out the "popped" elements or to replicate the top element. Replication gives the user an endless supply of repeatedly-used values -- useful for constant arithmetic.

> Quote:
>
> But as we have found in many, many areas of computer programming, a fixed limit size to nearly anything winds up being a big headache later. (remember DOS 640kbytes? ...

Your analogy is imperfect, I would say. The 640-kB memory "partition" was a serious encumbrance because there really is no limit to what one might wish a PC to do. However, I think there is a limit to what the vast majority of users would reasonably expect a true handheld calculator to do. Beyond that limit, a PC or even a PDA would be a better-sized tool for the task. One could argue that the HP-48/49 and TI-89 are overkill, but there was no practical alternative at the time they were designed.

If the task remains the same, the tool needn't get more elaborate. Has the shovel been fundamentally "modernized"? How many of us actually use an electric toothbrush?

> Quote:
>
> In summary: For nostalgia: fixed size. For modern design: unlimited stacks

"Nostalgia"? If modernity means replacing

x<>y, Rdn, Rup

with

DUP, SWAP, DROP, OVER, ROT, UNROT, ROLL, ROLLD, PICK, UNPICK, PICK3, DEPTH, DUP2, DUPN, DROP2, DROPN, DUPDUP, NIP, NDUPN

and the infamous (beep!) "DROP Error: Too few arguments",

then take me down "memory lane" every time!

-- Best regards from KS

Re: RPN stack levels

*Message #33 Posted by **Valentin Albillo** on 20 July 2005, 4:31 a.m.,*
in response to message #32 by Karl Schneider

Karl posted:

""Nostalgia"? If modernity means replacing x<>y, Rdn, Rup with DUP, SWAP, DROP, OVER, ROT, UNROT, ROLL, ROLLD, PICK, UNPICK, PICK3, DEPTH, DUP2, DUPN, DROP2, DROPN, DUPDUP, NIP, NDUPN and the infamous (beep!) "DROP Error: Too few arguments",

then take me down "memory lane" every time!"

Amen ! I couldn't agree more. And it isn't just the overwhelming plethora of dismayingly redundant stack operators, it's also that while any RPN user worth his/her salt can easily keep in mind the contents of a 4-level stack without resorting to pencil and paper, fully aware of what is where, the task becomes sort of a "Mission: Impossible" case when dealing with a significantly larger stack, not to mention an "arbitrary length" one.

That's one of the main reasons why it's fairly easy and even *fun* to write a program for an HP-15C, HP-41C, or HP-42S, while doing the same in RPL demands much more effort on the user's part and easily becomes an exercice in frustration.

Best regards from V.

Re: RPN stack levels

*Message #34 Posted by **htom** on 21 July 2005, 4:38 p.m.,*
in response to message #33 by Valentin Albillo

After a period where I used Forth, I began to desire a command that would take strings like ztxly and deliver those values to the LXYZT registers. The expanded version allowed the digits 0-9 as recall locations for values as well, and then the extended version a preceeding "." as an indirect operator. Someone remembered the old CDC "indirect on negative address" thing then, and that's when I came to my senses.

As someone else pointed out, the major problem with pushing an entire stack is arranging the return value(s). It's relatively simple to design the syntax of the command, but very expensive in time, codespace, or hardware.

(Designate initial values as LXYZT, stack values on return execution as lxyzt, and add a five character field that holds the locations of the five values to be returned. And the status of the enter flag.)

Re: RPN stack levels: My recommendation

*Message #35 Posted by **Hugh Evans** on 21 July 2005, 3:33 p.m.,*
in response to message #22 by Karl Schneider

Since OpenRPN allows free access to our source code end users can modify, compile, and distribute ROMs freely. Therefore changing the depth of the stack is entirely possible. *fix, the programming language of OpenRPN, uses an RPL style stack. For the sake of nostaglia a variety of emulators and simulators could be written or ported and used.

[Return to Index | Top of Index]

Go back to the main exhibit hall