HP Forum Archive 13

[Return to Index | Top of Index]

Tradeoffs: dynamic register/program allocation vs. Simply More Memory . . .

*Message #1 Posted by **Paul Brogger** on 25 June 2003, 5:54 p.m.*

I was playing with my 34c today, and got to thinking . . . (Here comes the scary part!)

I wonder how much ROM is devoted to dynamic allocation of a relative handful of memory locations between registers and programs, and how expensive was it to code and debug the relevant firmware?

Certainly at recent prices for memory and/or chip real estate, you'd think they'd Keep It Simply Simple and establish fixed amounts of program and register memory, as in my old 29C. (What, then, explains the HP-32S/SII?)

But was CMOS Continuous Memory *SO* expensive back in the 70's that the "shared" approach (with concomitant ROM space, program development, debugging, testing and documentation costs) was really a wise investment? It's probably a measure of how spoilt I've become, but it seems hard to believe . . .

Maybe someone could fire up the ol' Way-back Machine and clue us in on just how the internals of register vs. program memory management works in the Spice and low-end Pioneer programmables?

Re: Tradeoffs: dynamic register/program allocation vs. Simply More Memory . . .

*Message #2 Posted by **Trent Moseley** on 25 June 2003, 10:31 p.m.,*
in response to message #1 by Paul Brogger

It's a lovely thought. What is is is; what was was was. There is no way to change the past.

tm

Re: Tradeoffs: dynamic register/program allocation vs. Simply More Memory . . .

*Message #3 Posted by **Vieira, Luiz C.** on 26 June 2003, 3:49 a.m.,*
in response to message #1 by Paul Brogger

Hi, Paul;

I'm not delving into the specific subject of how is it done, but just to make a simple recall.

The HP34C and the HP38C/E are the first ones to show the movable partition. In both cases, program memory "request" determines the amount of memory available for storage registers. Except for the HP95C (I don't know how does it work), I think there are no other LED-display HP calculators wiht this movable partition. Both Texas TI58 and TI59 also have it.

The HP41 and all Voyagers show this, being the HP15C the only Voyager that allows user selection for the movable partition. The HP41 and the HP42 show the same control capacity, and I am not sure about the HP32S/SII.

The additional O.S. parts to allow the movalble partition to exists seem to be not so big as expected. The only thing that changes is that instead of checking for a fixed memory position or defining it, the system now checks for a value stored somewhere else. The HP55 accepts [STO] and [RCL] from [0] to [9] and from [.] [0] to [.][9]. The HP33C/E and the HP25/25C accept only [STO] and [RCL] from [0] to [7]. What happens if we key in [STO][8] in an HP25? The system is pre-programmed in such a way that [STO] 8 and [RCL] 8 are neither recorded in a program nor performed by keyboard direct operation. Does the O.s. perform a check or not? What is the diference of checking for [STO] 8 and [STO][CHS]? Both are not allowed in the HP33C/E or the HP25/25C and I don't know if they are both refused the same way. I don't dare guessing. If we take both TI58/59, any key is accepted after [STO] or [RCL], with a few exceptions. In their case, movable partition is user-defined by [OP] 17, no authomatic adjustment available, and [STO]/[RCL] are from 00 to 99 only.

Back to Spices, you can type in a program in the HP38E/C ot the HP34C thats uses any register, i.e., while typing in a program, storage register availability is not checked, of course. At the moment any register is accessed, a "go-no_go" is performed to see if it exists. I think that the movable partition "location" is stored and updated each time one register is turned into seven program steps/bytes. As programs NORMALY do not generate otehr programs, while a program is running the last value assumed for the movable partition is checked, and never updated, everytime a register "position" is called. This is not done in fixed-scheme memory calculators.

I think many considerations can be made, but what actualy happens on each system can only be actually "seen" if the O.S. is debuged. I think that the one with the most complex organization is the HP42S. It probably mannages mamory contents in a way closer to the HP28's way. The HP42S emulates the HP41 in a higher level processor, as we know. A better mamory mannagement is also needed to emulate all particular HP41 operation and data register. And the HP41 has an exclusive user-defined movalbe partition, like the HP15C. This has not been changed in the HP42S.

Comments? Corrections? All welcome.

Luiz C. Vieira - Brazil

Edited: 26 June 2003, 10:42 a.m. after one or more responses were posted

Re: Tradeoffs: dynamic register/program allocation vs. Simply More Memory . . .

*Message #4 Posted by **Valentin Albillo** on 26 June 2003, 5:54 a.m.,*
in response to message #3 by Vieira, Luiz C.

Very interesting post, Luiz, I've enjoyed reading it, but there's a point I would like to make. You say:

"I think that the one with the most complex organization is the HP42S. It probably mannages mamory contents in a way closer to the HP28's way."

Assuming you're not considering the HP28/28S/48/49 machines, I think the HP-15C has also quite a complex memory management to do, for it has to allocate and administer (quite scarce) memory for all these items:

- program memory (steps 001-448)
- dynamically allocated user storage registers (R02-R65)
- matrices (A-E)
- the parallel complex RPN stack (X,Y,Z,T and LAST X)
- scratch registers for SOLVE (5 registers)
- scratch registers for INTEGRATE (23 registers)

in all their diverse configurations of allocated/deallocated status and variable sizes. To that, you could also add the permanent real RPN stack (X,Y,Z,T,L) and permanent user storage registers (RI, R00, R01), which are available at all times.

Also, there's the fact that there are a number of 'hidden' data stored here and there which also need to be allocated and managed, such as the row and column exchanges done to an LU-decomposed matrix, which have to be remembered together with the actual element values in order to accurately compute in a future moment the inverse and/or determinant of the matrix, or to use it in a 'division' operation.

Finally, SOLVE can call INTEGRATE and INTEGRATE can call SOLVE (though no directly or indirectly recursive calls are allowed) and this has to be managed too, to make them share some of their allocated registers so that no more than 23 are used in that case, too.

The point is, of course, that all this seems complex enough to me. The 42S' management is much more complex, of course, but at least it doesn't have to deal with allocating memory for a parallel RPN stack ! :-)

You're right about the HP15C.

*Message #5 Posted by **Vieira, Luiz C. (Brazil)** on 26 June 2003, 12:37 p.m.,*
in response to message #4 by Valentin Albillo

Hello, Valentin;

thank you for your comments. As always, yours are also thoughtful, precise and teasing ;^).

The fact that leads me to conclude the HP42S as having one of the most complex O.S. for an RPN calculator is because it has a different, floating memory organization. Based on some facts, I tell you it uses pointers to existing "objects" instead of available memory to all of them.

As it happens in RPL-structured machines, replicated objects in the HP42S do not use space. Let's take a simple example and start by clearing STACK and LASTX:

```
[CLST] [+]  [×: 0.0000    ]   @ this places 0.0000 in all stack
[MEM]       [6977 Bytes   ]
```

This is what's shown when the calculator is completely cleared. Now type in any floating point number and check memory.

```
123.456      [×: 123.456_ ]
[MEM]        [6963 Bytes  ]
```

See? Fourteen bytes are automatically allocated to hold the new entered data. Now fill the stack with it and check available memory again:

```
[ENTER]
[ENTER]
[ENTER]      [×: 123.4560 ]
[MEM]        [6963 Bytes  ]
```

Should we conclude that the stack occupies 14 bytes? The fact is that *no extra bytes are used to hold copies of the same data in all stack registers*. Now let's clear all stack and enter different numbers at a time and check available memory for each entered data:

```
[CLST] [MEM]      [6977 Bytes  ]
123  [MEM]        [6963 Bytes  ]
[ENTER] 123 [MEM] [6950 Bytes  ]
[ENTER] 123 [MEM] [6936 Bytes  ]
[ENTER] 123 [MEM] [6923 Bytes  ]
```

Each entry consumes as many bytes as needed to hold a new number. The O.S. does not check if I'm entering the same numbers, it only requests new bytes to hold it. Each stack entry uses, in fact, 13.5 bytes, and the half byte called my attention to Saturn architecture and RPL organization. Why not thinking the HP42 is an RPL calculator with a stack limited to four register? It would be easy to implement some new short routines in the existing RPL O.S. to emulate the HP41.

You can go any further by testing what happens when complex numbers are generated and how much memory they request to exist. Also when variables are created. I had done that a long time ago, when I bought my first HP42S (R.I.P., poor girl...), but I cannot find my notes. It's revealing. I also had a

complete XROM map for it... all lost.

You see, the HP15C handles a small amount of memory and it does wonders with it. But it is somewhat easier when you know exactly where are all stack registers and their precise location, even if they grow in an exact number of five registers. Also, when computing SOLVE or INTEGRATE, I wonder if the 23 registers needed to accomplish the task are mapped in memory or not. I'm not sure if imaginary stack is created in the same portion of memory everytime it is activated, mostly because SOLVE and INTEGRATE do not use imaginary stack contents unless you explicitly use Re<>Im, and this technique is shown in the HP15C's Adv. Fcn. Hanbook. And one very convenient fact is that if complex mode is active, imaginary stack contents are cleared everytime a function is evaluated by SOLVE or INTEGRATE because both can only sample real data and the stack is filled with this sampled data before to evaluate function to be integrated or solved for. Well, in fact, if you set flag 8 or execute [f][I] or [f][Re<>Im] while computing SOLVE or INTEGRATE is different of activating complex mode before or after computing any of them. Where in memory will imaginary stack registers be created if they are created while running SOLVE or INTEGRATE procedure? In this case, is it possible to "pause" SOLVE or INTEGRATE computing and use complex numbers by direct keyboard operations? I'd like knowing it for sure and I did not test it.

There's more to talk about HP42S memory management and RPL memory "tricks", but I'll do that later, if needed.

Thanks, Valentin; I was not sure about when I'd post something about this.

Luiz C. Vieira - Brazil

Edited: 26 June 2003, 4:05 p.m.

Re: Tradeoffs: dynamic register/program allocation vs. Simply More Memory . . .

*Message #6 Posted by **Patrick** on 26 June 2003, 2:55 p.m.,*
in response to message #4 by Valentin Albillo

If memory serves(!), I seem to recall reading that some of the management information you refer to (LU stuff) is stored in the registers in non-standard format. The floating point format used in HP registers does not use all of the possible legal bit patterns. Some of these otherwise "illegal" bit patterns are used to denote special information, such as the fact that a register contains a matrix descriptor, or that a matrix is in LU form.

I believe I read this in the HP-15C Advanced Functions Handbook. I'll try to track it down...

Memory allocation: 34/41/15/42/32

*Message #7 Posted by **Karl Schneider** on 26 June 2003, 11:26 p.m.,*
in response to message #1 by Paul Brogger

Paul --

The amount of ROM code devoted to memory management in these units is a question to which I don't know the answer. It must have been a "necessary evil" for HP to address this issue in order to provide flexible products at reasonable manufacturing cost. There's certainly a lot of documentation about it in the old manuals! The RAM probably was comparitively more expensive back in the day. No pure-RPN HP calculator model through the '80s had more than 7 kB, for various reasons, I suppose.

Here are some comparisons between memory management between the RPN units 34C, 15C, 41CV, 42S, and 32Sii. I own and use all of them.

The 34C (and successor 11C) will automatically partition memory between numbered registers and programs. There are 70 bytes (instructions) of permament program memory, and 20 seven-byte convertible numbered storage registers labeled 0-9 and .0-.9. As more instructions are programmed, registers are converted without prompting, one at a time as necessary, to program storage. As programmed instructions are deleted, the memory is returned to data registers.

The 41C/CV/CX allows the user to manually select the number of numbered seven-byte data registers available for use (up to 63 for the C; 319 for the CV/CX) ; the remaining memory is available for programmed instructions. Any data storage (temporary or otherwise) needed by programs or functions will utilize the data registers. Micro-coded routines (e.g., Romberg integration on the Advantage module) will use the highest-numbered registers; RPN programs on the modules will tend to use specific lowest-numbered registers.

The 15C and 42S allow manual partitioning in a more sophisticated fashion. The user specifies how many numbered data registers to allocate (between 2 and 66 seven-byte registers on the 15C; many more eight-byte registers or 16-byte registers for complex values on the 42S); the remaining space is "pooled" and automatically allocated as necessary for all other purposes -- programs, matrices, imaginary/complex numbers, integration, root-finding, etc. This way, space for numbered registers is separate from the space needed for other applications.

The 32Sii has simple managment of its meager 390 bytes or so of memory: There is no "allocation command"; all memory is in a free pool for real-valued variables, programs, equations, and scratch registers for Romberg integration and root-finding (SOLVE). Eight bytes is allocated for a variable only if it contains a non-zero value(!)

The thing I've wondered about in the 34C is the memory requirements for SOLVE and INTEGRATE. For Romberg integration, the 15C requires 23 "free" registers; the 41C Advantage requires 32 available numbered data registers. The 34C, by contrast, requires no extra "scratch" space for these same advanced functions. I've also noticed that the 34C is slower to integrate than the 15C, although the 34C is a little faster interactively. Maybe this non-requirement causes the 34C to perform redundant calculations.

Re: Memory allocation: 34/41/15/42/32

*Message #8 Posted by **Valentin Albillo** on 27 June 2003, 4:39 a.m.,*
in response to message #7 by Karl Schneider

Karl posted:

"The thing I've wondered about in the 34C is the memory requirements for SOLVE and INTEGRATE. For Romberg integration, the 15C requires 23 "free" registers; the 41C Advantage requires 32 available numbered data registers. The 34C, by contrast, requires no extra "scratch" space for these same advanced functions. I've also noticed that the 34C is slower to integrate than the 15C, although the 34C is a little faster interactively. Maybe this non-requirement causes the 34C to perform redundant calculations"

Wrong. The HP-15C's SOLVE and INTEGRATE algorithms were lifted straight out of the 34C's existing code, and in their turn, were used for the 41C's Advantage ROM and later for the 71B's Math ROM (improved to account for recursivity, i.e: an INTEGRATEd function can call INTEGRATE itself, and so on up to 5 levels deep).

This means that the 34C's INTEGRATE code *does* use the scratch 'registers' you mention, but they are *permanently allocated exclusively* for SOLVE/INTEGRATE, and thus *never* available to the user for conversion to regular storage registers or program steps.

This is, of course, highly inefficient and was corrected immediately in the next model to implement that functionality, i.e.: the HP-15C, where said registers are only allocated for SOLVE/INTEGRATE if and when needed. Had the 34C memory management code implemented this feature, users would have enjoyed 23 additional registers (i.e: up to 20+1+23 = 44 storage registers) or 161 additional program steps (i.e.: up to 70+7*20+161 = 371 program steps). What a pity ! :-(

(BTW, you *can* actually 'navigate' and see the contents of the 34C's SOLVE/INTEGRATE registers by using some 'black-box' techniques, so to say ! A complete memory map for the 34C was published a looooong time ago in either PPC or Australian PPC Technical Notes (or both) but can't remember or check it now.

Best regards,

Re: Memory allocation: 34/41/15/42/32

*Message #9 Posted by **Karl Schneider** on 28 June 2003, 4:57 a.m.,*
in response to message #8 by Valentin Albillo

Valentin, Johnny --

Jeez, a fella just can't slip a sloppy statement or factual error past these keen-eyed and knowledgeable Forum participants! :-)

I stated imprecisely, "The 34C, by contrast, requires no extra "scratch" space for these same advanced functions."

What I really *meant* was, "The 34C does not require the user to reserve any 'scratch' space for these same advanced functions." That would have been a correct statement, but would have (ahem) required more thought and time to write.

I actually had assumed that the processor had some "scratch" registers available, but wasn't aware that 23 were used, as in the 15C. Valentin states that it was "a pity" that the 23 dedicated registers weren't made available to the user for data and program storage, but to do so without potentially (ahem) "consternating" the user would have required a method of explicit memory allocation -- something the 15/41/42 have, and the 34 doesn't.

This was the essence of Paul's query -- how much microcode is dedicated to memory management on the 34C, when it might have been simpler just to provide more user memory and fix the respective partition sizes?

As for the number of registers used by the Advantage module for INTEG (32) and SOLVE (13), I stand corrected -- by Johnny -- in that the highest available (unused) *program* registers, not numbered registers, are employed. This actually makes sense, if one thinks about it. (Serves me right for relying on my memory, and not consulting the Advantage manual.)

Re: Memory allocation: 34/41/15/42/32

*Message #10 Posted by **Johnny Billquist** on 27 June 2003, 6:05 a.m.,*
in response to message #7 by Karl Schneider

The HP-41 have more tricks in the memory allocation than mentioned so far. Apart from program/register allocation, it also allocates memory for the assigned keys for the user keyboard, memory for alarms (if you have the time module) and some advantage functions.

Anyhow, these things are allocated from the program part of memory, so user assigned keys will reduce the number of registers available for programs. The same for alarms.

If you call the SOLVE function in the Advantage pac, it will not use numbered registers, but unused program registers. 13 of them, to be exact.

The root finder uses 23 numbered registers. But some of thos registers must be set up before calling the function, it's somewhat understandable that they use normal registers.

INTEG also uses program registers. 32 of them.

SOLVE can call INTEG and vice versa, but they are not allowed to recurse.

DIFEQ uses eight numbered registers.

The complex number functions use five registers.

The vector calculator uses four registers.

Coordinate transformation uses 17 numbered registers.

Curve fitting uses 18 numbered registers.

TVM uses 10 numbered registers.

So, in short:

All programs that use numbered registers will use low numbered registers (starting from R00).

SOLVE and INTEG don't use any numbered registers at all, but program memory.

Thanks to ALL!

*Message #11 Posted by **Paul Brogger** on 27 June 2003, 1:40 p.m.,*
in response to message #10 by Johnny Billquist

Internal details like these are exactly what I was looking for.

Thanks to all for taking the time to share!

[Return to Index | Top of Index]

Go back to the main exhibit hall