Long Live the HP-16C !

Valentín Albillo (PPC #4747, HPCC #1075)

The HP-16C is a member of the infamous Voyager series released in July, 1982 (together with the HP-15C), thus the HP-16C's 23rd anniversary has just passed by, and it seems fitting to have a fond remembrance of this superb model, which HP, in their infinite wisdom, discontinued almost 17 years ago.

Dubbed "Computer Scientist" by HP (as opposed to "TI Programmer", the machine it was released to compete with), it offered tremendous capabilities for its time (and even today!) to anyone interested in the kind of logical/mathematical operations related to computer and microprocessor design and implementation. It works with four number bases (binary, octal, decimal and hexadecimal) in a variety of signed and unsigned modes, with *user-selectable word sizes* ranging from 1 to 64 bits, and boasting a full array of logical operators and all kinds of manipulations at the bit level, including all boolean operators, shifting and rotating, masking, setting/clearing/testing individual bits, the works !

Add to that extremely long battery life (many years!), full programmability similar to that of the HP-11C/15C (not the much simpler, less capable programming model of the HP-10C/12C) with insertion/deletion of program lines, subroutines, labels, loop control instructions, logical tests, flags, indirect addressing, and flexible memory partitionable between programs and data (up to a maximum of 203 steps of program memory), which would be retained after turning off the machine, and you'd certainly have an impressively capable machine, but that's not all !

As if to unabashedly crush any and all competition, this programmer's delight also features variable register size, allowing you to optimize memory usage by carefully selecting whatever register size is adequate for your application. Thus, for instance, specifying a register size of 4 bits would get you as many as 406 storage registers, (32 of them directly addressable) ! It also includes floating-point arithmetic with reciprocals and square root, exponential notation, and of course, classical 4-level RPN+LASTX, full stack manipulation commands, and to top it all, even "double" functions to compute exact products and divisions of *double word size*.

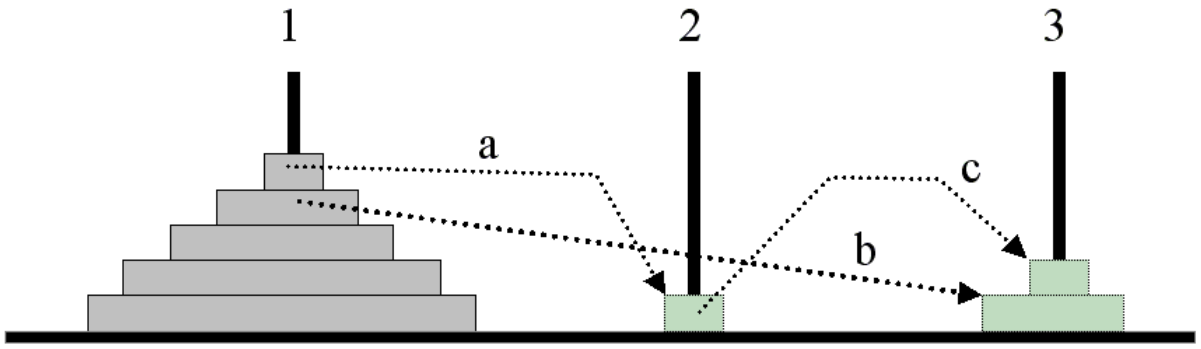
After this impressive but succint enumeration of its capabilities, it's no wonder that using and programming the HP-16C is extremely enjoyable and productive, and that's why most people who have one will pay high prices to try and get a second (or third!) backup unit, just in case, or to avoid having to carry their beloved machine to and fro, from home to work and back, everyday.

Now, let's show what the HP-16C can do, with this simple but revealing example.

A sample program: Towers of Hanoi

This is a small HP-16C program that I wrote specifically for this article, to show off its programming capabilities and versatility. Typical sample programs for the HP-16C (HP's or otherwise) always focus on simulating microprocessor instructions or converting floating point numbers from one machine format to another, so I thought it would be refreshing to include a program in quite another league (a puzzle no less) while still making use of the unique HP-16C instruction set and capabilities.

So here you are, a program to solve good old "Towers of Hanoi" puzzle. There's some legend or tale about tibetan monks having to move diamond rings from one spindle to another as fast and accurately as possible, with the world ending when they succeeded and such, but briefly the puzzle consists of an arbitrary number of discs (64 rings in the aforementioned legend), all of different sizes and placed ordered by size on a spindle, with the smallest disc on top, which all must be moved to a third spindle (using a second spindle as auxiliary) in a minimum number of moves, with the proviso that you must move one disc at a time, which must always be the top disc at some spindle, and you can't ever place a larger disc over a smaller one.



Thus, for example, in the figure above, the three spindles are labeled 1,2,3 and the five discs originally at spindle 1 must be moved to spindle 3 using 2 as auxiliary. The figure shows three moves being performed in order, (a),(b),(c), where (a) moves the smallest disc from 1 to 2, (b) moves the second smallest disc now on top from 1 to 3, then (c) moves the smallest disc back again on top of it, but this time both are in spindle 3 and spindle 2 is once again empty. The process ends, and the puzzle is solved, when all discs are in their initial order, but placed in spindle 3.

This particular puzzle has always been a favourite among programming textbooks, as it can be solved extremely easily using *recursion* and so serves a useful purpose to demonstrate what recursion is about, and how for the right problem it usually is the easiest and sometimes even the most efficient way.

For our present puzzle, using recursion would certainly be easiest, as this *2-line* program I wrote for the HP-71B using recursion to solve the N-disc puzzle convincingly shows:

```
1 DESTROY N @ @ STD @ INPUT "N=";N @ CALL M(1,3,2,N) @ DISP "OK"
2 SUB M(A,C,B,N) @ IF N THEN CALL M(A,B,C,N-1) @
  DISP "FROM";A;"TO";C @ CALL M(B,C,A,N-1)
```

Here's a sample run:

```
>RUN                FROM 3 TO 2          FROM 1 TO 3
 N=3                FROM 1 TO 3          OK
 FROM 1 TO 3        FROM 2 TO 1
 FROM 1 TO 2        FROM 2 TO 3
```

But the HP-16C, advanced as it is, doesn't have recursion available as part of its programming paradigm. What to do ? Well, for this particular puzzle there are very efficient, non-recursive methods to solve it, but for the didactic purposes of this article we'll do otherwise, namely *mimic recursion* on the HP-16C. How does one mimic recursion ? With arrays, that's how. This 7-liner would be the HP-71B version if recursion had to be mimicked:

```
1 ! *** TOWERS OF HANOI: NON-RECURSIVE VERSION - V. ALBILLO, 2005 ***
2 DESTROY ALL @ STD @ INPUT "N=";N @ INTEGER Z(100) @ A=1 @ B=3 @ C=5
3 IF N=1 THEN DISP "FROM";A;"TO";B @ ON Z(C-1) GOTO 1,2,3,4,5,6
4 Z(C)=A @ Z(C+1)=B @ Z(C+2)=5 @ B=6-A-B @ N=N-1 @ C=C+3 @ GOTO 3
5 A=Z(C-3) @B=Z(C-2) @DISP "FROM";A;"TO";B@ Z(C-1)=6 @A=6-A-B @ GOTO 3
6 C=C-3 @ N=N+1 @ IF C#5 THEN ON Z(C-1) GOTO 1,2,3,4,5,6
7 DISP "OK" @ END
```

(Incidentally, this program *can't be renumbered*, as it uses the **ON...GOTO** instruction as a poorman's "computed **GOTO**", which HP-71B's BASIC doesn't allow, so should you happen to renumber it, it wouldn't work !)

As stated, an array is used to hold the intermediate values and proper return addresses for each level of recursion. The size of this array, ultimately limited by available memory, will determine how deep the mimicked recursion can go and so the maximum number of discs for the puzzle. Our HP-16C program will be an optimized version of the non-recursive algorithm above, where the array is mimicked in its turn by using indirect addressing over a range of registers reserved for it. And this is where one of the most valuable HP-16C characteristics comes into play: *variable word size* !

In the program above, 3 array elements are used for each level of recursion, which would translate to 3 registers per level in the HP-16C, unless packing more than one element per register, which would make for a more complicated program. But using variable word size we can specify the smallest word size that still serves our purposes, and the HP-16C will make available as many registers of precisely that word size as memory allows. Here, specifying a word size of 8 provides us with *up to 111 registers*, allowing us to mimic recursion to much deeper levels than would be possible with the 56-bit registers available in most other classic HP calculators.

Program listing

```
01 43,22, A  g LBL A   31        1     1   61    42  A  f  SL
02        8     8      32       30     -   62    45  1     RCL 1
03    42 44  f WSIZE    33    44  3   STO 3  63    42 40  f  OR
04       24     DEC     34    22  3   GTO 3  64    44 31    STO(i)
05    42  3  f UNSGN    35 43,22, 0  g LBL 0  65        6     6
06        1     1      36    21  9   GSB 9  66    45  0     RCL 0
07    44  0     STO 0   37       33    Rv    67       30     -
08       30     -      38    44  0   STO 0  68    45  1     RCL 1
09    44  2     STO 2   39    43 34  g PSE    69       30     -
10    44  3     STO 3   40       33    Rv    70    43 21  g RTN
11        3     3      41    44  1   STO 1  71 43,22, 7  g LBL 7
12    44  1     STO 1   42       31    R/S   72    45  2     RCL 2
13 43,22, 3  g LBL 3    43        4     4    73    45  3     RCL 3
14    45  3     RCL 3   44    21  8   GSB 8  74       30     -
15    43 48     X#0?    45    44  0   STO 0  75        4     4
16    22  4     GTO 4   46    22  3   GTO 3  76       40     +
17    45  0     RCL 0   47 43,22, 1  g LBL 1  77    44 32     STO I
18    43 34  g PSE     48    45  3   RCL 3  78    43 21  g RTN
19    45  1     RCL 1   49        1     1    79 43,22, 9  g LBL 9
20       31     R/S     50       40     +    80    21  7     GSB 7
21 43,22, 1  g LBL 1    51    44  3   STO 3  81    43 23  g DSZ
22    21  9     GSB 9   52    45  2   RCL 2  82    45 31    RCL(i)
23    44 32     STO I   53    43  0   X#Y?   83    21  2     GSB 2
24    22 32     GTO I   54    22  1   GTO 1  84 43,22, 2  g LBL 2
25 43,22, 4  g LBL 4    55    42 10  f XOR    85       36     ENTER
26    21  7     GSB 7   56    43 21  g X<>(i) 86       36     ENTER
27        0     0       57 43,22, 8  g LBL 8  87        4     4
28    21  8     GSB 8   58    45  0   RCL 0  88    42  9  f RMD
29    44  1     STO 1   59    42 40  f  OR    89       34     X<>Y
30    45  3     RCL 3   60    42  A  f  SL    90    42  B  f  SR
                                             91    42  B  f  SR
```

Notes

- After entering the program, pressing **f MEM** should display: **P-0 r-014**
- **Rv** at steps 37 and 40 above are **Roll down** stack instructions.
- The following HP-16C-specific instructions are made good use of:

  WSIZE, DEC, UNSGN, XOR, OR, SL, SR

  most specially **WSIZE**, which allows us to have up to 111 registers at our disposal for the purpose of mimicking recursion to very deep levels.
- There's no need for a **RTN** instruction at step 92, as the end of program memory acts as an automatic **RTN** by default.
- This program could be significantly shorter if storage/recall arithmetic were available in the 16C which, alas, they aren't because of the 256-keycode limit.

Usage instructions

After keying in the program, let's try a sample run solving the 3-disc puzzle. Press:

```
   DEC                                    {sets decimal mode }
 3 GSB A   ->  (1 d)  ->  3 d             {move top disc from 1 to 3}
   R/S     ->  (1 d)  ->  2 d             {move top disc from 1 to 2}
   R/S     ->  (3 d)  ->  2 d             {move top disc from 3 to 2}
   R/S     ->  (1 d)  ->  3 d             {move top disc from 1 to 3}
   R/S     ->  (2 d)  ->  1 d             {move top disc from 2 to 1}
   R/S     ->  (2 d)  ->  3 d             {move top disc from 2 to 3}
   R/S     ->  (1 d)  ->  3 d             {move top disc from 1 to 3}
   R/S     ->   0 d                       {puzzle solved}
```

As you may see, solving the 3-disc puzzle required 7 moves. In general, solving the N-disc puzzle will require 2^N-1 moves, which obviously grows exponentially with N. For N=64, as in the old legend, we would need

$$2^{64} - 1 = \underline{18,446,744,073,709,551,615 \text{ moves}}$$

I'm afraid that even for the HP-16C, the batteries wouldn't last that long.

Final remarks

In the HP-16C you can actually appreciate the enormous amounts of talent and dedication invested in bringing it to life, every functionality carefully crafted and optimized, every feature synergically integrating with the rest. That kind of utmost quality is what I think the HP-16C represents best, no wonder it's still so intensely sought for. I own one, and frankly, wouldn't mind getting yet another. Just in case.