

What I would change in the HP 35s

Martin Cohen
mjcohen@acm.org

1. Introduction

I had previously written some routines for the HP 33s and wanted to do the same for the HP 35s. However, some of the “features” in the 35s (combined with the end of my involuntary retirement) led me to not do that. Instead, I will state the changes I would like to make to the 35s.

The changes are in two groups:

- Access to the parts of complex numbers and vectors
- Indirect addressing.

2. Access to parts

Although the 35s handles complex numbers better than the 33s, it is (I would say) notorious for its inability to accurately get the real part of a complex number. I would replace the “ARG” key with a “CPARTS” (for “complex parts”) key. This would display a menu with the following entries (all acting on the X register and replacing it):

- REAL – return the real part
- IMAG – return the imaginary part
- ABS – return the absolute value
- ARG – return the argument
- CONJ – return the conjugate

The new vectors are a useful addition to the 35s, although they seem a little odd to me. Nonetheless, I would replace the “ABS” key (not needed because of the “CPARTS” menu) with a “VPARTS” key with the following entries (the first three acting on the X register and replacing it):

V1-> - return the first component of the vector
V2-> - return the second component of the vector
V3-> - return the third component of the vector
->VX – convert X to [X, 0, 0]
->VXY – convert X and Y to [X, Y, 0]
->VXYZ – convert X, Y, and Z to [X, Y, Z]

If X and Y both contain vectors, I would also have “/” (divide) return their cross product, replacing them.

3. Indirect addressing

The 33s did indirect addressing with the special “I” register distinct from the regular A-Z variables. When a reference was made to “(I)”, the contents of this register was used as an index into memory.

The 35s changed this in two ways:

1. The regular variables “I” and “J” contained the indirect locations.
2. Addresses were accessed by the special keys “(I)” and “(J)”.

While these changes made some things easier (such as a doubly-nested loop), they also made some things much harder. For example, it is now extremely awkward to write a loop that will process sequentially the regular variables A through Z (or through any variable beyond I). This is because the loop variables (I and J) are both loop controllers and regular variables.

I would change this in two ways:

First, allow ANY variable (in A-Z) to be used for indirect addressing via a “IND” key (replacing “(J)”). “IND” would act as a modifier to “STO” and “RCL” (AS “+”, “-“, “*”, and “/” are now). For example, “STO IND C” would use the contents of C to specify the actual memory location accessed.

Although this would eliminate such compounds as “RCL* (I)”, the additional power would, in my opinion, be well worth it. For example, loops nested three or more deep could be easily done, and a loop referencing variables A-Y could be done using Z as the loop variable.

Another use for “IND” that would provide a capability that is not in the 35s is in algebraic expressions. Currently, there is no way in the 35s to refer to any variables other than A-Z in algebraic expressions. If “IND” were available, the keys “RCL IND A” could be converted to “IND A” in an expressions – this would cause the contents of A to be interpreted as a memory address, and the contents of this location used in the computation. To make expressions more powerful, the keys “STO variable”, instead of acting the same as “RCL variable”, could take the most recent expression and store that value in the variable, and then continue with the expression.

An even more powerful capability would be that of using an arbitrary expression for the indirect address (e.g. “RCL IND (A+B/2)”), but this would require careful handling to be done right – I leave it as an exercise for the reader.

As the saying goes, “Free advice is sometimes worth it.” This is mine.