User-Oriented Suggestions for Floating-Point and Complex-Arithmetic Forth Standard Extensions

Ferren MacIntyre

Center for Atmospheric-Chemistry Studies
Graduate School of Oceanography
University of Rhode Island
Narragansett, Rhode Island 02882-1197

Thomas Dowling

Miller Microcomputers
61 Lake Shore Rd.
Natick, Massachusetts 01760

Abstract

Governmental, industrial, and editorial acceptance of Forth requires that it conform to some recognized, formal standard, (preferably sanctioned by ANSI). Proposed here are standard extension wordsets for floating-point (FP) and complex (CP) arithmetic, based upon three years of extensive use – and rewriting – of the MMSFORTH words with the 8087 numerical coprocessor. The models for the proposed extensions are IEEE 754 for constraints and the HP-15C pocket calculator for scope.

Familiar words are merely preceded by `F` or `C`. Many words such as `-F-` and `F\` (for `FSWAP F-` and `FSWAP F/`) name single 8087 instructions. Short words are mandatory because of the natural length of FP equations. Less familiar words include `-F` for negation and `FRT2` for `FROT FROT`, symmetrical with `FROT` and as often needed.

Left open for community discussion is the choice of FP number indication on input. Large numbers are well represented by an embedded `E`, as `12.34E56`, but `3.E0` is prolix. `3E` is confusible with hex, `3.` with double-length integer. The European decimal `3,` is unique, but some do not like the comma.

Adequate output formatting capability is essential and is provided. Over 100 data types are subsumed under a single "grandfather" defining word, all stored with `IS`.

Not discussed are possible differences resulting from different IEEE 754 hardware implementations (e.g.: Intel's 8087 and 80287, etc.).

Why Floating Point?

The first question asked by nearly all Forth programers will be why Forth needs a floating-point (FP) standard. If one works with digitized, bounded data, scaled-integer arithmetic is natural and adequate. For predictable functions like sines and logarithms, a look-up table and integer interpolation is fast and accurate. However, proponents of scaling have not persuaded scientific users to abandon the perceived advantages of letting the machine do the scaling.

An unrelated, but indicative, reason for FP Forth is that 3 out of 4 FORTRAN compilers at the URI Academic Computing Center give erroneous results for complex trigonometric functions of small angles, forcing one to dismantle complex calculations and write them as real halves. The IBM PC, with 8087, MMSFORTH and the CP wordset in the appendix, gives accurate answers.

An ultimately more compelling reason for an FP standard is the policy of many institutions of avoiding non-standardized products. The absence of a formal standard prevents the US Government

from recognizing, American industry from using, and professional journals from publishing, Forth in FP applications. Thus, whether or not an FP standard seems desirable to us individually, it is essential to the acceptability of the language in the real world.

The second question which will be asked is how our proposal differs from FVG, the Forth Vendors' Group FP Standard (Tracy and Duncan, 1984). The easiest way to explain our position is to call ours a Forth Users' Group Standard. The difference is fundamental: Vendors have a vested interest in minimizing the amount of work they must do to produce and maintain a product. Professional scientific users, on the other hand, want a product which maximizes their productivity. This is an important distinction which is apparently not widely appreciated among the Forth community, many of whom use Forth to explore computer languages rather than to crunch numbers with serious intent. We suggest strongly that it is time for Forth to grow up in this one respect. The fun and games – the exciting search for better approaches – may continue in other directions, but the language of mathematics is a nearly closed field whose needs are well known. It presents a standing target which we claim to have hit squarely with this proposal.

Forth and the IEEE 754 Standard

One of the avowed purposes of the ANSI/IEEE 754 Standard (IEEE 1985) is to allow programs to run on "any computer that conforms to this standard," after "minor editing and recompilation." Nearly all Forth experience to date with IEEE 754 hardware has been with Intel's realization, the 8087 numerical coprocessor. In order to ensure that editing is in fact minor, it might be well not to adopt a software Standard until the Forth community has accumulated some experience on Intel's 80287 and the forthcoming Motorola and National Semiconductor implementations of IEEE 754.

Many of the choices we once agonized about on our own – precision, rounding, error-handling, etc. – (Helmers 1981; Sand and Bumgarner 1983) have now been pre-empted. An aspect to which IEEE 754 pays much attention is exception handling, saying (Sec. 7):

> "There are five types of exceptions that shall be signaled when detected. The signal entails setting a status flag [or] taking a trap... With each exception should be associated a trap under user control... The default response to an exception shall be to proceed without a trap...
>
> "For each type of exception the implementation shall provide a status flag that shall be set on any occurrence of the corresponding exception when no corresponding trap occurs. It shall be reset only at the user's request. The user shall be able to test and to alter the status flags individually, and should further be able to save and restore all five at one time."

The five IEEE exceptions are invalid operations (e.g., square root of a negative number), division by zero, overflow, underflow, and inexact rounding.

Having operated in the default mode for three years without feeling a need for anything more, we are impressed with the careful planning behind the default options of the 8087. For all practical problems yet encountered, the exponent range (± 4932) is indistinguishable from zero at the low end and infinity at the high, so that the "overflows" and "underflows" that so plague operations on 32-bit mainframes simply do not occur.

Accordingly, we provide a minimal set of Processor Control reference words which allow the user to put the status word on the stack `@SW`, set exception flags and clear them `SET-EXCEPTIONS` and `CLEAR-EXCEPTIONS`, and put the "environment" on the stack `@ENV`, where it may be examined by an exception-handling routine. We also specify `TOFS`, an extension of the 8087 instruction `FXAM`, which reports the type of object (14 choices!) at the top of the F-stack in the form, for instance, `"TOFS is -Denormal"`. We designate these as reference words for the simple reason that we have never had occasion to use them.

Similarly, we provide reference words which set the processor's rounding mode. `FROUND+` and `FROUND-` round toward + and - infinity, respectively. We also include infinity controls `AFFINE`

($+\infty \neq -\infty$) and **PROJECTIVE** ($+\infty = -\infty$), and words **@CW** and **!CW** to fetch the processor control word to the stack and return it to the 8087.

More useful, and therefore not reference words, are the rounding controls **FROUND**, which rounds toward nearest or even, and **FTRUNC**, which rounds toward zero. Essential is **FINIT** to initialize the processor, i.e., set the defaults and clear the F-stack.

It is clear that no Forth programmers worked on the IEEE 754 specifications or upon Intel's 8087, since the single most frequent exception is overflow of the 8-deep F-stack, and this fatal exception is not signalled in any useful manner. It naturally results in the loss of the datum forced "off the bottom". However, since the F-stack is really a ring-and-pointer, the effect is to attempt to overwrite an occupied register. The attempt is unsuccessful but is not aborted and leaves garbage in the register, but the only "flag" which is set is a 10 pattern in one of 8 locations along the processor's tag word, which itself is relatively inaccessible and incapable of producing an interrupt.

We call attention to the overriding need for an interrupting hardware flag for this exception, and discuss a possible software fix below, but do not suggest this as part of the Standard, preferring to leave it to implementors in the same spirit with which Forth abjures bounds checking of arrays. We also leave the "trapping" proposed by IEEE 754 to implementors. Those who need traps may install them as they see fit.

Limitations of This Paper

Some reviewers felt that we should address wider questions such as "hardware versus software implementations, the merits or demerits of a separate FP stack, the AM9511, the Macintosh SANE interface, and other FP packages. Benchmarks ... would be useful." We comment on these points in turn.

Hardware vs. software. There is simply no contest here, but this fact may not be apparent to casual users. In a loop which adds two numbers 10,000 times, the 8087 may be only twice as fast as single-precision (32-bit) software FP. But in heavily numerical calculations, the 8087 may be 115 times faster.

Similarly, the 80-bit precision of the 8087 may not seem useful, since one seldom needs 19 decimal digits in a practical calculation. But this misses the point: The function of the additional precision is to reduce roundoff error in recursive calculations. Ricatti-Bessel functions, calculated by about 100 recursions, are accurate to only 1 significant figure with 32-bit software, but to 6 (or more) with the 8087 (MacIntyre 1984). We have not tried 80-bit FP software, which might solve the roundoff problem – but only at the cost of taking 300 times as long as the hardware.

For these reasons we believe that software FP has no place in serious calculations – a point which Moore has often made. This leads us to take the position that software FP should mimic the IEEE 754 hardware (necessarily with a separate FP stack), on the grounds that the principal reason to write such programs is in preparation for moving them to hardware.

Other approaches. Looking at the differences between our recommendations and those of the FVG group, we feel that we are breaking enough new ground, so that we limit our comments to those things that we have used professionally, where we compete successfully with FP operations on mainframes and supercomputers. The proposed wordset, although based on MMSFORTH, differs considerably from the version currently available, and this paper is not to be construed as recommending any particular implementation.

Having no experience with the AM9511, SANE, etc, we are not competent to discuss them. If others who know these approaches feel that they have something to contribute to the Forth Standard, we encourage them to write partisan papers presenting their merits in the best possible light, so that the Standards Team has some ideas to select from in making its decisions.

Benchmarks. We have done no benchmarking other than that incidental to getting our own numbers out, mainly because our central interest lies in the numbers rather than in the way the numbers are

obtained. Benchmarks are sensitive to hardware speed, to algorithmic efficiency, and to cleverness of implementation, but it is not clear how they affect considerations about the Standard unless they address very specific questions. A carefully constructed benchmark might aid the Standards Team in deciding between conflicting suggestions; in this case we welcome the opportunity to run some tests.

Desiderata

Although many of the words in the Appendices have been around for years (Peterson and Lennon 1981), the rationale for some of the less obvious choices is given below.

Short Words

Forth programmers are accustomed to "self-documenting" long names. But FP operations have different requirements, and experience shows that approaching Moore's goal of one-line definitions (Brodie 1984, p. 180) is difficult for the usual FP equation. Despite concurring with the one-line optimum, the first pass at converting FORTRAN programs to Forth (of which more below), invariably has words which cross block boundaries. Much of the problem is solved by factoring, but long words simply take up too much space to be useful. Dowling's FVG-style `FNEGATE` was the first word MacIntyre renamed, independently picking Harwood's (1981) obvious replacement, `-F`. As an example (from an almanac program), the cosine of the angle between the meridians of the sun at declination δ and depression R, and the observer at latitude La is given by:

$$\cos\theta = -(\sin\delta \sin La + \sin R)/\cos\delta \cos La$$

This can be written as one 64-character line of compressed code:

```
: cos-th  δ SIN La SIN F* R SIN F+ δ COS La COS F* F/ FNEGATE ;
```

but we prefer the more legible version with phrasing gaps:

```
: cos-th  δ SIN La SIN F*  R SIN F+  δ COS La COS F*  F/  -F ;
```

(Notice that the only way we got this equation on a single line at all was by exercising the option of dropping the `F` from FP function names, in favor of `I` on the smaller number of integer versions which might be simultaneously active. And yes, we realize that Moore's almanac probably took 6 blocks in scaled integer, but we suspect that it didn't deal with Milankovich cycles over geological time spans.)

There are two reasons for long FP definitions. The first is that some equations are best written as 8-line definitions simply to keep associated calculations together. More important is that 8087 arithmetic is fast, but 8-byte stores and fetches are slow. One never wants to store intermediate results if they can be left on the F-stack till used, but leaving a number on the F-stack while executing intermediate words makes for illegible code, and good practice suggests consuming such intermediates inside the definition that created them.

"Logical" Choices

The most controversial names proposed below are probably those which move data between the normal stack and the F-stack. These all have the form `FxY`, where `x` is an arrow pointing "from" `F<Y` or "onto" `F>Y` the normal stack, and `Y` indicates the type of object being moved. Other conventions may be equally logical, but the overriding objectives are two: To keep these words together in the glossary so the user can find them (hence they all start with `F`), and to provide a structure in which a given arrow keeps its name throughout (experiment with pronouncing the original MMSFORTH names `FS>`, `FS<`,and `FS>D` if you don't see the problem). We reiterate that for auditory clarity, the `>` arrow should be pronounced "onto" rather than "two", "too", or "to".

It goes without saying that any word which pushes an object onto the F-stack that does not automatically occupy all 80 bits, should set the unoccupied bits to zero (this has not always happened in the past).

Indices

The *sine qua non* of scientific programming is manipulation of indexed arrays. Factoring out short segments of such manipulations to keep definition length within bounds requires that the index be available for words which are many jumps removed from the loop which established the value of the index. Forth is clumsy at this. If the index is on the return stack, every jump to a new word means that attention must be paid to the index. After the first jump, `I'` can be used (at least in MMSFORTH, although Forth-83 seems to lack even this); after the second, things get complicated. As a result, words which have been factored out of a long definition tend to be called with `I<name>`; and to start with `>R`, and have `R> DROP` either as the last component or `R>` preceding the last indexed word (replacing `DROP`). A decision must be made each time, and the variety of approaches means that different symbols and different structures produce the same result, guaranteeing illegible code. If the arrays are multiply indexed, the problem is worse, and one resorts to the slower option of named index variables.

Although the 16-bit Novix NC4000 was not designed with indexed operations in mind, when it is used with a coprocessor its designers suggest that the first index be put in the "square root" register, and a second in the "multiply" register (but a third index must be put in 2-cycle-access memory). This suggestion also eases implementation of words such as `I+(->I+1)` and `I-( ->I-1)`, useful for the common situation in which adjacent members of an array are wanted. Nonetheless, if there be a 32-bit Novix in the offing, a handful of true index registers would be highly desirable!

A final need is for reference words like `FI = I F<S` which put indices directly on the F-stack, for times when terms in an infinite series are multiplied or exponentiated by the index, but note that if index-dependent terms are more complex (e.g.: n/n+1) it often pays to precompute them and store them.

–F–, F\, FRT2, and FATN2

`-F-` is a direct translation of the single 8087 instruction `FSUB`, and `F\` similarly translates `FDIV`. (`F-` and `F/` translate `FSUBR` and `FDIVR` respectively.) Implementing them in Forth saves an `FSWAP`, and the whole thrust of FP words should be to minimize the number of instructions written and executed.

`FRT2`, which does the "complicated" `ROT ROT` to the F-stack, is no harder to implement, and takes no longer, than `FROT`. Brodie calls `ROT ROT` a sign that the stack is too crowded (Brodie 1984, p. 203), but it is symmetrical with `ROT` and needed about as often. We have not implemented `FPICK` or `FROLL`, but include them as reference words.

Note that we suggest both `FATAN`, which provides the principal value of a single argument, and `FATN2`, which is the two-argument, four-quadrant arctangent. Since `FATAN` uses `FATN2` internally, writing both into the Standard costs only about 10 cells of memory.

Comparison operators

Although the 8087 instruction `FCOMP` suffices for all FP comparisons, one often writes binary-choice versions, because

```
F<= IF XXX THEN
```

is clearer than

```
FCOMP 0 <= IF XXX THEN.
```

However, because these require only a half line of trivial code, they are included as reference words.

IEEE 754 calls for comparisons to signal an "unordered" result when one of the comparands is not a properly formatted number (as might occur after division by zero, for instance).

Representation of FP Numbers

When the exponent is explicit, an embedded `E` suffices to indicate an FP number, as `1.23E456`. Often, however, one wants to enter non-exponented FP numbers from the keyboard, and `1.E0` is prolix and inelegant. `1E0` can be interpreted as a hex number; `1.0` looks like a double-length integer, and often both FP and double-length words must be active simultaneously. One possibility is to use an embedded decimal to signify an FP number, and a terminal decimal point for a double-length integer, but this could cause difficulties with financial packages.

Another alternative is the "European decimal point" or comma, as `12,34`. The trade-off here is possible confusion with Forth's traditional use of the comma as a compilation operator.

We have experimented with these conventions, and now use the comma, but suggest that our European colleagues be invited to discuss this particular point.

Printing and Formatting FP Numbers

Since output must be understood by a human before it becomes significant, formatting words are essential to a floating-point package. The number-entry and -display routines on a hand calculator use a large fraction of its total memory, devoting as much code to convenient formatting as to "useful" mathematics. Even with the printing words described in the appendices, 70% of a recent 10-block program (to belabor the acoustics of small bubbles) was devoted to I/O: 3 blocks of parameter definitions and descriptions, 2 blocks of words to print out and label intermediates and results, a block of input-acceptors, and a block for video graphs, all to support, debug, and tame 3 blocks of complex arithmetic, and every one of these supporting words written because it was needed.

The I/O format which we have settled on after many permutations has two basic printing words, `E.R` for right-justified exponential, and `F.R` for floating point. At a cost of two more lines of code, one can add the FORTRAN-like `G.R`, in which the machine makes a choice between `E.R` and `F.R`. For two more lines of code, one can switch the exponential `E.R` between `SCI.` (with an *ad lib* exponent) and `ENG.` (whose exponent is a multiple of three, as in a Hewlett-Packard calculator). Another pair of lines get a choice for `F.R` between `FLD.` (with a fixed number of decimal places) and `FLW.` (with a fixed number width). IEEE 754 Sec. 5.3 specifically requires rounding upon conversion to narrower formats.

The three printing words also come with trivial non-justified variants `E.`, `F.`, and `G.`, while `FE.` and `FF.` print all 16 digits.

We omit "`E`" between mantissa and characteristic, leaving only the sign of the characteristic as separator, just as calculators do. ("Feynman notation" avoids the "`E`" by writing the signed characteristic as a subscript, an option which is increasingly available on small printers, and may yet come to video displays.)

For complex numbers we normally use a single `C.` command which prints two numbers in `G.` format. A useful touch for an implementation, if not the Standard, is to have the second number followed by an indication of the modes in use: `i` for rectangular, `r` for polar radians, and ° for polar degrees. This removes a source of confusion, since

3.000	4.000i
5.000	0.927r
5.000	53.13°

are all the same number.

It goes without saying that to qualify as complying with the Standard, formatting words should handle the many types of F-stack objects appropriately--that is, presenting whatever information is contained in the F-stack location. MMSFORTH does not, for instance, cope optimally with denormals, but the fact that no one seems to have noticed this in three years suggests a need for a simple means of testing for compliance. A Standard is meaningless without a verification suite.

Alas, Babylon

Babylonian numbers (hour:minutes:seconds and degrees:minutes:seconds), which inevitably crop up in things like almanacs, solar design, and navigational programs, present a format problem. The easy Forth convention is to enter these as three integer stack items, but this differs from the wide-spread HP-15C convention in which the entry format is dd.mmss. Furthermore, we note that while sextants normally read in decimal minutes dd:mm.mm, theodolites read in seconds dd:mm:ss, offering three possible conventions--each of which will be seen as optimal by some users.

Our response is to suggest two reference words `F<60` and `F>60` to move FP Babylonian data from and onto the normal stack. Since I/O words like this do not need speed, and since these are clumsy to implement in assembler, we see these as existing only in high-level Forth. We suggest that these words might come in three format-variants, with the user selecting the variant he desires. (Thus in one program `F<60` might move hh mm ss, but in another, hh.mmss. In the extreme case of needing more than one format, a variant might be renamed.

Although this flexibility may sound confusing, it boils down to making a choice, at load time, among the following possibilities:

Format		Input		F-stack		Stack
dd mm ss	3#IN	12 34 56	F<60	12.582222	F>60	56 34 12
dd.mmss	D#IN	12.3456	F<60	12.582222	F>60	3456 12
dd.mmmm	D#IN	12.3456	F<60	12.576000	F>60	3456 12

where `3#IN`, not included in the Standard, would be something like

```
: 3#IN (-> hh mm ss) ." ? " PAD 15 EXPECT PAD 1-
    NUMBER NUMBER
NUMBER DROP ;
```

so that one could enter the three numbers as a single unit with a single carriage return. (`D#IN` is MMSFORTH's double-length-number read-in word.)

As we implemented the Babylonian words, they are symmetrical in function but not in detail. That is, if `F<60` picks up an I/O format and puts a calculating number on the F-stack, then `F>60` returns an I/O number--but one whose stack order is most suited for printing, possibly with punctuation marks. `F>60` does not, that is, return a number in the same double-length format which `F<60` found it, since we didn't know what to do with it after we got it back in this format. This asymmetry of form may be philosophically worrisome because while `F<60 F>60` is meaningful, `F>60 F<60` is not. Nevertheless, we find this eminently practical.

Similar asymmetric choices exist for other words, such as the `F<D F>D` pair. Should the number returned be the number sent, possibly with many decimal figures? Or will one more often plan to continue with integer calculations, and therefore desire truncation upon return? Because these are points upon which rational people can reasonably disagree, the Standards Team must specifically resolve them. We enter a strong plea for consideration of the most useful forms of asymmetry, with documentation and a published explanation for the rationale behind the final choice.

QUANs

`CONSTANT` and `VARIABLE` have seldom been used as intended. If one is not concerned with the possibility of putting code into ROM, the convention that `CONSTANT`s should not be changed by the program loses its significance, and one promptly adopts whichever data structure results in a minimum littering of `'` and `@` about the program.

The `QUAN` (Rosen 1982, Dowling 1983), for "quantity", is a data structure with three CFAs, one pointing to `@`, one to `!`, and one to `AT`, which returns the `QUAN`'s address. Complicated as this

seems, it does away with the problem of ' and @, and saves space if there are at least three uses of the name. More to the point, it relieves the programmer of remembering how he has defined data. Mentioning the QUAN by name puts its value on the stack. IS stores data into it, as in 54 IS <name>.

The most imporant feature of the QUAN is that it is intelligent to the extent that the programmer need not worry about what kind of QUAN is being stored. Since over 100 kinds of QUANs are proposed below, this feature eliminates scores of special words and the sort of unpleasant surprises that result from ! when you meant C!.

No one in our experience who has used FQUANs has evinced any desire for other FP data types. Unless there are applications which have not become evident to the people we know, we see no reason to propose FCONSTANT or FVARIABLE, although we include them as reference words. F@ and F! are also included.

A Grandparent Defining Word for Data types

In any serious application, one is likely to want a dozen or more data types, and experience shows that very similar definitions proliferate with time. As an alternative, 144 distinct data types can be built from 18 building blocks, subsumed under one patriarchal word which itself defines data-type-defining words. This single word is in the standard and is named DATA-TYPE. It is used in the form:

```
d w t a DATA-TYPE <quan-type>
```

where:	d = dimension	0 1 2 3;
	w = width	0 1 2 = 8-, 16-, and 32-bit integers;,
		or 32- and 64-bit FP numbers
		and 80-bit temporary real or BCD numbers;
	t = type	0=integer, 1=floating-point, 2=complex, 3=BCD;
	a = address	0=in-line, 1=indirect, 2=outside 64kb.

Thus 0 1 0 0 DATA-TYPE QUAN and QUAN SIFAKA makes SIFAKA a single 16-bit integer quan, while 3 2 2 2 DATA-TYPE 3TCLARRAY and 5 10 15 3TCLARRAY INDRI make INDRI a three-dimensional, 80-bit-wide, complex array 15,000 bytes long, stored in high memory outside of Forth's 64-kb sector. A number of self-consistent <quan-type> names are included in the reference set.

On the address-segmented 8088, moving outside a 64-kb sector (with a=2) imposes a 23% time penalty on memory references, a not inconsiderable trade-off swapping time for space. For arrays it is the saving of space that is important, while single FLQUANs make it convenient to store associated single parameters beside the large arrays in high memory, so that by judicious overlaying of data space and a virtual disk, one can transfer all varieties of data to floppy disk with a single block move (MacIntyre 1984b).

Both data stacks are used to store to an FARRAY, the datum on the F-stack, and the indices on the parameter stack, leading to structures like

```
0, I J IS DATA
```

to make DATA(I,J)=0.

Moore questions the utility of ARRAY (Brodie 1984, p.196), preferring to define his own variations at will, and indeed, this is an option which advanced users take advantage of. But array definition seems unfeasible in FP because the defining words are necessarily in machine language in any serious implementation, and so complex that they need the attentions of a professional.

Matrices?

Having arrays, the 8087, and cookbook matrix routines almost available (Startz 1985), one contemplates adding matrix operators to the Standard. We do not do this, preferring a separate

Matrix-Manipulation extension, but note that a little forethought now will greatly simplify life later. Taking the Hewlett-Packard 15C calculator as the conceptual model, we find that two-dimensional arrays and matrices differ in only two important ways:

Arrays need remember only one dimension, while matrices need both;
Matrix calculations need dynamic memory management to provide workspace.

Any object defined by **MATRIX** should be usable as an **ARRAY** (but the reverse is not necessarily true). One recommended option, which prepares for matrix operations, costs little, and eliminates the defining word **MATRIX**, is to store all dimensions with arrays.

Memory management may in actuality not be a problem, as memory become cheaper. The program can report an incipient error if it is about to run out of room for intermediate results, and for large data sets the programmer will know in advance that he must overwrite his own arrays.

Coding Conventions for FP

In writing long equations, clarity is aided by:

short words,
vertical alignment of operations,
judicious use of white space.

The goal of "one line per definition" is reasonably replaced by "one line per storage operation", analogous to Harris's (1984) ending a phrase when the stack depth is reduced.

The structural differences between algebra and RPN are such that good FORTRAN can often be translated into reasonable Forth by mindlessly working from right to left, making full use of **-F-** and **F**. Maximum stack occupancy is frequently only half as great as when translating from left to right. A useful strategy is to do this consistently on the first pass, even when the equation is simple. When the bugs are out is soon enough to think about and neaten up the code.

These suggestions are illustrated by the coding in Figs. 1 and 2, which also shows the imaginative naming achieved by translating a FORTRAN program into Forth. (As an adjunct of the "short names" philosophy, it would be a great boon if the ASCII standard had included the Greek alphabet!)

It is helpful if vectored names are distinguishable from words that one can depend upon, and we suggest the naming convention

|NAME "disjunctive-name" (*not* "or-name"!) Vectored word

for forward references which will later be filled by a choice of executable words. If words are created for the sole purpose of filling the vectored names, we suggest indicating the fact with compound names such as:

aaa|NAME bbb|NAME Executable words to be vectored.

As an example, in MMSFORTH one might write

```
VECT |EXP                          to define the vector,
: ENG|EXP ... ;                    and
: SCI|EXP ... ;                    to define the vectees, and then either
: SCI. [FIND] SCI|EXP IS |EXP ;    or
: ENG. [FIND] ENG|EXP IS |EXP ;    to fill the vector.
```

These provide the words by which one selects scientific or engineering notation, while

```
: $F.EXP    ... |EXP ... ;
```

sets up the format for **E.** to print in either exponential format. (Something similar could doubtless be done with Brodie's (1984) **MAKE** and **DOER**.)

Fig. 1. Segment of a typical scientific FORTRAN program translated into Forth in Fig. 2. Name changes made for the sake of brevity and logic include:

```
COEF(2,L) -> L 1 CF    ALPHA  -> α    ONEMZT -> 1-Z    DEL -> δ
DIVDIF    -> 1D        S(I,1) -> I dt      S(I,4)    -> I R .
```

```
   62 COEF(2,L) = DIVDIF - S(I,1)*(2.*S(I,4) + S(I+1,4))/6.
      COEF(4,L) = (S(I+1,4)-S(I,4))/S(I,1)
      GO TO 68
   63 ONEMZT = GAM*(1. - Z)
      IF (ONEMZT .EQ. 0.)
      ZETA = 1. - ONEMZT
      ALPHA = ALPH(ZETA)
      C = S(I+1,4)*S(I,6)
      D = S(I,4)/6.
      DEL = ZETA*S(I,1)
      BREAK(L+1) = TAU(I) + DEL
      COEF(2,L) = DIVDIF - S(I,1)*(2.*D + C)
      COEF(4,L) = 6.*(C*ALPHA - D)/S(I,1)
      L = L + 1
      COEF(4,L) = COEF(4,L-1)+6.*(1.-ALPHA)*C/S(I,1)*ONEMZT**3
      COEF(3,L) = COEF(3,L-1)+DEL*COEF(4,L-1)
      COEF(2,L) = COEF(2,L-1)+DEL*(COEF(3,L-1)+(DEL/2.)*COEF(4,L-1))
      COEF(1,L) = COEF(1,L-1)+DEL*(COEF(2,L-1)+(DEL/2.)*(COEF(3,L-1)
     *                    +(DEL/3.)*COEF(4,L-1)
```

Fig. 2. The FORTRAN segment of Fig. 1 rewritten in Forth, with some operations done elsewhere in the Forth words 63! and L63. The words 2I, 3I, 4I, and 3F, are repetitive DUPs or FDUPs, to put multiple copies on the stack. *2, /6, etc., are floating-point operations. +L increments L. Note: The imaginative naming; the surviving tendency for the Forth to represent a right-to-left reading of the FORTRAN; the storage protocol for FQUANS; the "one line per storage operation" convention; the vertical alignment; dragging the index along with the clumsy >R and R>; the use of -F- and F\; the utility of short names; and the logic of long definitions.

```
\ 850219 FM Taut spline 22/28 Piecewise polynomial coefficients
\ Normal case.
: L62 >R 3I 1+ R R FOVER FOVER *2 F+ /6 dt F*
                1D -F-                              L 1 IS CF
  ( R+ R) F- I dt F/                                L 3 IS CF R> ;
\ New knot inserted; Z > .5.
: A63    >R  63!
  C D *2 F+  I dt   F* 1D -F-                       L 1 IS CF
  I dt D α C F* -F- *6 F\                           L 3 IS CF +L
  L 1-    3 CF 1, α F- C F* *6
    1-Z   3F F*  F* I dt F* F/              F+      L 3 IS CF
  L 1- 2I 3 CF δ F*    2 CF F+                      L 2 IS CF
  L 1- 3I 3 CF δ F* /2 2 CF F+ δ F*    1 CF F+      L 1 IS CF
  L 1- 4I 3 CF δ F* /3 2 CF F+ δ F* /2 1 CF F+
                δ F*    0 CF F+                     L 0 IS CF R> ;
```

How Many Words in a Standard?

Many people at the '85 Rochester Conference were uncomfortable with the number of words proposed here, preferring to see maybe 15 in an FP Standard. No doubt there are reasons for this attitude, but we remind the minimalists that in the 8087 we have a device which is in many ways more powerful, exact, and versatile than a mainframe, and it requires an extensive vocabulary to take advantage of this machine. It has, for instance, 15 varieties of addition and subtraction instructions alone, and while Forth does not need all of these, we include nothing in the standard which we have not found useful. A number of apparently exotic words (e.g., `FSPLIT`) are in fact "internal" words, originally called by others, whose accessibility has seemed useful. These add little to complexity or length of code.

A useful criterion for the selection of a vocabulary is that the FP user should not be aware of the hardware. He should be able to do everything needful without ever having to write a machine-language word. Perhaps the easiest way to achieve this criterion is for an FP implementation to have *at least* the power of the HP-15C. The reason is simple: The 15C represents the culmination of a decade of Hewlett-Packard's market research into what people who do arithmetic need to do it with. What one can carry in a shirt pocket, he certainly should be able to find at a terminal!

Admittedly there is a difference between implementation and Standard, and this proposal still falls far short of reaching the capability of the 15C. By way of estimating the importance of the proposed words, the Appendices include a column labelled "Level" (for want of a better name) which refers each word to the following table.

Example	Level	Description
`FINIT`	0	One or two 8087 assembler instructions.
`FROT`	1	A line or two of 8087 assembler.
`FI`	2	Average user could write a slow version in high-level Forth.
`FSIN`	3	Complicated 8087. Left unfinished by 8087 designers.
`F.R`	4	Complicated 8087. Necessary for useful code.
`RECT`	-	Trivial word with sometimes complicated results.

Level 0 words are so simple, and necessary, that they will be written in by any user. Level 1 words will be written in by any user who can, by trial and error, imitate similar examples in his implementation. Since these words will appear in all implementations, they should be included in the Standard.

Level 2 could be reduced to Reference words, as they can be written in high-level Forth, using other available words. We include them because assembly-language versions of some are preferred for speed, and because users tend to add those they do not find.

Level 3 words implement functions whose underlying value is calculated by the 8087, but whose completion requires extensive coding and familiarity with the vagaries of the 8087: they would thus seem to belong in the Standard.

Level 4 words are so complicated that they are best supplied by professionals.

Both ends of this scaling indicate words which should be in the Standard, but for opposing reasons. Only the middle ground is truly negotiable, and we feel that we have made a utilitarian choice for these.

Dealing with F-Stack Limitations

We mentioned above the lack of an exception flag or interrupt for F-stack overflow. Considering that some of the complex inverse trigonometric functions require 7 of the 8 F-stack locations, it will be seen that careful attention must be paid to F-stack overflow. While dedicated Forth hackers may accept this as one more challenge, the average FP user may find it an extreme

annoyance. Unfortunately, the 8087 was not designed to deal with this problem, and there is neither an elegant fix, nor anything which we can recommend by way of warning signals which we feel is suitable for incorporation in the Standard.

To indicate what might be done in an implementation, we note that a zero tag word indicates a full F-stack. Every operation which pushes data onto the F-stack (such as the F<Y words, DATA-TYPE, FDUP, etc.) could include something like the following word:

```
: ?OVERFLOW @ENV ROT AAAA AND IF 2DROP 2DROP 2DROP ELSE ."
F-stack will overflow! " HEX 6 0 DO U. LOOP DECIMAL ABORT THEN ;
```

However, this is too high an overhead for those whose objective is speed rather than safety, so the overflow test should be made removable.

Several scientific users have requested not a trap, but a stack extension into which the F-stack could overflow seamlessly without user attention. This would be a kilobyte or so of dedicated memory logically continuous with the bottom of the F-stack, plus a couple of pointers and words to move 80-bit data between F-stack and extension. But this is slow in operation and complicated in detail, and every implementor we have talked to has felt that such a feature should be part of an implementation and not part of a standard. (This is, no doubt, the sort of complexity which makes Chuck Moore so wary of FP operations.) We do specify the words F>T and F<T to move 80-bit temporary reals between stack and F-stack, to provide temporary storage with no loss of precision.

No viable implementation of integer Forth lacks a way of non-destructively printing the contents of the parameter stack, and we should not expect FP users to do with less. It is a fact of life that the F-stack gets used for I/O operations on FP numbers, making it impossible to print out the complete contents of the F-stack in a straightforward way, but the 8087 does provide the tools needed, so we deem it important to make F-stack printability a requirement of the Standard, for which we specify the word .FS.

Thoughts on Complex Arithmetic

State-smart Rectangular | Polar CP words?

Some complex operators work best in rectangular representation; some in polar. One option is to convert all results to rectangular mode, regardless of internal operation: A second option is to have the words respond to a Rectangular | Polar flag (like a Degree | Radian flag for trigonometric functions). We have insufficient experience to make a recommendation for the Standard on this matter. The second option is sometimes convenient and has not, so far, gotten in the way of anything we have tried, or slowed things down noticeably. The idea deserves further discussion.

Special words

We include CROT and CRT2 as reference words against the day when hardware advances offer an F-stack deep enough to make them useful. The words RE, IM, and CONJ to get the real and imaginary part of a complex number, or take its complex conjugate, all have trivial high-level synonyms, as do i and i*, but in practice we find the descriptive names essential for clarity of code.

Festina Lente

Recent experience in implementing standards suggests that unfortunate choices can be minimized by the participation of a larger section of the Forth community than can make it to a given meeting. We suggest that proposed standards be published, used, and debated for a minimum of a year before they are officially acted upon.

Attributions and Acknowledgements

Dowling, as principal author of MMSFORTH, wrote the original version of most of the words in both Appendices, and added many improvements suggested by MacIntyre who used them heavily for three years, tinkered with many and added many.

MUG, the MMSFORTH User Group of eastern Massachusetts, has discussed this paper at length. John Rible (the other author of MMSFORTH), Jim Gerow, Dave Angel, Dave Lindbergh, Jack Pearson, and some anonymous reviewers have been especially helpful – although the latter make it clear that the debate is not yet settled.

David Jagerman and Mahlon Kelly contributed essential ideas, and the reviewers insisted upon the addition of processor control words and features for IEEE 754 compatibility.

References

Brodie, Leo 1984. *Thinking Forth* (Prentice-Hall, Englewood Cliffs, NJ).

Dowling, T. 1983. The QUAN concept expanded. *J. Forth Appl. Res.* 1(2):69-71.

Harwood, J.F. 1981. Forth Floating Point. *1981 Rochester Forth Standards Conf.* pp. 191-198.

Harris, K. 1984. Forth Coding Conventions. *1984 Rochester Forth Conf.* pp. 157-165.

Helmers, P.H. 1981. Some thoughts toward a Forth Floating-Point Standard. *1981 Rochester Forth Standards Conf.* pp. 199-211.

IEEE, 1985. IEEE Standard for Binary Floating-Point Arithmetic. *ANSI/IEEE Std.* 754-85, (Inst. of Elec. and Electronic Eng., 345 E 47th St, New York, NY 10017)

Knuth, D.E. 1981. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms* (Addison-Wesley, Reading, MA).

MacIntyre, F. 1984a. Number crunching with 8087 FQUANS: The Mie Equations. *J. Forth. Appl. Res.* 2(3):51-61.

MacIntyre, F. 1984b. Block pseudofiles. *J. Forth Appl. Res.* (in review).

Petersen, J.V., and Lennon, M. 1981. NIC-forth and Floating-Point Arithmetic. *1981 Rochester Forth Standards Conf.* pp. 213-217.

Rosen, E. 1982. QUAN and VECT – High-speed, low-memory-consumption structures. *Proc. Fourth FORML Conf.*

Sand, J.R., and J.O. Bumgarner 1983. Dysan IEEE P-754 Binary Floating-Point Architecture. *1983 Rochester Forth Appl. Conf.* pp. 185-194.

Startz, R. 1985. *8087 Applications and Programming for the IBM PC, XT, and AT.* 2nd ed. (Brady, NY) 291 pp.

Tracy, M. and Duncan, R. 1984. The FVG Standard Floating-Point Extension. *Dr Dobb's J.*, Sep. 1984, p.110-115.

Manuscript received June 1985.

Dr. Ferren MacIntyre received a Ph.D. in physical chemistry from MIT in 1965, and has worked as an oceanographer since, studying the physiochemical and small-scale dynamical properties of the sea surface and its subsets: oceanic bubbles and marine aerosol. His interest in Forth lies principally in number-crunching applications, but he has perforce had to develop supporting programs to do graphics and bibliographies in MMSFORTH, and most recently an image-analysis program in MacForth.

Tom Dowling has been the lead programmer for Miller Microcomputer Services since 1979, and has headed many major projects using Forth and other computer languages. He has a BS in Electrical Engineering from Pratt Institute and a MS in Electrical Engineering from Northeastern University.

Appendix I: Proposed FP Standard Extension

Unlike the parameter stack, the 8087 stack holds only 80-bit-wide normalized FP numbers or 10-character BCD strings. All flags, indices, addresses, etc. remain on the parameter stack. a b and c in stack comments below refer to normalized numbers on the FP stack, while items following a colon are on the parameter stack. ' is an address on the parameter stack. R in the "level" column indicates a reference word.

Name	Level	Stack
Stack operations		
FDUP	1	(a -> a a)
FSWAP	1	(a b -> b a)
FROT	1	(a b c -> b c a)
FRT2	1	(a b c -> c a b)
FOVER	1	(a b -> a b a)
FDROP	1	(a ->)
FPICK	4R	(a b .n. c : n -> a b .n. c n)
FROLL	4R	(a b .n. c : n -> a b .. c n)
Arithmetic		
F+	0	(a b -> a+b)
F-	0	(a b -> a-b)
F*	0	(a b -> ab)
F/	0	(a b -> a/b)
F\	0	(a b -> b/a)
-F	0	(a -> -a)
-F-	0	(a b -> b-a)
*2	0	(a -> 2a)
/2	0	(a -> a/2)
FABS	0	(a -> |a|)
1/X	0	(a -> 1/a) (alternate name: FINV)

Functions. The leading F is required for applications which also use similar integer functions, like the rapid trigonometric functions of MMSFORTH's TGRAPH, but another option in the interest of short names is to omit the F and rename the integer functions (e.g.: ISIN).

FINT	0	(a -> [a])
FRAC	0	(a -> a-[a])
FMOD	3	(a b -> a modulo b)
FSPLIT	2R	(a -> [a] a-[a])
FSGN	4	(a -> a : ?) a<0, ?=-1; a=0 ?=0; a>0 ?=1
FSQRT	0	(a -> $a^{1/2}$)
FLN	3	(a -> ln{a})
FLOG	3	(a -> $\log_{10}${a})
FEXP	4	(a -> exp{a})
F10^X	4	(a -> 10^a)
F^	4	(a b -> a^b)

(continued)

```
FSIN      3     (a -> sin{a})
FCOS      3     (a -> cos{a})
FTAN      3     (a -> tan{a})
FASIN     2     (a -> arcsin{a})
FACOS     2     (a -> arccos{a})
FATAN     3     (a -> arctan{a}) Principal value
FATN2     3     ( y x -> arctan{y/x}) 4-quadrant arctangent

FSINH     3     (a -> sinh{a})
FCOSH     3     (a -> cosh{a})
FTANH     3     (a -> tanh{a})
FASINH    2     (a -> arcsinh{a})
FACOSH    2     (a -> arccosh{a})
FATANH    3     (a -> arctanh{a}

DEGREES   -     Set angle flag to degree mode
RADIANS   -     Set angle flag to radian mode
```

Pseudorandom number generation being a subtle art (Knuth, 1981), its programming should be handled by professionals. Generating Gaussian FP numbers, for instance, is not a trivial problem. Consequently, we feel that any FP package worth its price should include:

```
FRAND     4R    (a -> pseudorandom FP number, 0 <= # <= a)
```

Constants

```
0,        0     ( -> 0) Alternate names  .0  0E0
1,        0     ( -> 1) Alternate names 1.0  1E0
PI        0     ( -> 3.141592653589793)
```

Processor Controls

```
FINIT              0     Initialize processor, clear F-stack
FROUND             2     Set rounding toward nearest
FROUND-            2R    Set rounding toward −∞
FROUND+            2R    Set rounding toward +∞
FTRUNC             2     Set truncation mode
AFFINE             2R    +Infinity ≠ −infinity
PROJECTIVE         2R    +Infinity = −infinity
@CW                4     Move processor control word from processor to stack
!CW                4     Move processor control word from stack to processor
@SW                4R    Move processor status word from processor to stack
SET-EXCEPTIONS     2R    Set exception flags and interrupts with mask
CLEAR-EXCEPTIONS   0R    Clear exception flags
@ENV               4R    Move 7-word 'environment' from processor to stack
TOFS               4R    Determine type of object at TOFS via condition codes
```

F-stack <-> Memory

<name>	4	Fetch value of FQUAN <name>
IS <name>	4	Store value into FQUAN <name>
AT <name>	4	Return address of FQUAN <name>
F@	4	(' :> a)
F!	4	(a :' ->)

I/O

.FS	4	Non-destructive printout of FP stack.
F#IN	4	Display ? and wait for FP number entry
F#OR	4	Display ? and wait for FP or CR; if CR, use number on F-8087 stack
FLW.	4	Set floating-point format of fixed number of numerals
FLD.	4	Set floating-point format of fixed number of decimals
SCI.	4	Set exponential format as x.nnn-p
ENG.	4	Set exponential format as xxx.nnn-3p
E.	4	(n ->) Print TOFS in exponential format with n decimals
F.	4	(n ->) Print TOFS in floating-point with n decimals
G.	4	(n ->) Machine choice of E. or F.
E.R	4	(n w ->) E. right-justified in field of width w
F.R	4	(n w ->) F. right-justified in field of width w
G.R	4	(n w ->) G. right-justified in field of width w
FE.	4	Full exponential format
FF.	4	Full floating-point format

Comparisons. Flag is left on parameter stack.

FCOMP	4	(a b -: ?) a<b -> -1, a=b -> 0, a>b -> 1, unordered ->
ABORT		
F<	2R	(a b -: ?) a<b -> 1
F=	2R	(a b -: ?) a=b -> 1
F>	2R	(a b -: ?) a>b -> 1
F0<	2R	(a -: ?) a<0 -> 1
F0=	2R	(a -: ?) a=0 -> 1
F0>	2R	(a -: ?) a>0 -> 1
F<=	2R	(a b -: ?) a<=b -> 1
F<>	2R	(a b -: ?) a<>b -> 1
F>=	2R	(a b -: ?) a>=b -> 1
FMAX	2	(a b -> larger of a,b)
FMIN	2	(a b -> smaller of a,b)

Stack transfers. All begin with F to keep them together in the glossary. All are Level 4.

"F-from-Y"	"F-onto-Y"	Move:
F<S	F>S	Integer
F<D	F>D	Double-length integer
F<T	F>T	80-bit temporary real format
F<SF	F>SF	32-bit floating point. Reference word
F<DF	F>DF	64-bit floating point. Reference word
F<B	F>B	10-characters of BCD.
F<60	F>60	Hours|Degrees Minutes Seconds

Index operations

I J	1	Inner- and outer- loop indices
I- J-	2	Index−1
I+ J+	2	Index+1
FI FJ	2	Put indices on F-stack

Data types

FVARIABLE	4R	Not recommended. See below.
FCONSTANT	4R	Not recommended. See below.

Using a second-generation "grandparent" defining word, it is possible to reduce the number of data-type words in the standard to one, with numerous controlled reference words.

DATA-TYPE	4 (d w t a ->), where:
d=dimension	0, 1, 2, or 3 dimensions.
w=width	0 1 2 = 8-, 16-, or 32-bit integers; or 32- and 64-bit FP numbers and 80-bit temporary real numbers, or BCD numbers.
t=type	0=integer, 1=floating point, 2=complex, 3=BCD.
a=address	0=in-line storage, 1=indirect, 2=long-address.

Used in the form d w t a DATA-TYPE <data-type-name> where <data-type-name> is a defining word from the following reference list:

d w t a		d w t a		d w t a	
0 0 0 0	QUAN	0 1 0 0	DQUAN		
0 0 0 1	IQUAN	0 1 0 1	DIQUAN		
0 0 0 2	LQUAN	0 1 0 2	DLQUAN		
0 1 1 0	FQUAN	0 2 1 0	DFQUAN	0 3 1 0	TFQUAN
0 1 1 1	FIQUAN	0 2 1 1	DFIQUAN	0 3 1 1	TFIQUAN
0 1 1 2	FLQUAN	0 2 1 2	DFLQUAN	0 3 1 2	TFLQUAN
0 1 2 0	CQUAN	0 2 2 0	DCQUAN	0 3 2 0	TCQUAN
0 1 2 1	CIQUAN	0 2 2 1	DCIQUAN	0 3 2 1	TCIQUAN
0 1 2 2	CLQUAN	0 2 2 2	DCLQUAN	0 3 2 2	TCLQUAN
0 1 3 0	BQUAN	0 2 3 0	DBQUAN	0 3 3 0	TBQUAN
0 1 3 1	BIQUAN	0 2 3 1	DBIQUAN	0 3 3 1	TBIQUAN
0 1 3 2	BLQUAN	0 2 3 2	DBLQUAN	0 3 3 2	TBLQUAN

For d=1, -QUAN is replaced by -ARRAY (or -QARRAY)
d=2, -QUAN is replaced by 2...ARRAY
d=3, -QUAN is replaced by 3...ARRAY

Appendix II: Proposed Complex-Arithmetic Standard Extension

The CP extension assumes that the FP extension has been loaded.

Name	Level	Stack

Stack operations

CDUP	0	(x yi -> x yi x yi) = FOVER FOVER
CSWAP	1	(x yi u vi -> u vi x yi)
CROT	1R	(x yi u vi w zi -> u vi w zi x yi)
CRT2	1R	(x yi u vi w zi -> w zi x yi u vi)
COVER	1	(x yi u vi -> x yi u vi x yi)
CDROP	0	(x yi ->) = FDROP FDROP
RE	0	(x yi -> x) = FDROP
IM	0	(x yi -> yi) = FSWAP FDROP

Complex arithmetic

C0	0	(-> 0 0)
C1	0	(-> 1 0)
i	0	(-> 0 1)
C+	1	(x yi u vi -> {x+u} {y+v}i)
C-	1	(x yi u vi -> {x-u} {y-v}i)
C*	4	(x yi u vi -> {xu-yv} {xv+yu}i)
C/	4	(x yi u vi -> {xu+yv}/{u^2-v^2} {yu-xv}i/{u^2-v^2})
i*	0	(x yi -> -y xi) = -F FSWAP
CONJ	0	(x yi -> x -yi) = -F
-C	0	(x yi -> -x -yi)
-C-	1	(x yi u vi -> {u-x} {v-y}i)
C*F	2	(x yi f -> xf yfi)
MAG	4	(x yi -> r) (Used by R>P)
PHASE	4	(x yi -> θ) (Used by R>P)

Choice and conversion of coordinates. (A possibility for discussion).

RECT	-	Set mode flag to rectangular. Expects and leaves x yi on F-stack.
POLAR	-	Set mode flag to polar. Expects and leaves r θ on F-stack.
R>P	4	(x yi -> r θ)
P>R	4	(r θ -> x yi)

Functions. (Actual results should be converted to rectangular or determined by the RECT|POLAR state. z = x + yi, and is a two-entry complex number.)

CLN	2	(x yi -> ln{z})
CEXP	2	(x yi -> exp{z})
CSQRT	2	(x yi -> $z^{1/2}$)
1/C	2	(x yi -> z^{-1})
C^	2	(x yi a -> z^a)

(continued)

```
CSIN      2   (x yi -> sin{z})
CCOS      2   (x yi -> cos{z})
CTAN      2   (x yi -> tan{z})
CASIN     2   (x yi -> arcsin{z})
CACOS     2   (x yi -> arccos{z})
CATAN     2   (x yi -> arctan{z})

CSINH     2   (x yi -> sinh{z})
CCOSH     2   (x yi -> cosh{z})
CTANH     2   (x yi -> tanh{z})
CASINH    2   (x yi -> arcsinh{z})
CACOSH    2   (x yi -> arccosh{z})
CATANH    2   (x yi -> arctanh{z})
```

I/O

```
C#IN      2   Display ? and wait for CP number entry
C#OR      2   Display ? and wait for CP or CR; if CR, use number on 8087 F-stack
C.        2   ( n ->) Output complex number pair in G. format
```