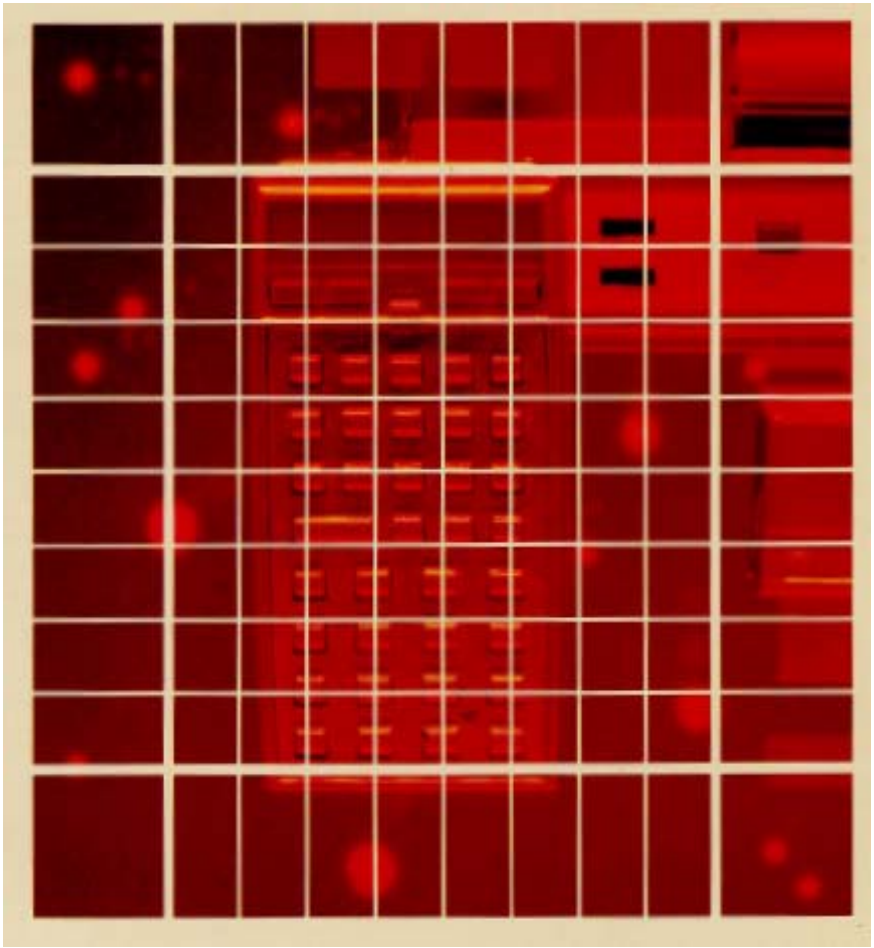
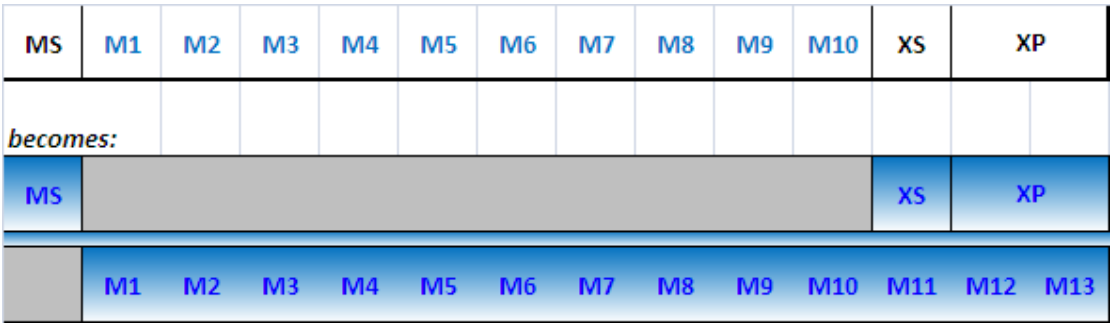
HP41 OS 13-digit Math Routines.

Scratching the surface 30 years later.





Ángel M. Martin.

February 2011

This compilation, revision A.1.5

13-digit Math Routines within the 41 OS.

Intro.

The purpose of this mini-paper is to document the usage of some of the 13-digit math routines within the 41 OS – for extended precision in the calculations of functions in the SandMath and the 41Z modules. It's not a comprehensive description of the routines, nor should it be seen as a complete overview of the 41 Math Rom (system ROM_1), which – I'm glad to say – still holds many secret chestnuts and undocumented treasures.

It's assumed that the reader is familiar with the "standard|" math routines in the OS, such as [AD2_10], [MP2_10], [LN10], etc. I'll make no attempt to explain those anything beyond a comparison point with the extended ones.

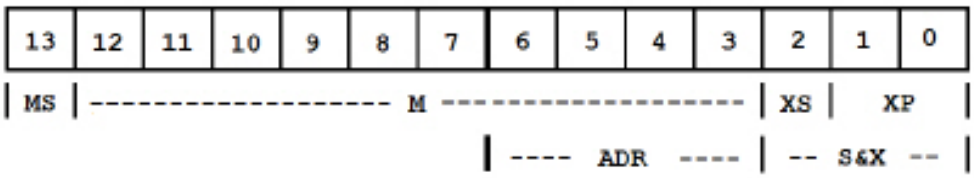
I'm not a computer science engineer; so pls. allow some imprecise rigor or unclear use of terms within the descriptions below – you've been warned!

The basics: Nibbles and digits.

The NUT CPU - a 30-year old classic by today's standards – uses 7-byte registers as main memory unit. Each byte has two nibbles, each nibble being 4 bits - which makes it a 56-bit CPU, if that means anything these days.

Nibbles (or digits) are labeled from right to left as 0 to 13, with nibble 0 holding the LSB. The typical assignment distribution of those fields is as follows:

- *nibble 13 holds the sign of the mantissa*, called the MS field. A zero denotes positive numbers, and 9 a negative number
- *nibbles 12 to 3 hold the mantissa*, called the M field and ranging from 0 to 9999999999
- *nibble 3 holds the sign of the exponent*, called the XS field - with zero positive and 9 negative exp.
- *nibbles 1 and 0 (that is: byte 0) hold the exponent*, called the XP field - ranging from 0 to 99. The combination of nibbles 2 to 0 is also called the S&X field.



Typically a number in the calculator is held in a single register, thus its precision is limited to 10 digits. This was good enough for 1980 standards for calculator design, but as algorithms get more complicated the numerical errors become more significant, rendering the system sub-optimal.

Yet the 41 OS math routines (in ROM1) were written with extended precision in mind. They employ two CPU registers to hold the numbers, extending the actual precision to 13 digits, as follows:

- *first register holds MS and the S&X,*
- *second register holds a 13-digit mantissa.*

From this it's clear that the routines should then operate taking a dual-register definition for each one of the input arguments (one if MONADIC and two if DUAL), and conversely should exit leaving the final result in two registers as well.

But in fact, they're even better than that.

Considering that the initial input data and the final output result can only have a 10-digit form, those routines must also accept 10-digit inputs, and must produce 10-digit outputs. No surprisingly they are indeed designed to ***accept input on either way***, and to output the results ***in both ways***. There's even a third hybrid form, with one operand in 13-digit form and another in 10-digit form.

This allows *intermediate operations to be done with 13-bit precision*, which is *good enough to achieve a 10-digit accuracy*, without rounding errors or loss of resolution.

The important think to realize is that there is only ONE set of math routines, and not dual sets. The routines always operate on a dual-mode basis, and exit with dual results placed in both 10-digit form [in register C], and 13-digit form [in arithmetic registers A&B].

Different entry points within the routines determine whether it's using the extended precision or not, just truncating the input values to 10-digit forms. So even the standard [AD2_10] routine produces a 13-digit output in registers A&B, even though most of the time this isn't taken advantage of and only the result placed in C is used.

So from now on we won't refer to either 10-digit or 13-digit routines anymore, as there's only one set – with a choice of manners to use them, depending on the syntax (entry points) and the richness of the data.

This also means *the execution time is not longer when the 13-digit forms are used*, thus there's no penalty in using them!

And besides that, to make things even more attractive, *the code listings get simplified if the extended precision syntax is used* – due to the way the routines have been written, and to the auxiliary routines used to facilitate repeated operations. So typically programs are shorter, saving vital bytes.

Those three reasons are surely good enough to justify the learning curve, won't you agree? So let's get to it, shall we…

Entry points and digit forms.

Clever as they were, the 41 OS programmers followed a nomenclature system to consistently label the different entry points for all routines in a common way, denoting the data form convention to use.

So a dual 10-digit form always uses the "2-10" string in their names, and the dual 13-digit form uses "2-13" – whereas the *one-of-each* hybrid uses "1-10" as descriptor. All perfectly clear, and very helpful to get your bearings when navigating the MCODE listings.

Let's first review what we already knew:-

- A single 10-digit input for MONADIC functions is expected to be in Register C. This of course notwithstanding additional requirements, like also being in register N (as it's done with [XFCT100],

- A dual 10-digit input for DUAL functions is expected to be in registers A for the first operand, and register C for the second. Ditto here w.r.t. additional requirements, like also using X/Y (like in [TOPOL] – this one probably the most complex routine in the OS alone!)

And here's the extension to the knowledge:

- A single 13-digit input for MONADIC functions is expected to be in the arithmetic registers A&B, as per the 13-digit convention explained before. Examples are: [LN13], [SQR13], [EXP13].

- A dual 13-digit input for DUAL functions is expected to be in registers A&B for the first operand, and registers C&M for the second one. The order is especially important for the division routine, of course. Examples are [AD2-13] , [MP2-13] and [DV2-13] & [X/Y13] (its reverse).

- A hybrid 10- and 13-digit input for DUAL functions is expected to be in registers A&B for the first operand (13-digit form following the extended convention), and in register C for the second operand (10-digit form following the standard convention). Examples are [AD1-10], [MP1-10] and [DV1-10].

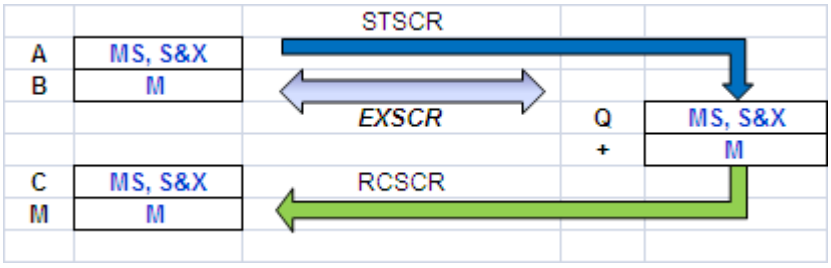
  Note that there aren't entry points for the reverse situation – operand 1 must be the 13-digit form

But wait, there's more. To further facilitate interoperability within all these routines, there are auxiliary ones that make the chain calculations almost as simple as if we were using an RPN stack. They are:

- [STSCR].- A routine to temporarily store a 13-digit result for later reuse, using scratch registers "Q" &"+" – using a slightly modified convention due to the compromised usage of the "+" register. (Now we know why Q can't be used in our MCODE so often!)

- [RCSCR].- Its trusty companion, a routine to restore the scratch registers into registers {C&M}, so their content can be used as second operand by a DUAL function. (It takes care of the subtleties used in [STSCR] to reverse the convention in compatible way).

- [EXSCR].- The third one. A routine to exchange (swap) the contents of registers {A&B} with the scratch registers. Nothing like the versatility of a swap sometimes! Very useful in hybrid operations (10-digit & 13-digit).



- [ADDONE], [SUBONE].- handy routines to add/subtract one (written ad-hoc in C) to/from the number held in registers {A&B}, leaving the result also in {A&B}. (Guess which native functions used these? Here's a hint: LNX+1 and E^X-1).

- [PI/2].- There's never enough number of decimal digits for an irrational number, won't you say? This routine gives 13 of them, and it's broadly used all along the Math ROM. Also note that it's placed in {C&M}, ready to be used as 13-digit second operand for addition, division, or multiplication steps.

- [LNC10], [LNC20], [LNC30], [LNC40], [LNC50], etc. As their names imply, routines to input the different values of the decimal log for those relevant values. (I'm a bit fuzzy here as I haven't really studied them all).

- [RTOD] and [DTOR].- Radians-to-Degrees and Degrees-to-Radians conversions. The result is also given in dual form, 13-digit in {A&B} plus 10-digit in C.

- [SQR13] .- 13-digit entry for the Square root routine. This is to be used with caution, as it doesn't always return correct results in chain calculations. If the mantissa result is zero, and it's to be followed by [ADD1-10] or [ADD2-13] then the mantissa sign field in B must be cleared prior to the addition step (as discovered by Jean-Marc Baillard).

- [TRGSET] and its multiple variants as per the flag settings, such as:
  - CF 0, CF 1 = [SIN]; - CF 0, SF 1 = [COS]; - SF 0, SF 1 = [TAN]
  This is an interesting case, which may have an implementation defect or simply wasn't supposed to be used as intermediate step in calculations. The result is given in dual form as well, ***but the mantissa sign of the 13-digit form is not always correct*** – thus it's necessary to use A=C MS if the 13-digit form is to be used in subsequent calculations.

Let's see some examples of how to use these handy routines, comparing the syntax and code reduction as things get more powerful:-

Example 1.- Adding X and Z.

No frills here: each input is a 10-digit form so our old-reliable routines do the work – and there's no additional value in using the 13-digit syntax, as we can't possibly "guess" the three missing digits!

```
READ 1(Z)
A=C ALL
READ 3(X)
[AD2-10]
```

Example 2. Calculate (X+Z)*Y

Here's a good chance to compare the " standard" vs. the "extended" syntax:

```
READ 1(Z)
A=C ALL
READ 3(X)
[AD2-10]
A=C ALL
READ 2(Y)
[MP2-10]
```

```
READ 1(Z)
A=C ALL
READ 3(X)
[AD2-10]    -so far so good, but watch:
READ 2(Y)   - uh? No need to obliterate A
[MP1-10]    - an elegant way to finish.
```

Note how we didn't need to save the intermediate result as 10-digit form into A, as it was ALREADY saved into A&B in 13-digit form. (so saving it would've destroyed it). Note also we saved one line whist gaining precision – a great deal.

Example 3. Calculate (X+Z)*(Y+T)

```
C=0 S&X
RAMSLCT
READATA
A=C ALL
READ 2(Y)
[AD2-10]
N=C
READ 1(Z)
A=C ALL
READ 3(X)
[AD2-10]
A=C ALL
C=N
[MP2-10]
```

```
C=0 S&X     - only way to read the T register
RAMSLCT     - select bottom of chip0
READATA     - get first operand to C
A=C ALL     - save it in A
READ 2(Y)   - so far so good, but watch:
[AD2-10]    - first intermediate result:
[STSCR]     - saved in scratch
READ 1(Z)
A=C ALL
READ 3(X)
[AD2-10]    - second intermediate result
[RCSCR]     - the trusty companion
[MP2-13]    - full dual 13-digit glory at last
```

In all fairness there's not such a code reduction here since the calls to [STSCR] and [RCSCR] required two bytes, whereas using N as scratch register is a single byte each way. Nevertheless the code clarity and enhanced accuracy are always there.

Example 4. Calculate Ln(X)+Y

This one also benefits from the extended entry points syntax, as follows:

```
READ 3(X)
CLRF 5
[LN10]
A=C ALL
READ 2(Y)
[AD2-10]
```

```
READ 3(X)   - get operand to C
CLRF 5      - flag the proper logarithm to use
[LN10]      - result in both 10 & 13-digit forms
READ 2(Y)   - get second operand to C
[AD1-10]    - hybrid multiplication to end
```

Example 5. Calculate SQR(X^2+Y^2) - (a.k.a the module.)

```
READ 3(X)
A=C ALL
[MP2-10]
N=C
READ 2(Y)
A=C ALL
[MP2-10]
A=C ALL
C=N
[AD2-10]
[SQR10]
```

```
READ 3(X)   - get first operand to C
A=C ALL     - duplicate in A
[MP2-10]    - square power in both 10/13-digit forms
[STSCR]     - 13-digit form saved in scratch
READ 2(Y)   - get second operand to C
A=C ALL     - duplicate in A
[MP2-10]    - result in both 10/13-digit forms
[RCSCR]     - recall first partial result to C&M
[AD2-13]    - all-13-digit addition to end
[SQR13]
```

Examples 6 & 7.- Calculate (X^3)+1 and (1/X^2) -1

Now without the corresponding comparisons, to expedite the illustration.-

a) Adding one

```
READ 3(X)
A=C ALL
[MP2-10]
READ 3(X)
[MP1-10]
[ADDONE]
```

b) Subtracting one

```
READ 3(X)
A=C ALL
[MP2-10]
[ON/X13]
[SUBONE]
```

The cube power is good example of combining the dual result into a chain calculation, in this case a hybrid product: economy of steps and calculation efficiency in one!

Example 8.- A couple of tricks to impress your friends.

Calculate 1/X^2, and divide a 10-digit number in N by its value (i.e. the reciprocal to [DV1-10]).

The first part is easy, let's do it in two different ways::

a) The sub-optimal way	*b) the full-resolution way*
READ 3(X)	READ 3(X)
[ON/X10]	[ON/X10]
[MP1-10]	**[STSCR]**
	[RCSCR]
	[MP2-13]

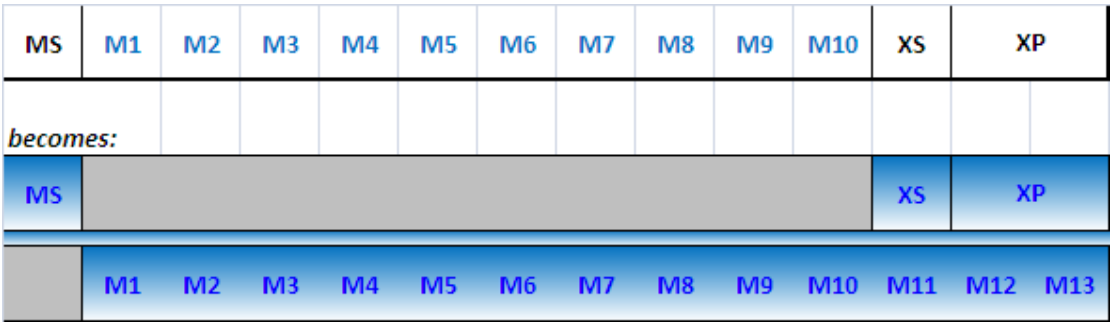
In the shorter implementation (on the left) we take advantage of the fact that the output from the [1/X] routine is dual, as 13-digit form in A&B and also as a 10-digit form in C. It's shorter but of course doesn't utilize the full information available and therefore the longer implementation (on the right) is recommended. If space is at a premium (as it eventually always is), the sub-optimal implementation is still better than a dual 10-digit one – which also takes man extra code line.

The second part is a little trickier. Since there isn't an entry point to reflect this kind of hybrid scenario, we'll upgrade the 10-digit value into a "fake"13-digit one in order to use the dual 13-digit division entry point, as follows:

[STSCR]	- saves A&B in scratch
C=N	- puts the 10-digit value to C
A=C ALL	- puts its sign and X&S in A
B=0 ALL	- a precaution to clear unused fields
C<>B M	- puts the 10-digit mantissa in B
[EXSCR]	- swaps the "fake" upgraded value and the genuine one
[RCSCR]	- *places the fake value into M&C*
[DV2-13]	- performs the division our way: {A,B} over {M,C}

Note that the last three steps can be replaced by this simplified way:

[RCSCR]	- brings {Q,+} to {M,C}
[X/Y13]	- divides {M,C} by {A,B} instead!



Example 9- Get the gloves off and calculate PROD [X+k], for k=1,2…6

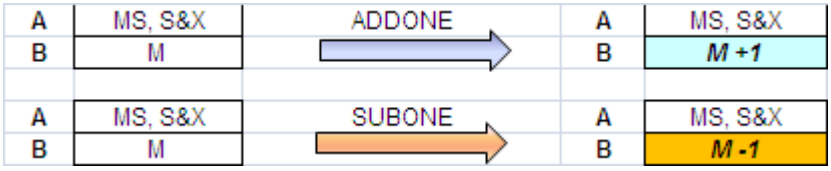
Now in a more realistic vein, we'll approach this one using a more detailed listing description as follows:

Highlights here are the usage of [EXSCR] and the initial conversion of the input value to a "fake" 13-digit form, so that the same syntax could be used inside of the multiplication loop.

0F8	READ 3(X)	x
10E	A=C ALL	A holds sign and S&X
02E	B=0 ALL	clears B
07A	A<>B M	B holds 13-digit mant
089	?NC XQ	x
064	->1922	[STSCR]
0E0	SLCT Q	
15C	PT= 6	loop counter
0A9	?NC XQ	k
064	->192A	[EXSCR]
0A0	SLCT P	
001	?NC XQ	x+k+1
060	->1800	[ADDONE]
0A9	?NC XQ	partial product
064	->192A	[EXSCR]
0D1	?NC XQ	x+k+1
064	->1934	[RCSCR]
149	?NC XQ	PP * (x+k+1)
060	->1852	[MP2-13]
0E0	SLCT Q	
3D4	PT=PT-1	loop 6 times
394	?PT= 0	
393	JNC -14d	
089	?NC XQ	**PROD(x+k) |k=0,1..6**
064	->1922	[STSCR]

The result of the loop is in 13-digit form, and therefore it's ready for further chain calculations as need may be.

Note that pointer P is used by both [ADDONE] and [MP2-13], thus we need to use Q instead for the loop counter.

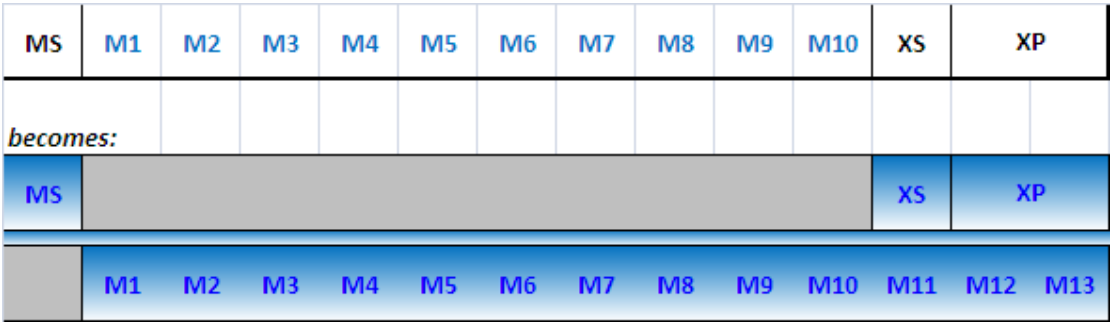


Example 10.- Calculate 2X + 1168,92649479

Notice of course that there are 12 digits in all required defining the value to add; therefore we'll have to use a dual 13-digit form addition. All we need is converting the given number to a proper 13-digit form in order to be able to use all its 12 digits in the adding operation.

This entails not only writing its mantissa in C, but also its sign and S&X into the M register to have a whole-defined value. Also let's not forget clearing the unused values in either register to avoid sporadic errors, so difficult to troubleshoot.

0F8	READ 3(X)	
10E	A=C ALL	
01D	*?NC XQ*	*Adds normalized*
060	*->1807*	*[AD2_10]*
04E	C=0 ALL	
130	LDI S&X	*S&X in M*
003	CON: 3	
158	M=C ALL	
04E	C=0 ALL	
35C	PT=12	
050	LD@PT- 1	
050	LD@PT- 1	
190	LD@PT- 6	
210	LD@PT- 8	
250	LD@PT- 9	***1168.92649479***
090	LD@PT- 2	
190	LD@PT- 6	
110	LD@PT- 4	
250	LD@PT- 9	
110	LD@PT- 4	
1D0	LD@PT- 7	
250	LD@PT- 9	
031	*?NC XQ*	
060	*->180C*	*[AD2-13]*



Example 11.- Pi in the (13-digit) sky.

Working with π has two variants, depending on whether it's used as first or second operand. Remember that [PI/2] writes a 13-digit version of π into the C&M registers, so it's ready as a second argument but requires some work it it's to be used as first:-

For instance, to calculate the square root of 2π

00E	A=0 ALL	
269	*?NC XQ*	
064	*->199A*	*[PI/2]*
1EE	C=C+C ALL	
1EE	C=C+C ALL	*2pi*
0EE	B<>C ALL	
305	*?NC XQ*	
060	*->18C1*	*[SQR13]*

Returned to both {A,B} (13-digit form) and C (10-digit form).

And to divide a 13-digit form number in {A,B} by π

269	*?NC XQ*	
064	*->199A*	*[PI/2]*
1EE	C=C+C ALL	*pi*
275	*?NC XQ*	*Ln(X)/Ln(Y)*
060	*->189D*	*[DV2-13]*

The 10-digit form of π is obtained by rounding the mantissa to just 10 digits, and by truncating the S&X field after the execution of [PI/2], as can be seen below in the well-known steps of the Coconut OS:

Header	1240	089	***"I"***	
Header	1241	010	***"P"***	
PI	**1242**	**2A0**	**SETDEC**	
	1243	*269*	*?NC XQ*	
	1244	*064*	*->199A*	*[PI/2]*
	1245	1EE	C=C+C ALL	
	1246	23A	C=C+1 M	*rounding?*
	1247	046	C=0 S&X	*truncation to 10-digit*
LXEX	1229	0EE	B<>C ALL	
RCL	**122E**	**18C**	**?FSET 11**	
	122F	*3B5*	*?C XQ*	
	1230	*051*	*->14ED*	*[R^SUB]*
HPRCL	**1231**	04E	C=0 ALL	
	1232	270	RAMSLCT	
	1233	0EE	B<>C ALL	
	1234	0E8	WRIT 3(X)	
NFRPRL	**1235**	*3B9*	*?NC GO*	
	1236	*002*	*->00EE*	*[NFRPR]*

A real-life example: Hyperbolic functions at last.

Yes there are many unanswered questions in the universe, but certainly one of them is why, oh why, didn't HP-MotherGoose provide a decent set of hyperbolic functions in the (otherwise pathetic) MATH-PAC, and worse yet -adding insult to injury- how come that error wasn't corrected in the Advantage ROM?

For sure we'll never know, so it's about time we move on and get on with our lives – whilst correcting this forever and ever. The first incarnation of these functions came in the AECROM module; I believe programmed by Nelson C. Crowle, a real genius behind such ground-breaking module - but it was also somehow limited to 10-digit precision.

Here they are in all their 13-digit splendor at last.-

First the inverse functions, using the logarithm and square root routines.

Header	A084	*08E*	*"N"*	
Header	A085	*009*	*"I"*	
Header	A086	*013*	*"S"*	*Ln[x+sqr(x^2+1)]*
Header	A087	*001*	*"A"*	
Header	A088	*008*	*"H"*	*Ángel Martin*
ASINH	**A089**	**248**	**SETF 9**	
ASINH	A08A	***03B***	***JNC +07***	
Header	A08B	*093*	*"S"*	
Header	A08C	*00F*	*"O"*	
Header	A08D	*003*	*"C"*	*Ln[x+sqr(x^2-1)]*
Header	A08E	*001*	*"A"*	
Header	A08F	*008*	*"H"*	*Ángel Martin*
ACOSH	**A090**	**244**	**CLRF 9**	
ACOSH	A091	0F8	READ 3(X)	
ACOSH	A092	*361*	*?NC XQ*	*(this includes SETDEC)*
ACOSH	A093	*050*	*->14D8*	*[CHK_NO_S]*
ACOSH	A094	3C4	ST=0	*in case it's called by XEQAX*
ACOSH	A095	10E	A=C ALL	
ACOSH	A096	*135*	*?NC XQ*	*x^2*
ACOSH	A097	*060*	*->184D*	*[MP2_10]*
ACOSH	A098	24C	?FSET 9	
ACOSH	A099	*001*	*?C XQ*	
ACOSH	A09A	*061*	*->1800*	*[ADDONE]*
ACOSH	A09B	24C	?FSET 9	
ACOSH	A09C	*009*	*?NC XQ*	
ACOSH	A09D	*060*	*->1802*	*[SUBONE]*
ACOSH	A09E	*305*	*?NC XQ*	
ACOSH	A09F	*060*	*->18C1*	*[SQR13]*
ACOSH	A0A0	0F8	READ 3(X)	
ACOSH	A0A1	*025*	*?NC XQ*	
ACOSH	A0A2	*060*	*->1809*	*[AD1_10]*
ACOSH	A0A3	*121*	*?NC XQ*	
ACOSH	A0A4	*06C*	*->1B48*	*[LN13]*
ACOSH	A0A5	*331*	*?NC GO*	*Overflow, DropST, FillXL & Exit*
ACOSH	A0A6	*002*	*->00CC*	*[NFRX]*

Header	A0A7	*08E*	*"N"*	
Header	A0A8	*001*	*"A"*	
Header	A0A9	*014*	*"T"*	*1/2[ln(1+x) - ln(1-x)]*
Header	A0AA	*001*	*"A"*	
Header	A0AB	*008*	*"H"*	*Ángel Martin*
ATANH	**A0AC**	**0F8**	**READ 3(X)**	
ATANH	A0AD	*361*	*?NC XQ*	*(this includes SETDEC)*
ATANH	A0AE	*050*	*->14D8*	*[CHK_NO_S]*
ATANH	A0AF	3C4	ST=0	*in case it's called by XEQAX*
ATANH	A0B0	*1CD*	*?NC XQ*	*Ln(x+1)*
ATANH	A0B1	*06C*	*->1B73*	*[XLN1+X]*
ATANH	A0B2	*089*	*?NC XQ*	*Ln(x+1)*
ATANH	A0B3	*064*	*->1922*	*[STSCR]*
ATANH	A0B4	0F8	READ 3(X)	
ATANH	A0B5	2BE	C=-C-1MS	*-x*
ATANH	A0B6	00E	A=0 ALL	
ATANH	A0B7	35C	PT= 12	***Build "1" in A***
ATANH	A0B8	162	A=A+1 @PT	
ATANH	A0B9	*01D*	*?NC XQ*	*1-x*
ATANH	A0BA	*060*	*->1807*	*[AD2_10]*
ATANH	A0BB	3C4	ST=0	
ATANH	A0BC	*121*	*?NC XQ*	*Ln(1-x)*
ATANH	A0BD	*06C*	*->1B48*	*[LN13]*
ATANH	A0BE	2BE	C=-C-1MS	*Sign change*
ATANH	A0BF	11E	A=C MS	*-Ln(1-x)*
ATANH	A0C0	*0D1*	*?NC XQ*	*Ln(x+1)*
ATANH	A0C1	*064*	*->1934*	*[RCSCR]*
ATANH	A0C2	*031*	*?NC XQ*	
ATANH	A0C3	*060*	*->180C*	*[AD2-13]*
ATANH	A0C4	04E	C=0 ALL	
ATANH	A0C5	35C	PT= 12	***Builds "2" in C***
ATANH	A0C6	090	LD@PT- 2	
ATANH	A0C7	*269*	*?NC XQ*	
ATANH	A0C8	*060*	*->189A*	*[DV1-10]*
ATANH	A0C9	*331*	*?NC GO*	*Overflow, DropST, FillXL & Exit*
ATANH	A0C10	*002*	*->00CC*	*[NFRX]*

We could've used a "fake" 13-digit format for X to have access to [ADDONE] and [SUBONE], but that wouldn't have added any more precision to the result.

Instead we used the [LN1+X] routine, which internally does the same thing anyway. And of course we'll utilize its 13-digit output!

On the other hand, we use the scratch registers to store the partial result Ln(1+x), so that it'll be fully used as 13-digit value in the final subtraction step.

$$\operatorname{arsinh} x = \ln\left(x + \sqrt{x^2+1}\right)$$

$$\operatorname{arcosh} x = \ln\left(x + \sqrt{x^2-1}\right); x \geq 1$$

$$\operatorname{artanh} x = \frac{1}{2}\ln\frac{1+x}{1-x}; |x| < 1$$

And here are the direct functions, a festival of exponential routines for you:-

Header	A0CB	08E	*"N"*	
Header	A0CC	009	*"I"*	*sh(x)=1/2[e^x-e^-x]*
Header	A0CD	013	*"S"*	
Header	A0CE	008	*"H"*	*Ángel Martin*
SINH	**A0CF**	**148**	**SETF 6**	
SINH	A0D0	***033***	***JNC +06***	
Header	A0D1	093	*"S"*	
Header	A0D2	00F	*"O"*	*ch(x)=1/2[e^x+e^-x]*
Header	A0D3	003	*"C"*	
Header	A0D4	008	*"H"*	*Ángel Martin*
COSH	**A0D5**	**144**	**CLRF 6**	
COSH	A0D6	188	SETF 11	*Go noisy!*
COSH	A0D7	0F8	READ 3(X)	
COSH	A0D8	0EE	B<>C ALL	
SHYP	**A0D9**	**0EE**	**B<>C ALL**	*subroutine use*
SHYP	A0DA	*361*	*?NC XQ*	*(this includes SETDEC)*
SHYP	A0DB	*050*	*->14D8*	*[CHK_NO_S]*
SHYP	A0DC	044	CLRF 4	
SHYP	A0DD	*029*	*?NC XQ*	
SHYP	A0DE	*068*	*->1A0A*	*[EXP10]*
SHYP	A0DF	*089*	*?NC XQ*	*e^x*
SHYP	A0E0	*064*	*->1922*	*[STSCR]*
SHYP	A0E1	*239*	*?NC XQ*	*e^-x*
SHYP	A0E2	*060*	*->188E*	*[ONIX13*
SHYP	A0E3	**14C**	**?FSET 6**	*true if SINH*
SHYP	A0E4	*013*	*JNC +02*	
SHYP	A0E5	2BE	C=-C-1 MS	*Sign change*
SHYP	A0E6	11E	A=C MS	*ditto in A*
SHYP	A0E7	*0D1*	*?NC XQ*	*e^x*
SHYP	A0E8	*064*	*->1934*	*[RCSCR]*
SHYP	A0E9	*031*	*?NC XQ*	
SHYP	A0EA	*060*	*->180C*	*[AD2-13]*
SHYP	A0EB	04E	C=0 ALL	
SHYP	A0EC	35C	PT=12	***build "2" in C***
SHYP	A0ED	099	LD@PT- 2	
SHYP	A0EE	*269*	*?NC XQ*	
SHYP	A0EF	*060*	*->189A*	*[DV1-10]*
SHYP	A0F0	18C	?FSET 11	*subroutine mode?*
SHYP	A0F1	***3A0***	***?NC RTN***	
SHYP	A0F2	*331*	*?NC GO*	*Overflow, DropST, FillXL & Exit*
SHYP	A0F3	*002*	*->00CC*	*[NFRX]*

Here the usage of a subroutine and CPU flag 11 is due to the fact that the LogGamma function (elsewhere in the SandMath) requires calculating the hyperbolic sine as intermediate step, otherwise it could have a shorter listing, and without using register B either.

$$\sinh x = \frac{e^x - e^{-x}}{2} \qquad \cosh x = \frac{e^x + e^{-x}}{2} \qquad \tanh x = \frac{e^{2x} - 1}{e^{2x} + 1}$$

Note also the common ending in these, suitable to yet further space savings just by using a JNC call to the appropriate section. We have however left it like that for clarity.

Header	A0F4	*08E*	*"N"*	
Header	A0F5	*001*	*"A"*	*th(x)=[e^2x-1]/[e^2x+1]*
Header	A0F6	*014*	*"T"*	
Header	A0F7	*008*	*"H"*	*Ángel Martin*
TANH	**A0F8**	**0F8**	**READ 3(X)**	
TANH	A0F9	*361*	*?NC XQ*	*(this includes SETDEC)*
TANH	A0FA	*050*	*->14D8*	*[CHK_NO_S]*
TANH	A0FB	10E	A=C ALL	
TANH	A0FC	*01D*	*?NC XQ*	*2x*
TANH	A0FD	*060*	*->1807*	*[AD2_10]*
TANH	A0FE	044	CLRF 4	
TANH	A0FF	*035*	*?NC XQ*	*e^2x*
TANH	A100	*068*	*->1A0D*	*[EXP13]*
TANH	A101	*089*	*?NC XQ*	
TANH	A102	*064*	*->1922*	*[STSCR]*
TANH	A103	*001*	*?NC XQ*	*e^2x+1*
TANH	A104	*060*	*->1800*	*[ADDONE]*
TANH	A105	*0A9*	*?NC XQ*	
TANH	A106	*064*	*->192A*	*[EXSCR]*
TANH	A107	*009*	*?NC XQ*	*e^2x-1*
TANH	A108	*060*	*->1802*	*[SUBONE]*
TANH	A109	*0D1*	*?NC XQ*	*e^2x+1*
TANH	A10A	*064*	*->1934*	*[RCSCR]*
TANH	A10B	*275*	*?NC XQ*	*[e^2x-1] / [e^2x+1]*
TANH	A10C	*060*	*->189D*	*[DV2-13]*
TANH	A10D	*331*	*?NC GO*	*Overflow, DropST, FillXL & Exit*
TANH	A10E	*002*	*->00CC*	*[NFRX]*

(*) Thanks to Thomas Klemm for pointing out a shorter way to calculate TANH using the scratch registers to store the exponential value.

Go ahead and set your machine in FIX 9, and bang the keyboard with esoteric inputs until you get blue in the face – do you find rounding errors somewhere?☺ - Hint: use www.wolframalpha.com as reference.

Putting it all together – Gamma for x>0 as a 13-digit function.

Just to go with a bang, here's an example that consolidates many of the tips & tricks seen before. We'll now write a routine to calculate the Gamma function of positive real numbers. We'll use the Lanczos approximation with 12-digit coefficients, as follows:

$$\Gamma(z) = \frac{\sum_{n=0..N} q_n z^n}{\prod_{n=0..N} (z+n)} (z+5.5)^{z+0.5} e^{-(z+5.5)}$$

q_0 =	75122.6331530
q_1 =	80916.6278952
q_2 =	36308.2951477
q_3 =	8687.24529705
q_4 =	1168.92649479
q_5 =	83.8676043424
q_6 =	2.5066282

Our function should give exact integer results for the natural numbers, since it's well-known that Γ(x) = (x-1)!. And here's the source code for such a feat:

Header	A2B2	*081*	*"A"*	
Header	A2B3	*00D*	*"M"*	*Gamma(x) by Lanczos*
Header	A2B4	*00D*	*"M"*	***x<71, and #-n***
Header	A2B5	*001*	*"A"*	
Header	A2B6	*007*	*"G"*	*Ángel Martin*
GAMMA	**A2B7**	**2CC**	**?FSET 13**	*Skip if running prgm*
GAMMA	A2B8	*027*	*JC +04*	
GAMMA	A2B9	*3B5*	*PORT DEP:*	***Displays "Running..." message***
GAMMA	A2BA	*08C*	*XQ*	
GAMMA	A2BB	*33B*	*->AF3B*	*[SUMMING]*
GAMMA	A2BC	*379*	*PORT DEP:*	***Calculates Gamma***
GAMMA	A2BD	*03C*	*XQ*	*needs argument in X*
GAMMA	A2BE	*2C1*	*->A2C1*	*[GAMMA]*
GAMMA	A2BF	*331*	*?NC GO*	*Overflow, DropST, FillXL & Exit*
GAMMA	A2C0	*002*	*->00CC*	*[NFRX]*
GAMMA	**A2C1**	**0F8**	**READ 3(X)**	
GAMMA	A2C2	*361*	*?NC XQ*	*(includes SETDEC)*
GAMMA	A2C3	*050*	*->14D8*	*[CHK_NO_S]*
GAMMA	A2C4	2EE	?C#0 ALL	*single-case zero*
GAMMA	A2C5	***05B***	***JNC +11d***	
GAMMA	A2C6	10E	A=C ALL	
GAMMA	A2C7	04E	C=0 ALL	
GAMMA	A2C8	2DC	PT= 13	***Build "-71" in C***
GAMMA	A2C9	250	LD@PT- 9	
GAMMA	A2CA	1D0	LD@PT- 7	
GAMMA	A2CB	050	LD@PT- 1	
GAMMA	A2CC	226	C=C+1 S&X	
GAMMA	A2CD	*01D*	*?NC XQ*	
GAMMA	A2CE	*060*	*->1807*	*[AD2_10]*
GAMMA	A2CF	2FE	?C#0 MS	*Set Carry if Negative*
GAMMA	A2D0	*289*	*?NC GO*	*"Out of Range"*
GAMMA	A2D1	*002*	*->00A2*	*[ERROF]*

The function uses a subroutine because gamma is also used in the Bessel functions code – implemented all in MCODE in the SandMath Module for integer and real orders, but as they say, that's another story…

We start by making sure we'll be within the 41 numeric range. Since we know that factorials for numbers greater than 69 will exceed it, we impose the restriction that x must be less than 71. Nevertheless there's also a final overflow check at the very end.

Then we take on the product loop as seen in example #9 before. Note the handy utilization of the scratch registers routines, as well as the "fake" upgrade of x to a 13-digit form for convenience sake.

GAMM1	A2D2	0F8	READ 3(X)	x
GAMM1	A2D3	10E	A=C ALL	A holds sign and S&X
GAMM1	A2D4	02E	B=0 ALL	clears B
GAMM1	A2D5	07A	A<>B M	B holds 13-digit mant
GAMM1	A2D6	089	?NC XQ	x
GAMM1	A2D7	064	->1922	[STSCR]
GAMM1	A2D8	0E0	SLCT Q	
GAMM1	A2D9	15C	PT= 6	loop counter
GAMM1	A2DA	0A9	?NC XQ	k
GAMM1	A2DB	064	->192A	[EXSCR]
GAMM1	A2DC	0A0	SLCT P	
GAMM1	A2DD	001	?NC XQ	x+k+1
GAMM1	A2DE	060	->1800	[ADDONE]
GAMM1	A2DF	0A9	?NC XQ	partial product
GAMM1	A2E0	064	->192A	[EXSCR]
GAMM1	A2E1	0D1	?NC XQ	x+k+1
GAMM1	A2E2	064	->1934	[RCSCR]
GAMM1	A2E3	149	?NC XQ	PP * (x+k+1)
GAMM1	A2E4	060	->1852	[MP2-13]
GAMM1	A2E5	0E0	SLCT Q	
GAMM1	A2E6	3D4	PT=PT-1	loop 6 times
GAMM1	A2E7	394	?PT= 0	
GAMM1	A2E8	393	JNC -14d	
GAMM1	A2E9	089	?NC XQ	PROD(x+k) |k=0,1..6
GAMM1	A2EA	064	->1922	[STSCR]

Note also that we save the partial result in the scratch registers for later use.

Next comes the long but simple polynomial term, written using Honer's method to take advantage of its efficiency. Also partially seen in example #10, here's where we use registers C&M for the coefficients, as second operands for the multiplication. This is done to use all digits up, for extended precision in the calculations.

At the end of this we'll divide the partial result by the result from the productory loop, using the scratch registers again.

GAMM2	A2EB	0F8	READ 3(X)	x
GAMM2	A2EC	10E	A=C ALL	
GAMM2	A2ED	04E	C=0 ALL	
GAMM2	A2EE	35C	PT=12	
GAMM2	A2EF	090	LD@PT-2	
GAMM2	A2F0	150	LD@PT-5	
GAMM2	A2F1	010	LD@PT-0	**2.50662827511**
GAMM2	A2F2	190	LD@PT-6	
GAMM2	A2F3	190	LD@PT-6	
GAMM2	A2F4	090	LD@PT-2	
GAMM2	A2F5	210	LD@PT-8	
GAMM2	A2F6	090	LD@PT-2	
GAMM2	A2F7	1D0	LD@PT-7	
GAMM2	A2F8	150	LD@PT-5	
GAMM2	A2F9	*135*	*?NC XQ*	*q6*x*
GAMM2	A2FA	*060*	*->184D*	*[MP2_10]*
GAMM2	A2FB	04E	C=0 ALL	
GAMM2	A2FC	226	C=C+1 S&X	*S&X in M*
GAMM2	A2FD	158	M=C ALL	
GAMM2	A2FE	04E	C=0 ALL	
GAMM2	A2FF	35C	PT=12	
GAMM2	A300	210	LD@PT-8	
GAMM2	A301	0D0	LD@PT-3	
GAMM2	A302	210	LD@PT-8	
GAMM2	A303	190	LD@PT-6	**83.8676043424**
GAMM2	A304	1D0	LD@PT-7	
GAMM2	A305	190	LD@PT-6	
GAMM2	A306	010	LD@PT-0	
GAMM2	A307	110	LD@PT-4	
GAMM2	A308	0D0	LD@PT-3	
GAMM2	A309	110	LD@PT-4	
GAMM2	A30A	090	LD@PT-2	
GAMM2	A30B	110	LD@PT-4	
GAMM2	A30C	*031*	*?NC XQ*	*q6*x+q5*
GAMM2	A30D	*060*	*->180C*	*[AD2-13]*
GAMM2	A30E	0F8	READ 3(X)	
GAMM2	A30F	*13D*	*?NC XQ*	*x*(q6*x+q5)*
GAMM2	A310	*060*	*->184F*	*[MP1_10]*
GAMM2	A311	04E	C=0 ALL	
GAMM2	A312	130	LDI S&X	*S&X in M*
GAMM2	A313	003	CON: 3	
GAMM2	A314	158	M=C ALL	
GAMM2	A315	04E	C=0 ALL	
GAMM2	A316	35C	PT=12	
GAMM2	A317	050	LD@PT-1	
GAMM2	A318	050	LD@PT-1	
GAMM2	A319	190	LD@PT-6	
GAMM2	A31A	210	LD@PT-8	
GAMM2	A31B	250	LD@PT-9	**1168.92649479**
GAMM2	A31C	090	LD@PT-2	
GAMM2	A31D	190	LD@PT-6	
GAMM2	A31E	110	LD@PT-4	
GAMM2	A31F	250	LD@PT-9	
GAMM2	A320	110	LD@PT-4	
GAMM2	A321	1D0	LD@PT-7	
GAMM2	A322	250	LD@PT-9	
GAMM2	A323	*031*	*?NC XQ*	
GAMM2	A324	*060*	*->180C*	*[AD2-13]*

GAMM2	A325	0F8	READ 3(X)	
GAMM2	A326	*13D*	*?NC XQ*	*x*(q4+x*(q6*x+q5))*
GAMM2	A327	*060*	*->184F*	*[MP1_10]*
GAMM2	A328	04E	C=0 ALL	
GAMM2	A329	130	LDI S&X	*S&X in M*
GAMM2	A32A	003	CON: 3	
GAMM2	A32B	158	M=C ALL	
GAMM2	A32C	04E	C=0 ALL	
GAMM2	A32D	35C	PT=12	
GAMM2	A32E	210	LD@PT- 8	
GAMM2	A32F	190	LD@PT- 6	
GAMM2	A330	210	LD@PT- 8	
GAMM2	A331	1D0	LD@PT- 7	
GAMM2	A332	090	LD@PT- 2	***8687.24529705***
GAMM2	A333	110	LD@PT- 4	
GAMM2	A334	150	LD@PT- 5	
GAMM2	A335	090	LD@PT- 2	
GAMM2	A336	250	LD@PT- 9	
GAMM2	A337	1D0	LD@PT- 7	
GAMM2	A338	010	LD@PT- 0	
GAMM2	A339	150	LD@PT- 5	
GAMM2	A33A	*031*	*?NC XQ*	*q3+x*(q4+x*(q6*x+q5))*
GAMM2	A33B	*060*	*->180C*	*[AD2-13]*
GAMM2	A33C	0F8	READ 3(X)	
GAMM2	A33D	*13D*	*?NC XQ*	*x*(q3+x*(q4+x*(q6*x+q5)))*
GAMM2	A33E	*060*	*->184F*	*[MP1_10]*
GAMM2	A33F	04E	C=0 ALL	
GAMM2	A340	130	LDI S&X	*S&X in M*
GAMM2	A341	004	CON: 4	
GAMM2	A342	158	M=C ALL	
GAMM2	A343	04E	C=0 ALL	
GAMM2	A344	35C	PT=12	*build q2 in C*
GAMM2	A345	0D0	LD@PT- 3	
GAMM2	A346	190	LD@PT- 6	
GAMM2	A347	0D0	LD@PT- 3	
GAMM2	A348	010	LD@PT- 0	
GAMM2	A349	210	LD@PT- 8	***36308.2951477***
GAMM2	A34A	090	LD@PT- 2	
GAMM2	A34B	250	LD@PT- 9	
GAMM2	A34C	150	LD@PT- 5	
GAMM2	A34D	050	LD@PT- 1	
GAMM2	A34E	110	LD@PT- 4	
GAMM2	A34F	1D0	LD@PT- 7	
GAMM2	A350	1D0	LD@PT- 7	
GAMM2	A351	*031*	*?NC XQ*	
GAMM2	A352	*060*	*->180C*	*[AD2-13]*
GAMM2	A353	0F8	READ 3(X)	
GAMM2	A354	*13D*	*?NC XQ*	
GAMM2	A355	*060*	*->184F*	*[MP1_10]*

Yes this is long, but stick around a little longer, the best is yet to come…
(after all programming is not always such an exciting ride).

GAMM2	A356	04E	C=0 ALL	
GAMM2	A357	130	LDI S&X	*S&X in M*
GAMM2	A358	004	CON: 4	
GAMM2	A359	158	M=C ALL	
GAMM2	A35A	04E	C=0 ALL	
GAMM2	A35B	35C	PT=12	*builds q1 in C*
GAMM2	A35C	210	LD@PT- 8	
GAMM2	A35D	010	LD@PT- 0	
GAMM2	A35E	250	LD@PT- 9	
GAMM2	A35F	050	LD@PT- 1	
GAMM2	A360	190	LD@PT- 6	***80916,6278952***
GAMM2	A361	190	LD@PT- 6	
GAMM2	A362	090	LD@PT- 2	
GAMM2	A363	1D0	LD@PT- 7	
GAMM2	A364	210	LD@PT- 8	
GAMM2	A365	250	LD@PT- 9	
GAMM2	A366	150	LD@PT- 5	
GAMM2	A367	090	LD@PT- 2	
GAMM2	A368	*031*	*?NC XQ*	
GAMM2	A369	*060*	*->180C*	*[AD2-13]*
GAMM2	A36A	0F8	READ 3(X)	
GAMM2	A36B	*13D*	*?NC XQ*	
GAMM2	A36C	*060*	*->184F*	*[MP1_10]*
GAMM2	A36D	04E	C=0 ALL	
GAMM2	A36E	130	LDI S&X	*S&X in M*
GAMM2	A36F	004	CON: 4	
GAMM2	A370	158	M=C ALL	
GAMM2	A371	04E	C=0 ALL	
GAMM2	A372	35C	PT=12	*builds q0 in C*
GAMM2	A373	1D0	LD@PT- 7	
GAMM2	A374	150	LD@PT- 5	
GAMM2	A375	050	LD@PT- 1	
GAMM2	A376	090	LD@PT- 2	
GAMM2	A377	090	LD@PT- 2	***75122,633153***
GAMM2	A378	190	LD@PT- 6	
GAMM2	A379	0D0	LD@PT- 3	
GAMM2	A37A	0D0	LD@PT- 3	
GAMM2	A37B	050	LD@PT- 1	
GAMM2	A37C	150	LD@PT- 5	
GAMM2	A37D	0D0	LD@PT- 3	
GAMM2	A37E	*031*	*?NC XQ*	***SUM(qi*x^i)***
GAMM2	A37F	*060*	*->180C*	*[AD2-13]*
GAMM3	**A380**	*0D1*	*?NC XQ*	***PROD(x+k) |k=0,1..6***
GAMM3	A381	*064*	*->1934*	*[RCSCR]*
GAMM3	A382	*275*	*?NC XQ*	
GAMM3	A383	*060*	*->189D*	*[DV2-13]*
GAMM3	A384	*0AE*	*A<>C ALL*	*Store 13-digit form:*
GAMM3	A385	*128*	*WRIT 4(L)*	*L holds sign and S&X*
GAMM3	A386	*0EE*	*C<>B ALL*	*N holds 13-digit mant*
GAMM3	A387	*070*	*N=C ALL*	***SUM / PROD***

Lastly it comes the more involved phase, where we've used registers N and 4(L) as temporary scratch as well, since Q & "+" are already taken. Note that we're using the exponential form of the power expression for simplicity:

(x+5,5)^(x+0.5)* exp[-(x+5.5)] = exp[(x+0,5)*Ln(x+5.5) – (x+5.5)]

GAMM4	A388	0F8	READ 3(X)	
GAMM4	A389	10E	A=C ALL	x
GAMM4	A38A	04E	C=0 ALL	
GAMM4	A38B	19C	PT=11	*0,5*
GAMM4	A38C	150	LD@PT- 5	
GAMM4	A38D	*01D*	*?NC XQ*	
GAMM4	A38E	*060*	*->1807*	*[AD2_10]*
GAMM4	A38F	*089*	*?NC XQ*	*x+0.5*
GAMM4	A390	*064*	*->1922*	*[STSCR]*
GAMM4	A391	04E	C=0 ALL	
GAMM4	A392	35C	PT=12	*5*
GAMM4	A393	150	LD@PT- 5	
GAMM4	A394	*025*	*?NC XQ*	*x+5.5*
GAMM4	A395	*060*	*->1809*	*[AD1_10]*
GAMM4	A396	3C4	ST=0	
GAMM4	A397	*121*	*?NC XQ*	*Ln(x+5.5)*
GAMM4	A398	*06C*	*->1B48*	*[LN13]*
GAMM4	A399	*0D1*	*?NC XQ*	
GAMM4	A39A	*064*	*->1934*	*[RCSCR]*
GAMM4	A39B	*149*	*?NC XQ*	*(x+0.5)*Ln(x+5.5)*
GAMM4	A39C	*060*	*->1852*	*[MP2-13]*
GAMM4	A39D	*0A9*	*?NC XQ*	*x+0.5*
GAMM4	A39E	*064*	*->192A*	*[EXSCR]*
GAMM4	A39F	04E	C=0 ALL	
GAMM4	A3A0	35C	PT=12	*5*
GAMM4	A3A1	150	LD@PT- 5	
GAMM4	A3A2	*025*	*?NC XQ*	*x+5.5*
GAMM4	A3A3	*060*	*->1809*	*[AD1_10]*
GAMM4	A3A4	2BE	C=-C-1 MS	
GAMM4	A3A5	11E	A=C MS	*-(x+5.5)*
GAMM4	A3A6	*0D1*	*?NC XQ*	*(x+0.5)*Ln(x+5.5)*
GAMM4	A3A7	*064*	*->1934*	*[RCSCR]*
GAMM4	A3A8	*031*	*?NC XQ*	
GAMM4	A3A9	*060*	*->180C*	*[AD2-13]*
GAMM4	A3AA	044	CLRF 4	
GAMM4	A3AB	*035*	*?NC XQ*	
GAMM4	A3AC	*068*	*->1A0D*	*[EXP13]*
GAMM5	A3AD	*138*	*READ 4(L)*	*restore 13-digit partial result*
GAMM5	A3AE	*158*	*M=C ALL*	*M has sign and exp*
GAMM5	A3AF	*0B0*	*C=N ALL*	*C has 13 digit mantissa*
GAMM5	A3B0	*149*	***?NC GO***	
GAMM5	A3B1	*062*	*->1852*	*[MP2-13]*

That's all folks; hope this journey through the archeological SW department has been interesting to you all.

THE END.