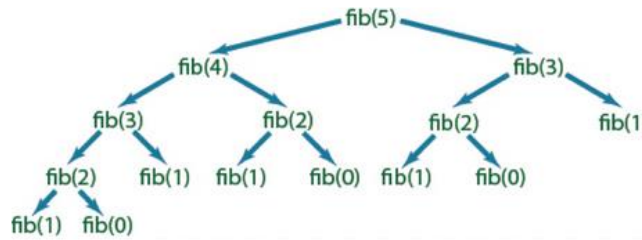
Recursion and Modular Math
HP-41 Module



Introduction and Credits.

This HP-41 module provides a short collection of functions and routines about Recursion, Modular Math, and Numeric arithmetic field. The recursion section is based on T.W. van der Berg's seminal article published in the PPCCJ – reproduced later in this manual. I added a few more examples using his method, optimized the code for a ROM layout and converted some of the housekeeping routines to MCODE for faster operation.

The first thing to say is that the HP-41 reduced memory and limited programming capabilities are not well-suited for recursive code, and let's not talk about its ultra-slow coconut CPU chip. Certainly, the SY-41CL and DM-41X make the subject more palatable these days, as well as computer simulators like V41 from Warren Furlow. Yet, it could be argued that other methods less demanding on resources are better on the HP-41 platform – and you'd be right, but the whole point of this module was to investigate the possibilities and document the results, even if they're not earth-shattering. In fact, in its "NUMERICAL" section, the module also includes alternative MCODE functions for most of the recursive routines which, when compare to the recursive code, run circles around them and then some more.

The "MODULAR MATH" section comprises a handful set of functions on the elusive and intriguing subject of modulus math. They are contributed by Greg McClure and Jean-marc Baillard.

Finally, a couple of other programs deal with the calculation of decimal digits of pi and e. In particular the MCODE function **MDOP** written by Peter Platzer, is a remarkable implementation even if it requires Q-RAM to hold the results, so dust off your HEPAX RAM for the task.

Without further ado, here is a list of the functions in the Main FAT table.

XROM#	Function	Description	Author
09.00	**-RECURSION**	Section Header	*n/a*
09.01	**Σ1/N**	Harmonic Numbers	*Ángel Martin*
09.02	**ΣN^X**	Generic Faulhaber's	*Ángel Martin*
09.03	**$B2**	Begin procedure- 2D	*Ángel Martin*
09.04	**$B3**	Begin procedure - 3D	*Ángel Martin*
09.05	**$E2**	End procedure - 2D	*Ángel Martin*
09.06	**$E3**	End procedure - 3D	*Ángel Martin*
09.07	**"$I2**	Initialize pointers - 2D	*Ángel Martin*
09.08	**"$I3**	Initialize pointers - 3D	*Ángel Martin*
09.09	**"ACKER**	Ackermann Function	*T.W. van der Berg*
09.10	**"#ACK**	Procedure Subroutine	*T.W. van der Berg*
09.11	**"CATN**	Catalan Numbers	*Ángel Martin*
09.12	**"#CT**	Procedure Subroutine	*Ángel Martin*
09.13	**"FACT**	Factorial	*T.W. van der Berg*
09.14	**"#FCT**	Procedure Subroutine	*T.W. van der Berg*
09.15	**"FIBO**	Fibonacci Numbers	*T.W. van der Berg*

09.16	**"#FIB**	Procedure Subroutine	*T.W. van der Berg*
09.17	**"HANOI**	Hanoi Towers	*T.W. van der Berg*
09.18	**"#DSC**	Disc subroutine	*T.W. van der Berg*
09.19	**"#TWR**	Tower subroutine	*T.W. van der Berg*
09.20	**"HARM**	Harmonic Numbers	*Ángel Martin*
09.21	**"#HM**	Procedure Subroutine	*Ángel Martin*
09.22	**"STIR**	Stirling Numbers	*Ángel Martin*
09.23	**"COMB**	Binomial Coefficients	*Ángel Martin*
09.24	**"#ST**	Procedure Subroutine	*Ángel Martin*
09.25	**"PART**	Partitions	*T.W. van der Berg*
09.26	**"#Q**	Procedure Subroutine	*T.W. van der Berg*
09.27	**CLRTN**	Clear RTN stack	*Ángel Martin*
09.28	**RTNE?**	Is RTN Stack Empty? (no levels)	*Doug Wilder*
09.29	**RTNF?**	Is RTN Stack Full? (six levels used)	*Ángel Martin*
09.30	**RTNS**	Get # of used RTN levels	*Ángel Martin*
09.31	**XQ>GO**	Drop last RTN level	*Hakan Thörgren*
09.32	**MANYDIGOFPI**	*Section Header*	*n/a*
09.33	**ΣDGT**	Sum of mantissa digits	*Ángel Martin*
09.34	**MREV**	Mantissa Digit Reversal	*Ángel Martin*
09.35	**MDOP _ _ _"_**	Many Digits of Pi	*Peter Platzer*
09.36	**"PI000"**	pi to 1000 Decimal Places	*Ron Knapp*
09.37	**"E2900"**	Compute e to 2900 Places	*Ron Knapp*
09.38	**SKIP**	Skips one program line	*Erik Blake*
09.39	**-NUMERICAL**	*Section Header*	*n/a*
09.40	**APERY**	Apery Numbers	*Jean-Marc Baillard*
09.41	**BELL**	Bell Numbers	*Ángel Martin*
09.42	**BN2**	Bernoulli Numbers	*Ángel Martin*
09.43	**FIB**	Fibonacci Numbers	*Ángel Martin*
09.44	**FIBI**	Inverse Fibonacci	*Ángel Martin*
09.45	**MLN**	Multinomial Coefficient	*Jean-Marc Baillard*
09.46	**ΣFIB**	Sum of Fibonacci numbers	*Ángel Martin*
09.47	**ΣIFIB**	Sum of Inverse Fibonacci numbers	*Ángel Martin*
09.48	**PHI**	Golden Ratio ~1.61803398875	*Ángel Martin*
09.49	**BINETN**	Binet Formula for Integers	*Ángel Martin*
09.50	**BINETX**	Binet Formula for Real arguments	*Ángel Martin*
09.51	**-MODULARMTH**	*Section Header*	*n/a*
09.52	**1/M**	Inverse Modulus	*Jean-Marc Baillard*
09.53	**CONG**	Congruence Equation	*McClure-Martin*
09.54	**GCD**	Greatest Common Denominator	*Ángel Martin*
09.55	**LCM**	Least Minimum Multiple	*Ángel Martin*
09.56	**M+**	Modulus Addition	*Greg McClure*
09.57	**M-**	Modulus Subtraction	*Greg McClure*
09.58	**M***	Modulus product	*Greg McClure*
09.59	**M^**	Modulus power	*Greg McClure*
09.60	**M^2**	Modulus Square power	*Greg McClure*
09.61	**SQRTM**	Modulus Square Root	*Jean-Marc Baillard*
09.62	**UV**	Auxiliary routine for 1/M	*Jean-Marc Baillard*
09.63	**“ROUT**	Reads E2900 Results	*Ron Knapp*

Miscellaneous Number Functions

The module includes a few short functions useful for numerical analysis, cryptography, and games.

Function	Description	Input	Output
ΣDGT	Sum of mantissa digits	Value in X	Sum in X, x in LastX
MREV	Mantissa digit reversal	Value in X	Result in X, x in LastX
PHI	Golden Ratio	n/a	φ in X, stack lifted
BINETN	Binet formula for integers	n in X	f(n)
BINETX	Binet formula for real values	x in X	f(x)

They're described below.

- **MREV** performs a mantissa digit reversal of the value in X. The result is placed in X and the original number is saved in LastX.

  Example: reverse the mantissa digits of pi
  PI, MREV => 4.562951413 (in FIX 9)

- **ΣDGT** sums all the mantissa digits of the value in X. The result is placed in X and the original number is saved in LastX.

  Example: sum the mantissa digits of pi:
  PI, ΣDGT => 40.00000000

  Example: The short routine below calculates the **digital root** of the number in X, simply using **ΣDGT** repeated times until its result is a single-digit integer (i.e. less than 10).

```
01  LBL "DGRT"        05  X>Y?
02  9                 06  GTO 00
03  *LBL 00           07  END
04  ΣDGT
```

- **PHI** lifts the stack and places the golden ration in X, φ =~ 1.618033989
  Note: this function is used as a 13-digit subroutine in the calculation of Fibonacci numbers with the Binet formula.

- **BINETN** implements the well-known Binet formula for integer input values. The result is the n-th Fibonacci number obtained directly without any iterations.

  $$F_n = \frac{\varphi^n - (-\varphi)^{-n}}{\sqrt{5}}$$

  Example: Calculate f(9)

  9, XEQ "BINETN" => 34.00000000

- **BINETX** implements an extension for non-integer real input values to calculate the interpolated Fibonacci numbers. This provides an easy expression for the determination that guarantees real values also for the interpolated Fibonacci numbers:

$$f_x^* = \frac{\varphi^x - \cos(\pi x)\varphi^{-x}}{\sqrt{5}}$$

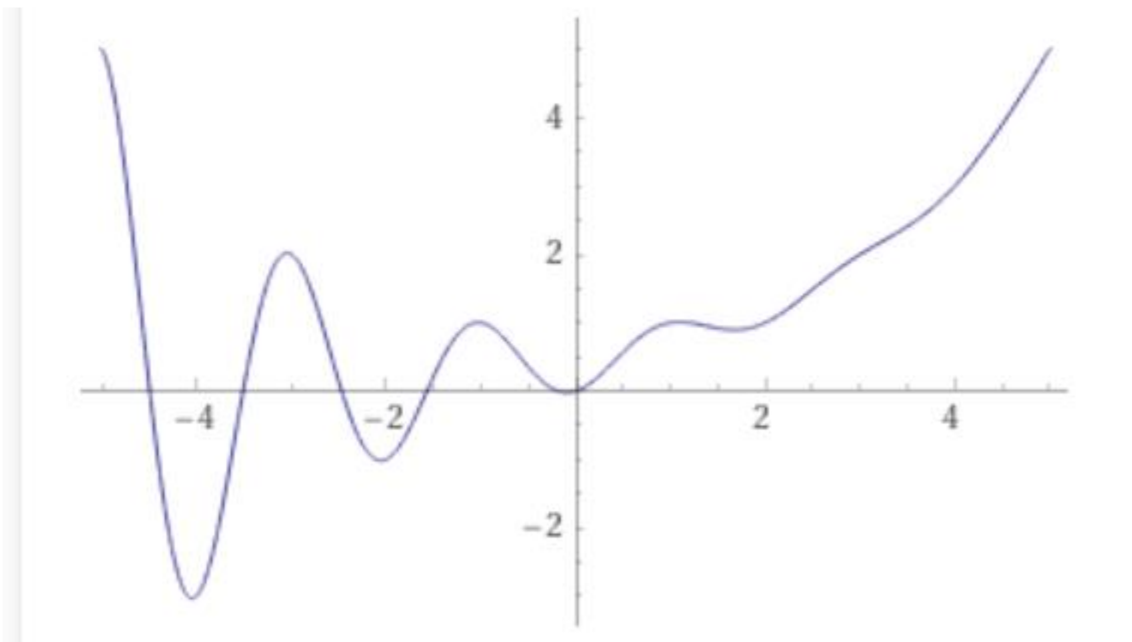
Example: Calculate f(π)

PI, XEQ "BINETX" => 0.043896342

In fact, this modified formula produces the real parts of the complex results obtained applying Binet's formula directly with complex arguments – where the term -φ^-n clearly yields a result in the complex domain: (-φ)^(-n) = exp(-n . ln (-φ))

Note: You can refer to the 41Z Module manual for the complex case, implemented in that module with the function **ZFIB**.

See below the graphical representation of [Binet(x) for arguments between [-5 . 5]]



Obviously, the values for integer arguments coincide with the natural Fibonacci number, since the term cos(πn) is equal to +/- one.

Number Theory Functions

A set of numerical constants and series is also available in the module, some of them as a faster alternative of the recursive routines to showcase the MCODE advantage.

Function	Description	Input	Output
APERY	Apery Numbers	Index n in C	n-th. Apery number
BELL	Bell Numbers	Index n in X	n-th. Bell number
BN2	Bernoulli Numbers	Index n in X	n-th. Bernoulli number
FIB	Fibonacci Numbers	Index n in X	n-th. Fibonacci number
FIBI	Inverse Fibonacci	Index n in X	n-th/ inverse Fibonacci
MLN	Multinomial Coefficient	n in Y, k in X	C(n,k)
ΣFIB	Sum of Fibonacci	Range n in X	Sum[fib(n)]
ΣIFIB	Sum of Inverse Fibonacci	Range n in X	Sum[1/fib(n)]
Σ1/N	Harmonic Number	n in X	Result in X, n in LastX
ΣX^N	Faulhaber formula	n in Y, x in X	Result in X, x in LastX

- **Σ1/N** calculates the Harmonic number of the argument in X, that is the sum of the reciprocals of the natural numbers (which excludes zero) lower and equal to n. It will be used in the calculation of the Kelvin functions and the Bessel functions of the second kind, K(n,x) and Y(n,x).

$$H_n = \sum_{k=1}^{n} \frac{1}{k}.$$

Example: calculate H(5) and H(25).

5, XEQ "Σ1/N" => 2.283333333
25, XEQ "Σ1/N" => 3.815958178

- **ΣN^X** Calculates a generalized value of the Faulhaber's formula for integer values of x. – The few first integer values of x have explicit formulas for the result, but that's not the case for a general value - which can also be non-integer. Obviously for x=-1 this function returns identical results than **Σ1/N**, albeit slower due to the additional complexity of the definition of the term.

Example: Check the triangular (x=1) and pyramidal (x=2) formulas for n=10 – which are particular cases of the Faulhaber's Formula, involving Binomial coefficients and Bernoulli's numbers. See the link below for details: http://en.wikipedia.org/wiki/Faulhaber%27s_formula

10, ENTER^, 1, XEQ "**ΣN^X**" => 55.00000000
10, ENTER^, 2, XEQ "**ΣN^X**" => 385.0000000

$$T_n = \sum_{k=1}^{n} k = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2} = \binom{n+1}{2}$$

$$P_n = \sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6} = \frac{2n^3 + 3n^2 + n}{6}.$$

Apéry Numbers. { **APERY** } (See JM Baillard's *reference page*.)

Named after the French mathematising Roger Apéry, (University of Caen in Normandy), these numbers are defined by:

$$A(n) = \sum_{k=0}^{n} \binom{n}{k}^2 \binom{n+k}{k}^2$$

The first few are: 1 5 73 1445 33001 .... [see Sloane's A005259]
These numbers may also be computed by the formula An = 4F3 (-n , -n , n+1 , n+1 ; 1 , 1 , 1 ; 1) where ${}_4F_3$ is a generalized hypergeometric function.

These numbers grow very quickly so the MCODE function presents the result in ALPHA to allow for exponents larger than 99. The Mantissa is left in X and the exponent in Y. If the function is part of a running program no ALPHA output will be shown.

Examples:

41, XEQ "APERY" => 4.944386782 E59
100, XEQ "APERY" => 2.824655679 E149
329, XEQ "APERY" => 1.990511251 E499

Note: In 1979, Apéry proved that zeta(3) is irrational. Since then, $\zeta(3)$ is called Apéry's Constant. It has an approximate value of: 1.202056903159594285399

Bell Numbers. { **BELL** } (See jm Baillard;s *reference page*)

In combinatorial mathematics, the Bell numbers count the possible partitions of a set, i.e. the Bell number Bn counts the number of different ways to partition a set that has exactly n elements.

Bell numbers are defined by the iterative sequence below:

B(0) = 1 and
B(n+1) = Σ{k=0..n} Cn,k B(k) if n > 1

where Cnk = n! / [k!(n-k)!] are the binomial coefficients.

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k.$$

Examples:

10, XEQ "BELL" => 115,975.0000
89, XEQ "BELL" => 5.225728472 E99

Bernoulli Numbers { **BN2** } (see JM Baillard *reference page*)

The Bernoulli numbers could be computed by the relations:

B(0) = 1 ;
B(0) + Cn+1,1 B(1) + Cn+1,2 B(2) + ... + Cn+1,n B(n) = 0

where Cnk = n! / [k!(n-k)!] are the binomial coefficients

If the convention B1=−1/2 is used, this sequence is also known as the first Bernoulli numbers; with the convention B1=+1/2 is known as the second Bernoulli numbers. Except for this one difference, the first and second Bernoulli numbers agree. Since Bn=0 for all odd n>1, and many formulas only involve even-index Bernoulli numbers, some authors write Bn instead of B2n.

Example:

10, XEQ "BN2" => B(10) = -0.075757576

Note however that this recurrence relation is unstable, and the results are quite inaccurate for large n. The generating function below is often used to avoid that:

$$\frac{t}{e^t - 1} = \frac{t}{2}\left(\coth\frac{t}{2} - 1\right) \qquad = \sum_{m=0}^{\infty} \frac{B_m^- t^m}{m!}$$

Multinomial Coefficients. { **MLN** } (See JM Baillard's *reference page*.)

Multinomial coefficients are an extension of the Binomial coefficient, using multiple indexes instead of two. For example, if "k" is the number of variables we have:

P = (n1 , n2 , ... nk) ! = n ! / (n1! n2! .... nk!) ; where n = n1 + n2 + ... + nk

$$\binom{n}{k_1, k_2, \ldots, k_{r-1}} = \frac{n!}{k_1!k_2!\ldots k_{r-1}!k_r!}$$

The function **MLN** expects the input values stored in data registers starting in R01, The number of variables "k" is entered in the stack' X-register.

Example: Calculate (76 , 107 , 112 , 184) !

16 STO 01 24 STO 02 41 STO 03 48 STO 04
4 XEQ "MLN" => P = 9.22755891 9 E69

Fibonacci Numbers { FIB , FIBI }

These functions calculate the Fibonacci and the Fibonacci Inverse numbers using the well-known recurrent relationship:

f(0) = 0 ,
f(1) = 1 ;
f(n) = f(n-2) +f(n-1)

And the "Fibonacci Inverse" defined as

f'(0) = 0
f'(1) = 1
f'(n) = 1/f('n-2) + 1/f'(n-1).

Note that this is **not* the same as the inverse of Fibonacci*, which would simply be 1/F(n)

Examples:

10, XEQ "FIB" => 55.00000000 ; LASTX, XEQ : FIBI" => 0.683299104
25, XEQ "FIB" => 75,025.00000 ; LASTX, XEQ "FIBI" => 0.707165965

Sum of Fibonacci numbers { ΣFIB , ΣFIBI }

Here we're calculating the sum of the first n Fibonacci numbers, starting at f(0)=0 until f(n).

An interesting fact is the sum of the first Fibonacci numbers with odd index up to f(2n−1) is the 2n-th. Fibonacci number, and the sum of the first Fibonacci numbers with even index up to f(2n) is the (2n+1)-th. Fibonacci number minus 1:

Moreover, the general expression below relates the sum to the sequence value:

Σ{0..n)F(n) = f(n+2)-1

Example:

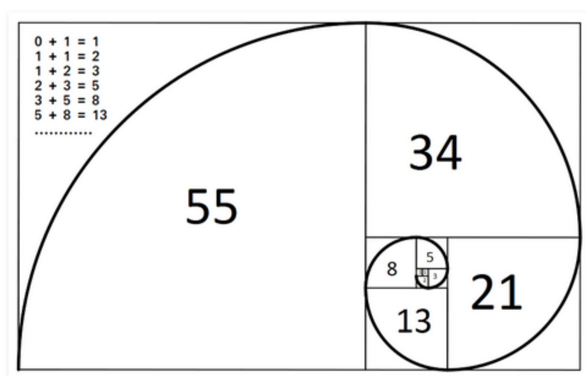
15, XEQ "ΣFIB" => 1,596.000000

Verifying the formula above:

17, XEQ "FIB" => 1,597.000000

Example:

15, XEQ "ΣFIBI" => 3.357233149



Recursion on the HP-41 - by T.W. van der Berg (10079)

From PPC Calculator Journal V11N9 - Nov/Dec 1984

Here is my answer to your latest request for more input from the members: an article about recursion in the HP41. In a few examples I will make the method to implement recursive algorithms on the HP41 clearer (there is an elegant way to do it). The examples in this article are:

- Factorials
- Fibonacci
- Ackermann function
- Towers of Hanoi
- Partitions
- Prefix, Infix and Postfix rotations
- Harmonic Numbers
- Stirling Numbers
- Binomial Coefficient
- Catalan Numbers

In the future I hope to send more programs. - Happy recursive programming!

Recursion on the HP-41

A well-known example of recursion is the function for n factorial, n!

$$n! = \begin{cases} n*(n-1)! & \text{If } n>1 \\ 1 & \text{if } n=1 \end{cases}$$

Or written as an algorithm:

Algorithm 1: Factorial

```
Function FAC(n:natural) :natural;
Begin
        If n=1   Then FACT:= 1
                  else  FACT:= n*FAC(n-1)
End
```

The job consists on translating this algorithm into HP41 language. The main problems are:

- How to implement local variables (each call of FAC creates a new variable n);
- How to handle the large amount of return addresses (the HP41 can only handle 6 levels of subroutine calls).

A solution for the latter to problems is a memory stack. Each time you call FAC the current value of n is pushed onto the stack. The return address is also pushed onto the stack.

Translation of algorithm 1:

1) Define a stack (in the data register area of the calculator)
2) Define two stack pointers (**sp1**, **sp2**). Sp1 points to n; sp2 points to the return address. (It is much easier to work with two stack pointers)
3) Translate all statements in algorithm 1.

This will require two FOCAL programs, a CALLING program and a CALLED subroutine. The calling program initializes the stack pointers and repeatedly calls the procedure subroutine until the algorithm reaches the final boundary condition. *The called subroutine should not be executed by itself*, since it lacks the proper initialization pointer and stack definitions.

We'll start with the called subroutine, which implements the algorithm using HP41 instructions. You can NOT run this program. You always need another program to call the subroutine and to initialize the stack pointers **sp1** and **sp2**. Below are both #FCT (called program) and the calling program for LBL "#FCT"

Called Routine	**Comment**
LBL "#FCT"	function FAC
ISG 01	begin
NOP	
ISG 02	(
NOP	sp1:=sp1*2
STO IND 01	sp2;=sp2+2
CLA	sp1^:=n
X<>M	sp2:=RTN
X<> a	)
X<> M	
ASTO IND 02	
1	
X=Y?	if n=1 then FAC:=1
GTO "END"	
-	
XEQ "#FCT"	Else:
RCL IND 01	FAC:=FAC(n-1)*n
*	
LBL "END"	end
CLA	
ARCL IND 02	
X<> M	RTN:=sp2^
X<> a	sp2:=sp2-2
X<> M	sp1:=sp1-2
DSE 02	)
NOP	
DSE 01	
NOP	
RTN	

Driver Program	**Comment**
LBL "FACT"	Calling program
10.00002	initialize stack pointers
STO 01	the stack starts at R10
11,00002	
STO 02	
DSE 01	sp1:=8.00002
NOP	
DSE 02	sp2:=9.00002
NOP	
"ENTER N"	
PROMPT	
XEQ "#FCT"	
END	

Instructions:	Result:	Example:		Display
1) XEQ "FACT"	ENTER N	XEQ "FACT"	=>	ENTER N
2) n, R/S	n!	5, R/S	=>	120

The essential point of #FCT is the implementation of the statements 'begin' and 'end'. Note the symmetry: "begin" = "end" ^-1 (i.e LBL "END" is the reverse sequence of steps defined in the "begin' section of the code inscribed in the rectangle, therefore it basically undoes it in preparation for the next iteration - execution of the #FCT function).

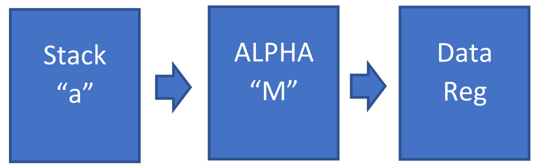
As you see #FCT *keeps calling itself while the argument is greater than 1*. Each iteration u ses two data registers to save the pointers, therefore this scheme produces the following arrangement:

R10	n1	sp1=10	R14	n3	sp1=14
R11	RTN1	sp2=11	R15	RTN3	sp2=15
R12	n2	sp1=12	R16	…	sp1=
R13	RTN2	sp2=13	R16	…	sp2=

A few comments are in order.

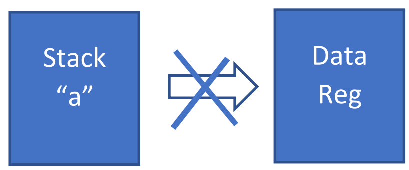
Note that **sp1** is a BCD number (the current value of n) but **sp2** is a binary number since the return addresses are in hex. This is handled by the ASTO/ARCL instructions storing and recalling the contents of the "a" register holding the upper half of the HP-41 RTN stack. Register M is just an intermediate location needed for the transaction in-between "a" and the data registers holding all its different iterations.

CLA	clears M
X<> M	stores n in M, clears X
X<> a	clears "a", saves RTN in X
X<> M	moves RTN to M, brings n back to X
ASTO IND 02	saves RTN in data register



All this dancing around is required to avoid the data normalization that occurs when using STO/RCL instructions, which you'd normally tend to utilize when having to copy data between register "a" and the data registers:

RCL a
STO IND 02



Which would've been more intuitive – but alas, we need to use ALPHA data instead, which means using the M register and ASTO instead of STO. (We could have used non-normalizing functions such as PEEKR/POKER but that's another story entirely and would have required a capable additional module).

Remarkably, only the "a' register needs to be backed-up since it's the one getting overflowed (capacity exceeded). The last three return addresses held in register "b" will be managed by the O/S itself, pushed into the "a" register as the number of subroutine levels increases. Another subtle effect of the method is that only one subroutine level is ever used in the "ä" register, which is backed-up at every iteration so there's no time for it to accumulate multiple levels (up to three). This has a small drawback though, because the backup also happens even when there's no data in "ä" – that is three data registers are used without a real strict need for it. Indeed, a small price to pay for the sake of an scalable algorithm.

It's worth noting that each iteration is adding one more RTN address to the stack (all pointing at the same program step after the XEQ "#FCT" instruction!), and that when finally when the term n=1 is reached, all the additions will be run sequentially, decreasing the RTN stack one at a time and executing the ending part of the program (LBL END") the same number of times.

Note that the RECURSE module implements this approach in an optimized way. First, the **sp1** pointer is held in N(6) and **sp2** is held in M(5), freeing so R01 and R02, and the memory stack starts in data register R01 instead of R10. Secondly, it includes dedicated MCODE functions for the begin: and end: procedures, including two subroutines for the backup and restore of the RTN stack – replacing the X<> steps in the FOCAL counterpart. Thus, [LVUP] copies the "a" register in the indirect location pointed at by M (moves it one level up), and [LVDN] recalls it from said location into "a" (moves it one level down).

There are two sets of MCODE functions for the begin:/end: housekeeping. The first set **$BEG2** / **$END2** used for routines that need two stack pointers (such as **FACT**), and **$BEG3** / **$END3** for those routines that employ two indexes and therefore need three stack pointers – more about this later.

A few more examples not part of the original article are included in the RECURSE module:

Algorithm 1.2: Harmonic numbers.

Definition of Harmonic numbers as a recurrent expression:

H(0)= 0
H(n) = H(n-1)+1/n   n>1

XEQ "HARM"   =>   N=?
10, R/S   =>`   2.928968254

Algorithm 1.3: Catalan numbers

Named after the Belgian mathematician Eugene Charles Catalan, they're defined as:

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!\,n!} = \prod_{k=2}^{n} \frac{n+k}{k} \qquad \text{for } n \geq 0.$$

And they satisfy the following recurrence relation implemented here:

Cn+1 = Cn (4n-2)/(n+1) , n>1

XEQ "CATN"   =>   N=?
7, R/S   =>   429.0000000

Algorithm 1.4: Stirling numbers of the 1st kind

Stirling numbers of the first kind S(n, k) are defined by the following recurrence relation:

S(n, 0) = 0 ;
S(n, k) = S(n-1, k-1) - (n-1) S(n-1, k) ,   1 <= k <= n

XEQ "STIR"   =>   N^K=?
6, ENTER^, 3, R/S   =>   - 115.0000000
12, ENTER^, 7, R/S   =>   -2,060,416.000

Algorithm 1.5: Binomial Coefficient

Commonly, a binomial coefficient is indexed by a pair of integers n ≥ k ≥ 0. It is the coefficient of the x^k term in the polynomial expansion of the binomial power (1 + x)^n, and is given by the formula:

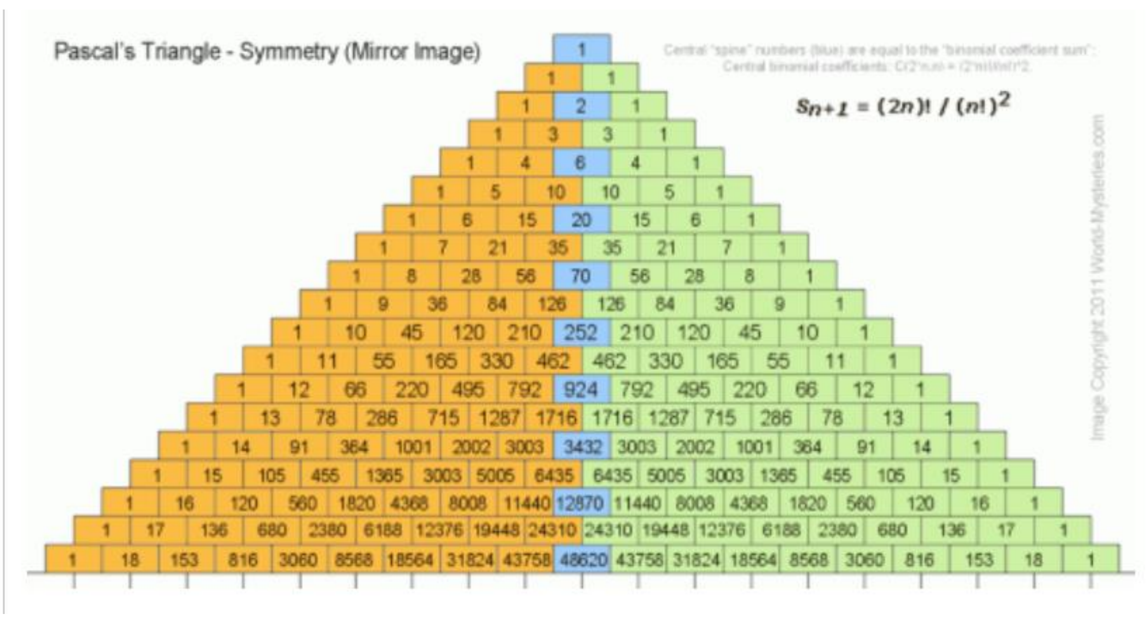
$$\binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

Arranging the numbers in successive rows for n = 0 , 1 , 2 ,… gives a triangular array called Pascal's triangle, satisfying the recurrence relation used in this implementation:

C(n,k) = C(n-1,k-1) + C(n-1, k)

XEQ "COMB"	=>	N^K=?	
6, ENTER^, 3, R/S	=>	20.00000000	
R/S	=>	N^K=?	
13, ENTER^, 5, R/S	=>	1,287.000000	(takes a *__*very* long time__*)

Note: obviously speed is not this method's forte. If you need a more practical solution you're encouraged to check the implementation of the Combinations function available in the SandMath and in a few other math modules.



Algorithm 2: Fibonacci numbers.

Definition of Fibonacci numbers:

F(0) = 0
F(1) = 1
F(n) = F(n-1) + F(n-2) ; if n>1

```
Function FIB(n:natural) :natural ;
Begin
        If n<= 1        then FIB:=n
                        Else  FIB:=FIB(n-1)+FIC(n-2)
End.
```

Which has been implemented in the corresponding pair of FOCAL programs shown below.

LBL "#FIB"	function FIB
ISG 01	begin procedure
NOP	
ISG 02	
NOP	
STO IND 01	
CLA	
X<> M	
X<> a	
ASTO IND 02	
1	if n<=1 then FIB:=n
X<>Y	
X<>Y?	
GTO 01	
-	
CHS	else:
XEQ "#FIB"	FIB:=FIC(n-1)+FIB(n-2)
X<> IND 01	
2	
-	
XEQ "#FB"	else:
RCL IND 01	FIB:=FIC(n-1)+FIB(n-2)
+	
LBL 01	end procedure
CLA	
ARCL 02	
X<> M	
X<> a	
X<> M	
DSE 02	
NOP	
DSE 01	
NOP	
RTN	

LBL "FIBO"	Calling program for #FB
8.0002	init stack pointers
STO 01	sp1 will start at 10
9,0002	
STO 02	sp2 will start at 11
"ENTER N"	
PROMPT	prompts for index
XEQ "#FIB"	compute F(n)
END	

Note the simplification in the calling program compared with the factorial case: the pointers are given directly, also there's no need for a global label to "END". Those were explicit there for didactical purposes only.

Note as well that *two calls to the recursive function* #FIB *are needed for each iteration*; one for n-1 and another for n-2.

Instructions:	Results:	Example:		Display:
1) XEQ "FIBO"	ENTER N	XEQ "FIBO"	=>	ENTER N
2) N, R/S	F(n)	5, R/S	=>	5

Algorithm 3: Ackerman function.

The Ackerman function is defined as: A(m,n) = n+1 , m=0
= A(m-1, 1) , n=0
= A(m-1, A(m. n-1)) , m>0 , n>0

This is different from the previous examples on several accounts. For starters, we're now dealing with two indexes, therefore will need to add a third pointer, **sp3**, and the corresponding data registers in the stack. The function has been Implemented with the following program: (warning: very slow!)

LABEL #ACK"	function ACKER
ISG 01	begin procedure
NOP	
ISG O2	(
NOP	sp1:=sp1+3
ISG 03	sp2:=sp2+3
NOP	sp3:=sp3+3
STO IND 01	
X<>Y	
STO IND 02	
CLA	
X<> M	
X<> a	
X<> M	
ASTO IND 03	
X=0?	If m=0 then
GTO 00	ACKER:=n+1
X<>Y	if n=0 then
X=0?	ACKER:=ACKER(m-1, 1)
GTO 01	
1	ACKER:=
-	ACK(m-1, ACK(m, n-1))
XEQ "#ACK"	
RCL IND 02	
1	
-	
X<>Y	
XEQ "#ACK"	compute A(m-1, x)
GTO 02	
LBL 00	ACKER:=n+1
X<>Y	
1	
+	
GTO 02	
LBL 01	ACKER:=ACKER(m-1, 1)
X<>Y	
1	
-	
1	
XEQ "#ACK"	
LBL 02	end procedure
CLA	
ARCL IND 03	
X<> M	
X<> a	
X<> M	
DSE 03	
NOP	
DSE 02	
NOP	
DSE 01	
NOP	
RTN	

LBL "ACKER"	
7.00003	the stack starts at R10
STO 01	sp1:=7.00003
8.00003	sp2:=8.00003
STO 02	sp3:=9.00003
9.00003	
STO 03	
"ENTER M^N"	input indexes
PROMPT	
XEQ "#ACK"	compute A(m,n)
END	

Three calls to the recursive function #ACK are needed for each iteration, which contributes to the slowness of the program – about 45-50 seconds on V41 with TURBO mode (!)

Instructions:	**Results:**	**Example:**	**Display**
XEQ "ACKER"	ENTER M^N	XEQ "ACKER" =>	ENTER M^N
M, ENTER^, n, R/S	A(m,n)	3, ENTER, 2, R/S =>	29

Algorithm 4: Towers of Hanoi.

Repeated here is the article published in PPCCJ V8N3 p22.

Given three pegs (A, B, and C), N discs of varying size stacked in order of size (large on the bottom, small on the top) on peg A.

Problem: In the smallest number of moves, one disc at a time, in such a way that a disc is never placed on the top of a smaller one, move the N discs (similarly stacked) from peg A to pag B.

Algorithm: Two sections are involved:

MDISC(A, B): moves a disc from peg A to peg B.
MTOWER(A, B, C, N) : moves a tower of N discs from peg A to peg B via peg C

```
Procedure MTOWER(var A,B,C :peg ; N  :natural ) ;
        Begin
           If N=1 then  MDISC(A, B)
                  Else  MTOWER(A, B, C, N-1)
                        MDISC(A, B)
                        MTOWER(C, B, A, N-1)
        End.
```

Here's the FOCAL code that implements this procedure. As expected, this example is more complex than the previous ones so it has a more demanding resource utilization, such as data register usage.

LBL "#MTWR"	procedure MTOWER
ISG 00	begin
NOP	
CLA	
X<> M	
X<> a	
X<> M	
ASTO IND 00	
X<>Y	MTWR(A,B,C,N)
DSE T	
XEQ "#MTWR"	
ISG T	
NOP	
X<> Z	
CLA	end
ARCL IND 00	
X<> M	
X<> a	
X<> M	
DSE 00	
NOP	
RTN	
LBL "#MDSC"	procedure MDISC
STO 06	save stack

RND	
STO 05	
RDN	
STO 04	
RDN	
RCL IND 04	
RCL IND 05	
RCL IND 06	
DSE Y	
NOP	moves disc from peg A
X<> IND Z	to peg B
X<> IND Y	
X<> IND Z	
ISG Z	
NOP	
STO IND 06	
RDN	
STO IND 05	
RDN	
STO IND 04	
RDN	
RCL 04	
RCL O5	
RCL 06	
X<> L	swap (X, L)
"PEG A:"	display peg A

AVIEW	
RDN	
RCL 01	
XEQ 01	
“PEG B:”	display peg B
AVIEW	
RDN	
RCL 02	
XEQ 01	
“PEG C:”	display peg C
AVIEW	
RDN	
RCL 03	
XEQ 01	
ADV	layout instructions
CLD	
X<> L	swap (X, L)
LBL 01	display a peg
ISG X	
NOP	
DSE X	
RTN	
LBL 02	
VIEW IND X	
ISG X	
GTO 02	
RTN	
LBL “HANOI”	calling program
FIX 0	
“ENTER N”	
PROMPT	
RCL X	
RCL X	
10	
STO 00	
DSE 00	
NOP	
+	
STO 02	
*	
STO 03	
RDN	
RCL 01	
1	
-	
E3	
/	
+	
ST+ 01	R01 :=
LASTX	R01+(9+2N)/1000
+	
ST+ 02	R02 :=
LASTX	R02+(9+4N)/1000
+	
ST+ 03	R03 :=
RDN	R03+(9+4N)/1000
RCL 01	
1	
LBL 00	
STO IND Y	
ISG X	
NOP	
ISG Y	
GTO 00	
R^	
1	
2	T = N
3	Z = pointer to R01 (A)
XEQ “#MTWR”	Y = pointer to R02 (B)
END	X = pointer to R03 I

Memory usage:

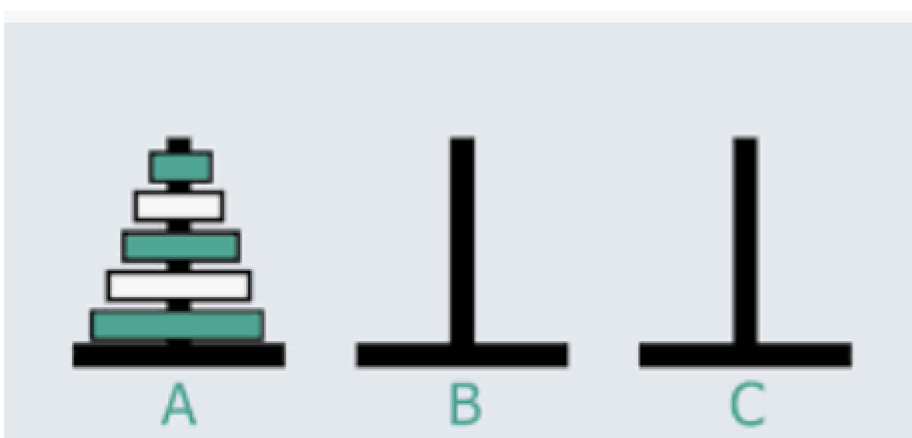
R00 :	stack pointer	R10 to R(10+N-1) :	stack
R01 :	pointer to peg A	R(10+N) to R(10+2N-1) :	peg A
R02 :	pointer to peg B	R(10+2N) to R(10+3N-1) :	peg B
R03 :	pointer to peg C	R(10+3N) to R(10+4N-1) :	peg C

Notice the implementation of ‘var’ in procedure MTOWER. If a call by reference is used (denoted by var) then use pointers to the variables. (you can find more about call by reference or call by value in books about software engineering).

Instructions:	Results
XEQ "HANOI"	ENTER N
N, R/S	PEG A:
	Sequence of numbers
	PEG B:
	Sequence of numbers, … etc.

The sequence of numbers are the numbers of the discs. Disc 1 is the smallest disc. Disc N is the largest disc. Each time a disc is moved from a peg to another peg the contents of each peg is displayed (peg A, peg B, peg C).

Example:	**Display:**		
XEQ "HANOI"	ENTER N		
5 , R/S	PEG A: 2345	PEG B: 1	PEG C:
	PEG A: 345	PEG B: 1	PEG C: 2
	PEG A: 345	PEG B:,	PEG C: 12
	PEG A: 45	PEG B: 3	PEG C: 12
	PEG A: 145	PEG B: 3	PEG C: 2
	. . . . .	. . . . . .	. . . .
	. . . .	. . . . . .	. . . .
	PEG A:	PEG B: 12345	PEG C:



Algorithm 5: Partitions.

You can write a positive integer m as a sum of positive terms (0 < term <= n)

```
6 = 1 + 1 + 1 + 1 +1 +1
  = 2 + 1 + 1 + 1 +1
  = 2 + 2 + 1 + 1
  = 2 + 2 + 2
  = 3 + 1 + 1 + 1
  = 3 + 2 + 1
  = 3 + 3
  = 4 + 1 +1
  = 4 + 2
  = 5 + 1
  = 6
```

The number of partitions Q(m, n) is the number of different ways the number can be written as sum of terms, thus Q(6, 6) = 11

Algorithm:

```
Function Q(m, n : natural) :natural  ;
        begin
           if n=1 or m=1 then Q: = 1
           else
             begin
                If m<n then Q: = 1 + Q(m, m-1)
                           Else Q:= Q(m, n-1) + Q(m-n, n)
             end
        end
```

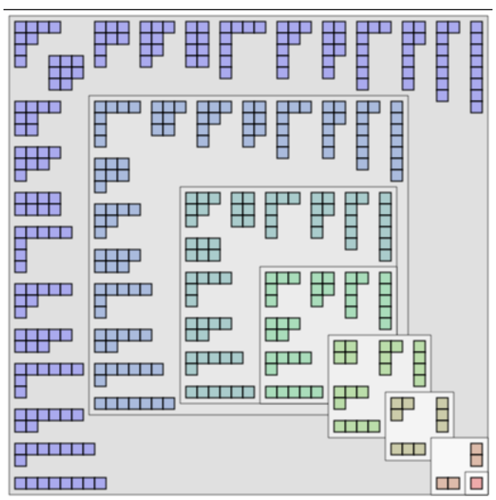
And shown below is the program that implements it. This one more standard as it uses the same structures as those seen in the examples before HANOI., now with three stack pointers because of the existence of two numeric inputs, m and n

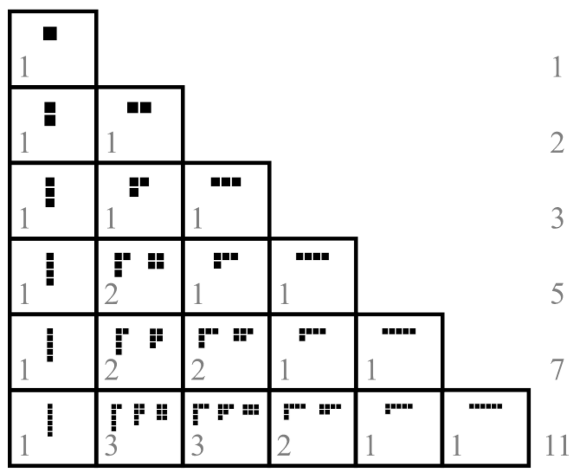
LBL "#Q"	function Q
ISG 00	begin
NOP	
ISG 01	
NOP	
ISG 02	
NOP	
STO IND 02	
X<>Y	
STO IND 01	
CLA	
X<> M	
X<> a	
X<> M	
ASTO IND 00	if m=1 or n=1 X<>Y
X=Y?	then Q=1
GTO 00	
X<> Z	if m<=n then
RDN	Q:=Q(m,m-1)+1
X<=Y?	
GTO 01	
X<>Y	Q(m,n-1)+Q(m,n-1)
1	
-	
XEQ #Q"	compute Q(m,n-1)
X<> IND 01	
RCL IND 02	
ST- Y	
XEQ "#Q"	compute Q(m,m-1)
1	
+	
LBL 00	end
CLA	
ARCL IND 00	

```
X<>  M
X<>  a
X<>  M
DSE  02
NOP
DSE  01
NOP
DSE 00
NOP
RTN
```

LBL "PART"	calling program
7,00003	
STO 00	
8,00003	can also use { 1 +}
STO 01	
9.00003	
STO 02	
"ENTER M^N"	
PROMPT	
XEQ #Q"	
END	

Instructions:	Result:	Example:	Display:
XEQ "PART"	"ENTER M^N"	XEQ "PART"	ENTER M^N
m ENTER n, R/S	Q(m,n)	6 ENTER^6, R/S	11





Modifications made in the RECURSE module.

The algorithms reviewed have a similar structure that have been implemented as common subroutines shared by all of them, as shows below:

- Initialization of pointer values **$I2, $I3**
- Begin of the procedure **$BEG2, $BEG3**
- Math on the partial results within each program
- Ending of the procedure. $**END2, $END3**

You can set User flag F10 to see the register numbers used to store the partial results.

Routine listing.

Even if in the module the routines are in MCODE a translation to FOCAL is provided below for your convenience. Note how we use the ALPHA registers M,N,O to hold the data register pointers **sp1**, **sp2**, and **sp3** instead of R00, R01, R02.

```
23  *LBL "$I2"
02  "N=?"
03  PROMPT
04  -2.00002
05  STO M
06  E
23  +
23  STO N
23  RDN
10  RTN

11  *LBL "$I3"
12  "N^K=?"
13  PROMPT
14  -3.00003
15  STO  M
16  E
17  +
18  STO  N
19  E
20  +
21  STO  O
22  RDN
23  END
```

```
01  LBL "$END3"
02  DSE  O(7)
03  NOP
04  LBL  "$END2"
05  DSE  N(6)
06  NOP
07  XEQ "LVDN"
08  DSE  M(5)
09  NOP
10  RTN

11  LBL "$BEG3"
12  ISG  O(7)
13  NOP
14  STO IND 7(O)
15  X<>Y
16  LBL  "$BEG2"
17  ISG  N(6)
18  NOP
19  STO IND N(6)
20  ISG M(5)
21  NOP
22  LBL "LVDN"
23  CLA
24  ARCL 02
25  X<>  M
26  X<>  a
27  X<>  M
28  RTN

29  LBL "LVUP"
30  CLA
31  ARCL  IND 00
32  X<>  M
33  X<>  a
34  X<>  M
35  END
```

RTN Stack Functions

The table below summarizes the RTN stack functions included in the module:

Function	Description	Input	Output
CLRTN	Clear RTN stack	RTN Stack contents	Erases pending addresses
RTNE?	Is RTN Stack Empty? (no levels)	RTN Stack contents	Yes if L=0
RTNF?	Is RTN Stack Full? (six levels used)	RTN Stack contents	Yes if L=6
RTNS	Get # of used RTN levels	RTN Stack contents	Number in X, stack lifted
XQ>GO	Drop first RTN level	RTN Stack contents	Last addr removed

None of the recursion routines described before make use of these functions but nevertheless they're related to the same subject, thus their inclusion in the module. They provide enhanced control of the program flow, so your routines can become more powerful and flexible.

Background information:

The OS has provision for up to six levels of subroutines; that is your FOCAL programs can have up to five chained XEQ calls to other programs or subroutines. The program pointer (PC) and the first two pending return addresses are stored in status registers b(12), the third is stored as two halves on each register, and the remaining three in status register a(11).

b(12):

R	3	A	D	R	2	A	D	R	1	P	C	N	T	
13	12	11	10	9	8	7	6	5	4	3	2	1	0	

a(11):

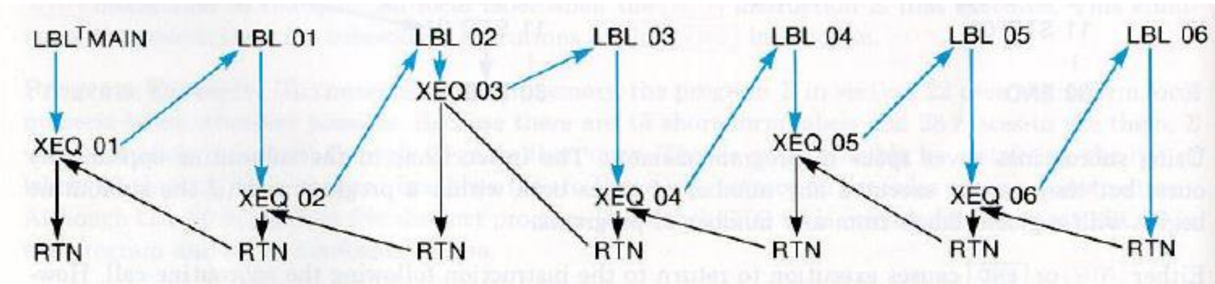
A	D	R	6	A	D	R	5	A	D	R	4	A	D	
13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Getting Information on Subroutine Levels usage.

- **RTNS** returns the number of pending RTN levels to the X register. Obviously, the result will be zero if executed in manual mode, as no pending subroutines exist. The stack is lifted.
- **CLRTN** clears all return addresses in the RTN stack. It does not alter the current program pointer so it's safe to use in a running program.
- **RTNE?** and **RTNF?** are used to check whether the RTN stack is empty (no levels) or full (six levels used). They behave like the standard test functions in the calculator, returning YES/NO and skipping a line in a running program if the condition is false.
- **XQ>GO** removes one pending routine address off the RTN stack and shifts the rest one level down. No output to X is produced.

Let's see a few examples of utilization.

The diagram below is taken from the HP-41CX manual, Vol. II page 302. It shows an example of six RTN stack levels utilization by a main program calling six subroutines – not the only way to get there but certainly a clear one.



Example1:

Using **RTN?** Is a good way to control whether an information message should be displayed. This situation arises frequently when using a FOCAL routine both as its own function or as a subroutine of another larger program, when displaying the partial result isn't desired.

LBL "SUBRTN"
. . .
. . .
"MESSAGE"
RTN?
RTN
PROMPT
END

Example2:

RTNF? provides a control safety check to prevent RTN Stack overflowing. Simply add it before the XEQ instructions and **SKIP** the XEQ call if the result is affirmative:

. . .
. . .
RTNF?
SKIP
XEQ "XYZ"
. . .
. . .

Example3:

XQ>GO is the best way to cancel a pending return, very useful in cases when the called subroutine has encountered either an error condition or a game-changer result; calling this function will cancel the return to the calling point.

Modular Math (by Greg McClure and JM Baillard)

The following description was written by Greg, taken from the GJM ROM manual.

For those acquainted with modular math, the following modulus functions are provided:

- **M+** performs Z+Y MOD X. It works for values up to 10 digits and takes into consideration the sign of the values. **M+** handles differently signed parameters. It is in MCODE and uses 13-digit math, making it much faster.
- **M-** performs Z-Y MOD X. The same comments apply as in **M+**.
- **M*** performs Z*Y MOD X. The same comments apply as in **M+** and **M-**.
- **M^2** performs Y^2 MOD X. It really uses much of the same code as **M***, it is actually doing a Y*Y MOD Z.

With the above MCODE routines, the following FOCAL functions taken from Jean-Marc Baillard run much quicker…

- **1/M** performs 1/Y MOD X. This function may or may not have an answer. Remember, the definition is "Return the value that, when multiplied by Y MOD X yields 1". This value may not exist. The function will stop with "DATA ERROR" if this is the case.
- **SQRTM** performs SQRT(Y) MOD X. This function will return either 0 (no solution) or the control number of the registers containing the answers. Remember, the definition is "Return the value that, when multiplied by itself, returns Y MOD X". So if 1.002 is returned to X then the answers are in R1 and R2. They should be considered as dual answers, that is, +R1, -R1, +R2, and –R2 (that would be 4 answers).

Where is ***M/***? Well actually this is the congruence function (if AX = B MOD C then X = B/A MOD C). The **CONG** function solves AX=B MOD C, expecting A in Z, B in Y, and modulus C in X. This may or may not yield an answer (for example 2X = 3 MOD 10 has no solutions), so it is possible that the function will stop with a DATA ERROR. If it doesn't, then X will contain the primary answer, and Y will contain the value that can be added or subtracted any integer number of times for the other answers. For example, to solve 2X = 4 MOD 10, do:

2, ENTER^, 4, ENTER^, 10, XEQ "CONG" ;

the result is X = 2, Y = 5. This means the solution set is {…, -8, -3, 2, 7, 12, 17 …}.

The Alpha register is used, so it will be cleared if a solution is found. If not, then synthetic registers M, N, and O will contain the reduced A, B, and C (a **GCD** is performed on A, B, and C before **1/M** is performed and this is saved in M, N, and O). This may be useful in determining why the DATA ERROR occurred. BTW I have listed Ángel Martin as a co-author, since he did much of the grunt work to help determine the method of solution needed. Once I read all the info, applying **1/M**, **GCD**, and **M*** was a simple matter.

Program listing

The FOCAL routines are listed in the next couple of pages for your reference.

```
LBL "1/M"
RAD
E
XROM "UV"
STO Z
FRC
X#0?
GTO 00
RDN
FRC
X=0?
GTO 01
*LBL 00
CLX
LN
*LBL 01
CLX
RCL 02
ABS
MOD
END
```

```
LBL "UV"        auxiliary subroutine
STO 00
CLX
STO 02
STO 03
E
STO 01
STO 04
*
+
LBL 01
STO  T
MOD
ST-  Y
X<>  Z
/
RCL 01
X<>  02
STO 01
X<>Y
*
ST-  02
CLX
RCL 03
X<>  04
STO 03
LASTX
*
ST-  04
RDN
X#0?
GTO 01
X<>Y
ST/  00
RCL 03
RCL 01
RCL 00
ST*  Z
*
END
```

```
01  LBL "SQRTM"
02  STO Z
03  2
04  /
05  INT
06  X<> Z
07  MOD
08  0
09  STO 00
10  SF 10
11  *LBL 01
12  CLX
13  RCL Z
14  ST* X
15  LASTX
16  MOD
17  X#Y?
18  GTO 02
19  X<> T
20  ISG 00
21  CLX
22  STO IND 00
23  *LBL 02
24  DSE Z
25  GTO 01
26  FS?C 10
27  GTO 01
28  RCL 00
29  E3
30  /
31  X#0?
32  ISG X
33  END
```

```
1   LBL "M^"
2   SIGN
3   STO 00
4   RDN
5   STO 02
6   X<>Y
7   STO 01
8   GTO 03
9   *LBL 01
10  2
11  MOD
12  X#0?
13  GTO 02
14  LASTX
15  ST/ 02
16  RCL 01
17  R^
18  M^2
19  STO 01
20  GTO 03
21  *LBL 02
22  ST- 02
23  RCL 00
24  RCL 01
25  R^
26  M^2
27  STO 00
28  *LBL 03
29  LASTX
30  RCL 02
31  X#0?
32  GTO 01
33  RCL 00
34  END
```

```
01  LBL "CONG"
02  STO M(5)
03  RDN
04  STO N(6)
05  RDN
06  STO O(7)
07  RCL M(5)
08  GCD
09  RCL N(6)
10  GCD
11  ST/ M(5)
12  ST/ N(6)
13  ST/ O(7)
14  RCL O(7)
15  RCL M(5)
16  1/M
17  RCL N(6)
18  RCL M(5)
19  M*
20  RCL M(5)
21  CLA
22  X<>Y
23  END
```

Many Digits of Pi. *(by Peter Platzer, MoHPC Forum)*

The module includes the remarkable and impressive MCODE implementation of the Spigot algorithm by Peter Platzer, published in the Museum of HP Calculators forum. His description is available in the appendix, but here are the highlights:

The code asks for three inputs: The page where the MLDL ram starts to use, the number of digits and the base b to use (max = 5 for 5 digits at a time). One can set Flag 0 and the calc will stop at each group of digits and wait for a key to be pressed, otherwise it just keeps calculating …

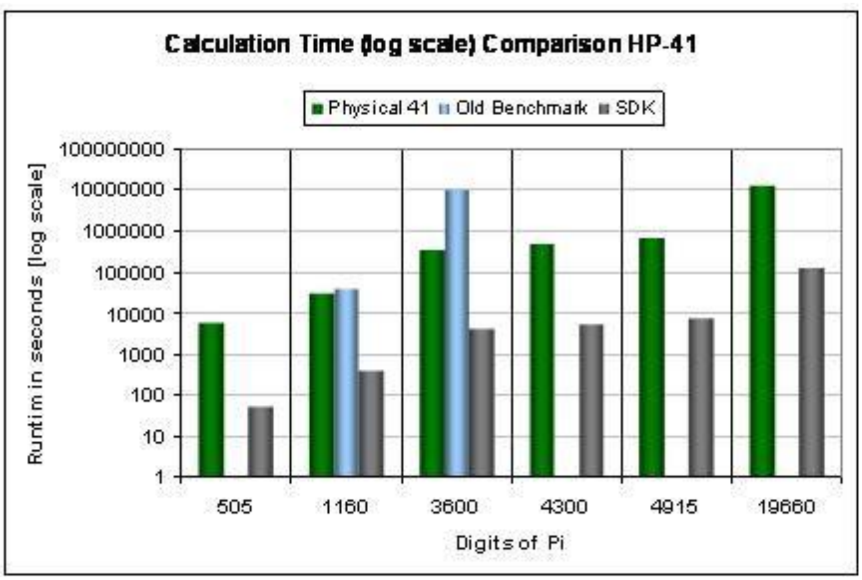
Setting Flag 1 will store the found digits in the same compressed format – each group of up to 5 digits is stored in 2 words, with the right nibble converted to hex. They are stored in reversed order though

In manual execution the function prompts for the number of digits to calculate (limited to 1999 by the prompt) and the destination page where to store them. This needs to be a q-RAM page to allow writes into it. The maximum number of digits is 4095 – which will fill up the page in its entirety.

The screens below show an example to calculate 1,046 digits to be stored in page B:

….

In an unmodified HP-41 it delivers 1,160 digits in about 9 hours 3,600 digits in about 4 days , and 4,915 digits in about 8 days. The chart below shows a comparison with the previous record-holding approaches described in the article.



Extended precision: Pi to 1,000 places. *(by Ron Knapp, PPCCJ V8N6 p69)*

"*Compute the first 1,000 decimal digits of Pi in less than 11 hours, 30 minutes*". That was the friendly challenge put out by the PPC 'Journal", especially to members of the TI Personal Calculator Club, approximately a year ago. This challenge was repeated in the "Calcu-letter" of Popular Science Magazine, July 1981.

Up to the present time, I have heard of no serious attempts to eclipse this record. So,-- I decided to improve my own program. The program listed below computes Pi to 1,000 decimal places in just 8 hours, 30 minutes.

Ed. note: with 2x machines, and some will run Faster, (fastest reported so far was Emett Ingram (17) at 2.8x) a 4 hour, 1,000 digit Pi program is the state of the PPC art. How long will it be before someone places 100,000 digits of Pi on a cassette? A printer on the HP-IL would take nearly 45 minutes to print it on 70 feet of paper at 20 digits per line, 2 lines per second.

The first 1.000 decimal places of Pi contains 93 0s, 116 1s, 103 2s, 102 3s, 93 4s, 97 5s, 94 6s, 95 7s, 101 8s, and 106 9s. Below is "3 dot" followed by the first 1,000 decimals of Pi.

3 . 1 4 1 5 9 2 6 5 3 5 8 9 7 9 3 2 3 8 4 6 2 6 4 3 3 8 3 2 7 9 5 0 2 8 8 4 1 9 7 1 6 9 3 9 9 3 7 5 1 0 5 8 2 0 9
7 4 9 4 4 5 9 2 3 0 7 8 1 6 4 0 6 2 8 6 2 0 8 9 9 8 6 2 8 0 3 4 8 2 5 3 4 2 1 1 7 0 6 7 9 8 2 1 4 8 0 8 6 5 1 3 2
8 2 3 0 6 6 4 7 0 9 3 8 4 4 6 0 9 5 5 0 5 8 2 2 3 1 7 2 5 3 5 9 4 0 8 1 2 8 4 8 1 1 1 7 4 5 0 2 8 4 1 0 2 7 0 1 9
3 8 5 2 1 1 0 5 5 5 9 6 4 4 6 2 2 9 4 8 9 5 4 9 3 0 3 8 1 9 6 4 4 2 8 8 1 0 9 7 5 6 6 5 9 3 3 4 4 6 1 2 8 4 7 5 6
4 8 2 3 3 7 8 6 7 8 3 1 6 5 2 7 1 2 0 1 9 0 9 1 4 5 6 4 8 5 6 6 9 2 3 4 6 0 3 4 8 6 1 0 4 5 4 3 2 6 6 4 8 2 1 3 3
9 3 6 0 7 2 6 0 2 4 9 1 4 1 2 7 3 7 2 4 5 8 7 0 0 6 6 0 6 3 1 5 5 8 8 1 7 4 8 8 1 5 2 0 9 2 0 9 6 2 8 2 9 2 5 4 0
9 1 7 1 5 3 6 4 3 6 7 8 9 2 5 9 0 3 6 0 0 1 1 3 3 0 5 3 0 5 4 8 8 2 0 4 6 6 5 2 1 3 8 4 1 4 6 9 5 1 9 4 1 5 1 1 6
0 9 4 3 3 0 5 7 2 7 0 3 6 5 7 5 9 5 9 1 9 5 3 0 9 2 1 8 6 1 1 7 3 8 1 9 3 2 6 1 1 7 9 3 1 0 5 1 1 8 5 4 8 0 7 4 4
6 2 3 7 9 9 6 2 7 4 9 5 6 7 3 5 1 8 8 5 7 5 2 7 2 4 8 9 1 2 2 7 9 3 8 1 8 3 0 1 1 9 4 9 1 2 9 8 3 3 6 7 3 3 6 2 4
4 0 6 5 6 6 4 3 0 8 6 0 2 1 3 9 4 9 4 6 3 9 5 2 2 4 7 3 7 1 9 0 7 0 2 1 7 9 8 6 0 9 4 3 7 0 2 7 7 0 5 3 9 2 1 7 1
7 6 2 9 3 1 7 6 7 5 2 3 8 4 6 7 4 8 1 8 4 6 7 6 6 9 4 0 5 1 3 2 0 0 0 5 6 8 1 2 7 1 4 5 2 6 3 5 6 0 8 2 7 7 8 5 7
7 1 3 4 2 7 5 7 7 8 9 6 0 9 1 7 3 6 3 7 1 7 8 7 2 1 4 6 8 4 4 0 9 0 1 2 2 4 9 5 3 4 3 0 1 4 6 5 4 9 5 8 5 3 7 1 0
5 0 7 9 2 2 7 9 6 8 9 2 5 8 9 2 3 5 4 2 0 1 9 9 5 6 1 1 2 1 2 9 0 2 1 9 6 0 8 6 4 0 3 4 4 1 8 1 5 9 8 1 3 6 2 9 7
7 4 7 7 1 3 0 9 9 6 0 5 1 8 7 0 7 2 1 1 3 4 9 9 9 9 9 9 8 3 7 2 9 7 8 0 4 9 9 5 1 0 5 9 7 3 1 7 3 2 8 1 6 0 9 6 3
1 8 5 9 5 0 2 4 4 5 9 4 5 5 3 4 6 9 0 8 3 0 2 6 4 2 5 2 2 3 0 8 2 5 3 3 4 4 6 8 5 0 3 5 2 6 1 9 3 1 1 8 8 1 7 1 0
1 0 0 0 3 1 3 7 8 3 8 7 5 2 8 8 6 5 8 7 5 3 3 2 0 8 3 8 1 4 2 0 6 1 7 1 7 7 6 6 9 1 4 7 3 0 3 5 9 8 2 5 3 4 9 0 4
2 8 7 5 5 4 6 8 7 3 1 1 5 9 5 6 2 8 6 3 8 8 2 3 5 3 7 8 7 5 9 3 7 5 1 9 5 7 7 8 1 8 5 7 7 8 0 5 3 2 1 7 1 2 2 6 8
0 6 6 1 3 0 0 1 9 2 7 8 7 6 6 1 1 1 9 5 9 0 9 2 1 6 4 2 0 1 9 8 9

Program listing.-

```
*LBL "PIE3"
*LBL A
" PI -?-"
AVIEW
CLRG
FIX 3
4
STO 09
E5
ST/ Y
STO 04
X^2
STO 05
X<>Y
427
+
STO 02
239
X^2
STO 07
LASTX
E2
*
STO 13
RDN
X^2
STO 08
94 E-5
STO 11
14.0139
STO 12
25
STO 10
*LBL 00
RCL 11
ST+ 12
RCL 12
RND
STO 00
RCL 07
RCL 02
INT
ENTER^
ST* Z
2
-
ST- Z
*
RCL 10
*
STO 06
CLX
STO 01
X<>Y
RCL 13
*
ENTER^
GTO 02
*LBL 01
RCL 06
ST/ Z
MOD
X<>Y
INT
X<>Y
RCL 04
ST* Z
*
ENTER^
*LBL 02
RCL 06
ST/ Z
MOD
STO 03
RDN
INT
+
RCL 05
ST- Y
X<>Y
RCL IND 00
+
X>0?
ISG 01
*LBL 03
X<0?
+
RCL 01
RCL 04
ST/ Z
*
ENTER^
*LBL 02
RCL 08
ST/ Z
MOD
R^
INT
LASTX
FRC
RDN
+
X<>Y
INT
RCL 04
ST* T
ST* Z
*
STO IND 00
RDN
ENTER^
*LBL 03
RCL 08
ST/ Z
MOD
X<>Y
INT
ST+ IND 00
RDN
+
STO 01
RCL 03
RCL 04
*
ENTER^
ISG 00
GTO 01
DSE 02
GTO 00
4096 E-7
STO 08
1439.00006
STO 02
837 E-6
STO 11
115.115
STO 12
80
STO 13
```

```
5 E6
STO 07
.75
STO 06
*LBL “Q”
RCL 11
ST+ 12
RCL 12
RND
STO 00
STO 03
SF 00
*LBL 05
RCL 02
INT
ENTER^
ENTER^
*LBL 02
2
-
ST* Z
RCL 10
ST* Z
X<>Y
*
2
ST- L
CLX
LASTX
ST* T
ST- Y
RDN
*
R^
ST+ T
X^2
R^
+
+
FC? 00
GTO 02
RCL 13
*
3
DSE 02
GTO 03
*LBL 02
RCL 07
*
RCL 06
*LBL 03
X<>Y
RDN
/
STO 01
CLX
R^
ENTER^
GTO 09
*LBL 08
RCL 01
ST/ Z
MOD
X<>Y
INT
X<>Y
RCL 04
ST* Z
*
ENTER^
*LBL 09
RCL 01
ST/ Z
MOD
RDN
INT
+
RCL IND 00
-
X>0?
GTO 02
DSE 00
*LBL 03
DSE IND 00
ISG 00
RCL 05
+
*LBL 02
STO IND 00
R^
RCL 04
*
ENTER^
ISG 00
GTO 08
RCL 03
STO 00
FS?C 00
GTO 05
CLX
ENTER^
DSE 02
FS? 00
GTO 04
*LBL 11
X<> IND 00
RCL 04
/
FRC
LASTX
INT
RCL 08
*
FRC
LASTX
INT
ST+ IND 00
RDN
X<>Y
RCL 05
ST* T
ST* Z
*
RCL 08
*
FRC
X<>Y
LASTX
INT
R^
+
RCL 05
-
+
X>0?
ISG IND 00
X>0?
GTO 03
RCL 05
+
*LBL 03
ISG 00
GTO 11
GTO “Q”
*LBL 04
```

```
RCL 03
STO 00
RCL 10
X^2
3
Y^X
LASTX
*
STO 08
CLX
*LBL 13
RCL IND 00
X<>Y
RCL 04
ST/ Z
*
ENTER^
*LBL 02
RCL 08
ST/ Z
MOD
R^
INT
LASTX
FRC
RDN
+
X<>Y
INT
RCL 04
ST* T
ST* Z
*
STO IND 00
RDN
ENTER^
*LBL 03
RCL 08
ST/ Z
MOD
X<>Y
INT
ST+ IND 00
RDN
+
ISG 00
GTO 13
114.013
STO 00
215
STO 03
CLX
*LBL 06
RCL IND 03
+
RCL IND 00
-
0
X<>Y
X<0?
X>0?
GTO 02
RCL 05
+
DSE Y
*LBL 02
STO IND 00
RDN
DSE 03
DSE 00
GTO 06
BEEP
RTN
*LBL E
SF 21
CLA
FIX 0
14.114
STO 00
SF 29
RCL IND 00
ACX
ADV
CF 29
ISG 00
*LBL 07
XEQ 10
ISG 00
FS? 00
RTN
" "
XEQ 10
ADV
CLA
ISG 00
GTO 07
AVIEW
RTN
*LBL 10
RCL IND 00
RCL 04
/
INT
LASTX
FRC
RCL 04
XEQ 12
" "
XEQ 12
RTN
*LBL 12
*
RCL Y
X=0?
GTO 03
LOG
INT
*LBL 03
RCL 09
X<>Y
X=Y?
GTO 02
-
0
*LBL 14
ARCL X
DSE Y
GTO 14
*LBL 02
ARCL T
ACA
CLA
END
```

Extended precision: E to 2,900 places. (by Ron Knapp, PPCCJ V9N1 p12)

This program is an abbreviated version designed to compute the decimal places of "e" to the greatest possible limit allowed in an HP-41CV or an HP-41C with a Quad Memory module. The program does the initialization including setting the SIZE to 294 data registers.

R01 shows the count-down number at all times. Originally this indicates the number of terms of the series necessary to obtain the accuracy desired. The number of terms yet to be computed is continuously displayed to allow the operator to know the progress of the computation. When the count-down number reaches zero the execution can proceed to the readout (or printout) routine, which displays 10 digits at a time, broken into two groups of five digits each, for easy reading. All leading and ending zeros are shown.

Instructions:

XEQ "E2900"	Will take around 25 minutes at TURBO50 speed !
XEQ "R"	To see/Print the results

```
01  LBL "R"        Readout results
02  FIX 0
03  CF 29
04  "2,"
05  AVIEW
06  4
07  ST+  03
08  LBL 06
09  CLA
10  SF 01
11  RCL IND 03
12   E5
13   /
14  FRC
15  LASTX
16  INT
17  LBL 07
18  ENTER^
19  ENTER^
20  4
21  X<>T
22  X=0?
23  GTO 08
24  LOG
25  INT
26  –
27  0
28  X=Y?
29  GTO 09
30  LBL 08
31  ARCL X
32  DSE  Y
33  GTO 08
34  LBL 09
35  ARCL  Z
36  FC?C 01
37  GTO 10
38  "/-  "         ; two spaces
39  R^
40   E5
41  *
42  GTO 07
43  LBL 10
44  AVIEW
45  ISG 03
46  GTO 06
47  END
```

Program listing. -

```
*LBL "E2900"
294
PSIZE
CF 01
CF 02
4.004
STO 00
1112
STO 01
E
STO 03
.293
STO 03
*LBL e
RCL 01
ENTER^
VIEW X
DSE 01
E10
X<>Y
ISG Z
*LBL 00
RCL 01
X<>Y
*
X>Y?
GTO 01
DSE 01
GTO 00
SF 01
ENTER^
*LBL 01
R^
LASTX
X<>Y
RCL 01
3
FC? 01
DSE X
*LBL 02
+
-
E
ENTER^
*LBL 03
X<> L
ST* Y
X<> L
ST+ Y
ST+ L
DSE Z
GTO 03
*
+
*LBL 04
E5
*
ENTER^
R^
ST/ Z
MOD
X<>Y
INT
E5
X>Y?
GTO 05
/
INT
E
ST- 00
X<>Y
ST+ IND 00
RDN
ST+ 00
CLX
LASTX
FRC
E5
*
LASTX
*LBL 05
*
X<> IND 00
LASTX
/
INT
ST+ Y
X<> L
FRC
X<>Y
E5
ST* Z
*
ENTER^
R^
ST/ Z
MOD
LASTX
RDN
X<>Y
INT
ST+ IND 00
CLX
+
+
ISG 00
GTO 04
X<>Y
/
RND
E
ST- 00
X<>Y
ST+ IND 00
R^
E-10
*
ST* 02
RCL 02
LASTX
X>Y?
SF 02
FS? 02
ST/ 02
E-3
RCL 00
FRC
FC?C 02
+
RCL 03
X<Y?
X<>Y
RDN
4
+
STO 00
FC?C 01
GTO e
END
```

EXTENDED PRECISION -- "e" to 2900 PLACES

```
2.71828 18284 59045 23536 02874 71352 66249 77572 47093 69995 95749 66967 62772 40766 30353 54759 45713 82178 52516 64274
  27466 39193 20030 59921 81741 35966 29043 57290 03342 95260 59563 07381 32328 62794 34907 63233 82988 07531 95251 01901
  15738 34187 93070 21540 89149 93488 41675 09244 76146 06680 82264 80016 84774 11853 74234 54424 37107 53907 77449 92069
  55170 27618 38606 26133 13845 83000 75204 49338 26560 29760 67371 13200 70932 87091 27443 74704 72306 96977 20931 01416
  92836 81902 55151 08657 46377 21112 52389 78442 50569 53696 77078 54499 69967 94686 44549 05987 93163 68892 30098 79312

  77361 78215 42499 92295 76351 48220 82698 95193 66803 31825 28869 39849 64651 05820 93923 98294 88793 32036 25094 43117
  30123 81970 68416 14039 70198 37679 32068 32823 76464 80429 53118 02328 78250 98194 55815 30175 67173 61332 06981 12509
  96181 88159 30416 90351 59888 85193 45807 27386 67385 89422 87922 84998 92086 80582 57492 79610 48419 84443 63463 24496
  84875 60233 62482 70419 78623 20900 21609 90235 30436 99418 49146 31409 34317 38143 64054 62531 52096 18369 08887 07016
  76839 64243 78140 59271 45635 49061 30310 72085 10383 75051 01157 47704 17189 86106 87396 96552 12671 54688 95703 50354

  02123 40784 98193 34321 06817 01210 05627 88023 51930 33224 74501 58539 04730 41995 77770 93503 66041 69973 29725 08868
  76966 40355 57071 62268 44716 25607 98826 51787 13419 51246 65201 03059 21236 67719 43252 78675 39855 89448 96970 96409
  75459 18569 56380 23637 01621 12047 74272 28364 89613 42251 64450 78182 44235 29486 36372 14174 02388 93441 24796 35743
  70263 75529 44483 37998 01612 54922 78509 25778 25620 92622 64832 62779 33386 56648 16277 25164 01910 59004 91644 99828
  93150 56604 72580 27786 31864 15519 56532 44258 69829 46959 30801 91529 87211 72556 34754 63964 47910 14590 40905 86298

  49679 12874 06870 50489 58586 71747 98546 67757 57320 56812 88459 20541 33405 39220 00113 78630 09455 60688 16674 00169
  84205 58040 33637 95376 45203 04024 32256 61352 78369 51177 88386 38744 39662 53224 98506 54995 88623 42818 99707 73327
  61717 83928 03494 65014 34558 89707 19425 86398 77275 47109 62953 74152 11151 36835 06275 26023 26484 72870 39207 64310
  05958 41166 12054 52970 30236 47254 92966 69381 15137 32275 36450 98889 03136 02057 24817 65851 18063 03644 28123 14965
  50704 75102 54465 01172 72115 55194 86685 08003 68532 28183 15219 60037 35625 27944 95158 28418 82947 87610 85263 98139

  55990 06737 64829 22443 75287 18462 45780 36192 98197 13991 47564 48826 26039 03381 44182 32625 15097 48279 87779 96437
  30899 70388 86778 22713 83605 77297 88241 25611 90717 66394 65070 63304 52795 46618 55096 66618 56647 09711 34447 40160
  70462 62156 80717 48187 78443 71436 98821 85596 70959 10259 68620 02353 71858 87485 69652 20005 03117 34392 07321 13908
  03293 63447 97273 55955 27734 90717 83793 42163 70120 50054 51326 38354 40001 86323 99149 07054 79778 05669 78533 58048
  96690 62951 19432 47309 95876 55236 81285 90413 83241 16072 26029 98330 53537 08761 38939 63917 79574 54016 13722 36187

  89365 26053 81558 41587 18692 55386 06164 77983 40254 35128 43961 29460 35291 33259 42794 90433 72990 85731 58029 09586
  31382 68329 14771 16396 33709 24003 16894 58636 06064 58459 25126 99465 57248 39186 56420 97526 85082 30754 42545 99376
  91704 19777 80085 36273 09417 10163 43490 76964 23722 29435 23661 25572 50881 47792 23151 97477 80605 69672 53801 71807
  76360 34624 59278 77846 58506 56050 78084 42115 29697 52189 08740 19660 90665 18035 16501 79250 46195 01366 58543 66327
```

Appendix. A few MCODE Listings.

Header	ACD7	08E	"N"	
Header	ACD8	02F	"/"	**Harmonic numbers:**
Header	ACD9	031	"1"	**Σ(1/n) for n=1,2,....x**
Header	ACDA	04E	"Σ"	Ángel Martin
ΣI/N	ACDB	2CC	?FSET 13	
	ACDC	2A9	?NC XQ	Shows "RUNNING" - clears F8
	ACDD	13C	->4FAA	[RUNMSG]
	ACDE	0F8	READ 3(X)	
	ACDF	361	?NC XQ	(includes SETDEC)
	ACE0	050	->14D8	[CHK_NO_S]
	ACE1	05E	C=0 MS	No Negative numbers!
	ACE2	088	SETF 5	Take Integer
	ACE3	0ED	?NC XQ	leaves result in 13-digit form
	ACE4	064	->193B	[INTFRC] - doesn't need DEC
	ACE5	19D	?NC XQ	Calculates H(n)
	ACE6	12C	->4B67	[HARMN4]
	ACE7	0B0	C=N ALL	10-digit result
	ACE8	331	?NC GO	Overflow, DropST, FillXL & Exit
	ACE9	002	->00CC	[NFRX]

HRMN	4B67	070	N=C ALL	argument
	4B68	1A0	A=B=C=0	zero trinity - initial sum!
expects CPU in DEC m	4B69	089	?NC XQ	uses {Q,+} (!)
	4B6A	064	->1922	[STSCR]
	4B6B	0B0	C=N ALL	argument to C
	4B6C	2EE	?C#0 ALL	Carry set if NOT Zero
	4B6D	0A9	?NC GO	leave result in {A,B}
	4B6E	066	->192A	[EXSCR]
	4B6F	128	WRIT 4(L)	k as Counter to L
	4B70	3CC	?KEY	
	4B71	360	?C RTN	bail out upon key depressed
	4B72	22D	?NC XQ	1/k
	4B73	060	->188B	[1/X_10]
	4B74	0D1	?NC XQ	Σ(k)
	4B75	064	->1934	[RCSCR]
	4B76	031	?NC XQ	Σ(k)+1/k
	4B77	060	->180C	[AD2-13]
	4B78	070	N=C ALL	new Partial Sum in N
	4B79	089	?NC XQ	Σ(k) + 1/k
	4B7A	064	->1922	[STSCR]
	4B7B	138	READ 4(L)	Recall n
	4B7C	1FD	?NC XQ	{A,B} = C-1
	4B7D	100	->407F	[DECC10]
	4B7E	373	JNC -18d	

Header	A6F6	098	"X"	SUM(N^X),, N=0,1,2,...Y
Header	A6F7	01E	"^"	(equals to HARMN if x=-1)
Header	A6F8	00E	"N"	
Header	A6F9	04E	"Σ"	Ángel Martin
ΣN^X	A6FA	2CC	?FSET 13	Skip if running PRGM
	A6FB	2A9	?NC XQ	Shows "RUNNING" - clears F8
	A6FC	13C	->4FAA	[RUNMSG]
	A6FD	244	CLRF 9	default condition
	A6FE	1A5	?NC XQ	Check for valid entries
	A6FF	100	->4069	[CHKST2]
	A700	2FE	?C#0 MS	Set Carry if negative
	A701	023	JNC +04	
	A702	248	SETF 9	flag it as negative
	A703	05E	C=0 MS	make it positive
	A704	0E8	WRIT 3(X)	required by [XY^X]
	A705	04E	C=0 ALL	
	A706	070	N=C ALL	mantissa
	A707	128	WRIT 4(L)	sign and exp
	A708	0B8	READ 2(Y)	Final Counter value (n)
	A709	05E	C=0 MS	No Negative numbers!
	A70A	088	SETF 5	Take Integer
	A70B	0ED	?NC XQ	leaves result in 13-digit form
	A70C	064	->193B	[INTFRC] - doesn't need DEC
	A70D	2FA	?C#0 M	Carry set if NOT Zero
	A70E	0DB	JNC +27d	(n-1) = 0
	A70F	0A8	WRIT 2(Y)	required by [XY^X]
	A710	3CC	?KEY	bail out upon key pressed
	A711	360	?C RTN	
	A712	0F8	READ 3(X)	Exp. Must be in X and C
	A713	3C4	ST=0	Base must be in Y
	A714	045	?NC XQ	C=y^C - uses SCR (!)
	A715	06C	->1B11	[XY_TO_X]
	A716	2F6	?C#0 XS	See if we overflow?
	A717	289	?C GO	"Out of Range"
	A718	003	->00A2	[ERROF]
	A719	24C	?FSET 9	Skip if original exp was positive
	A71A	239	?C XQ	inverse if negative
	A71B	061	->188E	[ON/X13
	A71C	138	READ 4(L)	restore 13-digit partial result
	A71D	158	M=C ALL	M has sign and exp
	A71E	0B0	C=N ALL	C has 13 digit mantissa
	A71F	031	?NC XQ	{A,B} = {A,B} + {C,M}
	A720	060	->180C	[AD2-13]
	A721	0AE	A<>C ALL	
	A722	128	WRIT 4(L)	sign and exp
	A723	0CE	C=B ALL	
	A724	070	N=C ALL	mantissa
	A725	0B8	READ 2(Y)	Recall n
	A726	1FD	?NC XQ	{A,B} = C-1
	A727	100	->407F	[DECC10]
	A728	32B	JNC -27d	[LOOP]
	A729	138	READ 4(L)	restore 13-digit partial result
	A72A	158	M=C ALL	M has sign and exp
	A72B	1A0	A=B=C=0	
	A72C	0B0	C=N ALL	C has 13 digit mantissa
	A72D	031	?NC XQ	{A,B} = {A,B} + {C,M}
	A72E	060	->180C	[AD2-13]
	A72F	331	?NC GO	Overflow, DropST, FillXL & Exit
	A730	002	->00CC	[NFRX]

Header	AF06	094	"T"	
Header	AF07	007	"G"	Sum of Mantissa Digits
Header	AF08	004	"D"	
Header	AF09	04E	"Σ"	Angel Martin
ΣDGT	AF0A	0B1	?NC XQ	Mantissa Digit SUM
	AF0B	10C	->432C	[SDGT4]
	AF0C	1F5	?NC XQ	CONVERT THE BINARY TO DECIM
	AF0D	0C4	->317D	[BIN--D]
	AF0E	0EE	B<>C ALL	put result in X
	AF0F	331	?NC GO	Overflow, DropST, FillXL & Exit
	AF10	002	->00CC	[NFRX]
ΣDGT4	432C	0F8	READ 3(X)	
	432D	00E	A=0 ALL	initial sum =0
	432E	39C	PT= 0	
NXTDGT	432F	33C	RCR 1	
	4330	3C6	RSHFC S&X	
Mantissa Digit Sum	4331	3C6	RSHFC S&X	
	4332	146	A=A+C S&X	add to previous sum
	4333	3DC	PT=PT+1	
	4334	0D4	?PT= 10	
	4335	3D3	JNC -06	[NEXTD]
	4336	3E0	RTN	

Header	ABE3	096	"V"	
Header	ABE4	005	"E"	Mantissa Reversal
Header	ABE5	012	"R"	
Header	ABE6	00D	"M"	Ángel Martin
MREV	ABE7	0F8	READ 3(X)	
	ABE8	10E	A=C ALL	safekeep C.MS and C.X
	ABE9	2DC	PT= 13	counter
	ABEA	250	LD@PT- 9	will do 10 times
	ABEB	01C	PT= 3	fixed position
	ABEC	3FA	LSHFA M	scroll A.M left
	ABED	102	A=C @PT	copy digit
	ABEE	3DA	RSFHC M	next C.M digit
	ABEF	27E	C=C-1 MS	decrease counter
	ABF0	3E3	JNC -04	no, do next
	ABF1	0AE	A<>C ALL	yes copy result to C.M
	ABF2	331	?NC GO	Overflow, DropST, FillXL & Exit
	ABF3	002	->00CC	[NFRX]

Header	AF50	*093*	***"S"***	
Header	AF51	*00E*	***"N"***	***Returns #RTN stack levels***
Header	AF52	*014*	***"T"***	
Header	AF53	*012*	***"R"***	*Ángel Martin*
RTNS	**AF54**	**006**	**A=0 S&X**	RESET counter
	AF55	01C	PT= 3	
	AF56	***338***	***READ 12(b)***	
	AF57	07C	RCR 4	*move PC away*
	AF58	2EA	?C#0 PT<-	*is RTN1 there?*
	AF59	*0E3*	*JNC +28d*	*no, exit*
	AF5A	166	A=A+1 S&X	*yes, increase count*
	AF5B	07C	RCR 4	*and*
	AF5C	2EA	?C#0 PT<-	
	AF5D	*0C3*	*JNC +24d*	
	AF5E	166	A=A+1 S&X	
	AF5F	07C	RCR 4	
	AF60	056	C=0 XS	
	AF61	2E6	?C#0 S&X	*partial exists?*
	AF62	***02F***	***JC +05***	*yes, jump over*
	AF63	***2F8***	***READ 11(a)***	*read next reg*
	AF64	056	C=0 XS	*clear XS digit*
	AF65	2E6	?C#0 S&X	*partial there?*
	AF66	*07B*	*JNC +15d*	*no, exit*
PARTL	AF67	166	A=A+1 S&X	*increase count*
	AF68	***2F8***	***READ 11(a)***	*read register again*
	AF69	23C	RCR 2	*move leftover*
	AF6A	2EA	?C#0 PT<-	
	AF6B	*053*	*JNC +10d*	
	AF6C	166	A=A+1 S&X	
	AF6D	07C	RCR 4	
	AF6E	2EA	?C#0 PT<-	
	AF6F	*033*	*JNC +06*	
	AF70	166	A=A+1 S&X	
	AF71	07C	RCR 4	
	AF72	2EA	?C#0 PT<-	
	AF73	*013*	*JNC +02*	
	AF74	166	A=A+1 S&X	
EXIT	AF75	*17D*	*?NC GO*	*[BIN-BCD] plus [RCL]*
	AF76	*0C6*	*->315F*	*[ATOX20]*

Header	AF11	*090*	***"P"***	
Header	AF12	*009*	***"I"***	
Header	AF13	*00B*	***"K"***	
Header	AF14	*013*	***"S"***	*Erik Blake*
SKIP	**AF15**	*141*	*?NC XQ*	
	AF16	*0A4*	*->2950*	*[GETPC]*
	AF17	*3E5*	*?NC XQ*	
	AF18	*0A8*	*->2AF9*	*[SKPLIN]*
	AF19	*0DD*	***?NC GO***	
	AF1A	*08E*	*->2337*	*[PUTPC]*

PHI13	**AC00**	**2A0**	**SETDEC**	
	AC01	1A0	A=B=C=0	zero trinity
here!	AC02	35C	PT= 12	
	AC03	050	LD@PT- 1	
	AC04	190	LD@PT- 6	
	AC05	050	LD@PT- 1	
	AC06	210	LD@PT- 8	
	AC07	010	LD@PT- 0	
	AC08	0D0	LD@PT- 3	
	AC09	0D0	LD@PT- 3	
	AC0A	250	LD@PT- 9	
	AC0B	210	LD@PT- 8	
	AC0C	210	LD@PT- 8	
	AC0D	1D0	LD@PT- 7	
	AC0E	110	LD@PT- 4	
	AC0F	250	LD@PT- 9	
	AC10	0EE	C<>B ALL	
	AC11	3E0	RTN	
Header	AC12	*089*	***"I"***	***Golden Ratio***
Header	AC13	*008*	***"H"***	
Header	AC14	*010*	***"P"***	*Ángel Martin*
PHI	**AC15**	**18C**	**?FSET 11**	
	AC16	*3B5*	*?C XQ*	*Stack lift*
	AC17	*051*	*->14ED*	*[R_SUB]*
	AC18	*3E9*	*?NC XQ*	*Golden Ratio in {A,B}*
	AC19	*08C*	*->23FA*	*[GSB000] -> [PHI13]*
	AC1A	0DA	C=B M	
	AC1B	23A	C=C+1 M	*rounded to 10-digits*
	AC1C	*0E8*	***WRIT 3(X)***	
	AC1D	3E0	RTN	

Header	AF84	*0BF*	***"?"***	
Header	AF85	*005*	***"E"***	
Header	AF86	*00E*	***"N"***	***RTN Stack Empty?***
Header	AF87	*014*	***"T"***	
Header	AF88	*012*	***"R"***	*Doug Wilder*
RTNE?	**AF89**	**338**	**READ 12(b)**	
	AF8A	01C	PT= 3	
	AF8B	04A	C=0 PT<-	*clear current PC*
	AF8C	00E	A=0 ALL	
	AF8D	*0B1*	***?NC GO***	*checks if A=C*
	AF8E	*05A*	*->162C*	*[XYN]*

Header	AF8F	*08F*	*"O"*	
Header	AF90	*007*	*"G"*	***Pop Address (FOCAL)***
Header	AF91	*03E*	*">"*	
Header	AF92	*011*	*"Q"*	*PPCJ V13 N2 p14*
Header	AF93	*018*	*"X"*	*Håkan Thörngren*
XQ>GO	**AF94**	**2F8**	**READ 11(a)**	
	AF95	0EE	C<>B ALL	
	AF96	338	READ 12(b)	
	AF97	0AE	A<>C ALL	
	AF98	29C	PT= 7	
	AF99	3EA	LSHFA PT<-	
	AF9A	3EA	LSHFA PT<-	
	AF9B	3EA	LSHFA PT<-	
	AF9C	3EA	LSHFA PT<-	
	AF9D	0AE	A<>C ALL	
	AF9E	01C	PT= 3	
	AF9F	0CA	C=B PT<-	
	AFA0	07C	RCR 4	
	AFA1	328	WRIT 12(b)	
	AFA2	0CE	C=B ALL	
	AFA3	04A	C=0 PT<-	
	AFA4	07C	RCR 4	
	AFA5	***073***	***JNC +14d***	
Header	AFA6	*08E*	*"N"*	
Header	AFA7	*014*	*"T"*	***Clear RTN stack***
Header	AFA8	*012*	*"R"*	
Header	AFA9	*00C*	*"L"*	
Header	AFAA	*003*	*"C"*	*Ángel Martin*
CLRTN	**AFAB**	**338**	**READ 12(b)**	
	AFAC	0E0	SLCT Q	
	AFAD	2DC	PT= 13	
	AFAE	0A0	SLCT P	
	AFAF	05C	PT= 4	
	AFB0	052	C=0 P-Q	
	AFB1	328	WRIT 12(b)	
	AFB2	04A	C=0 PT<-	
	AFB3	2E8	WRIT 11(a)	
	AFB4	**3E0**	**RTN**	

Header	AF84	*0BF*	*"?"*	
Header	AF85	*005*	*"E"*	
Header	AF86	*00E*	*"N"*	***RTN Stack Empty?***
Header	AF87	*014*	*"T"*	
Header	AF88	*012*	*"R"*	*Doug Wilder*
RTNE?	**AF89**	**338**	**READ 12(b)**	
	AF8A	01C	PT= 3	
	AF8B	04A	C=0 PT<-	*clear current PC*
	AF8C	00E	A=0 ALL	
	AF8D	*0B1*	***?NC GO***	*checks if A=C*
	AF8E	*05A*	*->162C*	*[XYN]*

Header	ABBC	*08D*	***"M"***	
Header	ABBD	*003*	***"C"***	***Least Common Multiple***
Header	ABBE	*00C*	***"L"***	*Ángel Martin*
LCM	**ABBF**	**108**	**SETF 8**	
	ABC0	*02B*	*JNC +05*	*[MAIN]*
Header	ABC1	*084*	***"D"***	
Header	ABC2	*003*	***"C"***	***Greatest Common Divisor***
Header	ABC3	*007*	***"G"***	*Ángel Martin*
GCD	**ABC4**	**104**	**CLRF 8**	
MAIN	ABC5	*11D*	*?NC XQ*	*Naturalize inputs*
	ABC6	*134*	*->4D47*	*[NATXY]*
	ABC7	0B8	READ 2(Y)	
	ABC8	10E	A=C ALL	
	ABC9	0F8	READ 3(X)	
LOOP	ABCA	158	M=C ALL	*M = Mk-1*
	ABCB	070	N=C ALL	C in N
	ABCC	044	CLRF 4	
	ABCD	*171*	*?NC XQ*	*C=MOD<A,C>*
	ABCE	*064*	*->195C*	*[MOD10]*
	ABCF	2EE	?C#0 ALL	*Mk = MOD(Mk-1, Mk-2)*
	ABD0	***02B***	***JNC +05***	
	ABD1	10E	A=C ALL	*put Mk in A*
	ABD2	198	C=M ALL	*bring Mk-1 to C*
	ABD3	0AE	A<>C ALL	*Mk-1 in A, MK in C*
	ABD4	*3B3*	*JNC -10d*	*loop back*
EXIT	ABD5	*01D*	*?NC XQ*	*Adds normalized A and C*
	ABD6	*060*	*->1807*	*[AD2_10]*
	ABD7	***10C***	***?FSET 8***	
	ABD8	***04B***	***JNC + 09***	
	ABD9	*239*	*?NC XQ*	
	ABDA	*060*	*->188E*	*[ON/X13*
	ABDB	0B8	READ 2(Y)	
	ABDC	*13D*	*?NC XQ*	
	ABDD	*060*	*->184F*	*[MP1_10]*
	ABDE	0F8	READ 3(X)	
	ABDF	*13D*	*?NC XQ*	
	ABE0	*060*	*->184F*	*[MP1_10]*
DONE	ABE1	*369*	***?NC GO***	*Overflow, DropST, FillXL & Exit*
	ABE2	*002*	*->00DA*	*[NFRXY]*

Header	A431	0B2	"2"	
Header	A432	002	"B"	
Header	A433	024	"$"	
$B2	A434	304	CLRF 1	increase level
	A435	02B	JNC +05	[MRG1]
Header	A436	0B2	"2"	
Header	A437	005	"E"	
Header	A438	024	"$"	
$E2	A439	308	SETF 1	decrease level
MRGE1	A43A	004	CLRF 3	two pointers
	A43B	05B	JNC +11d	[MERGE]
Header	A43C	0B3	"3"	
Header	A43D	002	"B"	
Header	A43E	024	"$"	
$B3	A43F	304	CLRF 1	increase level
	A440	02B	JNC +05	[MRG2]
Header	A441	0B3	"3"	
Header	A442	005	"E"	
Header	A443	024	"$"	
$E3	A444	308	SETF 1	decrease level
MRGE2	A445	008	SETF 3	three pointers
MERGE	A446	30C	?FSET 1	decrease level?
	A447	023	JNC +04	no, skip
	A448	379	PORT DEP:	yes, restore one RTN level
	A449	03C	XQ	and decrease pointers
	A44A	08C	->A48C	[LVDN]
INCRS	A44B	2A0	SETDEC	
	A44C	04E	C=0 ALL	
	A44D	35C	PT=12	
	A44E	00C	?FSET 3	tree pointers?
	A44F	01B	JNC +03	no, skip
C=3	A450	0D0	LD@PT- 3	yes, C=3
	A451	013	JNC +02	
C=2	A452	090	LD@PT- 2	C=2
BOTH	A453	30C	?FSET 1	ending?
	A454	013	JNC +02	no, skip
END	A455	2BE	C=-C-1 MS	yes, sign change
BEGIN	A456	268	WRIT 9(Q)	save for later
	A457	00C	?FSET 3	three pointers?
	A458	09B	JNC +19d	no, skip the third
	A459	10E	A=C ALL	offset to A
	A45A	1F8	READ 7(O)	
	A45B	01D	?NC XQ	add/subtract offset
	A45C	060	->1807	[AD2_10]
	A45D	1E8	WRIT 7(O)	
	A45E	30C	?FSET 1	decrease?
	A45F	067	JC +12d	yes, no need to save sp3
	A460	260	SETHEX	
	A461	0F8	READ 3(X)	get current m
	A462	070	N=C ALL	store in N
	A463	1F8	READ 7(O)	**sp3 pointer**

	A464	361	?NC XQ	check for valid IND addr
	A465	138	->4ED8	[EXIST40] - leaves ADR in A.X
	A466	029	?NC XQ	Write N in ADR(A.X)
	A467	124	->490A	[WRTADR]-1
	A468	179	?NC XQ	swap sp3 and sp1
	A469	120	->485E	[X<>Y4
	A46A	2A0	SETDEC	current "n" in X
TWOPTS	A46B	278	READ 9(Q)	get offset
	A46C	10E	A=C ALL	put in A for math
	A46D	1B8	READ 6(N)	stack pointer sp1
	A46E	01D	?NC XQ	add/subtract offset
	A46F	060	->1807	[AD2_10]
	A470	1A8	WRIT 6(N)	
	A471	30C	?FSET 1	decrease?
	A472	**057**	**JC +10d**	yes, no need to save sp1
	A473	260	SETHEX	
	A474	0F8	READ 3(X)	get current n
	A475	070	N=C ALL	store in N
	A476	1B8	READ 6(N)	**sp1 pointer**
	A477	361	?NC XQ	check for valid IND addr
	A478	138	->4ED8	[EXIST40] - leaves ADR in A.X
	A479	029	?NC XQ	Write N in ADR(A.X)
	A47A	124	->490A	[WRTADR]-1
	A47B	2A0	SETDEC	
DECRS	A47C	278	READ 9(Q)	get offset
	A47D	10E	A=C ALL	
	A47E	178	READ 5(M)	stack pointer #2
	A47F	01D	?NC XQ	add/subtract offset
	A480	060	->1807	[AD2_10]
	A481	168	WRIT 5(M)	
	A482	260	SETHEX	
	A483	30C	?FSET 1	decrease?
	A484	360	**?C RTN**	no, end here.
LVUP	**A485**	**2F8**	**READ 11(a)**	yes, back up RTN
	A486	070	N=C ALL	
	A487	178	READ 5(M)	
o IND_M	A488	361	?NC XQ	check for valid IND addr
	A489	138	->4ED8	[EXISTS40] - leaves ADR in A.X
	A48A	029	**?NC GO**	Write N in ADR(A.X)
	A48B	126	->490A	[WRTADR]-1
LVDN	**A48C**	**178**	**READ 5(M)**	
	A48D	38D	?NC XQ	Convert it to hex - uses F8
wo STK levels !	A48E	008	->02E3	[BCDBIN] - uses F8
	A48F	369	?NC XQ	check for valid IND addr
	A490	138	->4EDA	[EXIST44] - leaves ADR in A.X
rom IND_M	A491	0A6	A<>C S&X	
	A492	270	RAMSLCT	selects IND_M
	A493	038	**READATA**	
	A494	070	N=C ALL	store in N
	A495	046	C=0 S&X	
	A496	270	RAMSLCT	Select Chip0
	A497	0B0	C=N ALL	
	A498	2E8	WRIT 11(a)	write in RTN STK
	A499	**3E0**	**RTN**	

Library Subroutines called by the admin functions

	490A	0A6	A<>C S&X	*put adr in C[S&X]*
WRTADR	**490B**	**270**	**RAMSLCT**	*select register*
	490C	0F0	C<>N ALL	*get "EMANF" to C*
WRTSEL	**490D**	**2F0**	**WRTDATA**	*store in buffer*
SELRAM	490E	046	C=0 S&X	
	490F	270	RAMSLCT	*select Chip0*
	4910	3E0	RTN	done.

X<>Y4	**485E**	*149*	*?NC XQ*	*Select Chip0*
	485F	*024*	*->0952*	*[ENCP00]*
X<>Y0	4860	**0B8**	**READ 2(Y)**	*assumes Chip0 is selected*
	4861	10E	A=C ALL	
	4862	*3D5*	*?NC GO*	
	4863	*052*	*->14F5*	*[HALFXY]*

EXIST40	**4ED8**	*38D*	*?NC XQ*	*Convert it to hex - uses F8*
	4ED9	*008*	*->02E3*	*[BCDBIN] - uses F8*
EXIST44	**4EDA**	**0A6**	**A<>C S&X**	*; Save it in A*
	4EDB	*03B*	*JNC +07*	*; skip status register check*
VALSTK	**4EDC**	130	LDI S&X	
	4EDD	070	***<ST mask>***	
	4EDE	306	?A<C S&X	*; Is this a status register?*
	4EDF	*01F*	*JC +03*	*; No, check if exists*
	4EE0	1C6	A=A-C S&X	*; Point to status regs*
	4EE1	*3A0*	*?NC RTN*	*; Skip register check*
	4EE2	*255*	*?NC GO*	*check for valid addr*
	4EE3	*10E*	*->4395*	*[VALID]*

VALID	**4395**	**046**	**C=0 S&X**	
	4396	270	RAMSLCT	*Select Chip0*
	4397	378	READ 13(c)	
	4398	03C	RCR 3	*Obtain R00 address*
	4399	146	A=A+C S&X	*absolute address: R00+Last#*
VALD10	**439A**	130	LDI S&X	
	439B	200	CON: 512	*max reg# on the CX +1*
	439C	306	?A<C S&X	*Carry if address<512*
	439D	*381*	*?NC GO*	*Displays"NonExistent"*
	439E	*00A*	*->02E0*	*[ERRNE]*
	439F	**3E0**	**RTN**	***Returns with addr in A***