Programmable Arbitrary Precision Calculator for Windows

S. Jason Olasky
874 New Mark Esplanade
Rockville, MD 20850
301-294-9419
Compuserve [70471,2501]

Features
Primitives
General Notes
Standard Mathematical Functions
Stack and Register Operations
Financial Functions
Calendar and Time Functions
Support for Hex, Octal, and Binary Numbers
Miscellaneous Operations
Programming
Financial and Date Calculators
Options Dialog
Stack and Register Windows
INI File
Registration

Features

Arbitrary precision, up to 1075 digits
Arbitrary number of decimal places with rounding
Fully editable input
Commas allowed in input
Recall of previous input lines
Error checking
Virtual stack and memory register array
Programmable
Use an editor of your choice
Standard financial functions
Date functions based on Julian Day Number
Hexadecimal, Octal, and Binary conversions

Primitives

OPCODE	FUNCTION
+	Add
-	Subtract
*	Multiply
/	Divide
^	Power
%	PerCent
\	Reciprocal
ABS	AbsoluteValue
ACOS	ArcCosine
ASIN	ArcSine
ATAN	ArcTangent
BE	Toggle BE
BIN	ShowBinary
CD	Toggle CD
CF	Compounding Frequency
CHS	ChangeSign
CLST	ClearStack
CLRF	ClearFinReg
CLRG	ClearRegisters
CLS	ClearScreen
COS	Cosine
DATE	Date
DEG	SetDegrees
D>R	ConvertDegreesToRadians
DOW	DayOfTheWeek
DROP	Drop
DUP	Dup
E	e
EXP10	Exp10
EXPE	ExpE
FACT	Factorial
FIX	FixNotation
FRACT	FractionPart
FV	Future Value
HEX	ShowHex
HMS	HMS
HRS	HRS

I	Interest Rate
IND	SetIndirectFlag
INT	IntegerPart
JDN	Julian Day Number
LOG	Log
LN	Ln
MOD	Modulo
N	N
OCT	ShowOctal
OVER	Over
PF	Payment Frequency
PI	Pi
PICK	Pick
PMT	Payment
PREC	SetPrecision
PV	Present Value
RAD	SetRadians
R>D	ConvertRadiansToDegrees
RCL	Recall
RCLM	RecallMultiple
ROLL	Roll
ROT	Rot
SCI	SciNotation
SHOWS	ShowStack
SIN	Sine
SQR	Square
SQRT	SquareRoot
STO	Store
STOM	StoreMultiple
SWAP	Swap
TAN	Tangent
VIEW	ViewReg
XCH	Exchange

General Notes

This is a programmable arbitrary precision RPN (reverse polish notation) line oriented calculator, which includes financial and date functions in addition to many standard mathematical functions. The extended precision math routines are derived from the Bigcalc extended precision calculator written by Judson D. McClendon. Virtual arrays are used for the stack and memory registers, so there are no effective limits on the stack size or number of registers. One register requires slightly more than 1K, so 1.1MB of disk storage will suffice for 1000 registers.

This calculator is programmable, using a notation that is a hybrid between that used in the HP-41 calculator and FORTH. All opcodes are case insensitive. One character opcodes, i.e. +, -, *, /, \, ^ , % do not need to be separated from the previous number or operator by a space, but all other opcodes do. Thus 123 3 4+^ is valid but 123 3STO will be flagged as an error. Numbers may be preceded by a + or - sign.

The stack is based on the FORTH model rather than the HP calculator model; that is, the stack expands and contracts as numbers are entered or result from operations upon numbers already on the stack. As a general rule, any operation upon one or more numbers removes those numbers from the stack. To re-utilize a number, additional copies may be added to the stack with stack operations or the number may be stored in a register. Certain stack operations refer to a number on the stack relative to the top of the stack. Thus the topmost number is stack[0], the nextmost stack[1], etc. See the pick and roll operators for examples.

The prompt shows the number of elements on the stack. Thus the prompt 2> indicates that there are two numbers on the stack. If the stack is non-empty, the number on top of the stack will be displayed before the prompt. The stack is implemented as a virtual array, so there are no limits on the number of elements on the stack. Some stack operations allow indirect references, that is the number on the stack is used as an index to a register the contents of which will be utilized for the operation.

There are an unlimited number of memory registers available, as the memory registers are stored as a virtual array in a disk file. See "Virtual Arrays in C" by Mark Tichenor, published in the May 1988 issue of Dr. Dobb's Journal. This means that some register operations may be slowed due to the need for disk access; however, a ramdisk may be used by setting an environment variable named TEMP to the RAMDISK drive. Some operations that use a block control word are limited to a maximum of 1000 registers (000-999);

The financial functions utilize the first ten registers, either to save financial variables or for scratch registers, so if you want to store data that won't be overwritten by the financial functions, you will need to use higher numbered registers.

Registers are indexed using standard C array notation, so the first register is reg[0] the second reg[1], etc. The array index is retrieved from the stack, so to store the number on top of the stack in register 0, the command would be 0 STO, with the result that the size of the stack would be reduced by one. To retrieve this number, the command would be 0 RCL, and the number that was stored in reg[0] would then be the topmost number on the stack, which has increased in size by 1.

The trigonometric functions can operate with angles expressed as either degrees or radians. An indicator on the status bar indicates which mode has been set. The default mode is to express angles in radians. To set the angles mode to degrees, enter deg, and to set the angles mode to radians, enter rad. Degrees may be converted to radians and vice-versa using the D>R and R>D operators.

The default precision is 20 digits, and the default number of decimal places for fixed notation is 2. The precision and number of decimal places may be changed using the prec, fix, and sci operations. Increasing the precision will slow down calculations. Numbers may be entered using either fixed or scientific notation, eg 123,456.789 or 1.23456789E5, and commas may be used.

The screen buffer will hold between 500 and 9999 lines, depending on the option set. To go to the top of the buffer, use CTRL-Home and to go to the bottom of the buffer use CTRL-End. The arrow, pg-up, and pg-down keys work as expected, except that the left and right arrow keys only apply to the current input line. To recall previous entry lines, use the CTRL-uparrow and CTRL-downarrow.

Standard Mathematical Functions

The following descriptions of the operations implemented include stack diagrams showing the effect of the operation on the stack. In these diagrams the topmost element of the stack is on the right, the elements on the stack are numbered to indicate the order they were placed on the stack, and the result of the operation is shown.

+	Replaces the top two numbers on the stack with their sum. n1 n2 --- n1+n2
-	Replaces the top two numbers on the stack with their difference . n1 n2 --- n1-n2
*	Replaces the top two numbers on the stack with their product n1 n2 --- n1*n2
/	Replaces the top two numbers on the stack with the result of dividing the second from the top by the topmost. n1 n2 --- n1/n2
^	Replaces the top two numbers on the stack with the result of raising the second from the top to the power of the topmost. n1 n2 --- n1^n2
%	Replaces the top two numbers on the stack with the result of multiplying the second from the top by the topmost and then dividing by 100. n1 n2 --- n1*n2/100. Note: To add or subtract a percentage to a number, first duplicate the number then calculate the percentage and add or subtract it. Thus to add 15% to a number use the sequence of operations: dup 15%+
\	Replaces the top number on the stack with its inverse. n1 --- 1/n1
ABS	Replaces the top number on the stack with its absolute value. n1 --- abs(n1)
SQRT	Replaces the top number on the stack with its square root. n1 --- sqrt(n1)
SQR	Replaces the top number on the stack with its square. n1 --- sqr(n1)
FACT	Replaces the top number on the stack with its factorial. n1 --- n1!
INT	Replaces the top number on the stack with its integer part. n1 --- int(n1)

FRACT	Replaces the top number on the stack with its fractional part. n1 --- fract(n1)
MOD	Replaces the top two numbers on the stack with the second from the top modulo the topmost. n1 n2 --- mod(n1,n2)
SIN	Replaces the top number on the stack with its sine. n1 --- sin(n1)
ASIN	Replaces the top number on the stack with its arcsine. n1 --- arcsin(n1)
COS	Replaces the top number on the stack with its cosine. n1 --- cos(n1)
ACOS	Replaces the top number on the stack with its arccosine. n1 --- arccos(n1)
TAN	Replaces the top number on the stack with its tangent. n1 --- tan(n1)
ATAN	Replaces the top number on the stack with its arctangent. n1 --- arctan(n1)
LOG	Replaces the top number on the stack with its logarithm to the base 10. n1 --- log10(n1)
EXP10	Replaces the top number on the stack with 10 raised to the number. n1 --- 10^n1
LN	Replaces the top number on the stack with its natural logarithm. n1 --- ln(n1)
EXPE	Replaces the top number on the stack with e raised to the number. n1 --- e^n1
PI	Adds pi to the stack. --- pi
E	Adds e to the stack. --- e
CHS	Changes the sign of the top number on the stack. n1 --- -n1
SCI	The integer part of the number on top of the stack will be used to set the number of decimal places and numbers will be displayed in scientific notation based on the precision and number of decimal places set.

n1 --.

FIX

The integer part of the number on top of the stack will be used to set the number of decimal places and numbers will be displayed as fixed numbers based on the precision and number of decimal places set. 0 fix means don't display a decimal point or any fractional part of the number.

n1 --

PREC

The integer part of the number on top of the stack will be used to set the precision of succeeding calculations. It will not affect results already derived. Numbers will be displayed in fixed or scientific format as previously set. If scientific notation is being used, the precision will not be lower than the number of decimal places plus 3.

n1 ---

Stack and Register Operations

CLST — Clears the stack. Clearing the stack sets the stack pointer to 0..

n1 ... nm ---

CLRG — Clears all currently allocated registers to zero. Initially there are ten registers. Whenever a register is referenced beyond the currently allocated registers, additional virtual registers are allocated up to and including the new register. Clearing the registers does not reduce the number of registers that have been allocated. Does not affect the stack.

STO — Uses the integer part of the number on top of the stack as an index, and stores the nextmost number on the stack in the indexed register. If you think of the index number as part of the sto command, the effect is really to store the number on top of the stack, but from the point of view of the stack, the index has briefly been added to the stack. If STO is directly followed by one of the operators + - * / % ^, the operation will be performed using the next to last stack entry and the contents of the register, e.g. 1 3 STO+ will increment the contents of register 3 by 1, 10 3 STO/ will divide the contents of register 3 by 10. If a STO operation is preceded by the keyword IND, the contents of the indexed register will be used as an indirect register address.

n1 n2 ---

RCL — Uses the integer part of the number on top of the stack as an index, and recalls the number in the indexed register to the stack. Note that unlike storing a number which removes the number from the stack, recalling a number leaves it in the register. Has the effect of replacing the number on top of the stack by the contents of the register indexed by that number. If a RCL operation is preceded by the keyword IND, the contents of the indexed register will be used as an indirect register address.

n1 --- n2

STOM — Stores the contents of the stack based on the block control word. The block control word is retrieved from the register on top of the stack. Starts with the top element on the stack not counting the block control word pointer and stores it in the first register in the block and increments until done. At the end, numbers will be stored in the reverse order they were entered on stack.

n1 n2 ... nm r ---

RCLM — Recalls the contents of a block of registers to the stack based on the block control word. The block control word is retrieved from the register on top of the stack. Starts with the last register in the block and decrements to restore the stack in the same order it was originally.

r --- n1 n2 ... nm

DUP — Adds a copy of the number on top of the stack to the stack.

n1 -- n1 n1

DROP — Deletes the number on top of the stack from the stack.

n1 ---

SWAP — Swaps the two numbers on top of the stack.

n1 n2 --- n2 n1

ROT — Rotates the third from the top of the stack to the top of the stack. Equivalent to 2 Roll.

n1 n2 n3 --- n2 n3 n1

OVER — Adds a copy of the second number from the top to the stack.

n1 n2 -- n1 n2 n1

PICK — Uses the integer part of the topmost number on the stack to index a number on the stack. A copy of the indexed number is added to the stack. Like STO, the index number is really part of the pick command, but from the point of view of the stack, the index is briefly the top of stack element. Indexing for PICK is zero-based, with the top of the stack (index omitted) being 0, etc.

Example: *3 PICK*

n1 n2 n3 n4 3 --- n1 n2 n3 n4 n1

ROLL — Uses the integer part of the topmost number on the stack to index a number on the stack. The indexed number is removed from the stack and moved to the top of the stack, Like STO, the index number is really part of the pick command, but from the point of view of the stack, the index is briefly the top of stack element. Indexing for ROLL is zero-based, with the top of the stack (index omitted) being 0, etc.

Example: *3 ROLL*

n1 n2 n3 n4 3 --- n2 n3 n4 n1

XCH — Exchanges the number on the stack below the index with the contents of the indexed register.

Example: *3 XCH*

n1 3 -- n2 where n2 was the contents of register 3 and register 3 now contains n1.

Financial Functions

The financial functions, N, I, PV, PMT, and FV are those first implemented in the HP-92, as modified for the HP-41 in the PPC ROM. A full explanation is provided in the PPC ROM User's Manual (1981), from which the algorithm was derived and the examples taken, as well as in many standard references.

The rule for financial calculations using these functions is that money paid out is considered negative and money received is considered positive in sign.

There are five financial variables. N (number of periods), I (interest rate), PV (present value), PMT (payment), and FV (future value), and given any three of them, the other two can be calculated. This calculator includes two additional parameters, CF, compounding frequency (number of times the interest rate is compounded during the period for which the interest rate is I%), and PF, payment frequency (number of payment periods during the period for which the interest rate is I%). These additional parameters simplify the solution of some complex financial problems. In addition, there are toggles for beginning of period/end of period payment and continuous/discrete compounding.

The financial functions either store a number in a register (N - 1, I - 2, PV - 3, PMT - 4, FV - 5) or calculate that variable based on the other numbers entered, depending on whether the data entry flag is set. This flag is automatically set whenever a number is entered or calculated and turned off when a financial operation is performed. See the examples for details. A financial calculator is provided to simplify use of the financial functions.

N	Either stores the number of periods in register 1, or calculates N based on two other values. n1 --- --- n1
I	Either stores the interest rate in register 2, or calculates I. n1 --- --- n1
PV	Either stores the present value in register 3, or calculates PV. n1 --- --- n1
PMT	Either stores the payment in register 4, or calculates PMT. n1 --- --- n1
FV	Either stores the future value in register 5, or calculates FV. n1 --- --- n1
CF	Sets the compounding frequency. n1 --
PF	Sets the payment frequency. n1 --
BE	Toggles the beginning/end payment flag. Does not affect the stack.
CD	Toggles the continuous/discrete payment flag. Does not affect the stack.
CLRF	Clears registers 1 through 5 used for financial variables, sets CF=PF=1, sets

BE to E and CD to D.

Examples

Monthly payment.

A couple purchases a $50,000 house, borrowing $40,000 at 8.5% for 30 years less one month. What is their monthly payment.

clrf 40,000 pv 8.5 12/ i 30 12 1- n pmt*
result is PMT = $307.75

Internal rate of return.

The couple above then sold their house 18 months later, netting $25,000. At what annual interest rate would they have had to invest their original $10,000 and $307.75 monthly payments to obtain $25,000.

*clrf 18 n 25,000 fv -10,000 pv -307.75 pmt i 12**
result is I = 38.51%

Simple interest.

Find the annual simple interest rate (%i) for an $800 loan to be repaid at the end of one year with a single payment of $896.

clrf 1 n -800 pv 896 fv i
result is APR = 12.0%

Compound interest

Find the future value of an $800 loan after one year at a nominal rate of 12% compounded monthly. No payments are specified, so the payment frequency is set equal to the compounding frequency.

clrf 12 n 12 dup cf pf 12 i -800 pv fv
result is FV = 901.46

Periodic payment.

Find the monthly end of period payment required to fully amortize the loan in the preceding example. A fully amortized loan has a future value of zero. Use data retained from preceding example.

0 fv pmt
result is PMT = $71.08

Conventional mortgage.

Find the number of monthly payments necessary to fully amortize a loan of $100,000 at a nominal rate of 13.25% compounded monthly, if end of period payments of $1,125.75 are made.

clrf 12 dup cf pf 13.25 i 100,000 pv -1,125.75 pmt n
result is N = 360.10

Final payment.

Using the same data as in the preceding example, find the amount of the final payment if n is changed to 360. The final payment is equal to the regular payment plus any balance remaining (FV) at the end of the last period.

360.0 n fv 4 rcl+
result is final PMT = $1,234.62

Balloon payment.

On long term loans, small changes in the periodic payments can result in large changes in the future value. If the monthly payment in the preceding example is rounded down to $1,125 what is the additional balloon payment due with the final payment?

-1,125 pmt fv
result is balloon payment of $3,579.99

Canadian mortgage.

Find the monthly end-of-period payment necessary to fully amortize a 25 year $85,000 loan at 11% compounded semiannually.

clrf 2 cf 12 pf 25 12 n 11 i 85,000 pv pmt*
result is PMT=818.15

European mortgage.

The "effective annual rate" (EAR) is used in some European countries instead of the nominal annual rate commonly used in the US and Canada. For a 30 year $90,000 mortgage at 14% EAR compute the monthly end-of- period payments, noting that when using an EAR, the compounding frequency is set to 1.

clrf 12 pf 360 n 14 i 90,000 pv pmt
result is PMT=$1,007.88

Bi-weekly savings.

Compute the future value of bi-weekly savings of $100 for three years at a nominal annual rate of 5.5% compounded daily. Note that it is necessary to toggle the BE flag to beginning of period.

clrf be 365 cf 26 dup pf 3 n 5.5 i -100 pmt fv*

result is FV=$8,489.32

Present value of an annuity.

What is the present value of $500 to be received at the beginning of each quarter over a 10 year period if money is being discounted at a 10% nominal annual rate compounded monthly. Note that it is necessary to toggle the BE flag.

clrf be 12 cf 4 dup pf 10 n 10 i 500 pmt pv*
result is PV=$12,822.64

Balloon payment.

Compute the monthly end-of-period payment on a 3 year $20,000 loan at 15% nominal annual rate compounded monthly, with a $10,000 balloon payment due at the end of the 37th period. Note that the balloon payment must be discounted one period to make it coincide with the last regular payment.

clrf 12 dup dup cf pf 3 n 15 i 20,000 pv pmt*
note: at this point the effective monthly interest rate as a decimal fraction is in register 6; throw away the number on the stack (monthly payment without the balloon) and continue as follows*:*
drop -10,000 6 rcl 1+ / fv pmt
result is PMT=474.39

Effective rate using a 365/360 basis.

Compute the effective annual rate (%APR) for a nominal annual rate of 12% compounded on a 365/350 basis.

clrf 3 fix 365 dup n cf 360 pf 12 i -100 pv fv 3 rcl +
result is APR=12.935%

Mortgage with points.

What is the true APR of a 30 year, $75,000 loan at a nominal rate of 13.25% compounded monthly, with monthly end-of-period payments of $844.33 if 3 points are charged? The PV must be reduced by the dollar value of the points to establish an effective PV. Because the payments remain the same, the true APR will be higher than the nominal rate.

clrf 12 dup dup cf pf 30 n 75,000 dup 3% - pv -844.33 pmt i*
result is APR=13.69%

Equivalent payments.

Find the equivalent monthly payment required to amortize a 20 year $40,000 loan at 10.5% nominal annual rate compounded monthly, with ten annual payments of $5,029.71 remaining. Compute the PV of the remaining annual payments then change n and PF to a monthly basis and compute the equivalent monthly payment.

clrf 12 cf 10 n 10.5 i -5,029.71 pmt pv

this calculates the PV of remaining payments which is $29,595.88

12 dup pf 10 n pmt*

result is monthly PMT=$399.35

Perpetuity with continuous compounding.

If you can purchase a single payment annuity with an initial investment of $60,000 that will be invested at a 15% nominal annual rate compounded continuously, what is the maximum monthly return you can receive without reducing the principal. If the interest rate is constant and the principal is not disturbed the payments can go on indefinitely. Note that the term n of a perpetuity is immaterial and can be set to any non-zero value.

clrf cd 12 dup pf n 15 i 60,000 dup fv chs pv pmt

result is PMT=$754.71

Calendar and Time Functions

The calendar functions JDN and Date are inverses. JDN computes the Julian Day Number for a given calendar date and Date converts a JDN to a calendar date. The valid range for dates is from March 1 of the year 0 CE (Common Era, also known as AD) to sometime in the far future. All dates are assumed to follow the Gregorian calendar, which was originally devised in 1582, adopted in 1752 by the British Empire, including the then American colonies, and as late as 1927 for Turkey. It is possible to use a somewhat more complicated algorithm to allow for dates in either the Gregorian or the Julian calendars, and including dates BCE (Before the Common Era, also known as BC) but this program does not do so.

The Julian Day Number should not be confused with the Julian Calendar. The Julian Day Number is the number of whole days that have elapsed since a certain reference time in the past. The JDN is widely used in astronomy and elsewhere in calculations involving dates. The reference time is January 1, 4713 BCE, Julian Calendar, at noon.

It should be noted that the day of the week corresponding to a given date can be easily calculated given the JDN. The day of the week is (JDN + 1) MOD 7, where 0 = Sunday, 1 = Monday, etc.

A Date Calculator is provided to simplify use of the calendar functions.

JDN	Converts a calendar date in the form YYYY.MMDD to the corresponding JDN. n1 --- n1
DOW	Displays the day of the week corresponding to the julian day number on the stack. n1 ---
DATE	Converts a JDN to a calendar date in the form YYYY.MMDD n1 --- n2

Examples

The attack on Pearl Harbor occured on December 7, 1941. What day of the week was it?

1941.1207 jdn dow
result is Sunday

What day of the year is July 4, 1993?

1993.0704 jdn 1992.1231 jdn -
result is July 4, 1993 is the 185th day of the year

What day is 90 days after July 4, 1993?

4 fix 1993.0704 jdn 90+ date
result is 1993.1002 or October 2

The time functions HMS and HRS allow converting between times expressed as decimal numbers and times expressed as hours, minutes, seconds. The time functions have precision only to the hundredth of a second.

HMS	Converts time expressed as a decimal number H.DDDDDD to H.MMSSCC, where H is hours, DDDDDD is some fraction of an hour, MM is minutes, SS is seconds, and CC is hundredths of a second. Result is rounded to CC. H.DDDDDDDD --- H.MMSSCC
HRS	Converts a time expressed as H.MMSSCC to H.DDDDDD H.MMSSCC --- H.DDDDDD

Support for Hex, Octal, and Binary Numbers

PAPCW is not a programmer's calculator, but limited support is provided for hexadecimal, octal, and binary numbers. Numbers may be displayed in hexadecimal, octal, or binary by using the HEX, OCT, or BIN keywords. Only the integer parts of the numbers will be used. Hexadecimal, octal, and binary integers are entered using the prefix 0x, 0o, or 0b, respectively, eg 0xABC, 0o5703 or 0b11001. The base conversion code uses long doubles (which have 64 bit mantissas) as an intermediate form, so the base conversions are limited to binary integers of 64 or fewer bits (16 hexadecimal digits).

Miscellaneous Operations

CLS	Clears the screen. Does not affect the stack.
VIEW	Displays selected registers based on the block control word bbb.eeeii where bbb is the first register to view, eee is the last register to view, and ii is the increment. The block control word is retrieved from the register on top of the stack. r --- Note: Registers may be continuosly monitored by using the view registers menu item.
SHOWS	Displays the stack from top to bottom. the stack is not affected. Note: The stack may be continuously monitored by using the view stack menu item.

Programming

Programs may be written using any editor and then loaded and executed. The default editor is NOTEPAD. Several sample programs are provided, including a mortgage amortization schedule, a linear regression program, a root finder, and a conversions program. A program may be automatically loaded and run by running PAPCW with the program name as an argument. Execution will begin with the first statement in the program.

Unless stated otherwise, programming operators can not be used interactively, but only within programs. A semicolon is used to begin a comment. Once a semicolon has been parsed, the rest of the line is skipped. For debugging, a single-step option is provided. While single step is enabled, PAPC will pause after each program line is executed, and a dialog message will provide you with options to step, quit which will return you to interactive mode, stop which will put your programm in a stopped state, or display the stack or registers.

LOAD	Loads the program(s) in a file. If you need to use a path in the filename, enclose the name in quotes, eg *load* "*c:\xyz\progname*" .The filename may be any legal DOS filename, although I have used a filetype of PRG in the samples. A file may contain multiple programs. There are no particular formatting requirements, but putting one operation on a line may be helpful for debugging. A program is terminated by an END statement. If you don't include an END statement, one will be appended when the file is loaded. The load command may be executed in interactive mode. Load statements in a program file will be executed immediately, as shown in the sample LOADALL program. Once a program has been loaded, it may be executed by saying XEQ progname where progname is the label of the entrypoint, or just by entering the name of the entrypoint. Example: LOAD xyz. Load may be invoked from the menu or the toolbar.
CAT	Displays a catalog of all labels that have been loaded. May be executed in interactive mode. CAT in a program file will be executed immediately.
CLP	Clears a program and deletes all labels that are in the program from the catalog. May be executed in interactive mode. CLP statements in a program file will be executed immediately. Example: CLP xyz
CLCAT	Clears all programs from the catalog. May only be executed in interactive mode.
LBL	Defines a label and enters it in the catalog. Examples: LBL a, LBL ThisIsAVeryLongLabel, LBL 01
END	Defines the end of a program, and functions like a RTN.
PUTS	Puts a string to the console Example: *PUTS* "*This is a string to put to the console\n*". Note that unlike the C function puts (), a newline is not automatically appended. Use \n or \r for a newline, \t for tab. Set tabs with TABSET, Example: *10 tabset PUTS* "*\tTAB1\tTAB2\tTAB3\n*" The default tab setting is 5.
PROMPT	Puts a string to the console and then gets a line of input which is parsed and executed. If a NULL line is entered (ie, just a <CR>) the next instruction is executed, otherwise the next line is skipped. The reason for this is to provide a mechanism to escape from a data entry loop.

Example:

```
LBL DATAENTRY PROMPT "Enter a number: "
GTO NEXT ; only if a number wasn't entered .
; data entry processing
GTO DATAENTRY
LBL NEXT       ; continue
```

XEQ — Calls a program or subroutine, the next instruction in the calling program will be executed when a RTN or END instruction is executed. May be used in interactive mode to execute a program. Ten levels of subroutines are allowed. XEQ is not required for a previously defined label (backward reference), which will be resolved without it, but is necessary for a label that has not yet been defined (forward reference).

Examples:

interactive	*XEQ myprogram or myprogram*
program	*XEQ subxyz ...*

GTO — Transfers control to a LBL

Example: *GTO xyz*

RTN — Returns from a subroutine call, or to interactive mode if the program was invoked with an initial parameter

JMP — JMP takes the number on top of the stack, or the contents of a register if IND is used, adds that to the program instruction counter and then transfers to that instruction. JMP may be used to implement a simple case mechanism.

Example:

```
LBL test2
PROMPT "Enter a number 1, 2, or 3, or CR to exit: " ; next line executed if just a CR is entered
GTO done ; here if a number was entered
1 < GTO error
3 > GTO error
JMP
GTO one
GTO two
GTO three
```

STOP — Returns immediately to interactive mode. The program can be resumed where it left off using RUN.

RUN — Resume the program where it left off. Can only be used from interactive mode.

PAUSE — Pauses program and throws up the debugging dialog

ISG	Increment and skip if greater. Uses a block control word bbb.eeeii to control program flow. The block control word is retrieved from the register on top of the stack. ii is added to bbb, if the result is greater than eee, the next instruction is skipped. The updated block is stored back in the same register.
DSE	Decrement and skip if less than or equal. Uses a block control word bbb.eeeii to control program flow. The block control word is retrieved from the register on top of the stack. ii is added to bbb, if the result is equal to or less than than eee, the next instruction is skipped. The updated block is stored back in the same register.
PUTSTACK	Puts the number on the stack to the console using the field width (fixed mode only).
WIDTH	Set field width. A field width of 0 indicates no fixed field width (like default interactive mode).
==	Compares the second number on the stack to the number on top of the stack, which is dropped, If the numbers are equal, the next instruction is executed, otherwise it is skipped (DO IF TRUE rule).
!=	Compares the second number on the stack to the number on top of the stack, which is dropped, If the numbers are not equal, the next instruction is executed, otherwise it is skipped (DO IF TRUE rule).
>	Compares the second number on the stack to the number on top of the stack, which is dropped, If the number is > than the comparison number, the next instruction is executed, otherwise it is skipped (DO IF TRUE rule).
>=	Compares the second number on the stack to the number on top of the stack, which is dropped, If the number is >= than the comparison number, the next instruction is executed, otherwise it is skipped (DO IF TRUE rule).
<	Compares the second number on the stack to the number on top of the stack, which is dropped, If the number is < than the comparison number, the next instruction is executed, otherwise it is skipped (DO IF TRUE rule).
<=	Compares the second number on the stack to the number on top of the stack, which is dropped, If the number is <= than the comparison number, the next instruction is executed, otherwise it is skipped (DO IF TRUE rule).

Flag operations

There are ten user flags 0 through 9 that can be used in programs. Flags are indexed by a number on the stack, or indirectly by the contents of a register number.

FS	Set a flag. Examples: 3 FS, 3 IND FS
FC	Clear a flag. FS? Test whether a flag is set, if so execute the next instruction, otherwise skip over it (DO IF TRUE).

FC?	Test whether a flag is clear, if so execute the next instruction, otherwise skip over it (DO IF TRUE).
FS?C	Test whether a flag is set and then clear it, if the flag was set execute the next instruction, otherwise skip over it (DO IF TRUE)
FC?C	Test whether a flag is clear and then clear it, if the flag was clear execute the next instruction, otherwise skip over it (DO IF TRUE)

Financial and Date Calculators

Calculator dialogs are provided to facilitate data entry for the financial and date functions. For the financial registers, operators as well as numbers can be entered and will be interpreted. The date calculator will put the calculated date in the clipboard. The default for the calendar option uses the *PC Magazine* utility **Wincmd** to invoke the Windows calendar and enter the date from the clipboard. If you have a calendar program that can accept a date from the keyboard, you can substitute it for calendar.exe.

Options Dialog

The options dialog allows you to enter the default directory for your PAPCW programs, the program to be used for editing programs, the calendar to be used by the date calculator, and the size of the scroll buffer (minimum of 500, maximum of 9999 lines). The default for the editor uses the *PC Magazine* **Wincmd** utility to invoke the windows notepad with a mask for PRG files. The default for the calendar uses the **Wincmd** utility to invoke the windows calendar with the date that the date calculator has placed in the clipboard.

Stack and Register Windows

There are menu entries available for opening and closing windows to watch the stack and registers. If these windows are open, there will be respectively S and R indicators on the status bar. Clicking on the S or R will bring the stack or register window to the top if that window was hidden by another window.

INI File

PAPCW maintains an INI file in the Windows directory which allows PAPCW to keep track of window positions and program options.

Registration

This program (The Programmable Arbitrary Precision RPN Calculator for Windows) is released as shareware. If you like it and plan to use it regularly, please send $20 to: S. Jason Olasky, 874 New Mark Esplanade, Rockville, MD 20850. The source code is available for an additional $20.

The programmable arbitrary precision rpn calculator for windows and all accompanying materials are provided "as is" without warranty of any kind. The entire risk of using the programmable arbitrary precision rpn calculator is assumed by you. S. Jason Olasky makes no warranty of any kind, express or implied, including but not limited to any warranties of merchantability and fitness for a particular purpose. IN NO EVENT WILL S. JASON OLASKY BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO DAMAGES FOR LOSS OF BUSINESS PROFITS, LOSS OF SAVINGS, BUSINESS INTERRUPTION, AND THE LIKE) ARISING OUT OF YOUR USE OR INABILITY TO USE THE PROGRAM. By using the programmable arbitrary precision rpn calculator for windows, you agree to the above limitations.

Portions of this code were derived from *Bigcalc* by Judson D. McClendon, *Parser* by Lloyd Zusman, and *Varray* by Mark Tichenor.

Notifications of bugs, suggested improvements, or comments in general are welcome.