1966

A study in the construction of a precision arithmetic programming system using list processing techniques

Andrew J. Kasarda
Lehigh University

A STUDY IN THE CONSTRUCTION OF A PRECISION ARITHMETIC PROGRAMMING SYSTEM USING LIST PROCESSING TECHNIQUES

by

Andrew James Kasarda

[illegible]

A Dissertation

Presented to the Graduate Faculty

of Lehigh University

in candidacy for the Degree of

Master of Science

in

Mathematics

Lehigh University

1966

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science.

May 27, 1966
(date)

Gerhard Rayna
Professor in charge

E. H. Cutler for Pitcher
Head of the Department

ACKNOWLEDGEMENTS

The author wishes to express his sincere appreciation to Professor G. Rayna for his valuable suggestions and helpful criticisms.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

This report describes the construction and implementation of the Precision Arithmetic Programming System designed for use on the GE - 225 Data Processing System. The system was written as a collection of subroutines in the LEWIZ programming language. Various list processing techniques were applied in the structuring of the data for the system to provide efficient and dynamic memory allocation. The Precision Arithmetic Programming System provides the ability to perform ordinary arithmetic operations on integer data of any order of magnitude without any loss of precision in the final result.

INTRODUCTION

Computers have a well established reputation for their ability to carry out various arithmetic operations on numbers with great speed and accuracy. Recently, however, digital computers have been applied, with varying degrees of success, to non-numeric types of data processing such as information retrieval, pattern recognition, and decision making. Although the use of digital computers in these areas is still mostly in the experimental stage, various studies have produced some rather useful intermediate results. The studies being carried on by H. Gelernter,[1] J. McCarthy,[2] and others dealing with the processing of lists of non-numeric data have produced a variety of list processing techniques, as well as list processing programming languages like LISP, IPL-V, and SLIP. Although these techniques were used on lists of non-numeric data, they can also be applied quite easily to lists of numeric data.

The purpose of this study was to construct a Precision Arithmetic Programming System (PAPS) for operating exactly on arbitrarily large, but finite, integers. Certain list processing techniques such as push down lists, threaded lists, and stacking proved to be quite useful in the implementation of the Precision Arithmetic Programming System.

TECHNIQUES

INTEGER REPRESENTATION

The Precision Arithmetic Programming System was written in the LEWIZ programming language for the GE-225 digital computer. The basic unit of information in the GE-225 computer is a word which consists of 20 binary digits. Two kinds of numbers can be used in LEWIZ, fixed point numbers which require one word of storage and floating-point numbers which require two words of storage. The largest fixed point number that is allowable in LEWIZ is in absolute value the integer 262,143 while the largest floating point number allowed in LEWIZ is approximately 10^{76}.

Since the PAPS was designed to perform arithmetic on arbitrarily large, finite integers, a system of storing these integers in a manner that would permit each decimal digit to be preserved in context had to be devised. The convention that was adopted was to factor each integer into three digit portions and store these portions, by place-value, as fixed point numbers. That is, each three digit portion of an integer would require one word of storage.

Before going any farther in this discussion, it will be useful to make some definitions. In the PAPS, the basic storage unit is called a data cell. A data cell can store any integer having up to three decimal digits along with a sign. It requires one word of storage. An integer, or number, will refer to an ordered collection of data cells, each containing a three-digit portion of some decimal integer. The ordering is by place value of the decimal integer portions. An integer may consist of just one data cell or it may consist of many data cells. Each data cell can represent any decimal integer whose absolute value is less than or equal to the integer 999.

INTEGER ADDRESSING

In order to be able to store integers of arbitrary size, it is necessary to construct an addressing scheme which will permit storage of decimal integers of various size as well as facilitate their referencing and manipulation, while using as little available memory space as possible.

The method of addressing that this system uses is similar to addressing schemes used in certain list processors. A given data cell is assigned three kinds of addresses: an index, a preceding link, and a next link. The index address is the actual location of a given data cell within a single dimension data array called the

DATA array. The preceding link address is the location (within the DATA array) of the data cell just preceding the given data cell. The preceding link is also a member of a single dimension array called the PLNK array, and this link has the same index within the PLNK array as the given data cell has within the DATA array. The third kind of address, the next link, is the location (within the DATA array) of the data cell just following the given data cell. The next link belongs to a single dimension array called the NLNK array, and its index is the same as the index of the given data cell. This addressing scheme is known as a threaded list structure. All addresses are three digit fixed point numbers ranging in value from 001 to 500, since each of the arrays, the DATA array, the PLNK array, and NLNK array consist of up to 500 words of storage. This is actually an upper limit to the size of the arrays and one may specify the size array desired within this range. A fourth array which is actually part of the DATA array is also required. It is called the list of available space (LOAS) array. This array contains all those indices referencing data cells in the DATA array. Initially, the LOAS array will contain every index in the DATA array, but as integers are input to the system, the DATA array fills up and the LOAS array empties. However, the LOAS array is actually dynamic in operation since as integers are no longer needed in the system, they are deleted and their data cell indices are returned to the LOAS array for use by new integers. The LOAS array is a pushdown list: that is, when an address is returned to the LOAS, the other addresses are pushed down one place and the returned address is put at the top of the LOAS array. Also, as an address is removed from the top of the LOAS, the next available address "pops up" to the top of the list. In other words, the LOAS array is a "first-in-last-out" type of list. Figure 1 illustrates the list structure of the DATA and LOAS arrays.

When refering to an integer it will be necessary to be able to distinguish between two representations. The actual or decimal representation of an integer refers to its familiar arithmetic form, sign preceding its high order digit followed by decimal digits with either a comma or a blank marking successive powers of a thousand. For instance, the integers

-64,721,805	or	-64 721 805
100,216	or	100 216

are given in their actual form. The internal representation of an integer is its cellular form along with its thread addresses. Figure 2 illustrates the internal and actual representations of an integer. Here it can be

	INDEX	DATA	PLNK	NLNK	
	100	372	0	98	Used part of DATA Array
	99	-601	0	97	
	98	21	100	0	(showing 2 numbers)
	97	-436	99	0	
LOAS Top →	96	0	0	95	(LOAS)
	95	0	96	94	Unused part of DATA Array
	94	0	95	93	
	93	.	.	.	
	.	.	.	.	
	.	.	.	.	
	3	0	.	.	
	2	0	3	1	
	1	0	2	0	

Figure 1. THE DATA ARRAY AND LOAS ARRAY

INTERNAL REPRESENTATION

INDEX	DATA	PLNK	NLNK
408	-076	0	327
327	-439	408	328
328	-201	327	329
329	000	328	035
035	-536	329	036
036	-771	035	261
261	-809	036	508
508	-334	261	429
429	-001	508	0

Actual Representation

-076 439 201 000 536 771 809 344 001

Figure 2. Integer Representations

seen that data cell 408 contains the high order three digits (-076) of the above integer. Since no higher order digits exist for this integer, its PLNK value is 0, while the next lower three digit part of the integer is located at data cell 327, that is, its NLNK value is 327. Similarly, data cell 429 contains the lowest order three digit part of the integer. Since no lower digits exist for the integer, its NLNK value is 0, while its PLNK value is 508, that is, the next higher order three digit part of the integer (-334) is stored in the data cell 508. The data cells between these end cells each contain the other three digit parts of the integer. It is clear that this method of integer storage is quite useful since it provides simple accessibility of an integer from either of its ends, its high order end or its low order end, depending on which is needed or desired.

A number is usually a value assigned to a variable by some arithmetic operation. The PAPS system provides the capability of labeling integers with reference names (satisfying the rules for naming variables described in the LEWIZ manual).[3] However, the actual value assigned to the reference name is not the value of the integer, since this would not be possible for such a large number. Instead, a reference key is constructed from the index of the high order data call and the index of the low order data cell of the integer by performing the following calculation:

Reference key = 1000 X high order cell index + low order cell index

This reference key is the value that given to the integer's reference name. Since this value often would be larger than the maximum fixed point number allowable, the key is stored in floating point form. Referencing an integer is accomplished by using the reference name whenever the integer is required. The PAPS system will decode the reference key and locate the requested integer. This produces the same effect as though the actual integer was called directly. For instance, if it were desired to name the integer given above in Figure 2, the reference name MAXN, then the key that would be assigned to MAXN, then the point number 4.0842900+05. Thus the system provides the same flexibility in selecting variable names as that which exists in LEWIZ.

THE SUBROUTINES

The precision arithmetic system is implemented through a collection of LEWIZ subroutines which perform various required tasks such as input and output, arithmetic, and file maintenance. The subroutines are classified according to the kind of task which they perform:

Utility Routines
Input and Output Routines
Arithmetic Routines

The Utility Routines

There are three subclasses of utility routines whose task is to perform the necessary file maintenance operations. Figure 3 gives a complete listing of the subroutines within each of the three subclasses.

The first subclass, the Link Address Routines, are those Utility routines which operate only on link addresses. They locate variable addresses, remove variables from memory, or put new ones into memory. The D V F A. subroutine, when called, will remove a variable from memory. It has one argument, the reference label to be removed from memory, as input. It removes the variable presented simply by returning all link addresses assigned to the variable to the List of Available Space (LOAS). A value of 0 is then assigned as the variable key and an exit

Link Address Routines

DVFA. (Ref. Name)

GETE. (Ref. Name)

GETS. (Ref. Name)

LKIN. (Link Address)

LKOT. (O)

Data Cell Routines

CHOP. (Link Address)

PREZ. (Count, Ref. Name)

RELZ. (Link Address)

REPK. (Ref. Name)

REST. (Count, Ref. Name)

SIGN. (Ref. Name)

Program Initialization Routines

SLAT. (Array Dimension)

Figure 3. THE UTILITY ROUTINES

from the subroutine is made. (A variable whose key is 0 will be recognized as an undefined variable).

Example: Let

INDEX	DATA	PLNK	NLNK
91	609	0	85
85	751	91	77
77	012	85	0

be the internal representation of the integer

609 751 012

Let X be the variable referencing this integer. The key assigned to X would be

X = 9.1077000+04

To remove X from use, the instruction

DVFA. (X)

would be used. The indices 91, 85, and 77 would be returned to the LOAS, and the variable X would be assigned a zero key value, and X would no longer be recognized as a valid variable in the system.

The GETE. subroutine performs the function of obtaining the link address of the data cell containing the low order 3-digits of the integer specified as the input argument to the GETE. subroutine. The GETE. subroutine performs the calculation

KEY - [integer part of (KEY/1000) X 1000]

and obtains as a result a 3-digit number which is the address of the desired data cell. This address is the output value of the subroutine.

Example:

If Y = 3.7105900+05 were the key of an integer, then the instruction

GETE. (Y)

would result in an output value

GETE (1) = 059.

The GETS. subroutine has the function of obtaining the other end of the integer referenced, that is, it gets the link address of the data cell containing the high order 3 digits of the number referenced.

The GETS. subroutine takes the key of the variable specified and performs the calculation

Integer part of (KEY / 1000)

obtaining a 3-digit number which is the address of the desired data cell. This address is the output value of the subroutine.

Example: If JOE = 4.3127900E5 were the key of some integer, then the instruction

GETS. (JOE)

would result in an output value

GETS (1) = 431

The next two subroutines are most often used by the system. They are the LKIN. and LKOT. subroutines.

The Link-In (LKIN.) subroutine has the task of returning a freed link address to the List of Available Space. Its input argument is the line address that is to be returned to the LOAS. There is no output value from the LKIN. subroutine.

Example: If the "top" of the LOAS array were

	INDEX	DATA	PLNK	NLNK
LOAS =	541	0	362	0
	362	0	508	541
	508	0	etc	362

before the LKIN. subroutine call, and if the link address to be returned were LINK = 677, then the instruction

LKIN. (LINK)

would result in the following modification of the LOAS

LOAS =	677	0	541	0 (new LOAS top)
	541	0	362	677
	362	0	508	541
	508	0	etc.	362
	.	.	.	.
	.	.	.	.
	.	.	.	.
	.	.	.	.

The Link-Out (LKOT.) subroutine does just the opposite of the LKIN. subroutine. The LKOT subroutine gets the next available link address from the "top" of the LOAS. This subroutine has no input argument, but its output is the next available address in the LOAS. The "pointer" referencing to "top" of the LOAS is reset to the next available number in the LOAS array. If the LOAS is emptied, the error message

LOAS EMPTY - PROGRAM STOP

is printed out on the printer and the program is terminated.

Example: If the LOAS were

	INDEX	DATA	PLNK	NLNK
TOP LOAS	444	0	501	0
	501	0	226	444
	226	0	etc	501
	.			
	.			
	.			

before the statement LKOT. (O) then the result after the instruction would be

	INDEX	DATA	PLNK	NLNK
new top LOAS	501	0	226	0
	226	0	etc.	501
	.			
	.			
	.			

The second subclass of subroutines listed in the Utility Routines grouping are the Data Cell Routines. They provide the ability to modify the data cells in various ways.

The PREZ. subroutine, Provide Ending Zeros, performs the task of attaching a given number of zero data cells to the low order end of an integer. This, in effect, multiplies a given integer by the number 10^{3N}, where N specifies how many zero data cells are to be attached. The PREZ. subroutine also generates the appropriate address linkage for the modified integer and resets its reference key. The PREZ. subroutine has to input arguments. The first argument specifies the number of zero data cells to be attached, and the second argument is the reference name of the integer that is to be modified. As output, the PREZ. subroutine returns a floating point number which is the new key value that is to be assigned to the reference name. The PREZ. subroutine is used by the system in the Multiplication subroutine to modify intermediate products generated by the Multiplication subroutine.

Example: Let the reference name and key

VOLT = 2.7932100+05

be assigned to the integer

167 208 441

Then to attach one zero data cell, the LEWIZ statement

VOLT = PREZ. (N, VOLT),

where N = 1, would be used. If the topmost address in the LOAS were 221 the external result would be

VOLT = 2.7922100+05

while internally, the cell line NLNK (441) would become 221, PLNK (221) would become 441, and NLNK (221) = 0.

In certain instances, an arithmetic routine can generate a result containing zero-valued high order data cells. Such a result would waste a great deal of valuable space. The Release Leading Zeros (<u>RELZ</u>,) subroutine has the task of removing these zero-valued data cells from a result and returning their indices to the LOAS. The input argument is the label referencing the address of the high order data cell of the result of an arithmetic operation. Its output is the address of the first non-zero data cell. However, if the result of an arithmetic operation were zero, then the RELZ. subroutine would return, as output, the address of the last or low order data cell whose value would be zero.

<u>Example</u>: Let the result of an arithmetic operation be the integer

000 021 359,

where the address of the high order zero data cell is 244, and let the address of the data cell containing the value 021 be 119. Then the LEWIZ statement

RELZ. (ADRS)

where ADRS = 244 at input, would result in

RELZ (1) = 119

as output. The index 244 would be returned to the LOAS. The RELZ. subroutine is used by the ADD subroutine to release leading generated zeros. However, since all of the arithmetic subroutines use the ADD. subroutine to generate a result, they each, in effect, use the RELZ. subroutine to remove leading generated zeros.

The SIGN. subroutine is used to homogenize the algebraic sign of each data cell of the internal representation of an integer. The subroutine when called examines the high order data cell for its algebraic sign. If the sign of that data cell is plus, then each remaining data cell of the given integer is given a plus sign as part of its internal representation. Similarly, if the sign of the high order data cell is minus, then each of the remaining data cells of the given integer is given a minus sign as part of its internal representation. Note however, that the sign of a zero-valued data cell is always plus.

As input, the SIGN. subroutine is given the reference name of the integer to be adjusted. After the signs have been set in each data cell of that integer, an exit is made from the subroutine. The SIGN. subroutine has no output so that the input argument is unaltered.

Example: If

INDEX	DATA	PLNK	NLNK
58	-140	0	64
64	059	58	71
71	326	64	0

were in internal representation of an integer that had been input to the system, then before it is freed as a valid internal integer, the SIGN. subroutine is called to make the sign of the integer homogeneous in its internal representation. The statement

SIGN. (X),

where X = 5.8071000+05 is the reference name of the integer, would cause the following internal result.

INDEX	DATA	PLNK	NLNK
58	-140	0	64
65	-059	58	71
71	-326	64	0

The SIGN. subroutine is called by the system in the READ. subroutine to perform the operation of making the algebraic signs of data cells of an integer uniform.

The FRAK. subroutine is used to obtain the individual numeric digits of the data cell referenced as the second input argument of the subroutine. The first input argument specifies which of the three digits is gotten. A number one signifies the low order digit of the data cell, a number two obtains the middle digit, and the number three yields the high order digit of the data cell as output of the FRAK. subroutine.

The REST. subroutine is used by the division subroutine to restore the result of the product of the trial quotient with the divisor in a form which can be correctly subtracted from the dividend. This is done by placing low order single place value zeros at the low order end of the above product to give the same order of magnitude as the dividend. The number of zeros required to do this is given as the first input argument of the REST. subroutine. The second argument of the REST. subroutine is the reference name of the product. The output of the REST. subroutine is a reference key of the re-stored product, where restoration of the product into data cells begins at the low order end starting with the low order zeros. For example, if the dividend was the number

123 456 789 000

and if the product to be subtracted initially was the integer

1 203 104

then this product would be repacked as

120 310 400 000.

The CHOP. subroutine determines the position of the highest non-zero place value in a given non-zero data cell. If the units place value is the only non-zero place value of the data cell, the output of the CHOP. subroutine is 1. If the tens position of the data cell is non-zero, the output is 2 and if the hundreds position is non-zero, the result is 3.

The REPK. subroutine is used by the division subroutine to repack the quotient. The quotient as generated by the DIV. subroutine is stored one digit per data cell. Before the quotient is released, it is repacked three digits per data cell beginning with the low order digit of the unpacked quotient. The result of the REPK. subroutine is the reference key of the repacked quotient.

The Program Initialization grouping of the subroutine performs the function of introducing the system to all variables and arrays that will be used.

The <u>SLAT</u>. subroutine sets the initial values of the link and data arrays for the system. It has one input argument, a label whose value is the Data Array dimension defined in the main program. It has no external output. The statement

SLAT. (DIM),

where DIM is the dimension of the Data Array, causes the system initialization to begin. First, the SLAT. subroutine generates the List of Available Space Array, which at this time would be full while the Data Array would be empty. The number of available spaces assigned to the LOAS is equal to the dimension value of the Data Array which is the SLAT. subroutine input argument.

<u>Example</u>: If DIM = 5, then the LEWIZ statement

SLAT. (DIM)

would initialize the LOAS array as follows:

INDEX	DATA	PLNK	NLNK
5	0	0	4
4	0	5	3
3	0	4	2
2	0	3	1
1	0	2	0

The top of the LOAS array is the index address 5. This value is stored under the reference name LOAS. (Note: The LOAS array is actually the unused portion of the Data Array.) The reference name LOAS always contains a value which is the address of the next available (unfilled) data cell.

The SLAT. subroutine must be called by the main program before any input or processing with the Precision Arithmetic System can take place.

The Input/Output Routines

The second major classification of subroutines in the Precision Arithmetic Programming System is the Input/Output grouping listed as:

READ. (O)

RITE. (Variable)

LIST. (Variable)

These subroutines perform all input and output operations for the system.

All input to the Precision Arithmetic System is via the READ. subroutine. The LEWIZ statement

REFERENCE NAME = READ. (O)

in the main program when processed with cause the punched card data to be read and stored by the system. An integer being entered must be broken up into pieces which are factors of 1000 and entered in descending value from the highest three digit factor of 1000 to the lowest order three digit piece. Actually, the input integer would be punched into the card exactly as it would be written, except that in place of the commas which are used to set off successive powers of 1000, one or more blanks are used for separation.

Example: The integer

61,235,960,001

could be punched as

061b235b960b001

or 61b235b960b1

or 61bb235bbb960bb1

where the lower case "b" represents a blank space (or card column). Note that high order zero need not be punched. One other number must be punched at the end of each input integer, the number 9999. This defines the end of an input integer, but it is not stored as part of the integer. Thus, the integer listed above would be punched as

61b235b960b001b9999

Note that at least one blank card column must be placed between the low order 3-digit piece of the input integer and the sentinel 9999. If the 9999 does not appear after an input integer, the system will continue to read the following numbers until either a 9999 is found or until the END OF DATA card is read. If any one of the portions of an input integer is greater in absolute value than 999 an error is generated and the error message

ERROR - DATA OUTSIDE ALLOWABLE RANGE

will be printed on the printer and the program will be terminated. If the integer is negative only the high order factor need have a minus sign. If the integer is positive, no sign is required.

The READ. subroutine will accept and store correct integer pieces, one at a time, in the Data Array in the order in which they are read in. The subroutine obtains the next available data cell from the LOAS and stores the first integer factor in the data cell. It generates a zero address for its preceding link (PLNK) address and then looks to see if there is another integer factor to be stored. If there is, the subroutine returns to the LOAS for the next available data cell address. The next link (NLNK) address of the first integer factor is the address of the second data cell. The READ. subroutine continues storing successive integer factors and generating their respective PLNK and NLNK addresses until a 9999 has been read. When the number 9999 has been read, the READ. subroutine assigns a zero address to the NLNK address of the last integer factor that was stored in the DATA array before the number 9999 was read. The subroutine then generates a reference key for the input integer. This key is computed by taking the address of the data cell containing the first integer factor input to the system, multiplying it by 1000, and then adding to this product the address of the data cell containing the last integer factor input to the system before the number 9999 was read. This result is often too large to be stored as a fixed point number so it is stored as a floating point number. This floating point number is the reference key of the input integer, and it is the output of the READ. subroutine. The READ. subroutine then calls on the SIGN. subroutine to make the sign of each factor of the integer the same as the sign of the first factor of the integer. This representation however, is only internal to the system. After exit has been made from the READ. subroutine, the value that the reference name will have will be the floating point integer reference key generated by the READ. subroutine. From that point on in the main program the integer that was input to the system is referenced by the reference name given to the integer by the READ. subroutine. Figure 4 gives an example of the processing of an integer.

The Precision Arithmetic Programming System has two output routines, the RITE. subroutine and the LIST. subroutine. The RITE. subroutine is used for most output operations to display integer data in its ordinary form. As input, this subroutine requires the reference key supplied by the reference name of the integer to be printed as output. The LEWIZ statement

RITE. (X)

when processed would cause the integer referenced by the reference name X to be printed on the printer. The subroutine locates the high order data cell of the requested integer and stores this data cell, character by character, into a fixed point output array of 120 words. Each character is stored in one word of this array. After a data cell has been stored in the array, a blank is stored following the low order character of the data cell. The subroutine then returns to the system to get the next data cell of the integer and stores it in the manner described above. This continues until either the entire integer is stored in the output array or until the array has been filled. In either case, the RITE. subroutine calls on the LEWIZ library subroutine ALFOUT. which then

Input Integer: -432 017 635 900 000 010

Reference Name: X

Subroutine Call:

X = READ. (O)

Internal Representation

INDEX	DATA	PLNK	NLNK
095	-432	0	094
094	-017	095	081
081	-635	094	123
123	-900	081	149
149	000	123	201
201	-010	201	0

Reference Name and Key

After Exit From

The READ. Subroutine:

X = 9.520100+05

Figure 4. THE READ. SUBROUTINE

prints the 120 word output array on one printer line. If the entire integer could not be stored in one output line, the RITE. subroutine returns for the remaining data cells storing them character by character in the output array and printing the output array using the ALFOUT. subroutine until the entire integer has been listed. When this occurs, an exit is made from the RITE. subroutine returning control to the main program. The output integer is listed with a sign appearing only before the high order digit if the integer is negative. Otherwise no sign is listed. The internal structure of the integer remains unchanged by the RITE. subroutine and is available for further processing by the system.

<u>Example</u>:

If X were the reference name of the integer given in Figure 4, then the LEWIZ statement

```
RITE. (X)
```

would produce the following print-out:

```
-432 017 635 900 000 010
```

For debugging and testing purposes, the system has an output subroutine called the <u>LIST</u>. subroutine. The statement

```
LIST. (X)
```

would list as output the entire internal representation of the integer referenced by the name X. This listing prints in separate columns the data cell index, data cell value, the address of the preceding data cell, and the address of the next data cell respectively. The listing begins with the high order data cell and continues listing through the low order data cell. The LIST. subroutine call given above would result in the listing of the integer referenced by X as shown in Figure 4.

The Arithmetic Routines

All of the subroutines that have been described so far in this paper are basically system support routines. Their purpose is to support the Precision Arithmetic Operations that the system has been created to perform; that is, they were designed to support the Arithmetic Routines.

The Arithmetic Routines are a grouping of four major arithmetic subroutines along with several minor subroutines. They are:

```
 ADD. (Dummy, Ref. Name 1, Ref. Name 2)
SUBT. (Dummy, Ref. Name 1, Ref. Name 2)
MULT. (Dummy, Ref. Name 1, Ref. Name 2)
 DIV. (Dummy, Ref. Name 1, Ref. Name 2)
IRGR. (Ref. Name 1, Ref. Name 2)
CSIN. (Ref. Name)
QKAD. (Address 1, Address 2)
DASN. (Ref. Name)
DOOM. (Ref. Name)
```

Since the LRGR, CSIN., and QKAD subroutines support the major arithmetic routines, they will be described first. The LRGR. subroutine finds the "larger of two integers in absolute value. That is, it determines which of two integers has the greatest absolute value. If the integers have the same absolute value, then the LRGR. subroutine selects as the larger number, the integer whose reference name is given as the first argument of the subroutine. If the integers have the same absolute value, but have opposite signs, then the LRGR. subroutine selects the other integer as the larger number. The smaller of the two numbers is also available as an output of the LRGR. subroutine since the subroutine always places the key of the larger value as the first output argument and the key of the smaller value as the second output argument. Examples (1), (2), below give an illustration of the LRGR. subroutine operation.

Example (1) Let NUM1=3.4152900E+05 and NUM2=2.6111000E+05 be the reference names and reference keys of the integers

213 408 611 001

and -106 259 340 321 459

respectively. Then the LEWIZ subroutine

LRGR. (NUM1, NUM2)

will produce as output

LRGR(1) = 2.6111000E+05 (NUM2)

LRGR(2) = 3.4152900E+05 (NUM1)

Example (2)

If VAR1=9.6095000E+04 and VAR2=4.3059000E+04 reference the integers

1 009

and -1 009 respectively, then the LRGR. subroutine

LRGR. (VAR1, VAR2)

will produce as output

LRGR (1) = 9.6095000E+04 (VAR1)

LRGR (2) = 4.3059000E+04 (VAR2)

The CSIN. subroutine performs the operation of changing the sign of a given integer. In the process, the sign of each data cell is complemented. This, in effect, makes the CSIN. subroutine the unary minus operation, since if X is an integer, then CSIN. (X) replaces X by -X. This subroutine finds use in the system in the SUBT. subroutine.

In certain instances it is necessary to add two data cells together to obtain a sum which is only an intermediate result to some more complex calculation. Rather than using up space in the data array it would be more convenient to use temporary memory locations not in the data array to store this sum. Both the sum and carry, if one occurred, would be made available to system, and no data cells would have been used. This sort of thing is done by the <u>QKAD</u>. subroutine a "quick" addition is performed on the two data cells whose addresses are given as arguments to the QKAD. subroutine. The sum, a 3-digit number, and the carry become the first and second output arguments, respectively of the QKAD. subroutine.

<u>Example</u>: Let ADRS1 reference the data cell containing the number 321, and let ADRS2 reference the data cell containing the number 864. Then the LEWIZ statement

```
QKAD. (ADRS1, ADRS2)
```

produces the output

```
QKAD (1) = 185
QKAD (2) = 1
```

Note that had the statement

```
ADRS1 = QKAD. (ADRS1, ADRS2)
```

been given, the result would have been

```
ADRS1 ( = QKAD (1) ) = 185
QKAD  (2) = 1
```

If ADRS2 was to be replaced by the carry value then the statement

```
ADRS2 = QKAD (2)
```

must be written before the QKAD. subroutine is called again.

The QKAD subroutine is used by the MULT. subroutine to generate an intermediate sum as described above.

The DASN. subroutine determines the algebraic sign of the integer referenced by the subroutine's input argument. This is done by examining the sign of the nonzero high order data cell of the integer referenced. If the sign of this data cell is plus, then as output the DASN. subroutine yields the number zero. If the sign of this data cell is minus, the number plus one is the output.

The DOOM. subroutine determines the order of magnitude of an integer in terms of the number of data cells it required for storage. The input argument is the reference key of the integer whose order of magnitude is to be determined. The output, then, is the number of data cells that integer required for storage.

Of all the subroutines in the Precision Arithmetic System, none is more important than the addition (ADD.) subroutine. The ADD. subroutine is the workhorse of the system. Each of the other main arithmetic subroutines require it, either directly or indirectly, in the performance of its operation.

The ADD. subroutine performs the addition of two integers adding one data cell at a time from each integer, beginning with the low order data cells and continuing until the sum of the two integers is completed.

The resultant sum can be placed in any on one of three possible locations depending on which is desired. If X and Y were the reference names of two integers whose sum is to be found, then the three possible storage locations are Z, X, and Y, where Z is a new reference name different from X and Y. Thus, the possible sums are

$$Z = X + Y,\ X = X + Y,\ Y = X + Y.$$

Because of these possible different sums, it is necessary to inform the ADD. subroutine which one it is to generate beforehand. This is done by providing the ADD. subroutine with three input arguments. The first input argument is a dummy reference name whose value is either the number 1, 2, or 3. These numbers tell the ADD. subroutine that the sum Z = X + Y, X = X + Y, or Y = X + Y respectively is to be generated. The second and third input arguments are the reference names of the two integers that are to be added. For instance, if it were desired to compute the sum Y = X + Y, then the LEWIZ statement

Y = ADD. (DUM, X, Y)

would be used, where the value of DUM would be the number 3.

Once called, the ADD. subroutine immediately calls on the LRGR. subroutine to determine which of the two integers is the larger. It then determines which sum is to be generated by examining the value of the first input argument of the ADD. subroutine. Depending on whether this value is 1, 2, or 3, the subroutine branches to one of three parts of the subroutine. That is, the ADD. subroutine is divided into three sections, one to generate the sum Z = X + Y if ADD (1) = 1, another to generate the sum X = X + Y if ADD (1) = 2, and a third to generate the sum Y = X + Y if ADD (1) = 3. The first section, Z = X + Y, requires a new set of data cells for storage of the sum and thus requires one method of address generation.

Since the sum being generated will be stored at a new place in the Data Array, it removes data cells from the LOAS to store the integer pieces generated by the addition of X and Y. An entirely new set of data cells is used to store the sum, and a new integer is created. The ADD. subroutine then computes the reference key for this new integer and stores it as its output argument ADD (1); and an exit is made.

The second and third sections, X = X + Y and Y - X + Y, each return the sum to X or Y depending upon which section has been called. However, the address generation in these sections becomes more complicated than that used in the first section. In the second section, the sums generated by adding the values in the consecutive data cells of X and Y beginning with the low order data cells of X and Y replace the values in the data cells used by the integer X. The integer that was referenced by the reference name X before the addition is destroyed and in place of that integer is a new integer which is the sum of X and Y. The three-digit integer pieces that make up the sum X + Y replace the three-digit integer pieces that made up the integer X. The same data cells that were used to store X are used to store the sum X + Y. If the sum requires more data cells than were used by X, they are taken from the LOAS. If the sum requires fewer data cells than were used by X, high order zero cells would have been generated. At this point, the ADD. subroutine calls on the RELZ. subroutine to release these useless data cells and return them to the LOAS. Then a reference key is generated and becomes the output of the subroutine.

The third section of the ADD. subroutine performs the address generation of the sum factors the same way as the second section, however, the location of the data cell factors of X and Y become more complicated. The ADD. subroutine has adopted the convention of always making the larger number the primary addend and the smaller number the secondary addend. That is, the larger number always appears as the top number of the sum, and the smaller number is always the bottom number. This convention makes it necessary to use a more complex switching process to obtain the proper data cell addresses of the addends. When the sum has been completed, the reference key is generated in the same way as described for the second section of the ADD. subroutine and an exit is made to the main program.

The actual calculation of the sum, regardless of the form that is to be the result, is accomplished by adding a data cell of the smaller number to the corresponding data cell of the larger number. This sum may or may not result in the generation of a carry; which may be +1 or -1. If a carry does occur it is accounted for by adding it to the sum of the next pair of consecutive data cells of the addends. Addition begins with the low order data cells of the addends and continues until the high order data cell of the primary addend has been processed. The secondary addend may or may not have had as many factors as the primary addend. If it did not, then zero-valued data cells are added to the remaining high order primary addend data cells to complete the sum. If both numbers were of the same order of magnitude, corresponding data cells would be added together, providing for any carries generated, until the high order data cells of each addend are added. If no carry occurred with this last sum, the process would terminate. If a carry were generated, one more data cell would be called for from the LOAS to store it, and processing would terminate. In all of the above situations, the sign that would be assumed by the sum would be the sign of the larger or primary addend. The problem of carry generation, as can be seen, has also added to the complexity of the ADD. subroutine structure.

An example of the ADD. subroutine's various output forms is given below.

Example: Let X and Y reference the integers 123 456 789 and 879 644 213 respectively. Let their internal representation be as follows:

X = 1.0014300E+5

INDEX	DATA	PLNK	NLNK
100	123	0	143
143	456	100	146
146	789	143	0

Y = 2.3345100E+5

INDEX	DATA	PLNK	NLNK
233	879	0	234
234	544	233	451
451	213	234	0

Each of the three possible sum forms are to be generated. For the first sum Z = X + Y, the value of the first input argument, DUMMY, is 1. Then the LEWIZ statement

Z = ADD. (DUMMY, X, Y)

when processed will cause the following internal and external results.

INDEX	DATA	PLNK	NLNK
301	001	0	326
326	003	301	519
519	001	326	477
477	002	519	0

Z = X+Y = 3.0147700E+05

Here it can be seen that all index addresses of the sum are different from those of both X and Y.

If the value of DUMMY is changed to the number 2, processing of the LEWIZ statement

X = ADD. (DUMMY, X,Y)

would have the following result:

INDEX	DATA	PLNK	NLNK
296	001	0	100
100	003	296	143
143	001	100	146
146	002	143	0

X = X + Y = 2.9614600E+05

Notice that in this situation the index addresses of X + Y are the same as the index addresses of the initial integer X except for the high order index of X + Y. This data cell was needed to store the carry from the sum of the two preceding data cells. This data cell was removed from the top of the LOAS and given to the new integer X = X + Y. The reference key of X has also been adjusted to the appropriate high order address.

Finally, if DUMMY is given the value 3, then the ADD. subroutine call statement

Y = ADD. (DUMMY, X, Y)

would produce the following result:

INDEX	DATA	PLNK	NLNK
296	001	0	233
233	003	296	234
234	001	233	451
451	002	234	0

Y = X+Y = 2.9645100E+05

As in the second sum form, the sum X + Y replaces the original integer Y by the sum X + Y. It too, requires one more data cell than was required for the original integer Y. The reference key of the sum is then adjusted and stored at Y.

It is quite possible that a zero sum or a sum smaller than either integer could be generated by the addition if the numbers had opposite signs. In this situation, the second and third sum forms would have required fewer data cells for storage of the sum than their respective original factors. In this case, they would have returned data cells to the LOAS. If the result were zero in any case, one data cell would be required for storage.

Subtraction is also provided for as one of the Precision Arithmetic Programming System's basic arithmetic operations. When two integers X1 and X2 are to be subtracted, the LEWIZ statement

Ref. Name = SUBT. (DUMMY, X1, X2)

is used. The input argument, DUMMY, performs the same task as was described in the ADD. subroutine. It tells where the result is to be stored. The other two inputs to the SUBT. subroutine are reference names of the integers to be subtracted. The label, Ref. Name, on the left-hand side of the equal sign in the SUBT. subroutine call statement is the reference name into which the reference key of the result of the subtraction is to be stored.

After the SUBT. subroutine has been called, the CSIN. subroutine is called to change the algebraic sign of the integer which is to be the subtrahend, that is, the third input argument of the SUBT. subroutine. The minuend remains unchanged. Once the sign has been altered, the SUBT. subroutine calls on the ADD. subroutine to add the two integers. This sum gotten from the ADD. subroutine is, in reality, the difference of the two integers since the sign of the integer to be subtracted was complemented before the addition occurred. The SUBT. also provides for three various result storage forms:

$$Z = X - Y,\ X = X - Y,\ Y = X - Y$$

where $Z \neq X$ and $Z \neq Y$. After the difference has been computed, the reference key of the difference is calculated and stored at the appropriate reference name. This reference key is the output of the SUBT. subroutine. Before exit from the SUBT. subroutine is made, the sign of the subtrahend is again complemented, but only if the difference has been stored at a reference name different from the subtrahend. Then exit is made to the main program. Since the results are similar to those gotten from the ADD. subroutine, no example will be given for the SUBT. subroutine.

The third arithmetic operation incorporated into the Precision Arithmetic Programming System is the operation of multiplication. It is performed by the <u>MULT.</u> subroutine which also has three input arguments. The first argument is the DUMMY which is used to specify the place where the product key is stored after the calculation has been made. As before the values the label DUMMY can assume are the integers 1, 2, or 3. The other two input arguments to the MULT. subroutine are the reference names of the two integers to be multiplied together. The MULT. subroutine is called by the LEWIZ statement

Ref. Name = MULT. (DUMMY, A, B)

where A and B are the reference labels of the integers to be multiplied together, and the label on the left-hand side of the equal sign, Ref. Name, is the reference name of the resultant product.

Operationally, the MULT. subroutine adopts the convention that the larger number is to be the multiplicand and the smallest number is to be multiplier. The LRGR. subroutine is used to determine which input argument is the larger. Then the generation of the product begins. The low order data cell multiplies each data cell in the multiplicand starting with the low order data cell of the multiplicand and continues through the consecutive higher order data cells until the highest order factor has been processed. The product is stored in data cells that were taken from the LOAS as each 3 digit factor was generated. The MULT. subroutine then looks for the next higher order factor of the multiplier. If none exists, the MULT. subroutine generates a new reference key and stores it at the requested reference name. If another factor exists for the multiplier, the

MULT. subroutine generates a reference key for the first product calculated and then proceeds to generate the product of the second factor of the multiplier with the multiplicand in the manner described above. This second product is also stored in new data cells taken from the LOAS. When this product has been completed, the PREZ. subroutine is called to adjust the magnitude of this second product by the factor 10^{3N}, where in this case $N = 1$. Then a reference key is generated and the ADD. subroutine is called to form the sum of these two intermediate products. The sum is stored by the ADD. subroutine in the data cells that were used to store the first of the intermediate products, and the reference key of the sum is then adjusted to the appropriate value. Since the second intermediate product is no longer needed, the DVFA. subroutine is called to remove it from the Data Array, and the links it used to store the factors are returned to the LOAS to be available for use again. After this has been accomplished, the ADD. subroutine obtains the next factor of the multiplier, forms its product with the multiplicand, adjusts this product by the factor 10^{3N}, $N = 2$, and adds this result to the sum of the first two intermediate products. Then that last product is removed from the Data Array, and processing of the next higher order multiplier factor is initiated.

This process continues in the manner described above until the last factor of the multiplier has been processed. The resulting integer which was accumulated at the location of the first product is the desired product. the MULT. subroutine then generates a reference key for this integer and stores it at the appropriate reference name, and exit to the main program is executed. If the product terminated after processing only one multiplier portion, that is, if the multiplier contained only one three digit portion, the MULT. would generate a reference key and store it at the appropriate place. If the reference name of the product happened to be either one of the input reference names, then the MULT. subroutine deletes that integer from the Data Array and replaces the old reference key by the one generated for the product. An example of the accumulation technique and result is given below.

Example:

Let X = 1.0009900+05 and Y = 9.8096000+04 be the respective reference names and reference keys of the following integers.

```
                222 333
      and       754 368 951
```

The LEWIZ statement

```
      Z = MULT. (DUMMY, X, Y), DUMMY = 1
```

would perform the following internal calculations:

INDEX	DATA	PLNK	NLNK	
92	251	0	93	
93	204	92	94	(MULT (1))
94	860	93	95	
95	683	94	0	

This is the result of the product of the factor 333 with the integer 754 368 951. The accumulation of intermediate products would be

```
         251    204    860    683 (MULT (1) before)
   167   469    907    122    000 (intermediate product)
   -------------------------------
   167   721    111    982    683 (MULT (1) after)
```

Since there are only two factors in the multiplier, this last sum stored at MULT (1) is the product of the two input integers. The internal storage representation of the product is

INDEX	DATA	PLNK	NLNK	
86	167	0	92	
92	721	86	93	
93	111	92	94	MULT (1)=Product
94	982	93	95	
95	683	94	0	

The reference key would be generated from the top and bottom index addresses and would be stored as

z = 8.6095000+04

The last of the arithmetic operations in the PAPS is the division operation. It is performed by the arithmetic routine called DIV. The DIV. subroutine has three input arguments, the first is the Dummy value referencing the location of the quotient. The second and third arguments are respectively the reference names of the dividend and the divisor. The DIV. subroutine is called by the LEWIZ statement

Quotient = DIV. (Dummy, Dividend, Divisor).

When the DIV. subroutine is called, it first determines the sign of each number in order to define the sign that will be assigned to the quotient and remainder. The DIV. subroutine then proceeds to perform the division with the absolute value of the divisor and the dividend. The order of magnitude of the divisor and the dividend is determined. This is used to locate the trial division point of the first digit of the quotient; that is, the order of magnitude of the quotient. For instance, let

DIVD = 6 993 207 143, DIVR = 53 214.

The order of magnitude of DIVD is four, since it contains four data cells, while the order of magnitude of DIVR is two. The difference of these two numbers gives a trial estimate of where the division point is located. Since, in the above example, the difference is 2, the location of the division point is tentatively over the last digit of the data cell whose place value is second from the high order data cell. In this case the division point is over the low order digit of the data cell containing the portion 993. When this has been done, the DIV. subroutine then examines the magnitude of the divisor. If the divisor is composed of only one data cell, division is carried on cell by cell, generating a quotient. This is the simple case. If it happens that the divisor is composed of more than one data cell, an entirely different method is used to obtain the quotient. A trial quotient must be obtained which will approximate as closely as possible the correct quotient digit. The DIV. subroutine breaks this task into three cases each of which is examined by the QUES. subroutine. This subroutine has not been defined separately since it is really an integral part of the DIV. subroutine. Its description is given now as part of the definition of the DIV. subroutine. The <u>QUES</u>. subroutine first obtains that number of digits of the dividend as are contained in the divisor, high order zeros excluded. These digits are then re-stored, starting with the low order digit of those obtained, three digits to a cell. When that has been completed, the QUES. subroutine determines which of the two numbers, the divisor or the partial dividend, is the larger. The result of this comparison is used to decide upon a quotient value. If the divisor is the larger number, the output of the QUES. subroutine is the number -1. If they are equal, its output is the number 0, and if the dividend is the larger number, the output of QUES. is the number 1. Using the information received from the QUES. subroutine, the DIV. subroutine proceeds to obtain a quotient value. First, the high order four digits of the divisor are stored in a temporary location called TDVR. Then the Div. subroutine generates a quotient using the first four or five digits of the dividend depending on whether the output of QUES. was non negative (0 or 1) or negative (-1). If the first four digits of the dividend (called TDVD) are used, a quotient value less than 10 is obtained. In the other case, however, using the first five digits of the dividend, it is possible to obtain a quotient equal to 10 (but not greater). If the quotient obtained in this instance were 10, the DIV. subroutine changes the quotient value to the number 9. Once the quotient is obtained, the DIV. subroutine proceeds to form the product of the divisor and the quotient. Before this result is subtracted from the dividend, the

DIV. subroutine determines the location of the division point. This point was obtained by the QUES. subroutine when it generated a temporary dividend from the high order digits of the dividend equal to the number of digits contained in the divisor. Thus, the division point is located at a point that many digits from the high order digit or that many plus one depending upon whether the output of QUES. was non negative or negative, respectively. This point position is then subtracted from the magnitude value of the dividend to obtain the number of low order zero digits have to be attached to the product generated above. This is necessary for the subtraction to be done correctly. Once the product has been prepared, it is subtracted from the dividend by the SUBT. subroutine. The difference then replaces the original dividend. The DIV. subroutine then examines the sign of the remainder. If it is negative, the trial quotient was off by at most one. Thus, the divisor is added to the remainder and the result replaces the incorrect remainder. Then one is subtracted from the quotient value making the correction complete. Now the Div. subroutine loops back, using the remainder as the new dividend and continues the division.

When the remainder finally becomes smaller than the divisor, the division has been completed. However, the quotient was stored one digit per word and so it must be repacked. This is accomplished by the REPK. subroutine. The remainder is then given the global reference name REMAIN, and the DIV. subroutine then determines the sign of the quotient and remainder. In any case, the quotient and remainder have the same sign. Finally, the DIV. subroutine stores the quotient at the location specified by the input Dummy, and an exit is made to the main program.

DISCUSSION AND CRITICISM

The Precision Arithmetic Programming System as described in this paper will permit the basic operations of addition, subtraction, multiplication, and division, as well as certain data manipulations that can be performed on integers of various magnitudes. The actual size of the integers that can be used by the system is restricted. The system allows a maximum of 500 three digit integer pieces to be stored. As the number of different integers increases, the average size of each integer decreases.

The MULT. and DIV. subroutines each require a temporary work space that is taken from the DATA array. The product of two thirty digit integers is at maximum a sixty digit integer. Since the MULT. subroutine uses an accumulator to generate intermediate products, the accumulator would require up to 20 data cells of temporary storage for operation.

Division requires a good deal of temporary storage to perform its various manipulations and intermediate calculations. For instance, the quotient as it is being generated stores only one digit per data cell. The QUES. subroutine also generates the partial dividend by taking the dividend apart digit by digit before repacking it.

The efficiency of the arithmetic routines varies. The ADD. and SUBT. subroutines both use accumulators so as not to waste space. The manner in which the calculations are made works well, although complementing a generated sum is not used. Instead, the larger number is always made the primary addend, and the sign of the sum is always the same as the sign of the primary addend. Carries were a major concern in the ADD. and SUBT. subroutines. Because of the convention adopted of always making the larger number the primary addend, two possible carry values had to be accounted for. An examination of the listing of the ADD. program will show the manner in which testing was performed.

The MULT. subroutine uses two accumulators, one to generate intermediate products, and the other to accumulate the sums of the intermediate products. After each intermediate product is generated, it is no longer needed and hence, it is disregarded and the space it used is returned to the available space array.

The DIV. subroutine is perhaps the most complicated of the routines. It is as efficient as can be made as far as its requirement for work space is concerned. However, efficiency of the DIV. subroutine is dependent upon its initial and intermediate results. In order to be at all efficient in performing a division of the type required, the DIV. subroutine must be able to generate a reasonable trial quotient to begin division, for otherwise, it would spend most of its time adjusting the incorrect quotient. That this is a problem can easily be demonstrated with an example. Suppose the DIV. subroutine was required to perform the following division:

$$199900 \,/\, \overline{100000000}$$

Choosing 1 as an initial trial quotient would result in a rather larger error, 80%. However, by examining the succeeding digits in the divisor, it would become obvious that the correct trial quotient should be 5. There are other similar marginal cases that could cause trouble if the initial trial quotient is incorrect. This error must be kept as low as possible.

The system that is used by the DIV. subroutine seems to account for these marginal cases, as well as, for ordinary situations without causing any significant error in generating a trial quotient. For example, the division problem

$$150099 \,/\, \overline{150000999}$$

would be handled in the following manner by the DIV. subroutine. First, the order of magnitude is used to determine an initial positioning of the division point. This would be located over the low order zero of the dividend. The QUES. subroutine then compares the divisor with the six high order digits of the dividend and finds the divisor to be larger. As a result, the division point is moved one place value to the right of the original estimate. Since the divisor is now smaller than the modified dividend being used, the first four digits of the divisor and the first five digits of the modified dividend are used to obtain an initial quotient value. Here

$$1500 \;/\overline{15000}$$

yields 10 but 10 is too large so it is reduced to 9. This becomes the trial quotient. Then the product

150099*9

is formed yielding 1350891. This then is subtracted from the modified dividend yielding 149 118. This value now becomes the modified dividend, and it is then compared with the divisor by the QUES. subroutine. The divisor is larger so the next digit of the dividend is attached yielding the new modified dividend 1491189. The QUES. subroutine then takes the number 1500 and divides it into 14911 obtaining 9 as the next trial divisor. This multiplies the divisor and the product is subtracted from the modified dividend yielding 140 298. This process continues in this manner until the final quotient is obtained. No error is generated, and hence, no unnecessary calculations are made.

Although the PAPS is relatively fast, it lacks the necessary storage space to be very useful in larger applications. If the subroutines were linked and stored on either magnetic tape or a disk, it would be much more useful. Linking would also permit various other routines such as the G. D. D., L. C. M., and some matrix operations to be included to amke the system more powerful.

The Precision Arithmetic Programming System in its present state of development is available for use and further development through the Lehigh University Computer Center. A complete deck listing and outputs of the system are on file.

APPENDIX I

LIST OF ILLEGAL SYMBOLS

ADD	FJON	DIVID	IT
ADRS	DIVR	I	IL
BARO	IFRST	FON	IQ
BET	ADRS1	BUOT	END
CSIN	AFTR	D	INT
CELL1	BLANK	CNTR	DIVD
DATA	AJ	ADRS2	E
DVFA	CHOP	A	AX
DO	CELL2	BOX	INDX
DRS1	DASN	B	ABS
FRAK	DIV	CARY	IZIP
FINL	C	CAR	BAX
GETE	DRS2	DOOM	IG
G2	FINS	DUM	LKOT
HOLD	DM2	CYCL	LOC2
DIG	GETS	CM1	MEMR2
INPT	G3	FIRST	MAKE
FR	HLOC	DVD	LOC
ET	FRST	G1	LC1
FT	IK	DVR	LOAS
IFIRST	DPV	HNBR	LT
FION	DOMS	G4	PREV

PLAS2	STAR2	QD	MEMR1
PVD	Y	PAJ	MINUS
OMD	REMP	NXT	NUM
REST	ALFOUT	QUOT	MOT
RS1	BIG	M3	LC2
R2	JK	LOT	NADR
RIG1	KOUNT	SLAT	PREZ
PLT	LIST	STRT2	PLAS1
Q2	LRGR	T	QKAD
N	MULT	SBT2	LK
REMAIN	LAST	TYPE	REPK
RADR	NLNK	SP	RELZ
M2	NI	SL1	RPT
P	OC	TL	R4
SIGN	LS	TDVR	OMR
STRT1	PLNK	SMALL	Q1
TEMP	PRES	TAR2	LEST
SBT1	QUES	QUS	MPY
SON	PVR	RST	MAX
STL	READ	ZIP	M1
VAR	RITE	ISTL	PROD
TR	RS2	IFT	QU
WIG	R3	KT	SUBT
STOR	RIG2	LKIN	S
X	Q	LOC1	TEM

THOW

SET

SYPE

SL2

SUM

TDVD

TAR1

STAR1

MQUS

IR

Z

```
005 330  ANDREW J KASARDA     SAMPLE INPUT AND OUTPUT OF THE PAPS
MAY 27 66        15 29.0
                                                                                PAGE 1.      MAY 27 66

 #      SEQ  LABL TYP                       STATEMENT                        C ZERO      NOT 0       PLUS      MINUS       ELSE

001.   000010       FIXDATA[300],PLNK[300],NLNK[300]                          [     ]    [     ]    [     ]    [     ]    [     ]
002.                D  ADD.[66],CSIN.[10],DASN.[10],DOOM.[11]                 [     ]    [     ]    [     ]    [     ]    [     ]
003.                D  DVFA.[11],FRAK.[12],GETE.[3],GETS.[3]                  [     ]    [     ]    [     ]    [     ]    [     ]
004.                D  LIST.[10],LKIN.[3],LKOT.[10],LRGR.[15]                 [     ]    [     ]    [     ]    [     ]    [     ]
005.                D  MULT.[23],PREZ.[16],READ.[16],DIV.[50]                 [     ]    [     ]    [     ]    [     ]    [     ]
006.                D  REPK.[11],REST.[14],RITE.[19],SIGN.[11]                [     ]    [     ]    [     ]    [     ]    [     ]
007.                D  SLAT.[11],SUBT.[11],QUES.[16],CHOP.[15]                [     ]    [     ]    [     ]    [     ]    [     ]
008.   000010       FIXLOC1,LOC2,TEMP,ADRS1,ADRS2,CARY,BARO,INPT              [     ]    [     ]    [     ]    [     ]    [     ]
009.   000020       FIXHOLD,HLOC,VAR,ADRS,FINS,STRT1,STRT2,MEMR1              [     ]    [     ]    [     ]    [     ]    [     ]
010.   000030       FIXMEMR2,NUM,PREV,FIRST,PRES,AFTR,LAST,DUM,OC             [     ]    [     ]    [     ]    [     ]    [     ]
011.   000040       FIXBLANK,MINUS,KOUNT,S,T,A[120],MAKE,LOC,IK,TEM           [     ]    [     ]    [     ]    [     ]    [     ]
012.   000050       FIXIT,SBT1,SBT2,HNBR,THOW,SON,NL                          [     ]    [     ]    [     ]    [     ]    [     ]
013.   000060       FIXDC,PLAS1,PLAS2,ZIP,BOX,QKAD.[2],MOT                    [     ]    [     ]    [     ]    [     ]    [     ]
014.   000070       FIXTYPE,FINL,CELL1,CELL2,SET,CAR,STL,C,CYCL               [     ]    [     ]    [     ]    [     ]    [     ]
015.   000075       FIXLC1,LC2,DRS1,DRS2,RELZ.[2],LOAS,LS                     [     ]    [     ]    [     ]    [     ]    [     ]
016.                FIXRS1,G1,G2,G3,SP,SYPE,SL1,SL2,DM1,DM2,DVD               [     ]    [     ]    [     ]    [     ]    [     ]
017.                FIXDVR,DIG,NADR,FRST,KT,BET,RS2,G4                        [     ]    [     ]    [     ]    [     ]    [     ]
018.                FIXRPT,R2,LT,R3,R4,FR,PVD,PVR,DPV,TR                      [     ]    [     ]    [     ]    [     ]    [     ]
019.                FIXRIG1,RIG2,JK,LK,WIG,OMD,OMR,PLT,IL,TL,ET,DOMS          [     ]    [     ]    [     ]    [     ]    [     ]
020.                FIXQ,Q1,Q2,QD,IQ,FT,LEST                                  [     ]    [     ]    [     ]    [     ]    [     ]
021.   000010       PL THE PRECISION ARITHMETIC SYSTEM                        [     ]    [     ]    [     ]    [     ]    [     ]
022.                PV ,,                                                     [     ]    [     ]    [     ]    [     ]    [     ]
023.   000015          N=300,SLAT.[N]                                         [     ]    [     ]    [     ]    [     ]    [     ]
024.   000020       PVLX=READ.[0]                                             [     ]    [     ]    [     ]    [     ]    [     ]
025.   000025          LIST.[X],RITE.[X]                                      [     ]    [     ]    [     ]    [     ]    [     ]
026.   000030       PVL,Y=READ.[0]                                            [     ]    [     ]    [     ]    [     ]    [     ]
027.   000035          LIST.[Y],RITE.[Y]                                      [     ]    [     ]    [     ]    [     ]    [     ]
028.   000040       PVL,AJ=READ.[0]                                           [     ]    [     ]    [     ]    [     ]    [     ]
029.   000045          LIST.[AJ],RITE.[AJ]                                    [     ]    [     ]    [     ]    [     ]    [     ]
030.   000050       PVL,B=READ.[0]                                            [     ]    [     ]    [     ]    [     ]    [     ]
031.   000055          LIST.[B],RITE.[B]                                      [     ]    [     ]    [     ]    [     ]    [     ]
032.   000060       PVL,PAJ=READ.[0]                                          [     ]    [     ]    [     ]    [     ]    [     ]
033.   000065          LIST.[PAJ],RITE.[PAJ]                                  [     ]    [     ]    [     ]    [     ]    [     ]
034.   000070       SL                                                        [     ]    [     ]    [     ]    [     ]    [     ]
035.   000080       PVLZ=1,SUM=ADD.[Z,X,Y],                                   [     ]    [     ]    [     ]    [     ]    [     ]
036.   000090          LIST.[SUM]                                             [     ]    [     ]    [     ]    [     ]    [     ]
037.   000100       PV                                                        [     ]    [     ]    [     ]    [     ]    [     ]
038.   000110          RITE.[SUM]                                             [     ]    [     ]    [     ]    [     ]    [     ]
039.   000120       SL                                                        [     ]    [     ]    [     ]    [     ]    [     ]
040.   000130       PVLZ=3,B=SUBT.[Z,AJ,B],                                   [     ]    [     ]    [     ]    [     ]    [     ]
041.   000140          LIST.[B]                                               [     ]    [     ]    [     ]    [     ]    [     ]
042.   000150       PV                                                        [     ]    [     ]    [     ]    [     ]    [     ]
043.   000160          RITE.[B]                                               [     ]    [     ]    [     ]    [     ]    [     ]
044.   000170       SL                                                        [     ]    [     ]    [     ]    [     ]    [     ]
045.   000180       PVLZ=1,MPY=MULT.[Z,SUM,AJ],                               [     ]    [     ]    [     ]    [     ]    [     ]
046.   000190          LIST.[MPY]                                             [     ]    [     ]    [     ]    [     ]    [     ]
047.   000200       PV                                                        [     ]    [     ]    [     ]    [     ]    [     ]
048.   000210          RITE.[MPY]                                             [     ]    [     ]    [     ]    [     ]    [     ]
049.   000220       SL                                                        [     ]    [     ]    [     ]    [     ]    [     ]
050.   000230       PVLZ=1,DIVID=DIV.[Z,PAJ,B],                               [     ]    [     ]    [     ]    [     ]    [     ]
051.   000240          LIST.[DIVID]                                           [     ]    [     ]    [     ]    [     ]    [     ]
052.   000250       PV                                                        [     ]    [     ]    [     ]    [     ]    [     ]
053.   000260          RITE.[DIVID]                                           [     ]    [     ]    [     ]    [     ]    [     ]
054.   000270       SL                                                        [     ]    [     ]    [     ]    [     ]    [     ]
055.   000280       PVLREMAIN,                                                [     ]    [     ]    [     ]    [     ]    [     ]
```

```
                                                                                  PAGE 2.      MAY 27 66

 #      SEQ   LABL TYP                     STATEMENT                        C ZERO     NOT 0     PLUS     MINUS     ELSE

056.  000290                LIST.[REMAIN]                                  [    ]   [    ]   [    ]   [    ]   [    ]
057.  000300          PV                                                   [    ]   [    ]   [    ]   [    ]   [    ]
058.  000310                RITE.[REMAIN]                                  [    ]   [    ]   [    ]   [    ]   [END ]




059.  000030ADD .           LRGR.[ADD[2],ADD[3]]                           [    ]   [    ]   [    ]   [    ]   [    ]   ADD .
060.  000040                LOC1=GETE.[LRGR[1]]                            [    ]   [    ]   [    ]   [    ]   [    ]   ADD .
061.  000050                LOC2=GETF.[LRGR[2]]                            [    ]   [    ]   [    ]   [    ]   [    ]   ADD .
062.  000051                DATA[GETS.[LRGR[1]]]                           [    ]   [    ]   [    ]   [ 51 ]   [ 52 ]   ADD .
063.  000052   51           THOW=-1000,SON=-1                              [    ]   [    ]   [    ]   [    ]   [ 53 ]   ADD .
064.  000053   52           THOW=1000,SON=1                                [    ]   [    ]   [    ]   [    ]   [ 53 ]   ADD .
065.  000060   53           CARY=BARO=VAR=PLNK[0]=NLNK[0]=0                [    ]   [    ]   [    ]   [    ]   [    ]   ADD .
066.  000070                ADD[1]-2                                       [ 40 ]   [    ]   [ 40 ]   [ 13 ]   [    ]   ADD .
067.  000080   13           ADRS1=LKOT.[VAR],ADRS2=LKOT.[VAR]              [    ]   [    ]   [    ]   [    ]   [    ]   ADD .
068.  000090                LAST=ADRS1,TEMP=LOC1                           [    ]   [    ]   [    ]   [    ]   [    ]   ADD .
069.  000100                PLNK[LAST]=ADRS2,NLNK[LAST]=0                  [    ]   [    ]   [    ]   [    ]   [    ]   ADD .
070.  000110   17           PLNK[ADRS1]=ADRS2,NLNK[ADRS1]=TEMP             [    ]   [    ]   [    ]   [    ]   [ 15 ]   ADD .
071.  000120   15           PLNK[LOC1]                                     [ 16 ]   [    ]   [    ]   [    ]   [    ]   ADD .
072.  000130   16           HOLD=DATA[LOC1]+DATA[LOC2]+CARY-BARO           [    ]   [    ]   [    ]   [    ]   [ 18 ]   ADD .
073.  000140                BARO=0                                         [    ]   [    ]   [    ]   [    ]   [ 19 ]   ADD .
074.  000150   18           HOLD=[DATA[LOC1]+THOW]+DATA[LOC2]+CARY-BARO    [    ]   [    ]   [    ]   [    ]   [    ]   ADD .
075.  000160                BARO=SON                                       [    ]   [    ]   [    ]   [    ]   [    ]   ADD .
076.  000170   19           HOLD-2000                                      [ 20 ]   [    ]   [ 20 ]   [ 21 ]   [    ]   ADD .
077.  000180   20           CARY=2,DATA[ADRS1]=HOLD-2000                   [    ]   [    ]   [    ]   [    ]   [ 28 ]   ADD .
078.  000190   21           HOLD-1000                                      [ 22 ]   [    ]   [ 22 ]   [ 23 ]   [    ]   ADD .
079.  000200   22           CARY=1,DATA[ADRS1]=HOLD-1000                   [    ]   [    ]   [    ]   [    ]   [ 28 ]   ADD .
080.  000210   23           HOLD+2000                                      [ 24 ]   [    ]   [ 25 ]   [ 24 ]   [    ]   ADD .
081.  000220   24           CARY=-2,DATA[ADRS1]=HOLD+2000                  [    ]   [    ]   [    ]   [    ]   [ 28 ]   ADD .
082.  000230   25           HOLD+1000                                      [ 26 ]   [    ]   [ 27 ]   [ 26 ]   [    ]   ADD .
083.  000240   26           CARY=-1,DATA[ADRS1]=HOLD+1000                  [    ]   [    ]   [    ]   [    ]   [    ]   ADD .
084.  000250   27           CARY=0,DATA[ADRS1]=HOLD                        [    ]   [    ]   [    ]   [    ]   [ 28 ]   ADD .
085.  000260   28           HLOC=TEMP,TEMP=ADRS1                           [    ]   [    ]   [    ]   [    ]   [    ]   ADD .
086.  000270                ADD[1]-2                                       [ 43 ]   [    ]   [ 43 ]   [    ]   [    ]   ADD .
087.  000280                ADRS1=ADRS2,ADRS2=LKOT.[VAR]                   [    ]   [    ]   [    ]   [    ]   [    ]   ADD .
088.  000290                LOC1=PLNK[LOC1]                                [ 30 ]   [    ]   [    ]   [    ]   [ 29 ]   ADD .
089.  000300   29           LOC2=PLNK[LOC2]                                [ 31 ]   [    ]   [    ]   [    ]   [ 17 ]   ADD .
090.  000310   31           DATA[LOC2]=0                                   [    ]   [    ]   [    ]   [    ]   [ 17 ]   ADD .
091.  000320   30           CARY                                           [ 33 ]   [    ]   [    ]   [    ]   [ 32 ]   ADD .
092.  000330   32           DATA[LOC1]=0,PLNK[LOC1]=0,LOC2=0               [    ]   [    ]   [    ]   [    ]   [ 31 ]   ADD .
093.  000340   33           PLNK[TEMP]=0,FIRST=TEMP,LKIN.[ADRS2],        C
   .  000345                LKIN.[ADRS1]                                   [    ]   [    ]   [    ]   [    ]   [    ]   ADD .
094.  000347                IFIRST=RFL7.[FIRST]                            [    ]   [    ]   [    ]   [    ]   [    ]   ADD .
095.  000350                ADD[1]=[1000*IFIRST+LAST]                      [    ]   [    ]   [    ]   [    ]   [    ]   ADD .
096.  000370   40           I=ADD[1],ADD[I]-LRGR[1]                        [ 41 ]   [    ]   [    ]   [    ]   [ 61 ]   ADD .
097.  000380   41           LAST=LOC1,TEMP=LOC1                            [    ]   [    ]   [    ]   [    ]   [    ]   ADD .
098.  000390   42           ADRS1=LOC1                                     [    ]   [    ]   [    ]   [    ]   [ 15 ]   ADD .
099.  000395   43           ADD[I]-LRGR[1]                                 [    ]   [ 63 ]   [    ]   [    ]   [    ]   ADD .
100.  000400                LOC1=PLNK[LOC1]                                [ 45 ]   [    ]   [    ]   [    ]   [    ]   ADD .
```

```
                                                                                   PAGE 3.        MAY 27 66

  #       SEQ   LABL TYP                 STATEMENT                      C ZERO     NOT 0     PLUS     MINUS      ELSE

 101.   000410          LOC2=PLNK[LOC2]                                 [  46]   [    ]   [    ]   [    ]   [  42]   ADD .
 102.   000420   46     DATA[LOC2]=0                                    [    ]   [    ]   [    ]   [    ]   [  42]   ADD .
 103.   000430   45     CARY                                            [  48]   [    ]   [    ]   [    ]   [  47]   ADD .
 104.   000440   47     LOC2=0,DATA[LOC1]=DATA[LOC2]=PLNK[LOC1]=0       [    ]   [    ]   [    ]   [    ]   [    ]   ADD .
 105.   000450          ADRS1=LKOT.[VAR]                                [    ]   [    ]   [    ]   [    ]   [  15]   ADD .
 106.   000460   48     NLNK[TEMP]                                      [  50]   [    ]   [    ]   [    ]   [    ]   ADD .
 107.   000465          PLNK[TEMP]=0,NLNK[TEMP]=HLOC                    [    ]   [    ]   [    ]   [    ]   [    ]   ADD .
 108.   000468   50     FIRST=TEMP                                      [    ]   [    ]   [    ]   [    ]   [    ]   ADD .
 109.   000469   49     IFIRST=RELZ.[FIRST]                             [    ]   [    ]   [    ]   [    ]   [    ]   ADD .
 110.   000470          ADD[1]=[1000*IFIRST+LAST]                       [    ]   [    ]   [    ]   [    ]   [.   ]   ADD .
 111.   000490   61     LAST=TEMP=LOC2,TEM=LKOT.[VAR],IK=0              [    ]   [    ]   [    ]   [    ]   [    ]   ADD .
 112.   000500   62     ADRS1=LOC2                                      [    ]   [    ]   [    ]   [    ]   [  15]   ADD .
 113.   000510   63     LOC1=PLNK[LOC1]                                 [  65]   [    ]   [    ]   [    ]   [    ]   ADD .
 114.   000520          LOC2=PLNK[LOC2]                                 [  66]   [    ]   [    ]   [    ]   [  62]   ADD .
 115.   000530   66     DATA[LOC2]=0,ADRS1=TEM                          [    ]   [    ]   [    ]   [    ]   [    ]   ADD .
 116.   000540          TEM=PLNK[TEM]=LKOT.[VAR],NLNK[ADRS1]=HLOC       [    ]   [    ]   [    ]   [    ]   [  15]   ADD .
 117.   000550   65     CARY                                            [    ]   [  66]   [    ]   [    ]   [    ]   ADD .
 118.   000560          FIRST=TEMP,PLNK[FIRST]=0,NLNK[FIRST]=HLOC       [    ]   [    ]   [    ]   [    ]   [    ]   ADD .
 119.   000570          PLNK[HLOC]=FIRST                                [    ]   [    ]   [    ]   [    ]   [  49]   ADD .



 120.   000010CHOP.     INT.[DATA[CHOP[1]]/100]                         [    ]   [  10]   [    ]   [    ]   [    ]   CHOP.
 121.   000020          INT.[DATA[CHOP[1]]/10]                          [    ]   [  15]   [    ]   [    ]   [    ]   CHOP.
 122.   000030          CHOP[1]=1                                       [    ]   [    ]   [    ]   [    ]   [.   ]   CHOP.
 123.   000040   15     CHOP[1]=2                                       [    ]   [    ]   [    ]   [    ]   [.   ]   CHOP.
 124.   000050   10     CHOP[1]=3                                       [    ]   [    ]   [    ]   [    ]   [.   ]   CHOP.



 125.   000020CSIN.     ADRS=GETS.[CSIN[1]]                             [    ]   [    ]   [    ]   [    ]   [    ]   CSIN.
 126.   000030   10     DATA[ADRS]=-DATA[ADRS]                          [    ]   [    ]   [    ]   [    ]   [    ]   CSIN.
 127.   000040          ADRS=NLNK[ADRS]                                 [.   ]   [    ]   [    ]   [    ]   [  10]   CSIN.
 #      SEQ  LABL TYP              STATEMENT                        C ZERO    NOT 0     PLUS     MINUS     ELSE
128.  000010DASN.   DATA[GETS.[DASN[1]]]                              [    ]   [    ]   [    ]   [  10]   [    ]   DASN.
129.  000020        DASN[1]=0                                         [    ]   [    ]   [    ]   [    ]   [.   ]   DASN.
130.  000030  10    CSIN.[DASN[1]],DASN[1]=1                          [    ]   [    ]   [    ]   [    ]   [.   ]   DASN.



131.  000010DIV .   DATA[GETS.[DIV[3]]]                               [  10]   [    ]   [    ]   [    ]   [  11]   DIV .
132.  000020  10 PL ERROR - DIVISION BY ZERO                          [    ]   [    ]   [    ]   [    ]   [END ]   DIV .
133.  000030  11    SYPE=DIV[1]                                       [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
134.  000040        SL1=DASN.[DIV[2]],SL2=DASN.[DIV[3]]               [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
135.  000050        DM1=DOOM.[DIV[2]],DM2=DOOM.[DIV[3]]               [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
136.  000060        DVD=GETS.[DIV[2]],DVR=GETS.[DIV[3]]               [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
137.  000070        DM2-1                                             [  13]   [  30]   [    ]   [    ]   [    ]   DIV .
138.  000080  13    DIG=NLNK[DVD]                                     [    ]   [  20]   [    ]   [    ]   [    ]   DIV .
139.  000090        FION=LKCT.[0],PLNK[FION]=NLNK[FION]=0             [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
140.  000100        DATA[FION]=INT.[DATA[DVD]/DATA[DVR]]              [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
141.  000110        FCN=LKOT.[0],PLNK[FON]=NLNK[FON]=0                [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
142.  000115        DATA[FON]=DATA[DVD]-[DATA[FION]*DATA[DVR]]        [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
143.  000117        REMAIN=[1000*FON+FON]                             [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
144.  000120        SL1-SL2                                           [  12]   [    ]   [    ]   [    ]   [    ]   DIV .
145.  000130        DATA[FION]=-DATA[FION],CSIN.[REMAIN]              [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
146.  000140  12    DIV[1]=[1000*FION+FION]                           [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
147.  000150  20    NADR=FRST=LKOT.[0],NXT=LKOT.[0]                   [    ]   [    ]   [    ]   [    ]   [.   ]   DIV .
148.  000160        PLNK[FRST]=0,NLNK[FRST]=NXT                       [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
149.  000170        KT=3,DIVD=DATA[DVD]                               [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
150.  000180        DIVD-DATA[DVR]                                    [  23]   [    ]   [  23]   [    ]   [    ]   DIV .
151.  000190  22    DIVD=DIVD*10+FRAK.[KT,DATA[DIG]]                  [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
152.  000195        DIVD-DATA[DVR]                                    [  23]   [    ]   [  23]   [    ]   [    ]   DIV .
153.  000200        DATA[NADR]=0                                      [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
154.  000210  23    DATA[NADR]=INT.[DIVD/DATA[DVR]]                   [    ]   [    ]   [    ]   [    ]   [  24]   DIV .
155.  000220        DIVD=DIVD-DATA[NADR]*DATA[DVR]                    [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
156.  000230  24    BET=NADR,NADR=NXT,NXT=LKOT.[0]                    [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
157.  000240        PLNK[NADR]=BET,NLNK[NADR]=NXT                     [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
158.  000245        [KT=KT-1]                                         [    ]   [  22]   [    ]   [    ]   [    ]   DIV .
159.  000250        KT=3,DIG=NLNK[DIG]                                [    ]   [  22]   [    ]   [    ]   [    ]   DIV .
160.  000260        LKIN.[NXT],LKIN.[NADR],NLNK[BET]=0                [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
161.  000270        FJON=LKCT.[0],PLNK[FJON]=NLNK[FJON]=0             [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
162.  000275        DATA[FJON]=DIVD                                   [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
163.  000280        REMAIN=[1000*FJON+FJON]                           [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
164.  000330  30    NADR=FRST=LKOT.[0],NXT=LKOT.[0]                   [    ]   [    ]   [    ]   [    ]   [  14]   DIV .
165.  000340        PLNK[NADR]=0,NLNK[NADR]=NXT                       [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
166.  000510        PVR=3*[DM2-1]+CHOP.[DVR]                          [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
167.  000350        RIG1=DVR,RIG2=NLNK[RIG1]                          [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
168.  000360        CHOP.[RIG1]-2                                     [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
169.  000370        TDVR=1000*DATA[RIG1]+DATA[RIG2]                   [  36]   [    ]   [  37]   [    ]   [    ]   DIV .
170.  000380  36    TR=3,TDVR=100*DATA[RIG1]+10*FRAK.[TR,DATA[RIGC    [    ]   [    ]   [    ]   [    ]   [  40]   DIV .
   .  000385        2]]+FRAK.[TR-1,DATA[RIG2]]
171.  000390  37    TR=3,TDVR=10*DATA[RIG1]+FRAK.[TR,DATA[RIG2]]      [    ]   [    ]   [    ]   [    ]   [  40]   DIV .
172.  000400  40    DIVD=DIV[2],PLT=1                                 [    ]   [    ]   [    ]   [    ]   [    ]   DIV .
```

```
                                                                          PAGE 5.      MAY 27 66

 #      SEQ   LABL TYP               STATEMENT                      C ZERO     NOT 0      PLUS      MINUS      ELSE

173.   000410           OMD=3*[DM1-1]+CHOP.[DVD]                    [    ]    [    ]    [    ]    [    ]    [  47]   DIV .
174.   000420   41      Z=1,MAX=SUBT.[Z,DIVD,DIV[3]]                [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
175.   000422           Z=DASN.[MAX],MAX=DVFA.[MAX]                 [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
176.   000450           Z-1                                         [  48]    [    ]    [    ]    [    ]    [    ]   DIV .
177.   000430           TDVD-TDVR                                   [  45]    [    ]    [  45]    [    ]    [    ]   DIV .
178.   000440   43      TDVD=TDVD*10+FRAK.[TR,DATA[DIG]]            [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
179.   000450           TDVD-TDVR                                   [  45]    [    ]    [  45]    [    ]    [    ]   DIV .
180.   000455           [IL=IL+1]-TL                                [    ]    [    ]    [    ]    [  25]    [    ]   DIV .
181.   000460           DATA[NADR]=0,TR=TR-1                        [    ]    [  44]    [    ]    [    ]    [    ]   DIV .
182.   000465   26      TR=3,DIG=NLNK[DIG]                          [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
183.   000466   25      TR=TR-1                                     [  26]    [    ]    [    ]    [    ]    [  43]   DIV .
184.   000470   44      BET=NADR,NADR=NXT,NXT=LKOT.[0]              [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
185.   000480           PLNK[NADR]=BET,NLNK[NADR]=NXT               [    ]    [    ]    [    ]    [    ]    [  43]   DIV .
186.   000490   45      DATA[NADR]=INT.[TDVD/TDVR],RADR=LKOT.[0]    [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
187.   000492           DATA[RADR]=DATA[NADR],PLNK[RADR]=NLNK[RADR]=0 [    ]  [    ]    [    ]    [    ]    [    ]   DIV .
188.   000500           BLOT=[1000*RADR+RADR]                       [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
189.   000520           JK=QUES.[PVR,DIVD],LK=QUES[2]               [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
190.   000530           DFV=3*[DOOM.[DIVD]-1]+CHOP.[WIG]-LK         [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
191.   000535           DATA[RADR]-10                               [    ]    [  46]    [    ]    [    ]    [    ]   DIV .
192.   000540           DATA[RADR]=9                                [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
193.   000550   46      E=1,DIVR=MULT.[E,DIV[3],BUOT]               [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
194.                    BLOT=DVFA.[BUOT]                            [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
195.   000560           DIVR=REST.[DPV,DIVR]                        [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
196.                    LIST.[DIVR]                                 [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
197.   000570           D=2,DIVD=SUBT.[D,DIVD,DIVR]                 [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
198.   000580      PVL  DIVR=DVFA.[DIVR],DASN.[DIVD]-1              [    ]    [  47]    [    ]    [    ]    [    ]   DIV .
199.   000590           DATA[RADR]=DATA[RADR]-1,AX=3                [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
200.   000600           DIVD=ADD.[AX,DIV[3],DIVD]                   [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
201.   000610   47      WIG=GETS.[DIVD]                             [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
202.   000611           TDVD=DATA[WIG],ET=CHOP.[WIG],TR=3           [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
203.                    DIG=NLNK[WIG]                               [  48]    [    ]    [    ]    [    ]    [    ]   DIV .
204.   000612      PVL  PLT=PLT-1                                   [    ]    [  34]    [    ]    [    ]    [    ]   DIV .
205.   000613           IL=0,ET-2                                   [  27]    [    ]    [  28]    [    ]    [    ]   DIV .
206.   000614           TL=4                                        [    ]    [    ]    [    ]    [    ]    [  41]   DIV .
207.   000615   27      TL=ET+1                                     [    ]    [    ]    [    ]    [    ]    [  41]   DIV .
208.   000616   28      TL=2                                        [    ]    [    ]    [    ]    [    ]    [  41]   DIV .
209.   000617   34      OMR=3*[DOOM.[DIVD]-1]+ET                    [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
210.   000618      PVL  DOMS=OMD-OMR,OMR-PVR                        [    ]    [    ]    [    ]    [  48]    [    ]   DIV .
211.   000619           OMD=OMR,TL=[[PVR-DOMS]+1]-ET                [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
212.   000620           BET=NADR,NADR=NXT,NXT=LKOT.[0]              [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
213.   000630           PLNK[NADR]=BET,NLNK[NADR]=NXT               [    ]    [    ]    [    ]    [    ]    [  41]   DIV .
214.   000650   48      REMAIN=DIVD                                 [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
215.   000660   14      IFRST=RELZ.[FRST],QUOT=[1000*IFRST+BET]     [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
216.   000670           DIV[1]=REPK.[QUOT]                          [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
217.   000680           SL1-SL2                                     [  49]    [    ]    [    ]    [    ]    [    ]   DIV .
218.   000690           CSIN.[DIV[1]],CSIN.[REMAIN]                 [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
219.   000700   49      TYPE-2                                      [    ]    [    ]    [  50]    [    ]    [    ]   DIV .
220.   000710           DVFA.[DIV[2]],DIV[2]=DIV[1]                 [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
221.   000720           DVFA.[DIV[3]],DIV[3]=DIV[1]                 [    ]    [    ]    [    ]    [    ]    [    ]   DIV .
```

```
  #      SEQ   LAFL  TYP                 STATEMENT                       C ZERO    NOT 0     PLUS     MINUS     ELSE

 222.   000010DOCM.    CNTR=1                                          [    ]   [    ]   [    ]   [    ]   [    ]   DOOM.
 223.   000020         RS1=GETS.[DOOM[1]]                              [    ]   [    ]   [    ]   [    ]   [    ]   DOOM.
 224.   000030   10    RS1=NLNK[RS1]                                   [  11]   [    ]   [    ]   [    ]   [    ]   DOOM.
 225.   000040         CNTR=CNTR+1                                     [    ]   [    ]   [    ]   [    ]   [  10]   DOOM.
 226.   000050   11    DCOM[1]=CNTR                                    [    ]   [    ]   [    ]   [    ]   [.   ]   DOOM.



 227.   000010DVFA.    LOC=GETS.[DVFA[1]]                              [    ]   [    ]   [    ]   [    ]   [    ]   DVFA.
 228.   000020   10    OC=NLNK[LOC]                                    [    ]   [    ]   [    ]   [    ]   [    ]   DVFA.
 229.   000025         LKIN.[LOC]                                      [    ]   [    ]   [    ]   [    ]   [    ]   DVFA.
 230.   000030         LOC=OC                                          [  11]   [    ]   [    ]   [    ]   [  10]   DVFA.
 231.   000040   11    DVFA[1]=0                                       [    ]   [    ]   [    ]   [    ]   [.   ]   DVFA.



 232.   000010FRAK.    M1=INT.[FRAK[2]/100]                            [    ]   [    ]   [    ]   [    ]   [    ]   FRAK.
 233.   000020         M2=INT.[[FRAK[2]/10]-M1*10]                     [    ]   [    ]   [    ]   [    ]   [    ]   FRAK.
 234.   000030         M3=INT.[FRAK[2]-[M1*100+M2*10]]                 [    ]   [    ]   [    ]   [    ]   [    ]   FRAK.
 235.   000040         FRAK[1]-2                                       [  11]   [    ]   [  12]   [  10]   [    ]   FRAK.
 236.   000050   10    FRAK[1]=M3                                      [    ]   [    ]   [    ]   [    ]   [.   ]   FRAK.
 237.   000060   11    FRAK[1]=M2                                      [    ]   [    ]   [    ]   [    ]   [.   ]   FRAK.
 238.   000070   12    FRAK[1]=M1                                      [    ]   [    ]   [    ]   [    ]   [.   ]   FRAK.
```

```
                                                                   PAGE 7.       MAY 27 66

 #      SEQ  LABL TYP                STATEMENT                C ZERO     NOT 0      PLUS       MINUS      ELSE

239.  000020GETE.    FINS=GETE[1]-[INT.[GETE[1]/1000]]*1000   [      ]  [      ]  [      ]  [      ]  [      ]   GETE.
240.  000030         GETE[1]=FINS                             [      ]  [      ]  [      ]  [      ]  [.     ]   GETE.




241.  000010GETS.    GETS[1]=INT.[GETS[1]/1000]               [      ]  [      ]  [      ]  [      ]  [.     ]   GETS.





242.  000010LIST.FIXINDX                                      [      ]  [      ]  [      ]  [      ]  [      ]   LIST.
243.  000020         INDX=GETS.[LIST[1]]                      [      ]  [      ]  [      ]  [      ]  [      ]   LIST.
244.  000030  10 PV  INDX,DATA[INDX],PLNK[INDX],NLNK[INDX]    [      ]  [      ]  [      ]  [      ]  [      ]   LIST.
245.  000040         INDX=NLNK[INDX]                          [.     ]  [      ]  [      ]  [      ]  [   10 ]   LIST.




246.  000010LKIN.    PLNK[LKIN[1]]=LOAS,NLNK[LKIN[1]]=0       [      ]  [      ]  [      ]  [      ]  [      ]   LKIN.
247.  000015         DATA[LKIN[1]]=0,NLNK[LOAS]=LKIN[1]       [      ]  [      ]  [      ]  [      ]  [      ]   LKIN.
248.  000020         LOAS=LKIN[1]                             [      ]  [      ]  [      ]  [      ]  [.     ]   LKIN.
```

```
                                                                                   PAGE 8.      MAY 27 66
   #      SEQ   LAPL  TYP                    STATEMENT                      C ZERO     NOT 0      PLUS      MINUS      ELSE
  249.   000010LKOT.     LKOT[1]=LOAS                                       [     ]   [     ]   [     ]   [     ]   [     ]   LKOT.
  250.   000020          LOAS=PLNK[LOAS]                                    [   10]   [     ]   [     ]   [     ]   [     ]   LKOT.
  251.   000030          NLNK[LOAS]=0                                       [     ]   [     ]   [     ]   [     ]   [.    ]   LKOT.
  252.   000040   10 PL  LOAS EMPTY - PROGRAM STOP                          [     ]   [     ]   [     ]   [     ]   [END  ]   LKOT.



  253.   000020LRGR.     MEMR1=GETE.[LRGR[1]]                               [     ]   [     ]   [     ]   [     ]   [     ]   LRGR.
  254.   000030          MEMR2=GETE.[LRGR[2]]                               [     ]   [     ]   [     ]   [     ]   [     ]   LRGR.
  255.   000035   15     STRT1=MEMR1,STRT2=MEMR2                            [     ]   [     ]   [     ]   [     ]   [     ]   LRGR.
  256.   000040          MEMR1=PLNK[MEMR1]                                  [   11]   [     ]   [     ]   [     ]   [     ]   LRGR.
  257.   000050   11     MEMR2=PLNK[MEMR2]                                  [   13]   [     ]   [     ]   [     ]   [   12]   LRGR.
  258.   000060   13     ABS.[DATA[STRT1]]-ABS.[DATA[STRT2]]                [.    ]   [     ]   [.    ]   [     ]   [   14]   LRGR.
  259.   000070   14     STOR=LRGR[1]                                       [     ]   [     ]   [     ]   [     ]   [   14]   LRGR.
  260.   000080          LRGR[1]=LRGR[2]                                    [     ]   [     ]   [     ]   [     ]   [     ]   LRGR.
  261.   000090          LRGR[2]=STOR                                       [     ]   [     ]   [     ]   [     ]   [     ]   LRGR.
  262.   000100   12     MEMR2=PLNK[MEMR2]                                  [.    ]   [     ]   [     ]   [     ]   [.    ]   LRGR.
                                                                            [     ]   [     ]   [     ]   [     ]   [   15]   LRGR.



  263.   000010MULT.     LRGR.[MULT[2],MULT[3]]                             [     ]   [     ]   [     ]   [     ]   [     ]   MULT.
  264.   000015          BIG=LRGR[1],SMALL=LRGR[2]                          [     ]   [     ]   [     ]   [     ]   [     ]   MULT.
  265.   000020          LC1=GETE.[BIG],PLNK[0]=NLNK[0]=0                   [     ]   [     ]   [     ]   [     ]   [     ]   MULT.
  266.   000030          LC2=GETE.[SMALL],CYCL=VAR=C=0                      [     ]   [     ]   [     ]   [     ]   [     ]   MULT.
  267.   000040          TYPE=MULT[1]                                       [     ]   [     ]   [     ]   [     ]   [     ]   MULT.
  268.   000060   10     DATA[LC2]                                          [   21]   [     ]   [     ]   [     ]   [     ]   MULT.
  269.   000070          FINL=DRS1=LKOT.[VAR],DRS2=LKOT.[VAR],CAR=0         [     ]   [     ]   [     ]   [     ]   [     ]   MULT.
  270.   000080          PLNK[DRS1]=DRS2,NLNK[DRS1]=0                       [     ]   [     ]   [     ]   [     ]   [     ]   MULT.
  271.   000082          TAR1=DATA[LC1],TAR2=DATA[LC2]                      [     ]   [     ]   [     ]   [     ]   [     ]   MULT.
  272.   000084          STAR1=TAR1☆TAR2                                    [     ]   [     ]   [     ]   [     ]   [     ]   MULT.
  273.   000100          DATA[DRS1]=GETE.[STAR1]                            [     ]   [     ]   [     ]   [     ]   [     ]   MULT.
  274.   000110          CELL1=GETS.[STAR1]                                 [     ]   [     ]   [     ]   [     ]   [     ]   MULT.
  275.   000120   11     LC1=PLNK[LC1]                                      [   12]   [     ]   [     ]   [     ]   [     ]   MULT.
  276.   000130          SET=DRS1,DRS1=DRS2,DRS2=LKOT.[VAR]                 [     ]   [     ]   [     ]   [     ]   [     ]   MULT.
  277.   000132          TAR1=DATA[LC1],TAR2=DATA[LC2]                      [     ]   [     ]   [     ]   [     ]   [     ]   MULT.
  278.   000134          STAR2=TAR1☆TAR2                                    [     ]   [     ]   [     ]   [     ]   [     ]   MULT.
  279.   000150          CELL2=GETE.[STAR2]                                 [     ]   [     ]   [     ]   [     ]   [     ]   MULT.
  280.   000160          DATA[DRS1]=QKAD.[CELL1,CELL2]+CAR                  [     ]   [     ]   [     ]   [     ]   [     ]   MULT.
  281.   000170          CAR=QKAD[2],CELL1=GETS.[STAR2]                     [     ]   [     ]   [     ]   [     ]   [     ]   MULT.
  282.   000180          PLNK[DRS1]=DRS2,NLNK[DRS1]=SET                     [     ]   [     ]   [     ]   [     ]   [     ]   MULT.
  283.   000190   12     SET=DRS1,DRS1=DRS2,DRS2=LKOT.[VAR]                 [     ]   [     ]   [     ]   [     ]   [   11]   MULT.
  284.   000200          DATA[DRS1]=QKAD.[CELL1,CAR]                        [     ]   [     ]   [     ]   [     ]   [     ]   MULT.
```

```
                                                                                     PAGE 9.      MAY 27 66

  #      SEQ   LABL TYP                 STATEMENT                           C ZERO     NOT 0      PLUS      MINUS      ELSE
 285.   000210          CAR=QKAD[2]                                       [  13]    [    ]    [    ]    [    ]    [    ]   MULT.
 286.   000220          CELL1=0                                           [    ]    [    ]    [    ]    [    ]    [  12]   MULT.
 287.   000230   13     STL=DRS1,PLNK[STL]=0,NLNK[STL]=SET                [    ]    [    ]    [    ]    [    ]    [    ]   MULT.
 288.   000235          LKIN.[DRS2]                                       [    ]    [    ]    [    ]    [    ]    [    ]   MULT.
 289.            14     1-[C=C+1]                                         [  17]    [    ]    [    ]    [    ]    [  15]   MULT.
 290.   000238   17     ISTL=RELZ.[STL]                                   [    ]    [    ]    [    ]    [    ]    [    ]   MULT.
 291.   000239          MULT[1]=[1000*ISTL+FINL]                          [    ]    [    ]    [    ]    [    ]    [  16]   MULT.
 292.   000240   15     ISTL=RELZ.[STL]                                   [    ]    [    ]    [    ]    [    ]    [    ]   MULT.
 293.   000245          PROD=[1000*ISTL+FINL]                             [    ]    [    ]    [    ]    [    ]    [    ]   MULT.
 294.   000250          PROD=PREZ.[CYCL,PROD]                             [    ]    [    ]    [    ]    [    ]    [    ]   MULT.
 295.   000255   16     1-@                                               [  18]    [    ]    [    ]    [    ]    [  19]   MULT.
 296.   000260   18     LC2=PLNK[LC2]                                     [  20]    [    ]    [    ]    [    ]    [    ]   MULT.
 297.   000270          LC1=GETE.[BIG],CYCL=CYCL+1                        [    ]    [    ]    [    ]    [    ]    [  10]   MULT.
 298.   000280   19     P=3,MULT[1]=ADD.[P,PROD,MULT[1]]                  [    ]    [    ]    [    ]    [    ]    [    ]   MULT.
 299.   000290          PROD=DVFA.[PROD]                                  [    ]    [    ]    [    ]    [    ]    [  18]   MULT.
 300.   000300   21     FINL=STL=LKOT.[VAR]                               [    ]    [    ]    [    ]    [    ]    [    ]   MULT.
 301.   000310          PLNK[FINL]=NLNK[FINL]=DATA[FINL]=0                [    ]    [    ]    [    ]    [    ]    [  14]   MULT.
 302.   000320   20     TYPE=2                                            [  22]    [    ]    [  23]    [    ]    [    ]   MULT.
 303.   000340   22     DVFA.[MULT[2]],MULT[2]=MULT[1]                    [    ]    [    ]    [    ]    [    ]    [    ]   MULT.
 304.   000350   23     DVFA.[MULT[3]],MULT[3]=MULT[1]                    [    ]    [    ]    [    ]    [    ]    [    ]   MULT.



 305.   000010PREZ.     PREZ[1]                                           [  12]    [    ]    [    ]    [    ]    [  15]   PREZ.
 306.   000015   12     PREZ[1]=PREZ[2]                                   [    ]    [    ]    [    ]    [    ]    [    ]   PREZ.
 307.   000020   15     DC=1,PLAS1=LKOT.[VAR],PLAS2=LKOT.[VAR]            [    ]    [    ]    [    ]    [    ]    [    ]   PREZ.
 308.   000030          PLNK[PLAS1]=BOX=GETE.[PREZ[2]],ZIP=GETS.[PREZC    [    ]    [    ]    [    ]    [    ]    [    ]   PREZ.
    .   000040          [2]],NLNK[BOX]=PLAS1
 309.   000050   16     DATA[PLAS1]=0,MOT=PLAS1,NLNK[PLAS1]=PLAS2         [    ]    [    ]    [    ]    [    ]    [    ]   PREZ.
 310.   000060          [DO=DO+1]-PREZ[1]                                 [    ]    [    ]    [  10]    [    ]    [    ]   PREZ.
 311.   000070          PLAS1=PLAS2,PLAS2=LKOT.[VAR]                      [    ]    [    ]    [    ]    [    ]    [    ]   PREZ.
 312.   000080          PLNK[PLAS1]=MOT                                   [    ]    [    ]    [    ]    [    ]    [  16]   PREZ.
 313.   000090   10     NLNK[PLAS1]=0,LKIN.[PLAS2]                        [    ]    [    ]    [    ]    [    ]    [    ]   PREZ.
 314.   000100          IZIP=ZIP                                          [    ]    [    ]    [    ]    [    ]    [    ]   PREZ.
 315.   000110          PREZ[1]=[1000*IZIP+PLAS1]                         [    ]    [    ]    [    ]    [    ]    [    ]   PREZ.
 #     SEQ    LABL TYP             STATEMENT                           C ZERO     NOT 0     PLUS      MINUS     ELSE
316.  000010QKAD.    LOT=QKAD[1]+QKAD[2]                                 [    ]    [    ]    [    ]    [    ]    [    ]   QKAD.
317.  000020         QKAD[1]=GETE.[LOT]                                  [    ]    [    ]    [    ]    [    ]    [    ]   QKAD.
318.  000030         QKAD[2]=GETS.[LOT]                                  [    ]    [    ]    [    ]    [    ]    [.   ]   QKAD.

319.  000010QUES.     QU=QUES[1],KT=0                                    [    ]    [    ]    [    ]    [    ]    [    ]   QUES.
320.  000020         Q1=FT=LKOT.[0],Q2=LKOT.[0]                          [    ]    [    ]    [    ]    [    ]    [    ]   QUES.
321.  000030         PLNK[FT]=0,NLNK[FT]=Q2                              [    ]    [    ]    [    ]    [    ]    [    ]   QUES.
322.  000040          GD=GETS.[QUES[2]],IQ=CHOP.[QD]                     [    ]    [    ]    [    ]    [    ]    [    ]   QUES.
323.  000050   12    DATA[Q1]=FRAK.[IQ,DATA[QD]]                         [    ]    [    ]    [    ]    [    ]    [    ]   QUES.
324.  000060         [KT=KT+1]-CU                                        [  10]    [    ]    [    ]    [    ]    [    ]   QUES.
325.  000070         IG=IQ-1                                             [    ]    [  11]    [    ]    [    ]    [    ]   QUES.
326.  000080         IG=3,QD=NLNK[QD]                                    [    ]    [    ]    [    ]    [    ]    [    ]   QUES.
327.  000090   11    Q=Q1,Q1=Q2,Q2=LKOT.[0]                              [    ]    [    ]    [    ]    [    ]    [    ]   QUES.
328.  000100         PLNK[Q1]=Q,NLNK[Q1]=Q2                              [    ]    [    ]    [    ]    [    ]    [  12]   QUES.
329.  000110   10    NLNK[Q1]=0,LKIN.[Q2]                                [    ]    [    ]    [    ]    [    ]    [    ]   QUES.
330.  000115         IFT=FT,GUS=[1000*IFT+Q1]                            [    ]    [    ]    [    ]    [    ]    [    ]   QUES.
331.  000120         MGUS=REPK.[QUS]                                     [    ]    [    ]    [    ]    [    ]    [    ]   QUES.
332.  000130         BAX=LRGR.[MQUS,DIV[3]],BAX-DIV[3]                   [    ]    [  15]    [    ]    [    ]    [    ]   QUES.
333.  000140         QUES[1]=-1,QUES[2]=QU+1                             [    ]    [    ]    [    ]    [    ]    [  16]   QUES.
334.  000150   15    QUES[1]=1,CUES[2]=QU                                [    ]    [    ]    [    ]    [    ]    [    ]   QUES.
335.  000170   16    MGUS=DVFA.[MQUS]                                    [    ]    [    ]    [    ]    [    ]    [.   ]   QUES.

336.  000020READ.    NUM=DUM=0                                           [    ]    [    ]    [    ]    [    ]    [    ]   READ.
337.  000030   15 CRDINPT                                                [    ]    [    ]    [    ]    [    ]    [    ]   READ.
338.  000040         NUM=NUM+1                                           [    ]    [    ]    [    ]    [    ]    [    ]   READ.
339.  000050         9999-INPT                                           [  16]    [    ]    [    ]    [    ]    [    ]   READ.
340.  000060         1000-ABS.[INPT]                                     [  10]    [    ]    [  11]    [  10]    [    ]   READ.
341.  000070   10 PL ERROR-DATA OUTSIDE ALLOWABLE RANGE                  [    ]    [    ]    [    ]    [    ]    [END ]   READ.
342.  000080   11    NUM-1                                               [  12]    [    ]    [    ]    [    ]    [  13]   READ.
343.  000090   12    PREV=0,FIRST=PRES=LKOT.[DUM],AFTR=LKOT.[DUM]        [    ]    [    ]    [    ]    [    ]    [    ]   READ.
344.  000100         DATA[FIRST]=INPT,PLNK[FIRST]=PREV,                C
   .                 NLNK[FIRST]=AFTR                                    [    ]    [    ]    [    ]    [    ]    [  14]   READ.
345.  000110   13    PRES=AFTR,AFTR=LKCT.[DUM]                           [    ]    [    ]    [    ]    [    ]    [    ]   READ.
346.  000120         DATA[PRES]=INPT,PLNK[PRES]=PREV,                  C
   .                 NLNK[PRES]=AFTR                                     [    ]    [    ]    [    ]    [    ]    [    ]   READ.
347.  000130   14    PREV=PRES                                           [    ]    [    ]    [    ]    [    ]    [  15]   READ.
348.  000140   16    LAST=PRES,NLNK[LAST]=0,LKIN.[AFTR]                  [    ]    [    ]    [    ]    [    ]    [    ]   READ.
349.  000145         IFIRST=FIRST                                        [    ]    [    ]    [    ]    [    ]    [    ]   READ.
```

```
                                                                                      PAGE 11.      MAY 27 66
  #      SEQ   LABL TYP                     STATEMENT                          C ZERO     NOT 0     PLUS     MINUS     ELSE
 350.   000150           READ[1]=[1000*FIRST+LAST]                             [     ]  [     ]  [     ]  [     ]  [     ]   READ.
 351.   000160           SIGN.[READ[1]]                                        [     ]  [     ]  [     ]  [     ]  [.    ]   READ.



 352.   000010RELZ.      $RELEASE LEADING ZEROS                                [     ]  [     ]  [     ]  [     ]  [     ]   RELZ.
 353.   000020    2      DATA[RELZ[1]]                                         [     ]  [.    ]  [     ]  [     ]  [     ]   RELZ.
 354.   000030           NL=NLNK[RELZ[1]]                                      [.    ]  [     ]  [     ]  [     ]  [     ]   RELZ.
 355.   000040           LKIN.[RELZ[1]]                                        [     ]  [     ]  [     ]  [     ]  [     ]   RELZ.
 356.   000050           REL7[1]=NL,PLNK[REL7[1]]=0                            [     ]  [     ]  [     ]  [     ]  [    2]   RELZ.



 357.   000010REPK.      G1=GETE.[REPK[1]],LEST=G2=LKOT.[0]                    [     ]  [     ]  [     ]  [     ]  [     ]   REPK.
 358.   000020           G3=LKOT.[0],PLNK[LEST]=G3,NLNK[LEST]=0                [     ]  [     ]  [     ]  [     ]  [     ]   REPK.
 359.   000030   10      DATA[G2]=DATA[G1],G1=PLNK[G1]                         [   11]  [     ]  [     ]  [     ]  [     ]   REPK.
 360.   000040           DATA[G2]=10*DATA[G1]+DATA[G2],G1=PLNK[G1]             [   11]  [     ]  [     ]  [     ]  [     ]   REPK.
 361.   000050           DATA[G2]=100*DATA[G1]+DATA[G2],G1=PLNK[G1]            [   11]  [     ]  [     ]  [     ]  [     ]   REPK.
 362.   000060           G4=G2,G2=G3,G3=LKCT.[0]                               [     ]  [     ]  [     ]  [     ]  [     ]   REPK.
 363.   000070           PLNK[G2]=G3,NLNK[G2]=G4                               [     ]  [     ]  [     ]  [     ]  [     ]   REPK.
 364.   000080   11      REPK[1]=DVFA.[REPK[1]],LKIN.[G3],PLNK[G2]=0           [     ]  [     ]  [     ]  [     ]  [   10]   REPK.
 365.   000090           IG=G2,REPK[1]=[1000*IG+LEST]                          [     ]  [     ]  [     ]  [     ]  [.    ]   REPK.



 366.   000010REST.      RPT=RS1=GETE.[REST[2]],R2=LT=LKOT.[0]                 [     ]  [     ]  [     ]  [     ]  [     ]   REST.
 367.   000020           R3=LKOT.[0],PLNK[R2]=R3,NLNK[R2]=0                    [     ]  [     ]  [     ]  [     ]  [     ]   REST.
 368.   000030           FR=1,[RST=REST[1]]-1                                  [   10]  [     ]  [     ]  [     ]  [     ]   REST.
 369.   000040   11      DATA[R2]=0,RST=RST-1                                  [   12]  [     ]  [     ]  [     ]  [     ]   REST.
 370.   000050           R4=R2,R2=R3,R3=LKCT.[0],PLNK[R2]=R3                   [     ]  [     ]  [     ]  [     ]  [     ]   REST.
 371.   000060           NLNK[R2]=R4                                           [     ]  [     ]  [     ]  [     ]  [     ]   REST.
 372.   000070   12      R4=R2,R2=R3,R3=LKCT.[0],PLNK[R2]=R3                   [     ]  [     ]  [     ]  [     ]  [   11]   REST.
 373.   000080           NLNK[R2]=R4,DATA[R2]=FRAK.[FR,DATA[RPT]]              [     ]  [     ]  [     ]  [     ]  [     ]   REST.
 374.   000090           [FR=FR+1]-4                                           [   13]  [     ]  [     ]  [   12]  [     ]   REST.
 375.   000100   13      FR=1,RPT=PLNK[RPT]                                    [   14]  [     ]  [     ]  [     ]  [   12]   REST.
```

```
                                                                          PAGE 12.      MAY 27 66

  #      SEQ   LABL TYP                 STATEMENT                     C ZERO     NOT 0     PLUS     MINUS     ELSE

376.   000110   14      PLNK[R2]=0                                    [    ]   [    ]   [    ]   [    ]   [    ]   REST.
377.   000115           IR=RELZ.[R2],REMP=[1000*[R+LT]                [    ]   [    ]   [    ]   [    ]   [    ]   REST.
378.   000120           REST[2]=DVFA.[REST[2]],REST[1]=REPK.[REMP]    [    ]   [    ]   [    ]   [    ]   [.   ]   REST.
379.   000130   10      DATA[R2]=FRAK.[FR,DATA[RPT]],FR=FR+1          [    ]   [    ]   [    ]   [    ]   [  12]   REST.


380.   000020RITE.      T=1,BLANK=48,MINUS=32,LOC=GETS.[RITE[1]]      [    ]   [    ]   [    ]   [    ]   [    ]   RITE.
381.   000030           DATA[LOC]                                     [    ]   [    ]   [    ]   [  10]   [  12]   RITE.
382.   000040   10      A[0]=MINUS                                    [    ]   [    ]   [    ]   [    ]   [  11]   RITE.
383.   000050   12      A[0]=BLANK                                    [    ]   [    ]   [    ]   [    ]   [    ]   RITE.
384.   000060   11      MAKE=ABS.[DATA[LOC]]                          [    ]   [    ]   [    ]   [    ]   [    ]   RITE.
385.   000065           A[T]=INT.[MAKE/100]                           [    ]   [    ]   [    ]   [    ]   [    ]   RITE.
386.   000070           A[T+1]=INT.[[MAKE/10]-A[T]*10]                [    ]   [    ]   [    ]   [    ]   [    ]   RITE.
387.   000080           A[T+2]=INT.[MAKE-[A[T]*100+A[T+1]*10]]        [    ]   [    ]   [    ]   [    ]   [    ]   RITE.
388.   000085           [T+2]-119                                     [  14]   [    ]   [    ]   [    ]   [    ]   RITE.
389.   000090           A[T+3]=BLANK                                  [    ]   [    ]   [    ]   [    ]   [    ]   RITE.
390.   000100   14      KCUNT=121-[T=T+4]                             [  13]   [    ]   [    ]   [    ]   [  15]   RITE.
391.   000110   15      LCC=NLNK[LCC]                                 [  16]   [    ]   [    ]   [    ]   [  11]   RITE.
392.   000120   16      S=T                                           [    ]   [    ]   [    ]   [    ]   [    ]   RITE.
393.   000130   17      A[S]=BLANK                                    [    ]   [    ]   [    ]   [    ]   [    ]   RITE.
394.   000140           120-[S=S+1]                                   [  19]   [    ]   [    ]   [    ]   [  17]   RITE.
395.   000150   13      LCC=NLNK[LCC]                                 [  19]   [    ]   [    ]   [    ]   [  18]   RITE.
396.   000160   18      T=1                                           [    ]   [    ]   [    ]   [    ]   [    ]   RITE.
397.   000175   19      ALFOUT.[%A%]                                  [    ]   [    ]   [    ]   [    ]   [    ]   RITE.
398.   000180           T-1                                           [  12]   [    ]   [    ]   [    ]   [.   ]   RITE.


399.   000020SIGN.      LCC=GETS.[SIGN[1]]                            [    ]   [    ]   [    ]   [    ]   [    ]   SIGN.
400.   000030   10      DATA[LOC]                                     [    ]   [    ]   [    ]   [  11]   [    ]   SIGN.
401.   000040           DATA[LOC]=ABS.[DATA[LOC]]                     [    ]   [    ]   [    ]   [    ]   [    ]   SIGN.
402.   000050           LCC=NLNK[LCC]                                 [.   ]   [    ]   [    ]   [    ]   [  10]   SIGN.
403.   000060   11      DATA[LOC]=-[ABS.[DATA[LOC]]]                  [    ]   [    ]   [    ]   [    ]   [    ]   SIGN.
404.   000070           LCC=NLNK[LCC]                                 [.   ]   [    ]   [    ]   [    ]   [  11]   SIGN.
```

```
                                                                                    PAGE 13.     MAY 27 66

  #      SEQ  LABL TYP                        STATEMENT                       C ZERO     NOT 0     PLUS     MINUS     ELSE

405.   000020SLAT.    LCAS=SLAT[1],IT=1                                       [     ]  [     ]  [     ]  [     ]  [     ]   SLAT.
406.   000030   11    DATA[IT]=0                                              [     ]  [     ]  [     ]  [     ]  [     ]   SLAT.
407.   000040         PLNK[IT]=IT-1,NLNK[IT]=IT+1                             [     ]  [     ]  [     ]  [     ]  [     ]   SLAT.
408.   000050         [IT=IT+1]-[N+1]                                         [     ]  [  11]  [     ]  [     ]  [     ]   SLAT.
409.   000060         NLNK[IT-1]=0                                            [     ]  [     ]  [     ]  [     ]  [.    ]   SLAT.


410.   000020SUBT.    HNBR=SUBT[1],CSIN.[SUBT[3]]                             [     ]  [     ]  [     ]  [     ]  [     ]   SUBT.
411.   000030         SUBT[1]=ADD.[SUBT[1],SUBT[2],SUBT[3]]                   [     ]  [     ]  [     ]  [     ]  [     ]   SUBT.
412.   000040         HNBR-2                                                  [  11]  [     ]  [.    ]  [  11]  [     ]   SUBT.
413.   000050   11    CSIN.[SUBT[3]]                                          [     ]  [     ]  [     ]  [     ]  [.    ]   SUBT.
414.   000320END   END                                                        [     ]  [     ]  [     ]  [     ]  [     ]   SUBT.


*****SYMBOL TABLE*****
                    ADD                          ADRS1                         ADRS2
                    ADRS                         AFTR                          A
                    BARO                         BLANK                         BOX
                    BET                          AJ                            B
                    CSIN                         CHOP                          CARY
                    CELL1                        CELL2                         CAR
                    DATA                         DASN                          DOOM
                    DVFA                         DIV                           DUM
                    DO                           C                             CYCL
                    DRS1                         DRS2                          DM1
                    FRAK                         FINS                          FIRST
                    FINL                         DM2                           DVD
                    GETE                         GETS                          G1
                    G2                           G3                            DVR
                    HOLD                         HLOC                          HNBR
                    DIG                          FRST                          G4
                    INPT                         IK                            IT
                    FR                           DPV                           IL
                    ET                           DOMS                          IQ
                    FT                           DIVID                         END
                    IFIRST                       I                             INT
                    FION                         FON                           DIVD
                    FJON                         BUOT                          E
                    DIVR                         D                             AX
                    IFRST                        CNTR                          INDX
```

```
          ABS                           BIG                           ISTL
          IZIP                          JK                            IFT
          BAX                           KOUNT                         KT
          IG                            LIST                          LKIN
          LKOT                          LRGR                          LOC1
          LOC2                          MULT                          MEMR1
          MEMR2                         LAST                          MINUS
          MAKE                          NLNK                          NUM
          LOC                           NL                            MOT
          LC1                           OC                            LC2
          LOAS                         *LS                            NADR
          LT                            PLNK                          PREZ
          PREV                          PRES                          PLAS1
          PLAS2                         QUES                          QKAD
         *PVD                           PVR                           LK
          OMD                           READ                          REPK
          REST                          RITE                          RELZ
          RS1                          *RS2                           RPT
          R2                            R3                            R4
          RIG1                          RIG2                          OMR
          PLT                           Q                             Q1
          Q2                            QD                            LEST
          N                             PAJ                           MPY
          REMAIN                        NXT                           MAX
          RADR                          QUOT                          M1
          M2                            M3                            PROD
          P                             LOT                           QU
          SIGN                          SLAT                          SUBT
          STRT1                         STRT2                         S
          TEMP                          T                             TEM
         *SBT1                         *SBT2                          THOW
          SON                           TYPE                          SET
          STL                          *SP                            SYPE
          VAR                           SL1                           SL2
          TR                            TL                            SUM
          WIG                           TDVR                          TDVD
          STOR                          SMALL                         TAR1
          X                             TAR2                          STAR1
          STAR2                         QUS                           MQUS
          Y                             RST                           IR
          REMP                          ZIP                           Z
         *ALFOUT

UNUSED MEMORY FROM [OCTAL]  14223 TO [OCTAL]  14325.
```

THE PRECISION ARITHMETIC SYSTEM

```
   3.0029500+05 $X=READ.[0]
           300             210            0           299
           299             359          300           298
           298             660          299           297
           297               1          298           296
           296               0          297           295
           295              21          296             0
  210 359 660 001 000 021

   2.9429100+05 $Y=READ.[0]
           294             -33            0           293
           293            -610          294           292
           292            -599          293           291
           291             -87          292             0
 -033 610 599 087

   2.9028900+05 $AJ=READ.[0]
           290             721            0           289
           289             399          290             0
  721 399

   2.8828700+05 $B=READ.[0]
           288             721            0           287
           287             200          288             0
  721 200

   2.8628200+05 $PAJ=READ.[0]
           286            -500            0           285
           285               0          286           284
           284            -100          285           283
           283            -637          284           282
           282            -119          283             0
 -500 000 100 637 119
```

```
  1.0000000+00 $Z=1     2.7628100+05 $SUM=ADD.[Z,X,Y]

         276             210               0            277
         277             359             276            278
         278             626             277            279
         279             390             278            280
         280             400             279            281
         281             934             280              0

 210 359 626 390 400 934
```

```
  3.0000000+00  $Z=3      2.8728700+05  $B=SUBT.(Z,AJ,B)

        287                199             0              0

 199
```

```
1.0000000+00 $Z=1      2.6028800+05 $MPY=MULT.[Z,SUM,AJ]

      260              151              0             269
      269              753            260             270
      270              224            269             271
      271              118            270             272
      272              408            271             273
      273              843            272             274
      274              386            273             288
      288              666            274               0

151 753 224 118 408 843 386 666
```

```
 1.0000000+00 $Z=1      2.5025300+05 $DIVID=DIV.[Z,PAJ,B]

        250             -251              0              251
        251             -256            250              252
        252             -787            251              253
        253             -121            252                0

-251  256  787  121
```

```
  2.5425400+05 $REMAIN

        254                 -40              0              0


-040


TYPE #END# STATEMENT EXECUTED.
CARDS REMAINING IN DECK ARE--

MAY 27 66        15 31.2
```

BIBLIOGRAPHY

1. Berkeley, E. C. "The Programming Language LISP: An Introduction and Appraisal." _Computers and Automn_. 13, Sept. 1964, 16-23.

2. Comfort, W. T. "Multiword List Items." _Commun. ACM_, 7, June 1964, 357-362.

3. Gelernter, H., Hansen, J. R., and Gerberich, C. L. "A Fortran-Compiled List Processing Language." _J. Assn. Computing Mchy_., 7, April 1960, 87-101.

4. McCarthy, J. "LISP-I·5 Programmers Manual." M.I.T., 1960.

5. Noonan, R. E., Smith, W. A. Jr., Rayna, G. "LEWIZ Programmer's Reference Manual." Lehigh Univ. Computing Lab., Sept. 1963, revised Oct. 1965.

6. Weizenbaum, J. "Symmetric List Processor." _Comm. ACM_ 6, Sept. 63, 524-544.

VITA

Andrew J. Kasarda was born on July 13, 1940, in the town of Phoenixville, Pennsylvania. His parents are Mr. and Mrs. Andrew Kasarda Sr. He attended the Pennsylvania State University and received a B. A. Degree in Mathematics in June 1962. He taught Mathematics and Computer Programming at Shippensburg State College from September 1962 to August 1964.