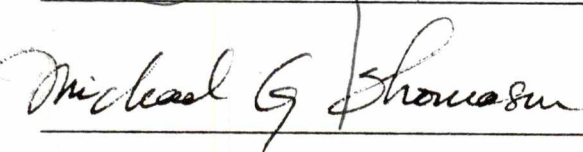


To the Graduate Council:

I am submitting herewith a thesis written by Jean K. Gaffney entitled "A Subset of MUMPS for the VM Operating System." I have examined the final copy of this thesis for form and content and recommend that it is accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.


Kevin C. O'Kane
Major Professor

We have read this thesis and
recommend its acceptance:

Accepted for the Council:


The Graduate School

A SUBSET OF MUMPS FOR THE
VM OPERATING SYSTEM

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Jean K. Gaffney

August 1983

to

Pat

ACKNOWLEDGEMENTS

I would like to thank all the people who have helped and encouraged me during the course of this degree.

In particular I would like to thank Dr. Kevin O'Kane for all his help and for sharing his MUMPS expertise with me. I would like to thank Dr. Charles Pfleeger and Dr. Michael Thomason for persuading the graduate school to admit me to the masters program. I am also grateful to Dr. Thomason for all his advice and for his helpful corrections to the thesis. I would like to express my thanks to all the members of the Computer Science Department for their help and guidance over the past four years. I would especially like to thank Dr. David Straight, not only for his technical assistance, but also for his moral support, which was often much needed. I am also indebted to my colleagues in the Computing Center for all their help and encouragement.

Finally, I would like to thank my husband, Pat, without whose help, encouragement and patience I would never have finished this degree.

ABSTRACT

MUMPS is a high level computer language that was designed at Massachusetts's General Hospital in 1967. The purpose of this thesis is to describe a PL/I version of MUMPS which is designed for an IBM computer running the VM operating system. Specifically, the thesis describes how a PDP-11 FORTRAN version of MUMPS was converted into a PL/I version suitable for a single-user environment. The problems that arose during the conversion are described in detail. The thesis concludes with an extensive examination of the ways in which the single-user version can be efficiently transformed into a multi-user system.

TABLE OF CONTENTS

CHAPTER	PAGE
1. INTRODUCTION	1
2. THE HISTORY OF MUMPS	3
Background	3
Computing in the 1960s	4
Design of MUMPS	5
Features of the MUMPS Language	7
MUMPS Applications	9
3. THE IMPLEMENTATION LANGUAGE	11
Input/Output	13
Interrupt Handling	15
Standard PL/I	16
4. THE PL/I IMPLEMENTATION	19
5. FUTURE IMPLEMENTATIONS	29
RSCS--Remote Spooling Communications Subsystem	29
DIAL	30
IUCV--Inter-User Communications Vehicle . . .	30
VMCF--Virtual Machine Communications Facilities	32
MSG--Special Message Facility	37
File Access	37
IFS--Interactive File Sharing Program	38
6. DISCUSSION	41
REFERENCES	43

APPENDICES	46
APPENDIX A Hardware Configurations	47
APPENDIX B Program Listing	50
VITA	163

LIST OF FIGURES

FIGURE	PAGE
1. A HIERARCHICAL STRUCTURE	6
2. RELATIONAL OPERATORS	8
3. PL/I COMPILERS	16
4. TECO COMMAND	22
5. TEMPORARY FILE PRINT PROGRAM	24
6. SEND/RECV PROTOCOL	36

CHAPTER 1

INTRODUCTION

MUMPS is a high level programming language that was introduced in 1967 by the medical staff of the Massachusetts General Hospital[3]. It was initially used for applications pertaining to the medical profession. However, today, because of its flexibility, MUMPS is used for many scientific and commercial applications.

In 1981 the only version of MUMPS that could be used on an IBM Computer was an implementation designed for an IBM Series 1. This was a serious shortcoming in view of the predominance of IBM machines other than this model. Consequently, in order to fill this gap, a decision was made in June 1981, to produce a MUMPS interpreter suitable for IBM mainframe computers. Initially, the interpreter would be for a single-user environment. Because of the large amount of string manipulation inherent in MUMPS, it was decided to produce this version in the high level language PL/I. The main purpose of this thesis is to describe the PL/I implementation.

The computer language MUMPS, its history and some existing applications are described in Chapter 2. Some reasons for choosing PL/I as the preferred language for the new version are given in Chapter 3. The details of converting an existing FORTRAN version of the interpreter to PL/I are

described in Chapter 4. Chapter 5 looks at how this single user version could be expanded to a multi-user version. In chapter 6 the possibility of implementing the MUMPS interpreter in FORTRAN 77 is examined and finally a 4341 hardware configuration suitable for running the MUMPS interpreter under VM is detailed.

It is worth noting that at present there are at least two other implementations of MUMPS for IBM computers. The first one is a FORTRAN '66 version suitable for an IBM mainframe computer[16] and the second one is a version which is written in assembler language for the IBM PC[10]. These versions are evidence of the growing popularity of the MUMPS language as a suitable vehicle for database applications.

CHAPTER 2

THE HISTORY OF MUMPS

Background

As a consequence of the increased awareness of computers most people today have heard of the languages FORTRAN and COBOL. However, only a limited number of people have heard of the language MUMPS. This language was developed in the late 1960s by a group of physicians and computer scientists at the Massachusetts General Hospital[3]. It grew out of the need to cope an ever increasing volume of complex, unstructured health care information. For example, the clinical information about a patient is derived from many sources: the doctor, the ward nurse, the X-ray department, the laboratory etc. The data from the various sources have to be correlated and made available for quick and easy access. To cope with these problems therefore, MUMPS was designed to be a computer language that incorporated a sparse hierarchical data base system and that provided the capability of a multi-user environment (the name MUMPS is an acronym derived from Massachusetts General Hospital Utility Multi-programming System). By considering some of the developments that were taking place in the field of computing at the time MUMPS was conceived, it is possible to understand some of the factors that influenced its design.

Computing in the 1960s

The advent of computers introduced the potential of being overwhelmed by large amounts of data. Papers were written on the subject and the term "data base" began to appear. This term was originally used in connection with military information systems to mean the set of data accessed by time sharing users[12]. Today its meaning is broader. A data base is a collection of data usually stored in records or files that are interrelated. The mid 1960s saw the introduction of Data Base Management Systems, DBMS. This is a set of software that controls access to the data base. Some early examples of these systems are Informatics' MARKIV, IBM's IMS and Intel's System 2000, all of which were hierarchical data base systems. This type of data base is one in which the relationship between records is strictly one of subordination or ownership (more usually stated as a parent- child relationship). To provide greater flexibility in the manipulation of data records, so called network and relational data base systems began to appear at the end of the 1960s.

Another occurrence that influenced the design of MUMPS was the advent in 1964 of a successful interactive terminal system. IBM introduced a system called SABRE, Semi-Automatic Business-Related Environment[11]. This system resulted from a joint project with American Airlines to design and implement an electronics reservation system. SABRE handled seat

reservations and passenger name records and automatically requested space on other airline flights when necessary.

Design of MUMPS

The physicians and computer scientists at the Massachusetts General Hospital decided that the optimal method for handling clinical data was to incorporate the concepts of a computer language with a data base management system which allowed multi- user access. At that time only hierarchical DBMS had been introduced on computers and so it was the obvious choice for MUMPS. It was also easy to impose a hierarchical structure on the information concerning a specific patient - see Figure 1.

However, the way in which the DBMS was implemented in MUMPS is interesting and deserves some comment. A usual way of storing data in a data base is in the form of an array of subscripted variables[20]. In most programming languages it is necessary to provide storage for the entire array even though only some of the elements of the array are to contain values. For example, to assign values to elements 1 and 100 of an array it is also usually required to provide storage for the intervening elements 2 through 99. This results in a considerable waste of space if these elements are never assigned values.

The DBMS in the MUMPS language was designed to avoid such wasted space and therefore only requires storage for those

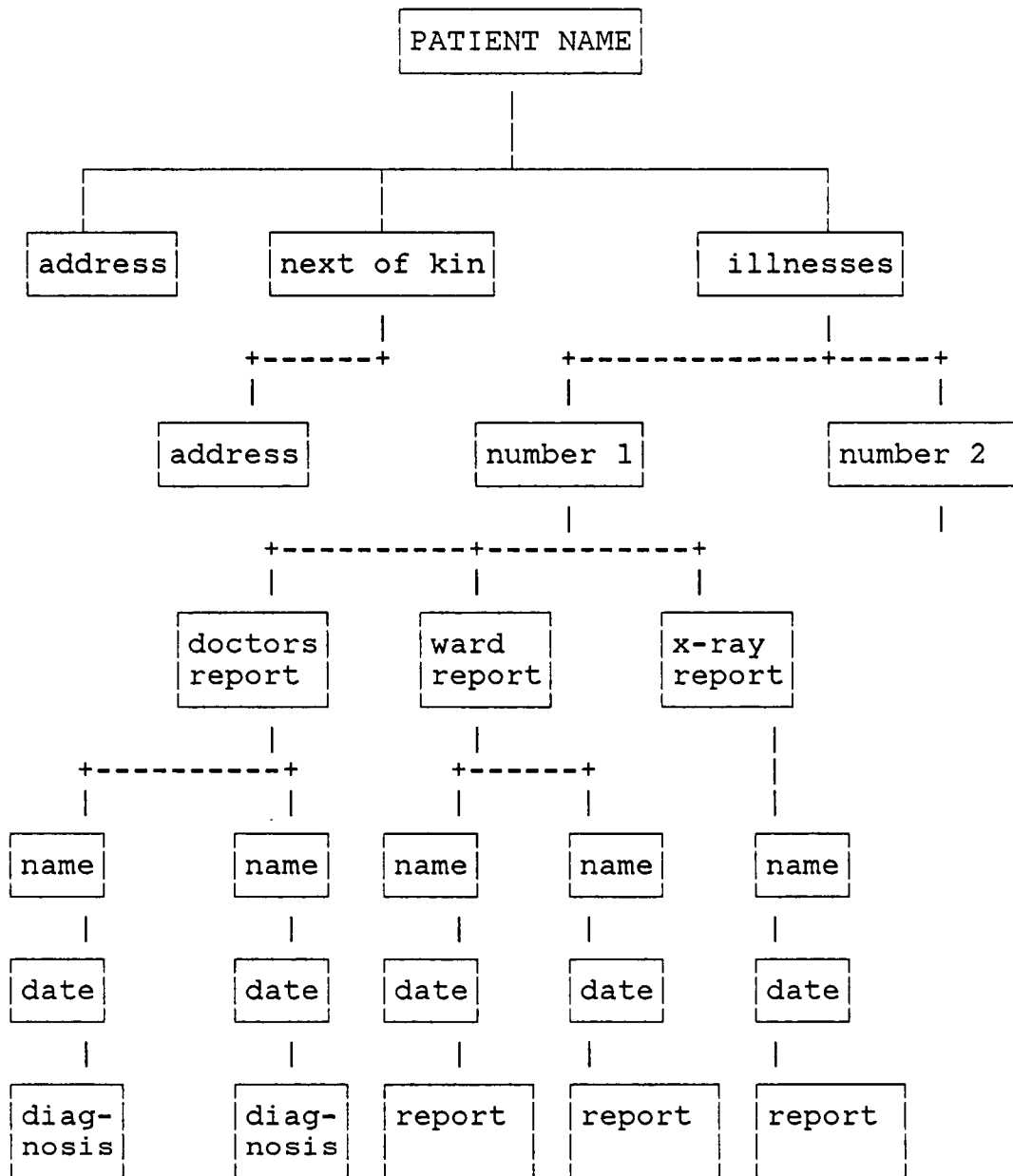


FIGURE 1. A HIERARCHICAL STRUCTURE

elements that are to receive values. It is for this reason that the DBMS used in the language is referred to as a "sparse" hierarchical DBMS. One result of this design feature is a degree of flexibility in the subscripting of variables. For example in some implementations of MUMPS it is possible to use subscripts, such as character strings, which may be more meaningful than integers, depending on the context. This flexibility is particularly relevant to applications programmers.

Not only was MUMPS designed as a sparse DBMS, but also the elements of the arrays could be of varying length. This allowed data, that could vary in length from the six characters needed to represent the date to the several hundred characters that might be needed to hold a doctor's diagnosis, to be held in the same data base.

By designing MUMPS as a multi-user interactive system the problem of correlating data from several different sources was addressed. Terminals could be situated at the relevant locations for the sources of information and the data entered directly into one composite data base.

Features of the MUMPS Language

The MUMPS language is unusual in that there is only one data type: a variable length character string consisting of up to 255 characters[13]. A number is treated as a special type of character string. This entails an internal conversion of

character strings in order to perform algebraic operations. There is therefore, a considerable overhead entailed in performing calculations.

MUMPS was designed principally to handle non-numeric data and therefore, as well as the usual arithmetic operators it has string relational operators. These are shown in Figure 2.

<u>Operator</u>	<u>Example</u>	<u>Result</u>
=	ANIMAL="CAT"	1 if value of animal is CAT 0 otherwise
[	TITLE["BLUE"	1 if the word BLUE is in the title 0 otherwise
]	TITLE]B	1 if the value of title follows B 0 otherwise
?	TITLE?20A	1 if title has exactly 20 alphabetics 0 otherwise

FIGURE 2. RELATIONAL OPERATORS

Another interesting aspect of MUMPS is 'indirection'. This allows data values to be interpreted as MUMPS code as the following example demonstrates.

EXAMPLE

a. SET TITLE="WRITE !,""THESIS"""

b. XECUTE TITLE

Statement b. executes the string contained in TITLE. This will cause the word THESIS to be written on a new line. An exclamation mark(!) in MUMPS means new line.

Full details of the MUMPS language are defined in the MUMPS Language Standard, ANSI XII.1-1977.

MUMPS Applications

The Department of Health Education and Welfare sponsored the development of MUMPS and the production of associated applications programs. As a result of this there now exists a plethora of MUMPS medical applications software. This software covers areas as diverse as hospital billing, hospital pharmacy systems, laboratory information systems and out patient scheduling systems. Indeed, some hospitals are completely committed to MUMPS and have written application programs to control all branches of hospital management. An example of this is described in a paper in the proceedings of the 1980 MUMPS Users' Group Meeting[17]. The subject of the paper is the Community General Hospital of Greater Syracuse. In 1972 the hospital obtained a PDP-15 and subsequently installed a history taking system. Between 1972 and 1978 the staff of the hospital wrote an admitting system, a financial and administrative system, a medical record system, a food service system, a chemistry control system and several other minor systems.

Moreover, the MUMPS language has now spread into many other fields and today there also exists pharmaceutical, inventory control, billing/accounts receivable, banking, corporate strategic planning and library software to name but a few areas.

CHAPTER 3

THE IMPLEMENTATION LANGUAGE

In recent years as the cost of computers has decreased, the acquisition of mini and micro computers has become possible for many more businesses. For example, ten years ago in the medical profession only large hospitals could afford a computer. Nowadays, most individual general practitioners can afford to buy their own. However, in order to take advantage of the existing MUMPS software it is also necessary for them to obtain a MUMPS interpreter.

There are many versions of MUMPS available for a variety of manufacturers' hardware. There are implementations for the DEC10, PDP-11, VAX, Apple, Harris, Superpet, NOVA and the IBM Series 1 to mention but a few. There is, however, one large range of hardware that, until recently, did not have a MUMPS interpreter. This is the IBM 360/370 range of computers. The subject of this thesis provides the basis for an interpreter that is suitable for these computers.

The design requirements of MUMPS imposed certain constraints on the choice of high level language for the new interpreter. Since MUMPS is designed to manipulate data, it follows that it should be capable of handling characters efficiently. It is well known that the FORTRAN '66 language is deficient in this respect[4]. Moreover, the FORTRAN '77 language provides only limited character handling facilities

and at the time this work was commenced a suitable compiler for FORTRAN '77 did not exist. Therefore, PL/I seemed the most appropriate language for the implementation[8]. Some of the reasons for this last statement are given in the following paragraphs.

PL/I allows variables to be declared as character strings of fixed or varying length. It supplies the operator 'concatenation' and several functions for manipulating strings. Some of the available functions are:-

BIT.....converts a data item to a bit string

CHAR.....converts a data item to a character string

INDEX.....finds a given substring in a string of
characters

LENGTH.....gives the number of characters in a string

STRING.....concatenates all the elements in an
aggregate variable into a string

SUBSTR.....extracts a specific substring

TRANSLATE...replaces a specified string with a new
string

VERIFY.....checks to see that all the bits or
characters in a given string occur in a
specified second string

There is an additional nice feature about varying length character strings that can be used to advantage. PL/I stores the current length in the two bytes immediately preceding the string[9]. These bytes are accessible by using the ADDR

function. If ADDR is used to find the address of a varying character string it returns the address of the length.

PL/I has the facility to declare a collection of different data types as a single structure. One advantage of this is that all variables concerned with a particular aspect can be gathered together under one name. This greatly improves the clarity of the program.

Other important aspects of the MUMPS language are the creation and manipulation of the data base, and its interactive facilities. PL/I provides excellent tools for interacting with the operating system. The sets of tools for controlling Input/Output and Interrupt Handling are now detailed.

Input/Output

The Input/Output (I/O) statements provided in PL/I are more numerous and flexible than those supplied in FORTRAN. There are two main types of I/O in PL/I, namely stream oriented and record oriented. Stream I/O is the simpler and more basic method and is almost completely machine independent. All data is considered to be a continuous stream of characters. On input this flow of characters is divided into data fields and assigned to variables. On output the value of variables is converted to characters and output in a continuous stream. There are various degrees of control that can be exercised over stream I/O, from the very simplest where

all the variable defaults are taken, to extremely complicated editing. The latter is similar to but more extensive than FORTRAN format statements.

The two stream I/O statements are called GET and PUT. As their names imply GET is for input and PUT is for output. These two statements are qualified according to the type of stream I/O chosen. The available choices are called:

list-directed

data-directed

and edit-directed stream I/O.

GET and PUT are followed by the name of the file associated with the data and the list of variables. For edit directed stream I/O, the data format is also required. The simplest form of stream I/O is data-directed. The name and value of the variable are contained in the data stream. List-directed specifies a list of variable names. The data are then assigned to each variable in turn. Edit-directed is the closest to FORTRAN I/O. It allows exact specification of the characteristics of the data and on output it also encompasses controlling the format of line, page, etc.

Examples

a. List-directed Specification

```
GET LIST(X,Y,Z);
```

The data read in would be of the form:

```
1.2,5.3,8.0
```

b. Data-directed Specification

```
GET DATA(X,Y,Z):
```

The data read in would be of the form:

```
X=1.2,Y=5.3,Z=8.0
```

c. Edit-directed Specification

```
GET EDIT(DAY,YEAR)
```

```
(F2,X2,F4);
```

This would result in two digits being assigned to DAY two digits being skipped and four digits being assigned to YEAR.

Record I/O is machine dependent and deals with groups of data divided into records. The two main record statements are READ and WRITE which will transmit a record at a time.

The PL/I 'declare' statement allows the characteristics of a file to be specified exactly. Together with the I/O statements PL/I has complete control over reading and writing files.

Interrupt Handling

A further control that PL/I affords the programmer is in control over interrupt handling. It allows the programmer to define the path the program should take when a specific interrupt occurs. This is done through the ON command. For instance,

```
ON ENDFILE(file-name) GO TO FINIS;
```

will transfer execution to the label FINIS when the end of the file is encountered.

There are several ON commands. For example ON KEY, ON CHAR, ON ATTENTION, ON ERROR to name a few. These ON commands can be further qualified by the ONCODE command which allows the precise condition code to be intercepted and examined.

Standard PL/I

The PL/I language is not confined to IBM computers[1]. In fact, as Figure 3 shows, most of the major machine ranges offer a PL/I compiler

<u>MANUFACTURER</u>	<u>RANGE</u>	<u>STANDARD</u>
Digital	VAX DEC-10	Subset G
CDC	CYBER 70 CYBER 170	ANSI
	6000 7600	ANSI
Honeywell	Series 60	ANSI
Univac	Series 1100	ANSI
Burroughs	B6000 B7000	IBM
Data General	ECLIPSE	ANSI

FIGURE 3. PL/I COMPILERS

Until 1976 the IBM definition of PL/I was the accepted standard. However, in 1976 an ANSI Standard, X3.53-1976 was established. Consequently, some of the compilers adhere to

IBM's standard and some to ANSI. Although the two standards are similar, there are a few significant differences which are now detailed.

a. Arithmetic Defaults.

The ANSI Standard makes all undeclared identifiers have the attributes BIN FIXED (fixed point binary). IBM's rule is more complicated. It is the same as in FORTRAN. If the variable name begins with I--->N the attributes are BIN FIXED otherwise the attributes are DEC FLOAT (floating point decimal).

b. Arrays

The statement $A = A + A(2)$ will add the value of $A(2)$ to each member of A when adhering to the ANSI Standard. However, by IBM's standard, as this is implemented in a loop, every member of A after $A(2)$ will be incremented by twice $A(2)$.

c. Preprocessor

The only preprocessor statement defined in the ANSI Standard is %INCLUDE which is used to include strings of external text into the program. This is much less powerful than IBM's preprocessor which has eleven additional statements. Full details of these are in IBM's PL/I Language Reference Manual.

d. Omissions

The ANSI Standard has omitted several statements and functions that are in the IBM compilers. For example

DISPLAY, SELECT, DO UNTIL, ANY, ALL, COMPLETION to name a few. Multi-tasking is also omitted from the ANSI Standard.

Of the vendors mentioned above, Burroughs has followed IBM's standard. All the others have followed the ANSI standard. In particular Digital has followed the proposed PL/I G(General Purpose) Subset, ANSI BSR X3.74.

This chapter has illustrated the power of PL/I. It was designed to encompass all the numeric features of FORTRAN and all the data handling features of COBOL. As can be seen it has done this and more. It was, therefore, thought to be the most suitable language available for implementing the new MUMPS interpreter.

CHAPTER 4

THE PL/I IMPLEMENTATION

At the beginning of this project it was decided that a convenient way to produce a new interpreter would be to convert an existing interpreter to PL/I. The choice of which interpreter to use for the conversion was simplified since there already exists at the University of Tennessee two FORTRAN versions of MUMPS designed for PDP-11s and DEC10s. These were written and are maintained by Dr. Kevin O'Kane[16]. The PL/I version was designed to maintain as closely as possible the flow and format of the existing FORTRAN versions. In this way it would be relatively easy to maintain all the versions in parallel.

There were essentially two methods by which the conversion could be accomplished. The first involves the use of an automatic conversion program which takes as its input the FORTRAN MUMPS interpreter and produces as output a PL/I version. The second relies upon the manual conversion of the FORTRAN code. This latter method is prone to numerous errors and is not as efficient as a good automatic program. For these reasons it was decided to use an automatic conversion program. In order to do this it was necessary to investigate the desirability of either using an existing program or of writing one. To determine the amount of work required for the latter a flow diagram of a suitable conversion program was

created. The conclusion arrived at was that writing such a program was prohibitive within the timeframe of this thesis.

The next step in the project entailed investigating existing conversion programs. Contrary to what might be expected there are very few programs for converting FORTRAN to PL/I. However, it was found that IBM had marketed such a program in the 1960s as an FDP (Field Developed Program) and there was in fact a copy of it at the University. This tape, however, proved to be so old it was unusable. Further investigation revealed that a tape containing VM software from Waterloo University contained a version of the program. There was also a copy of this tape at the University. Unlike the other tape this one was usable and therefore could be applied to this particular conversion.

The FORTRAN files were copied from the tape to a VM minidisk and a linkedit job stream was submitted through the VM RJE interface to create a load module.

A disadvantage of the IBM program is that it was written to convert IBM FORTRAN to IBM PL/I and not PDP-11 FORTRAN to PL/I. Therefore, at best it was only going to convert about 70% of the program. The remaining conversion would have to be performed manually.

The procedure that was adopted to perform the conversion was as follows:

- a. The MUMPS interpreter was stored as a series of files on the DEC-10. The leading page marks of each file had to

be stripped off so that the conversion program would not terminate after reading the initial page mark.

- b. The files were submitted one at a time to the IBM 370 as data to the conversion program. The JCL contained a `//*CMSUSER` card so that the converted file could be retrieved into a CMS minidisk.
- c. Two CMS EXECs were written to retrieve the output from the conversion program. One EXEC retrieved the punch file output and the other retrieved the listing file and submitted an RJE job to produce a hardcopy listing.
- d. The minidisk is only one cylinder so as it filled up an IEBUPDTE job stream was submitted to archive it to a disk dataset.

Many problems were encountered while trying to adhere to this procedure.

- a. PDP-11 FORTRAN allows comments to be put on the same line as a statement by preceding the comment with an exclamation mark. This sometimes caused the conversion program to loop. Therefore TECO was used to place these comments on a new line starting with a 'C' in column 1. TECO is a powerful text editor on the DEC-10. It was possible to use a one line TECO command to accomplish this (see Figure 4).
- b. Non-standard FORTRAN function names also caused the conversion program to loop. This was bypassed by

999<FN!\$CRC! \$-1L\$2T\$2L\$;\$>\$

Explanation

999	=	repeat factor.
		repeats everything within < >.
FN!\$	=	find every occurrence of the character '!' and replace it by the following string
<CR><LF>C!	\$ =	the replacement string. In this case it is..carriage return C! and two spaces.
-1L	=	go back to the start of the line
2T\$	=	print two lines.
		i.e the original line and the changed line
2l\$	=	now space over these two lines and continue
;\$	=	halt when search fails
\$	=	execute the command

FIGURE 4. TECO COMMAND

including the function calls in the declarations as dimension statements.

- c. The DEC-10 test version of Galaxy caused files sent from the 10 to be truncated. It was not immediately obvious that Galaxy was at fault as the result was similar to the conversion program receiving a page mark.

When these problems had been solved nearly all the MUMPS modules were run through the conversion program. This however left three or four modules that just kept looping for no apparent reason. To investigate the cause of this problem the temporary work file that is used by the conversion program was made into a permanent dataset. A small PL/I program was written to print this dataset as it is a keyed direct access dataset (see Figure 5). By examining the printout of this dataset, after a failing run of the conversion program, it could be seen which instruction was looping. It usually turned out to be an 'OR' statement. This is a probable bug in the conversion program.

The optimistic conversion rate of 70% turned out to be an overestimate. There were many expressions that the converter could not translate. In particular Boolean expressions, with more than one operator, caused unpredictable behavior. Sometimes they were completely commented out and other times they were partly translated and partly commented.

```

WRKPRT: PROC OPTIONS(MAIN);

      DCL SYSUT1 FILE RECORD KEYED ENV(REGIONAL(2));

      DCL PASSKEY CHAR(9);

      DCL PASSOPS CHAR(90);

      OPEN FILE(SYSUT1) SEQUENTIAL INPUT;

      ON ENDFILE(SYSUT1) GOTO EOF;

L1:    READ FILE(SYSUT1) INTO(PASSOPS) KEYTO(PASSKEY);

      PUT SKIP EDIT ('KEY',PASSKEY,'CARD',PASSOPS)

              (A,X(3),A,X(5),A,X(3),A);

      GOTO L1;

EOF:   CLOSE FILE(SYSUT1);

END WRKPRT;

```

FIGURE 5. TEMPORARY FILE PRINT PROGRAM

A large proportion of the problems that occurred during the translation stemmed from the fact that many of the arrays in the FORTRAN version were declared as type 'BIT' whereas in the PL/I version they were converted to 'CHAR' arrays. For example, the end of string marker, which is binary zero, could be denoted by a straightforward assignment in the original version i.e. $XD(1) = 0$. However, in the PL/I version the equivalent assignment had to be accomplished by the following sequence of statements

```
DCL AC(2)  CHAR DEF AB;  
DCL AB     FIXED BIN(1510);  
AB = 0;  
XD(1) = AC(1);
```

This meant that every zero that occurred in the program had to be examined to determine whether it was an end of string marker or a genuine zero. Inevitably some places where changes should have been made were missed and they proved extremely difficult to debug.

Changing the array definitions to character type also made some assignment statements illegal and these had to be overcome.

A major problem occurred with the conversion of COMMON blocks. To explain this, consider the following example of a common block, IO, that is defined by the statements

```
COMMON /IO/ A,B,C  
INTEGER A,B,C
```

which appear in a main program. In a subsequent FORTRAN subroutine it is only necessary to describe in detail the part of the common block that is going to be used. Therefore, in the above example if only the last variable of the block is to be specifically referenced then the remainder of the block can be declared by one name

```
COMMON /IO/ TEMP,C  
INTEGER TEMP(2),C
```

Similarly, if only the first variable is to be referenced then the statements

```
COMMON/IO/ A
```

```
INTEGER A
```

are adequate. The translation program converted the COMMON blocks to PL/I EXTERNAL structures. The above case produced inconsistent declarations as EXTERNAL structures must have exactly the same format and must be exactly the same length. This led to some difficult debugging as the PL/I compiler did not flag this as an error and store corruption occurred.

Another set of problems occurred when passing a part of a structure, which is an array, to a subroutine or a function. The only functions that can accept structures as arguments are ADDR, ALLOCATION and STRING. This problem was solved by passing the address of the array part of the structure to the subprogram. Within the subprogram the address was mapped to store.

Labels also caused some difficulties since they are held as four bytes in FORTRAN and eight bytes in PL/I. In the original version of MUMPS an array is used to hold the values of certain variables and a return label for nesting subroutine calls etc. This array is declared as fixed binary with precision 31. In the PL/I version the label therefore, had to be separated out and saved in an array declared as type label.

The FORTRAN version of MUMPS had been written for the PDP-11 and was based on the Ascii code. It had used Ascii codes 206 to 210 in its expression handling routine. These codes are outside the defined ASCII character set. However, some of these codes have legitimate meanings in EBCDIC, for example 210=K, these had to be changed and are now 251 to 255.

Another place where the difference between Ascii and EBCDIC gave rise to problems was in dealing with the alphabet. In Ascii A to Z are adjacent codes whereas in EBCDIC they have been divided into three groups. A to I, J to R and S to Z. Between these groups there are other legitimate codes. Thus, a single test in the FORTRAN version became a triple test in the PL/I version.

The I/O routines had to be rewritten for the PL/I version both from a file handling point of view and to use the more flexible PL/I statements that were described in the previous chapter.

One feature of PL/I that proved invaluable while debugging MUMPS was the PREPROCESSOR. PUT DATA and PUT LIST statements were inserted at regular intervals throughout the program. They were activated when the \$DEBUG variable was set to the value 1. This facility proved useful in tracing the flow of the program and for printing out values of the different variables at relevant points.

The scope of this thesis was to convert the subset of MUMPS that does not include Global file handling. This has

been accomplished. The Global file handling section is currently underway as another thesis topic.

CHAPTER 5

FUTURE IMPLEMENTATIONS

Future implementations of the PL/I version of the MUMPS interpreter should encompass a multi-user environment. This would allow several users to access a common set of data simultaneously. The VM operating system has several facilities that could be used to accomplish this: RSCS, DIAL, IUCV, VMCF and SMSG[7]. In this chapter four of these methods are summarized and one method, VMCF, will be looked at in detail.

RSCS--Remote Spooling Communications Subsystem

RSCS is an interrupt driven, multitasking system for a virtual machine. It contains line drivers for the communications links and provides facilities for transmitting spool files between virtual machines or between virtual machines and remote-job-entry stations. A virtual machine wishing to send a file to another virtual machine would 'spool' the file to the RSCS virtual machine. The very first entry in the file has to be the 'link-id' (location identifier) of the receiving virtual machine. The first entry is called a tag.

In order that MUMPS may use RSCS for communications a mechanism would have to be provided for remembering which communications came from which virtual machine to ensure that the responses were returned to the correct place. One way to

do this would be to have a special MUMPS command that contained the 'link-id' of the sender. The central MUMPS machine would then use this information to provide the tag for RSCS to direct responses to the correct destinations.

DIAL

DIAL logically connects a terminal to a previously logged on machine. This machine must have an address defined with a matching terminal type. For example if the DIAL command is issued from a 3270 the virtual machine must have previously executed a command of the form

```
DEF GRAF xxx 3270
```

defining a 3270 at address xxx.

It would be necessary to add a telecommunications procedure to MUMPS in order to handle the flow of information across the dialed lines. A disadvantage of this approach is that if the computer is also used for other timesharing systems then the need to dedicate a set of lines exclusively to MUMPS may prove inefficient.

IUCV--Inter-User Communications Vehicle

IUCV is a facility of VM which allows a program executing in one virtual machine to communicate with one or several virtual machines. IUCV is an extremely powerful facility which provides through the IUCV macro, seventeen different functions to control communications. These functions create

and destroy the paths over which communications take place. They send and reply to messages as well as being able to examine pending messages and subsequently receive or reject them.

A virtual machine has to be authorized in the VM directory to use IUCV, before it can establish a path. The receiving virtual machine must also be authorized. A path can then be established by means of the CONNECT and ACCEPT functions. Path descriptions are held in the Communications Control Table(CCT). Each machine has its own CCT.

Communications can begin when paths have been defined. This is done by the source machine issuing a SEND function. The receiving machine can then either immediately RECEIVE the message or it can ask for a description of the message through the DESCRIBE function and then either RECEIVE or REJECT it. The source and receiving machines each have three data areas defined through the IUCV functions: an external interrupt buffer, a receive area and an answer area. Virtual machines notify each other of impending messages by means of an external interrupt. It is therefore the responsibility of each virtual machine to monitor external interrupts, determine which function is requested and then look at the interrupt buffer to see which machine requested the function.

IUCV allows a virtual machine to have as many as 65,535 paths. It also allows communications to be established with

CP system services. Currently the two services supported are console communications and message system.

IUCV would be an ideal vehicle for establishing a MUMPS multi-user environment. MUMPS could be executing on one virtual machine and each user wishing to use MUMPS would log onto a virtual machine. The central machine could then communicate with each user through IUCV either by transferring data to the satellite machines answer area or by sending replies to its console through the IUCV CP support, depending on the type of MUMPS command.

VMCF--Virtual Machine Communications Facility

VMCF is a component of CP which allows data to be transferred from one virtual machine to another. Data can be transferred in blocks of up to 2048 bytes. The only limit on the total amount of data transferred is the size of the receiving virtual machine.

The VMCF component comprises seven control and five data transfer subfunctions. These subfunctions are invoked by Diagnose code X'68' and an associated parameter list ,VMCPARM. There is a unique external interrupt code associated with VMCF, code X'4001'. This interrupt informs a virtual machine of a pending data transfer.

The subfunctions are summarized in the following tables.

Control SubfunctionsFunction Purpose

AUTHORIZE	Make a virtual machine known to VMCF
UNAUTHORIZE	Permanently stop any VMCF activity
IDENTIFY	Let another virtual machine know it can communicate with you via VMCF
CANCEL	Cancel the message just sent to but not accepted by another machine
REJECT	Reject the message the message just arrived at your machine
QUIESCE	Temporarily halt any VMCF activity to your machine
RESUME	Restart after a QUIESCE

Data SubfunctionsFunction Purpose

SEND	Direct a message to another virtual machine
SEND/RECV	Direct a message to another virtual machine and ask for notification of a reply
SENDX	Direct a message to a virtual machine in a faster manner than SEND by eliminating some of the hand shaking protocol
RECEIVE	Notify sending machine you are ready to receive a message
REPLY	Directs a message to the sender

The data subfunctions SEND, SEND/RECV and SENDX implement three different protocols for transmitting data between virtual machines. If VMCF was chosen as the vehicle to implement a multi- user version of MUMPS the most suitable protocol would be SEND/RECV. In a MUMPS environment the receiving machine would be the one running MUMPS and the sending machine would be a machine logged on to use MUMPS. The steps the receiving and sending machines would have to take to exchange messages are outlined below.

Sending machine

a. AUTHORIZE

VMCF enabled.

SPECIFIC is specified so that this machine can only communicate with the receiving machine.

b. Check return code register

c. IDENTIFY

Tells receiving machine . . . I am ready to communicate.

Sends an external interrupt.

d. Checks return code register.

If it is zero issues . . .

e. SEND/RECV

Sends external interrupt.

f. Checks return code register. If it is zero, message was sent ok.

- g. Waits for a reply. When interrupt handler traps an interrupt a reply has been received.

Receiving machine

- a. AUTHORIZE

VMCF enabled.

Any VMCF authorized machine can communicate with this machine.

- b. Check return code register.

- c. Interrupt handler traps the interrupt sent by IDENTIFY and sets the return code.

- d. Interrupt handler traps the interrupt from the SEND/RECV and issues . . .

- e. RECEIVE

Tells sending machine . . . go ahead and send your message.

- f. REPLY

Sends a response. In this case the result of executing a MUMPS command.

- g. Check return code register. If zero the reply was received ok by the sending machine.

These steps are illustrated in Figure 6.

The sending and receiving machines have to provide external interrupt handling routines. External interrupts can be intercepted and subsequently examined by issuing the HNDEXT macro. The sending machine would go into a WAIT after

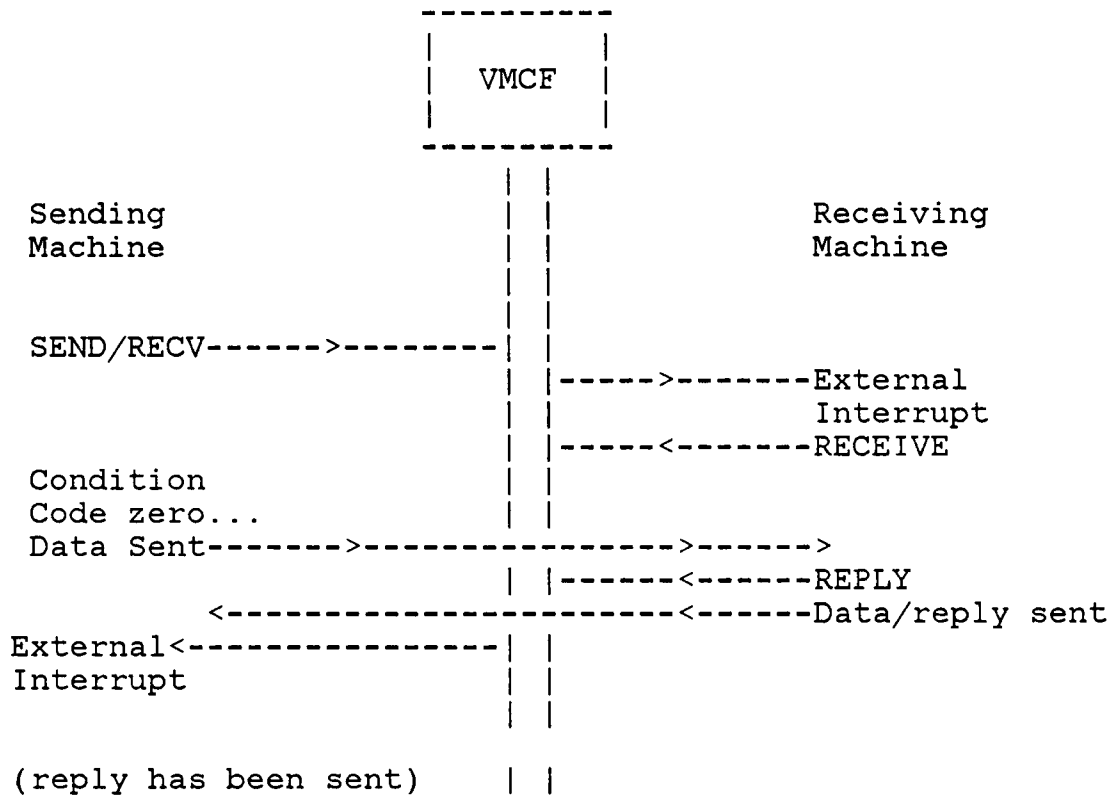


FIGURE 6. SEND/RECV PROTOCOL

it had issued SEND/RECV while it waits for an external interrupt to indicate a REPLY has been sent. An external interrupt might also be received from the console of the sending machine. This could be coped with by telling the user of the machine to reenter the next command after the REPLY has been received. Because the receiving machine must deal with interrupts from several machines the associated interrupt handler would of necessity be more complicated. However, this

is coped with by disabling VMCF after a VMCF interrupt has been received. It is reenabled after a REPLY has been successfully returned to the sending machine.

VMCF would be a suitable vehicle to implement a multi-user MUMPS system under VM.

SMSG--Special Message Facility

The Special Message Facility is a way of sending messages from one virtual machine to another. It is invoked by the CP command SMSG. SMSG can be qualified by VMCF or IUCV which means it can use either of these methods to transmit the message.

Special message bypasses the initialisation a sending machine would normally have to do for VMCF or IUCV. This is done by CP. However the message is limited in length to the size of the console buffer. For a 3270 this is 139 characters. A special message sent via VMCF uses the SENDX protocol.

Special message used on its own would be too limited for MUMPS. It might, however be possible to use it together with VMCF or IUCV.

File Access

Once a method for implementing a multi-user version of MUMPS has been decided upon the problem of controlling file access occurs. The normal way of sharing data in the VM environment is for one virtual machine to own the minidisk

containing the data and for the other virtual machines to link to this minidisk. The linkages are normally established as read only as it is very difficult to ensure the integrity of the data if several machines have write authority. This obviously would be too limiting for a multi-user version of MUMPS as each user would require the ability to enter as well as retrieve data.

There are two ways of solving this problem: either a routine for accessing the data could be written from scratch or an existing software package could be used. If the former way is chosen the minidisks containing the data could be owned by the central MUMPS virtual machine and access controlled by one of the methods just described. If the latter way is chosen an existing package that could be used is the VM/Interactive File Sharing Program (Program Number 5748-XXC)[6].

IFS--Interactive File Sharing Program

The Interactive File Sharing Program(IFS) runs in its own virtual machine. This machine owns the minidisks which contain or will contain the data files. The virtual machines wishing to access or create data files communicate with the central machine via the VMCF component of VM.

The access method used by IFS is VSAM and therefore all data files have to be VSAM files. IFS controls access to a file at the record level. It will allow several users to have write access to the same file but only one user at a time to

have access to a record. This maintains the integrity of the data in a file. The total number of files the IFS machine can control is 255. The total number of files each user can access is 15.

The Interactive File Sharing Facility can be divided into two parts: one part runs in the IFS machine and the other part runs on the application or user virtual machine. The later part is called the application interface and can be called by an application program written in PL/I, Cobol, RPG or assembler. IFS takes the user's request and passes it, via VMCF, to the IFS virtual machine. The part of IFS running in the IFS machine can be subdivided into three parts: the data manager, the request interpreter and the file maintenance system. The data manager is the person responsible for defining the minidisks which are to be made available for shared access. This person is also responsible for initialising these disks to contain VSAM data spaces, remembering that VSAM is the access method used. After this is done the File Maintenance System takes over and is responsible for controlling the creation and maintenance of shared files. The Request Interpreter takes the requests sent from the application interface passes the requests for file maintenance to the file maintenance system and deals with the rest of the requests itself. The Request Interpreter is the part of IFS that controls the access to the files.

This package could very easily provide the file access facility for a multi-user version of MUMPS but it does however have the drawback that it only controls VSAM data sets.

One further note of interest that might facilitate the design of a multi-user MUMPS is that version 3 of VM/SP is due to be released in September 1983. The announcement of this release states that CP support of IUCV has been enhanced and that CMS support for IUCV has been introduced. CMS support will allow a virtual machine to be initialized for IUCV communications and to invoke IUCV functions by a set of new CMS macros. An ECB macro and a CMS abend exit will also be provided. This will make it much simpler to invoke IUCV and makes communications via IUCV a very viable proposition for a multi-user version of MUMPS.

The two methods most suitable for implementing this multi-user version of MUMPS are VMCF and IUCV. VMCF is a well tried method that has been used for some time in packages such as DIRMAINT, the directory management program. IUCV, however is a more recent addition to VM and although not so well proven appears to be very powerful. It will also be much easier to use when version 3 of VM/SP is released.

CHAPTER 6

DISCUSSION

In this chapter FORTRAN '77 is considered as an alternative language for implementing a MUMPS interpreter and a possible 4341 hardware configuration is suggested.

When this thesis was started, there was no FORTRAN '77 compiler available at the University of Tennessee. However, it is worth noting that, if this thesis were started today, the choice of PL/I as the language to implement the MUMPS interpreter might not be so clear cut. Many of the facilities that led to the choice of PL/I as the language to implement a MUMPS interpreter are contained in FORTRAN '77[2]. It supports character variables and provides the character operator concatenation and character functions INDEX, LEN to manipulate strings. FORTRAN '77 also has a block IF facility, IF . . . THEN . . . ELSE IF . . . THEN . . . etc. However the disadvantages of FORTRAN '77 still outweigh its usefulness as a suitable language for implementing a MUMPS interpreter. There are major restrictions to character manipulation: varying length strings are not supported, the maximum length string is 500, null or empty strings are not allowed.

PL/I would still appear to be the better language to implement a MUMPS interpreter. The ability to be able to manipulate strings is crucial to MUMPS and PL/I still excels in this area.

If the features discussed in the previous chapter are implemented at a later date, a multi-user version of MUMPS which runs under VM will be available. This will introduce an interesting alternative for a MUMPS application that previously required a dedicated machine. Under VM, while MUMPS is running on a number of virtual machines, other applications such as text processing can be running on as many additional virtual machines as the hardware can reasonably support. It will also be possible to consider using the IBM 4341 series of machines. Appendix A depicts a possible IBM 4341 hardware configuration and also a (370) 3083 configuration.

This thesis, therefore has taken the first step towards establishing a comprehensive PL/I version of the MUMPS interpreter and it is hoped that future students will undertake the subsequent steps.

REFERENCES

REFERENCES

1. BARNES,R.A., PL/I for Programmers., North Holland 1980.
2. DAVIS,G.B., HOFFMAN,T.R., FORTRAN 77: A Structured Disciplined Style., McGraw-Hill 1983.
3. GREENES,R.A., PAPPALARDO,A.N., MARBLE,C.W., BARNETT,G.O., Design and Implementation of a Clinical Data Management System., Computers and Biomedical Research., Vol.2, pp.469-485.
4. IBM., A Guide to PL/I for Fortran Users.
5. IBM., IBM Journal of Research and Development - 25 Anniversary Issue.
6. IBM., IBM Virtual Machine Facility/370: VM/Interactive File Sharing General Information., GC24-5195
7. IBM., IBM Virtual Machine/System Product: System Programmer's Guide., SC19-6203
8. IBM., OS PL/I Checkout and Optimizing Compilers Language Reference., GC33-0009
9. IBM., OS PL/I Optimizing Compiler Execution Logic., SC33-0025
10. INNOVATIVE COMPUTER SYSTEMS INC., IBM Personal Computer MUMPS Manual.
11. JAREMA,D.R., SUSSENGUTH,E.H., IBM Data Communications: A Quarter Century of Evolution and Progress., IBM Journal of Research and Development., Vol.25, NO.5, Sep 1981.
12. McGEE,W.C., Data Base Technology., IBM Journal of Research and Development., Vol.25, No.5, Sep 1981.
13. MUMPS USERS GROUP., Standard MUMPS Pocket Guide., 1978
14. MUNNECKE,T., A Linguistic Comparison of MUMPS and COBOL., MUMPS Users Group.
15. MUNNECKE,T., MUMPS: Characteristics and Comparisons with Other Systems., Medical Informatics., Vol.2, pp.173-196
16. O'KANE,K.C., UT-MUMPS Users Guide., University of Tennessee.

17. PETH,C.F., Total Commitment in a Community Hospital.,
Proceedings of the 1980 MUMPS Users Group Meeting.
18. SAMMONS,G.L., Decision Support Systems for Corporate
Strategic Planning and Control., MUG Quarterly., Vol.XI,
NO.2/3, 1981.
19. WALTERS,R.F., Is MUMPS the Cure?., Microcomputing., Feb.
1982
20. WALTERS,R.F., BOWIE,J., WILCOX,J.C., MUMPS Primer.,
MUG., 1982.

APPENDICES

APPENDIX A

HARDWARE CONFIGURATIONS

Sample 3083 System with 500 Terminals

Unit	Description	Monthly Rental	Purchase
3083	Processor - 16MB	53700.00	1320000.00
3082	Processor Controller	5735.00	170000.00
3089	Power Unit	1282.00	38000.00
3087	Coolant Distribution Unit	2025.00	60000.00
3278	Display Console	118.00	2505.00
3203	Printer	2015.00	33875.00
1416	Train Cartridge	155.00	2930.00
3800	Printing Subsystem	13000.00	315000.00
	Burster-Trimmed-Stacker	1943.00	50000.00
	Two Channel Switch	371.00	9790.00
3380	DASD AA4	3613.00	98640.00

3380	DASD B04	2620.00	71600.00
3380	DASD AA4	3613.00	98640.00
3380	DASD B04	2620.00	71600.00
3880	Storage Controller	2703.00	66970.00
3880	Storage Controller	2703.00	66970.00
3420	Magnetic Tape	1085.00	24840.00
	6250 BPI	78.00	2000.00
	Total of Four Units	4652.00	107360.00
3803	Tape Controller	1435.00	34430.00
3274	Terminal Controller	946.00	18230.00
3178	Display/Typewriter	-----	1720.00
	Total of 500 Units	-----	860000.00
3287	Printer	314.00	5150.00
	Total of 140 Units	45640.00	765240.00
TOTAL		149209.00	4217540.00

Sample 4321 with 14 Terminals

4321	Processor	4455.00	64000.00
3278	Display Console	118.00	2505.00
3262	Line Printer	595.00	15040.00
3370	DASD A01	1463.00	35480.00
8809	Magnetic Tape	506.00	11960.00
3178	Display/Typewriter	-----	1720.00
	Total of 14 Units	-----	24080.00
TOTAL		7137.00	153065.00

APPENDIX B

PROGRAM LISTING

MUMPS: PROC OPTIONS(MAIN);

```

/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                      */
/*      KEVIN C. O'KANE                            */
/*      DEPARTMENT OF COMPUTER SCIENCE             */
/*      THE UNIVERSITY OF TENNESSEE                */
/*      KNOXVILLE, TENNESSEE 37916               */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                 */
/*      JANUARY 1981                              */
/*      IBM PLI VERSION                            */
/*-----*/

```

```

/*-----*/
/*
/*      PARAMETERS
/*
/*      ARGRET..IS USED WITH "REMOTE BLOCK" TYPE CONSTRUCTIONS.
/*      IT RECEIVES FROM AN ASSIGNED GOTO THE STATEMENT LABEL
/*      ADDRESS OF A RETURN POINT. THE SYSTEM FREQUENTLY USES THE
/*      REMOTE BLOCK TO INCREASE EFFICIENCY.
/*
/*      DTYPE..IS IN THE STRUCTURE PRSECOM. IT INDICATES THE
/*      RETURNED DATA TYPE OF AN EVALUATED ARGUMENT. IT ALSO
/*      INDICATES, HOW TO INTERPRET THE DATA. MOST EVALUATED
/*      RESULTS ARE RETURNED IN STRING XD. IF THE RESULT TYPE IS
/*      REAL, THE FIRST FOUR BYTES OF XD ARE THE FOUR BYTES OF
/*      THE REAL VARIABLE.
/*
/*      RLST..IS A UTILITY REAL VARIABLE
/*
/*      SYMLN..IS THE EXTENT OF THE SYMBOL TABLE. THE SYMBOL
/*      TABLE GROWS DOWN FROM THE TOP OF PD1. SYMLN IS THE VALUE
/*      THE CURRENT LOWEST EXTENT INDEX. SYMADDR IS RETURNED BY
/*      THE SYMBOL TABLE ROUTINE. IT IS THE INDEX(ADDRESS) OF THE
/*      START OF THE AREA IN THE SYMBOL TABLE WHERE A VARIABLE'S
/*      VALUE IS STORED. IT IS SET DURING RETRIEVE REQUESTS
/*
/*      XX..IS USED TO HOLD THE LOCATION OF THE NEXT TAB IN THE
/*      CURRENT COMMAND LINE.
/*
/*      P1PX..IS THE CURRENT POSITION IN THE COMMAND STRING ZD
/*
/*      RETZ..IS A REMOTE BLOCK RETURN ADDRESS
/*

```

```

/*
/*      ERNBR..IS USED TO GENERATE ERROR MESSAGES. IT IS A
/*      PARAMETER TO ERRMOD.
/*
/*      XPX..IS A POINTER INTO THE RESIDUE STRING XD.
/*
/*-----*/

```

```

DECLARE FILES(4) FILE;

```

```

DECLARE AC (2) CHAR DEF AB,
        AB FIXED BIN(15,0);

```

```

DECLARE TEXT(256) CHAR(1), /* I.O VARIABLES */
        TEXTADR POINTER,
        TERMSTR CHAR(100) VARYING,
        TERMARAY(100) CHAR DEF TERMSTR,
        OUTSTRG CHAR(255) VARYING EXTERNAL;

```

```

DECLARE 1 PTRNCOM EXTERNAL,
        2 DUMITEM FLOAT BINARY(53),
        2 BD(256) CHAR(1),
        2 IERR1 FIXED BINARY(31);

```

```

/*-----*/
/*
/*      IOCOMMC..STRUCTURE FOR GLOBAL ARRAYS
/*
/*-----*/

```

```

DECLARE 1 IOCOMMO EXTERNAL,
        2 DUMITEM FLOAT BINARY(53),
        2 ERR FIXED BINARY(31),
        2 G55 FIXED BINARY(31),
        2 G54 FIXED BINARY(31),
        2 D2 CHAR(1),
        2 F CHAR(1);

```

```

AB=0;   IOCOMMO.D2=AC(2); IOCOMMO.F=AC(2); /*INIT*/

```

```

DECLARE 1 FILESCO EXTERNAL,
        2 DUMITEM FLOAT BINARY(53),
        2 U8 CHAR(1),
        2 U9 CHAR(1),
        2 CLRGBL CHAR(1),
        2 TERM CHAR(1),
        2 G5RECS FIXED BINARY(31) INITIAL(473),
        2 G2RECS FIXED BINARY(31) INITIAL(499);

```

```

AB=0;
        FILESCO.U8=AC(2);
        FILESCO.U9=AC(2);
        FILESCO.CLRGBL=AC(2);

```

```

        FILESCO.TERM=AC(2);

        DECLARE 1 INOUTCC EXTERNAL,
                2 DUMITEM FLOAT BINARY(53),
                2 MTCUR FIXED BINARY(31);

/*-----*/
/*
/*      CODESCO..STRUCTURE FOR CODES FOR COMMA, PARENS ETC
/*      CODE 251 - OPEN PAREN
/*      CODE 252 - CLOSE PAREN
/*      CODE 253 - COMMA
/*      CODE 254 - QUOTE
/*
/*-----*/
        DECLARE 1 CODESCO EXTERNAL,
                2 DUMITEM FLOAT BINARY(53),
                2 COD251(2) CHAR(1),
                2 COD252(2) CHAR(1),
                2 COD253(2) CHAR(1),
                2 COD254(2) CHAR(1),
                2 COD255( 2) CHAR(1),
                2 XTAB( 2) CHAR(1),
                2 RNDX( 2) FIXED BINARY(31);

        AB=251;
                CODESCO.COD251(1)=AC(2);
                CODESCO.COD251(2)=AC(1); /*0*/
        AB=252;
                CODESCO.COD252(1)=AC(2);
                CODESCO.COD252(2)=AC(1); /*0*/
        AB=253;
                CODESCO.COD253(1)=AC(2);
                CODESCO.COD253(2)=AC(1); /*0*/
        AB=254;
                CODESCO.COD254(1)=AC(2);
                CODESCO.COD254(2)=AC(1);
        AB=255;
                CODESCO.COD255(1)=AC(2);
                CODESCO.COD255(2)=AC(1);
        AB=64;
                CODESCO.XTAB(1)=AC(2);
                CODESCO.XTAB(2)=AC(1);

        OPEN FILE(SYSPRINT);
        TEXTADR=ADDR(TEXT);

/*-----*/
/*
/*      PRSECOM..STRUCTURE FOR PARAMETERS TO SUBROUTINE PARSE
/*
/*-----*/

```

```

DECLARE 1 PRSECOM EXTERNAL,
        2 DUMITEM FLOAT BINARY(53),
        2 IORG FIXED BINARY(31),
        2 XPX FIXED BINARY(31),
        2 XPY FIXED BINARY(31),
        2 TOPX FIXED BINARY(31),
        2 T2 CHAR(1),
        2 V1D(257) CHAR(1),
        2 XD( 256) CHAR(1),
        2 XDSAVE( 256) CHAR(1),
        2 IERR FIXED BINARY(31),
        2 DTYPE CHAR(1),
        2 DUMITEM2( 1)FLOAT BINARY;

/*-----*/
/*
/*      DO COMMANDS
/*
/*      DO COMMANDS USE THE GOSUB STACK. IORG IS USED TO INDICATE
/*      THE ORIGIN IN PD1 WHERE THE NEW PROGRAM IS TO BE LOADED
/*      TERMINATION CAUSES THE RESTORATION OF THE ORIGINAL VALUE
/*      OF PD1LEN. DOSAVE IS USED FOR HOLDING ARGUMENTS
/*      TEMPORARILY.
/*-----*/

DCL TMP1( 2) CHAR(1),
    TMP2( 256) CHAR(1),
    TMP3( 262) CHAR(1);
AB=12;
    TMP1(1)=AC(2);
    TMP1(2)=AC(1);

DCL ZPDTMP (256) CHAR(1) STATIC; /*USED BY ZPAD*/
/*-----*/
/*
/*      UNLCM..STRUCTURE FOR PASSING INFORMATION BETWEEN PROCEDURES
/*-----*/

/*-----*/
/*
/*      GOSUB STACK
/*      1 - FORMER PD1CUR
/*      2 - FORMER P1PX
/*      3 - (THIS WAS USED FOR LABEL IN FORTRAN VERSION)
/*      4 - OFFSET I ZD OF COMMAND RESIDUE
/*      5 - ORIGINAL START OF XECUTE COMMAND
/*      6 - ORIGINAL PD1LEN
/*
/*      THE GOSUB STACK MANAGES CERTAIN STACKING FUNCTIONS
/*      FOR THE FOR, XECUTE AND DO COMMANDS. IT IS USED TO
/*      RECALL THE ENVIRONMENT THAT EXISTED AT THE TIME OF THE
/*

```

```

/*      COMMAND SO THAT SUBSEQUENT ARGUMENTS OF THE COMMAND AND      */
/*      COMMANDS ON THE SAME ORIGINAL LINE CAN BE PROCESSED. THE      */
/*      VALUES OF PD1CUR, P1PX AND PD1LEN ARE THOSE WHICH EXISTED    */
/*      WHEN THE COMMAND WAS INITIATED. THE ZD OFFSET RESIDUE IS     */
/*      THE PLACE ON THE LINE WHERE THE COMMAND ARGUMENT BEING       */
/*      EXECUTED ENDED. THE RETURN POINT IS USED IN CONJUNCTION      */
/*      WITH AN ASSIGNED GOTO. THE VALUE STORED IS THE STATEMENT     */
/*      LABEL IN THIS PROGRAM WHERE PROCESSING WILL CONTINUE.       */
/*      THE QUIT COMMAND SECTION OF THIS PROGRAM IS THE PART        */
/*      RESPONSIBLE FOR RE-LOADING AN ENVIRONMENT AND RETURNING      */
/*      TO THE SECTION OF THE CODE WHICH CONTINUES PROCESSING.      */
/*      THE NUMBER OF PERMITTED NESTINGS CAN BE ENLARGED BY         */
/*      ALTERING THE VALUE OF THE FIRST INDEX. GSBX GIVES THE       */
/*      CURRENT STACK TOP (0 IS EMPTY, >0 INDICATES AN EXTENT.)     */
/*      GSBX -- IS THE EXTENT OF THE STACK                          */
/*-----*/

```

```

DECLARE 1 UNLCM EXTERNAL,
        2 DUMITEM FLOAT BINARY(53),
        2 SYMLEN FIXED BINARY(31) INITIAL(4000),
        2 SYMADR FIXED BINARY(31),
        2 SYMFLG FIXED BINARY(15,0),
        2 GD(257) CHAR(1),
        2 G(15) FIXED BINARY(31),
        2 P FLOAT BINARY,
        2 PD(257) CHAR(1),
        2 VD(256) CHAR(1),
        2 DOSAVE(21) CHAR(1),
        2 XX8(5) CHAR(1),
        2 DATRIB CHAR(1),
        2 DVAR FIXED BINARY(31),
        2 XX9(6) CHAR(1),
        2 PD1LEN FIXED BINARY(31) INITIAL(0),
        2 PD1CUR FIXED BINARY(31) INITIAL(1),
        2 TRACE FIXED BINARY(31),
        2 ZD(256) CHAR(1),
        2 PD1(4000) CHAR(1) INITIAL((4000)' '),
        2 FORV(5,10) CHAR(1),
        2 FORIT(5) FIXED BINARY(1),
        2 FORLT(5) FIXED BINARY(1),
        2 FTMP(256) CHAR(1),
        2 FORIN(5)FLOAT BINARY,
        2 FORLV(5) FLOAT BINARY,
        2 FORNXT(5) FIXED BINARY(31),
        2 FORX FIXED BINARY(31),
        2 GOSUB(6,5)FIXED BINARY(31),
        2 GSBX FIXED BINARY(31);

```

```

/*-----*/
/*
/*      FOR LOOP CONTROL STRUCTURES.
/*
/*-----*/

```

```

/*      FOR LOOP COMMANDS INDUCE STACKING. THE ENVIRONMENT      */
/*      PRESENT WHEN A FOR COMMAND WAS EXECUTED IS SAVED IN THE */
/*      GOSUB STACK. DATA CONCERNING THE FOR ARGUMENTS ARE SAVED */
/*      HERE. THE NAME OF THE FOR VARIABLE IS SAVED IN FORV; A   */
/*      FLAG WHICH INDICATES IF THERE IS AN INCREMENT AND A LIMIT */
/*      TEST IS IN FORIT AND FORLT. FORIN CONTAINS THE AMOUNT OF  */
/*      OF THE INCREMENT IF INCREMENTS ARE USED. FORLV CONTAINS  */
/*      THE UPPER LIMIT IF ONE IS PRESENT. FORNXT POINTS TO THE ZD */
/*      OFFSET OF THE NEXT FOR ARGUMENT ...IF ANY. FORX IS THE    */
/*      STACK EXTENT. THE NUMBER OF PERMITTED FOR'S CONCURRENTLY */
/*      STACKED CAN BE INCREASED BY ALTERING THE DIMENSIONS. FTMP  */
/*      IS USED NOT ONLY FOR FOR'S BUT IN OTHER COMMANDS ALSO.   */
/*      ----- */

```

```

DECLARE GSUBL LABEL,
        GOSUBLAB(6,5) LABEL;

```

```

DECLARE GBLREF ENTRY,
        GLOBAL ENTRY,
        SQUASH ENTRY,
        GETSTR ENTRY,
        ERRNBR FIXED BINARY(31) STATIC,
        LOOKUP ENTRY RETURNS(FIXED BINARY(31) ),
        ISPOOL FIXED BINARY(31) STATIC,
        PD1OLD FIXED BINARY(31,0),
        OPNFLG( 5) CHAR(1) STATIC;
PD1OLD=0;

```

```

AB=0;

```

```

        OPNFLG(1)=AC(1);
        OPNFLG(2)=AC(1);
        OPNFLG(3)=AC(1);
        OPNFLG(4)=AC(1);
        OPNFLG(5)=AC(1);

```

```

DECLARE DIRFLG FIXED BINARY(1) STATIC,
        MTOLD FIXED BINARY(31) STATIC,
        PLSVE FIXED BINARY(31) STATIC,
        ITYPE CHAR(1),
        ITYPE2 CHAR(1);

```

```

DECLARE XWAIT( 4)FLOAT BINARY STATIC,
        XTIME FLOAT BINARY STATIC,
        RNDXX FLOAT BINARY,
        ERNBR FIXED BINARY(15) STATIC,
        PRNTSTR CHAR(100) VARYING,
        LIMIT FIXED BINARY(31),
        ZDLEN FIXED BINARY(31);

```

```

DECLARE ICTRL FIXED BINARY(31),
        P2PX FIXED BINARY(31) STATIC,
        IBLK FIXED BINARY(31) STATIC,
        ICH8 FIXED BINARY(31) STATIC,
        ICH9 FIXED BINARY(31) STATIC,

```

```

    POPX FIXED BINARY(31) STATIC,
    LAST(9) CHAR(1) EXTERNAL,
    CSTR(16) CHAR(1) STATIC
    INITIAL('S','G','W','I','O','Z','R','Q','X','F','H','E'
    , 'D','K','C','U'),
    XRDR(2) CHAR(1) STATIC,
    RSLT FLOAT BINARY,
    XDEQ FLOAT BIN DEF XD;

```

```

DECLARE RETZ LABEL,
    P1PX FIXED BINARY(31),
    EDIT FIXED BINARY(31),
    TMPX(4) FIXED BINARY,
    XPXH FIXED BINARY(31),
    JPX FIXED BINARY(31) STATIC,
    IBR FIXED BINARY(31) STATIC;

```

```

DECLARE REQ(4) CHAR(1) DEF RSLT,
    VER(5) FIXED BINARY(31),
    HOR(5) FIXED BINARY(31) STATIC,
    TPX FIXED BINARY(31),
    MTI( 4) FIXED BINARY(31) STATIC INITIAL(0,0,0,0),
    ARGRET LABEL,
    LRETZ LABEL,
    LPOSTC LABEL,
    BRANCH01(16) LABEL INITIAL(
    L570,L500,L440,L360,L340,L350,
    L260,L4000,L8000,L7000,
    L5100,L5200,
    L5300,L5400,L5500,L5600);

```

```

DECLARE IO FIXED BINARY(31),
    XX FIXED BINARY(31) STATIC,
    MT(4) FIXED BINARY(31),
    M FIXED BINARY(31) STATIC,
    X FIXED BINARY(31) STATIC,
    K FIXED BINARY(31) STATIC,
    L FIXED BINARY(31) STATIC,
    I FIXED BINARY(31) STATIC,
    J FIXED BINARY(31) STATIC;

```

```

DECLARE DATA FILE RECORD EXTERNAL,
    OUT FILE RECORD EXTERNAL,
    DUMM001 CHARACTER(1) INITIAL('+');

```

```

ON ATTENTION BEGIN;
PUT SKIP LIST(' BREAK KEY HIT. PROGRAM TERMINATED');
PUT SKIP;
GOTO L9999;
END;

```

```

XTIME=0;

```

```

OUTSTRG='';
L2:  CALL GBLOPN;
     AB=1; CLRGBL=AC(2); /*SUBSEQUENT CALLS WILL CLEAR FILES */
     CALL SCOPY(ZPAD(XTAB(1)||'W " UT PL/1 MUMPS"'),ADDR(PD1));
     CALL SCOPY(ADDR(PD1(2)),ADDR(ZD));
     ZDLEN=16;
     PD1OLD=0;
     PD1CUR=1;
     PD1LEN=20;
     IO=5;
     AB=0; PD1(PD1LEN+1)=AC(2); /*      NO XD EXISTS      */
     AB=0; XD(1)=AC(2);
     XPXH=0;
     AB=0; TMP1(2)=AC(2);
     MTCUR=1; /* CURRENT TERMINAL*/
/*-----*/
/*
/*      ERROR RESET ENTRY
/*      THIS CODE IS EXECUTED WHEN THE USER MUMPS PROGRAM HAS
/*      CRASHED. IT IS BRANCHED TO FROM THE VERY END OF THIS
/*      ROUTINE. IT RESTORES THE USER ENVIRONMENT.
/*
/*-----*/
L1:  IERR=0; /* RESET ERROR NUMBER AND INDICATOR FLAG*/
     IF (ERNBR=31) THEN SYMLLEN=4000;
     AB=0; XD(1)=AC(2);
     GSBX=0; /* ZERO GOSUB STACK*/
     FORX=0; /* ZERO FOR STACK*/
     XWAIT(MTCUR)=0.0;
     IO=5; /* SET $IO TO UNIT 5*/
     ICTRL=0; /* CONTROL-C FLAG SET TO NO CTRL-C RECEIVED*/
     RNDXX=0; /* RESET RANDOM NUMBER SEED TO NEW VALUE*/
     TOPX=0; /* SET LHS FLAG TO FALSE*/
     AB=0; T2=AC(2); /* MULTI ARGUMENT SWITCH*/
/*-----*/
/*
/*      RESTORATION POINT - RESTORE BASE-LINE PROGRAM
/*      THIS IS ENTERED FROM THE QUIT AND HALT COMMANDS WHEN
/*      PROGRAMS TERMINATE. IT RESTORES THE UNDERLYING CONTROL
/*      PROGRAM IN LOW USER MEMORY.
/*      THE UNDERLYING PROGRAM USES LOWER CASE VARIABLE X TO
/*      ELIMINATE CONFLICT WITH USER VARIABLES. USERS MAY NOT USE
/*      LOWER CASE VARIABLES.
/*
/*-----*/
L210: IF MTI(MTCUR)=0 THEN GOTO L205; /* INITIALIZED?*/
      CALL SCOPY(ZPAD(XTAB(1)||'W !,">:" R X X X G +0'),ADDR(PD1));
      IF PD1LEN<24 THEN PD1LEN=24; /* RESET IF NO PGM LOADED*/
L205: PD1CUR=1; /* STARTING POINT*/
      AB=0; TMP1(2)=AC(2);
      MTI(MTCUR)=1; /*INITIALIZED*/
/*-----*/

```

```

/*
/*      SET HORIZONTAL AND VERTICAL POINTERS
/*
/*      UNIT 5 IS TT:
/*
/*      SET GOSUB AND FOR STACKS EMPTY
/*
/*-----*/
L206:  DO I=1 TO 5;
      HOR(I)=0;
L209:  VER(I)=0;
      END;
      FORX=0;
      GSBX=0;
/*-----*/
/*
/*      ENTRY POINT FOR NEXT COMMAND LINE
/*
/*      PD1CUR GIVES THE INDEX IN PD1 OF THE CURRENT LINE START.
/*
/*      WE MOVE TO THE CHARACTER FOLLOWING THE TAB (IGNORE LEL IF
/*
/*      ANY). IF NO TAB IS PRESENT, THEN THE PROGRAM HAS ENDED.
/*
/*      LOCATE TAB CHARACTER
/*
/*      COPY RESIDUE OF LINE AFTER TAB INTO ZD ARRAY
/*
/*-----*/
L211:  XX=INDEX(ADDR(PD1(PD1CUR)),ADDR(XTAB),1);
      IF XX > 0 & PD1CUR+XX < PD1LEN THEN GOTO L212;
      /* NOT END OF PROGRAM*/
      AB=0; ZD(1)=AC(2);
      ZDLEN=0;
      GOTO L4000; /* INTERNAL QUIT*/
L212:  CALL SUBSTR(ADDR(PD1(PD1CUR)),ADDR(ZD),XX+1,-1);
      ZDLEN=LEN(ADDR(ZD));
      PD1OLD=PD1CUR; /* REMEMBER LINE START*/
      PD1CUR=PD1CUR+LEN(ADDR(PD1(PD1CUR)))+1; /* UPDATE PD1CUR*/
      P1PX=0; /* POINTER INTO ZD*/
/*-----*/
/*
/*      CONTINUE LINE ENTRY POINT
/*
/*-----*/
L240:  IF MTI(MTCUR) = 0 THEN GOTO L1; /* INITIALIZE*/
      P2PX=P1PX;
L241:  AB=0; T2=AC(2); /* NO MULTI ARG*/
      P1PX=P1PX+1;
      IF P1PX <= ZDLEN THEN GOTO L250; /* CONTINUE LINE*/
      IF FORX > 0 & GOSUB(1,GSBX) < 0 THEN GOTO L7015;
      /* LOOP TERMINATION*/
      ELSE GOTO L211; /* NEXT LINE*/
/*-----*/
/*
/*      DECODE COMMAND WORD
/*
/*-----*/
L250:  TMP1(1)=ZD(P1PX);

```

```

        IF TMP1(1) = ' ' THEN GOTO L211; /* COMMENT*/
/*-----*/
/*
/*      SCAN PAST LONG SPELLING - NO SPELLING CHECKS
/*
/*-----*/
L252:  AB=0;
        IF ZD(P1PX+1)=' ' | ZD(P1PX+1)=':' | ZD(P1PX+1)=AC(2)
          THEN GOTO L251;
        P1PX=P1PX+1;
        GOTO L252;
/*-----*/
/*
/*      LOOK-UP FIRST LETTER OF COMMAND WORD
/*
/*-----*/
L251:  DO I=1 TO 16;
        IF TMP1(1)=CSTR(I) THEN GOTO BRANCH01(I);
        END;
        GOTO L357;
/*-----*/
/*
/*      BAD COMMAND WORD
/*
/*-----*/
L255:  GOTO L9113;
/*-----*/
/*
/*      BAD ARGUMENT LIST
/*
/*-----*/
L256:  GOTO L9114;
/*-----*/
/*
/*      READ AND SWAP VECTOR MANAGEMENT
/*
/*-----*/
L260:  DIRFLG= '0'B; /* DIRECT MODE FLAG*/
        /* WHEN SET, SINGLE CHARACTER INPUT ENABLED*/
        ARGRET=L261;
        GOTO L5000; /* ARGSCAN*/
L261:  XPX=0;
        IF IO>4 THEN GOTO L300;
        IF OPNFLG(IO)=3|OPNFLG(IO)=1 THEN GOTO L300;
        ELSE GOTO L9126;
L300:  IF IO<5 THEN OPNFLG(IO)=1;
/*-----*/
/*
/*      NEW LINE
/*
/*-----*/
        IF XD(1)~='!' THEN GOTO L262;

```

```

        TEXT(1)='';
        CALL INOUT(TEXTADR,2,0);
        XPX=1;
        GOTO L263;

/*-----*/
/*
/*      NEW PAGE FORMAT
/*
/*-----*/
L262:  IF XD(1) -=' #' THEN GOTO L264;
        TEXT(1)='';
        CALL INOUT(TEXTADR,1,0);
        XPX=1;
        GOTO L263;

/*-----*/
/*
/*      TAB FORMAT
/*
/*-----*/
L264:  IF XD(1) -='?' THEN GOTO L280;
        CALL SCOPY(ADDR(XD(2)),ADDR(XD));
        XPX=0;
        AB=1; T2=AC(2); /* MULTI-ARG SWITCH*/
        CALL PARSE;
        IF IERR -=' 0' THEN GOTO L9001;
        IF DTYPE -=' 1' THEN GOTO L265;
        RSLT=XDEQ;
        GOTO L266;
L265:  CALL CVB(ADDR(XD),ADDR(RSLT));
L266:  I=RSLT;
        TEXT(1)='';
        CALL INOUT(TEXTADR,3,I);
        CALL SCOPY(ADDR(XDSAVE(XPY)),ADDR(FTMP));
        XPX=0;
        GOTO L311;

/*-----*/
/*
/*      TEXT TO BE PRINTED
/*
/*-----*/
L280:  IF XD(1) -='"' THEN GOTO L302;
        XPX=1;
L303:  XPX=XPX+1;
        IF XPX>LEN(ADDR(XD)) THEN GOTO L256; /* ERROR*/
        IF XD(XPX) -='"' THEN GOTO L303;
        IF XD(XPX+1) -='"' THEN GOTO L281;
        CALL SCOPY(ADDR(XD(XPX+1)),ADDR(XD(XPX)));
        GOTO L303;
L281:  AB=0; XD(XPX)=AC(2); /* TERMINATOR*/
        CALL INOUT(ADDR(XD(2)),BINARY(4),BINARY(0));
        XD(XPX)='"; /* REPLACE QUOTE*/
L263:  CALL SCOPY(ADDR(XD),ADDR(FTMP));

```

```

      GOTO L311;
/*-----*/
/*                                          */
/*      DIRECT CHARACTER READ MODE          */
/*                                          */
/*-----*/
L302: IF XD(1)~='*' THEN GOTO L320;
      CALL SCOPY(ADDR(XD(2)),ADDR(XD));
      DIRFLG='1'B;
/*-----*/
/*                                          */
/*      VARIABLE NAME IN ARGUMENT LIST      */
/*                                          */
/*-----*/
L320: TOPX=1; /* LHS SWITCH ON*/
      CALL PARSE;
      IF IERR~=0 THEN GOTO L9001;
      CALL SCOPY(ADDR(V1D),ADDR(VD));
/*-----*/
/*                                          */
/*      CHECK FOR INDIRECTION              */
/*                                          */
/*-----*/
L6600: IF VD(1)~='@' THEN GOTO L301;
      ARGRET=L6700;
      GOTO L670; /* PROCESS LHS INDIRECTION*/
L6700: GOTO L261; /* RETURN POINT */
/*-----*/
/*                                          */
/*      VARIABLE NAME FOUND INTO WHICH VALUE WILL BE ASSIGNED */
/*                                          */
/*-----*/
L301: IF XD(XPX+1)~=':' THEN GOTO L7201;
/*-----*/
/*                                          */
/*      TIMEOUT                            */
/*                                          */
/*-----*/
      AB=1; T2=AC(2);
      XPX=XPX+1;
      CALL PARSE;
      IF IERR~=0 THEN GOTO L9001;
      IF DTYPE='1' THEN GOTO L7202;
      CALL CVB(ADDR(XD),ADDR(RSLT));
      GOTO L7203;
L7202: RSLT=XDEQ;
L7203: CALL SCOPY(ADDR(XDSAVE(XPY)),ADDR(FTMP));
      XPX=0;
      IF RSLT>0 THEN GOTO L7204;
      TPX=0;
      GOTO L311;
L7204: AB=0; XD(1)=AC(2);

```

```

        TPX=0;
        XWAIT(MTCUR)=RSLT+0; /* UNTIL*/
        GOTO L324;
/*-----*/
/*                                          */
/*      HOLD RESIDUE OF LINE                      */
/*                                          */
/*-----*/
L7201: CALL SCOPY(ADDR(XD),ADDR(FTMP));
/*-----*/
/*                                          */
/*      DIRECT MODE - TEMPORARILY SET CHAR 12 OF JSW      */
/*                                          */
/*-----*/
        IF ~DIRFLG THEN GOTO L321;
        IF IO=5 THEN GOTO L325;
        ON ENDFILE(FILES(IO)) GOTO L327;
        GET FILE(FILES(IO)) EDIT (XD(1)) (A(1));
        AB=0; XD(2)=AC(2);
        GOTO L324;
L327:  AB=4; XD(1)=AC(2);
        AB=0; XD(2)=AC(2);
        GOTO L324;
L325:  AB=0; XD(1)=AC(2);
        GOTO L324;
/*-----*/
/*                                          */
/*      TERMINAL INPUT                                */
/*                                          */
/*-----*/
L321:  HOR(IO)=0;
        VER(IO)=VER(IO)+1;
        I=IO;
        AB=0; XD(1)=AC(2);
        IF I=5 THEN GOTO L330;
        CALL GETSTR(I,XD,BINARY(132),IERR,TPX);
        IF IERR THEN GOTO L256;
        ELSE GOTO L324;
L330:  GET EDIT(TERMSTR) (A(80));
        CALL SCOPY(ZPAD(TERMSTR),ADDR(XD));
        CALL TRIM(ADDR(XD));
/*-----*/
/*                                          */
/*      BRANCH TO INTERNAL SUBROUTINE TO ASSIGN VALUE TO      */
/*      VARIABLE - THIS SUBROUTINE IS ALSO USED BY "SET"      */
/*                                          */
/*-----*/
L324:  TPX=1; /* TIMEOUT*/
        ITYPE='2';
        RETZ=L311;
        GOTO L800;
/*-----*/

```

```

/*                                                    */
/*      CHECK FOR MORE ARGUMENTS                      */
/*                                                    */
/*-----*/
L311:  XPX=XPX+1;
      XWAIT(MTCUR)=0.0;
      IF XPX > LEN(ADDR(FTMP)) THEN GOTO L240;
      IF FTMP(XPX) ^= ',' THEN GOTO L256; /* ERROR*/
      DIRFLG='0'B;
      XPX=XPX+1;
      IF XPX > LEN(ADDR(FTMP)) THEN GOTO L256; /* ERROR*/
      CALL SUBSTR(ADDR(FTMP),ADDR(XD),XPX,-1);
      GOTO L261;

/*-----*/
/*                                                    */
/*      OPEN                                           */
/*                                                    */
/*-----*/
L340:  ARGRET=L341;
      GOTO L5000; /* ARGSCAN*/
L341:  IF LEN(ADDR(XD)) = 0 THEN GOTO L256;
/*-----*/
/*                                                    */
/*      EVALUATE UNIT NUMBER                          */
/*                                                    */
/*-----*/
L342:  XPX=0;
      AB=1; T2=AC(2);
      CALL PARSE;
      IF IERR ^= 0 THEN GOTO L9001;
L343:  IF DTYPE ^= '1' THEN GOTO L353;
      RSLT=XDEQ;
      GOTO L345;
L353:  CALL CVB(ADDR(XD),ADDR(RSLT));
/*-----*/
/*                                                    */
/*      DEVICES                                         */
/*                                                    */
/*-----*/
L345:  IF RSLT > 0 & RSLT < 5 THEN GOTO L347;
      GOTO L9119;
L347:  K=RSLT;
      IF OPNFLG(K) = 0 THEN GOTO L344;
      ELSE GOTO L9126;

/*-----*/
/*                                                    */
/*      GET FILE NAME                                  */
/*                                                    */
/*-----*/
L344:  IF XDSAVE(XPY) = ':' THEN GOTO L348;
      ELSE GOTO L9121;
L348:  CALL SCOPY(ADDR(XDSAVE(XPY+1)),ADDR(XD));

```

```

      XPX=0;
      T2= '1'B;
      CALL PARSE;
      IF IERR ~= 0 THEN GOTO L9001;
      IF DTYPE ~= '2' THEN GOTO L256;
      I=INDEX(ADDR(XD),ZPAD('/',1));
      IF I = 0 THEN GOTO L256;
      CALL SUBSTR(ADDR(XD),ADDR(TMP2),1,I-1);
      CALL SUBSTR(ADDR(XD),ADDR(TMP3),I+1,-1);
/*      OPEN FILE (FILES(I)) TITLE(TMP2); */
      OPNFLG(K)=1; /* INPUT*/
      GOTO L346;

/*-----*/
/*
/*      NEXT ARGUMENT
/*
/*-----*/
L346:  IF XPY > LEN(ADDR(XDSAVE)) THEN GOTO L240;
      IF XDSAVE(XPY) ~= ',' THEN GOTO L256;
      CALL SUBSTR(ADDR(XDSAVE(XPY+1)),ADDR(XD),1,-1);
      GOTO L342;
L355:  GOTO L9126;
/*-----*/
/*
/*      Z-INIT
/*
/*-----*/
L350:  IF ZD(P1PX) ~= 'G' | P2PX+2 ~= P1PX THEN GOTO L351;
      IF MTCUR ~= 1 THEN GOTO L352; /* ONLY CONSOLE ALLOWED*/
      CALL GBLOPN;
      GOTO L352;

/*-----*/
/*
/*      Z-EXIT
/*
/*-----*/
L351:  IF ZD(P1PX) ~= 'E' | P2PX+2 ~= P1PX THEN GOTO L356;
      IF MTCUR ~= 1 THEN GOTO L352;
      STOP;
L352:  P1PX=P1PX+2;
      GOTO L240;

/*-----*/
/*
/*      Z-SQUASH - COMPRESS GLOBAL FILES
/*
/*-----*/
L356:  IF ZD(P1PX) ~= ':' | P2PX+2 ~= P1PX THEN GOTO L357;
      IF MTCUR ~= 1 THEN GOTO L352;
      CALL SQUASH;
      GOTO L352;

/*-----*/
/*

```

```

/*      INTERNAL EDITOR CALLS                                */
/*      EDITOR CALLS CAN BE USED IN PROGRAM TEXT BUT THEY MUST */
/*      BE THE LAST THING ON THE LINE. MANY EDITOR CALLS      */
/*      DESTROY THE GOSUB AND FOR STACKS AND CAUSE EXECUTION  */
/*      TO REVERT TO PD1CUR=1                                  */
/*      */
/*-----*/
L357: CALL INPUT ;
      IF ERR ^= 0 THEN GOTO L255;
      GOTO L211; /* NOT STRICTLY PROGRAMMATIC*/
/*-----*/
/*      IF                                                    */
/*-----*/
L360: IF ZD(P1PX+1) ^= ' ' THEN GOTO L256; /* ERROR*/
      IF P1PX+2 > ZDLEN THEN GOTO L256;   /* ERROR*/
      IF ZD(P1PX+2) = ' ' THEN GOTO L391; /* NO ARGUMENT FORM*/
      ARGRET=L401;
      GOTO L5000; /* ARGSCAN*/
/*-----*/
/*      NO ARGUMENT FORM - CHECK $TEST (TPX)                */
/*-----*/
L391: P1PX=P1PX+2;
      IF TPX ^= 0 THEN GOTO L240; /* FALSE*/
      P1PX=ZDLEN;                /* FORCE END OF LINE*/
      GOTO L240;                 /* TRUE*/
/*-----*/
/*      EVALUATE ARGUMENT                                    */
/*-----*/
L401: XPX=0;
      AB=0; T2=AC(2);
      CALL PARSE;
      IF IERR ^= 0 THEN GOTO L9001;
L410: IF DTYPE ^= '1' THEN GOTO L411;
      RSLT=XDEQ;
      GOTO L412;
L411: CALL CVB(ADDR(XD),ADDR(RSLT));
L412: TPX=RSLT;
      IF TPX ^= 0 THEN GOTO L413;
      P1PX=ZDLEN;
      GOTO L240;
L413: IF XPY > LEN(ADDR(XDSAVE)) THEN GOTO L240; /* FINISH*/
      IF XDSAVE(XPY) ^= ',' THEN GOTO L256;      /* ERROR*/
      CALL SUBSTR(ADDR(XDSAVE(XPY+1)),ADDR(XD),1,-1);
      GOTO L401; /* NEXT ARGUMENT*/
/*-----*/
/*

```

```

/*      WRITE                                                    */
/*                                                    */
/*-----*/
L440:  ARGRET=L471;
      GOTO L5000; /* ARGSCAN*/
/*-----*/
/*                                                    */
/*      CHECK FOR FORMAT ARGUMENT (?,#,
/*                                                    */
/*-----*/
L471:  IF IO > 4 THEN GOTO L439;
      IF OPNFLG(IO) = 3 | OPNFLG(IO) = 2 THEN GOTO L439;
      ELSE GOTO L9126;
L439:  IF IO < 5 THEN OPNFLG(IO)=2;
      IF XD(1) ^= '!' & XD(1) ^= '#' & XD(1) ^= '?'
      & XD(1) ^= '*' THEN GOTO L472;
/*-----*/
/*                                                    */
/*      NEW LINE                                                    */
/*                                                    */
/*-----*/
      IF XD(1) ^= '!' THEN GOTO L473;
      TEXT(1)='';
      CALL INOUT(TEXTADR,2,0);
L477:  CALL SCOPY(ADDR(XD(2)),ADDR(XDSAVE));
      XPY=1;
      GOTO L491;
/*-----*/
/*                                                    */
/*      NEW PAGE                                                    */
/*                                                    */
/*-----*/
L473:  IF XD(1) ^= '#' THEN GOTO L475;
      TEXT(1)='';
      CALL INOUT(TEXTADR,1,0);
      GOTO L477;
/*-----*/
/*                                                    */
/*      TYPE OUTPUT                                                    */
/*                                                    */
/*-----*/
L475:  IF XD(1) ^= '*' THEN GOTO L481;
      CALL SCOPY(ADDR(XD(2)),ADDR(XD) );
      XPX=0;
      T2= '1'B;
      CALL PARSE;
      IF IERR ^= 0 THEN
      GOTO L9001;
      IF DTYPE ^= '1' THEN GOTO L482;
      RSLT=XDEQ;
      XD(1)=REQ(1); /* REQ IS DEFINED ON RSLT */
      XD(2)=REQ(2);

```

```

XD(3)=REQ(3);
XD(4)=REQ(4);
XD(5)=AC(1);
GOTO L483;
L482: CALL CVB(ADDR(XD),ADDR(RSLT));
XD(1)=REQ(1); /* REQ IS DEFINED ON RSLT */
XD(2)=REQ(2);
XD(3)=REQ(3);
XD(4)=REQ(4);
XD(5)=AC(1);
L483: CALL INOUT(ADDR(XD),BINARY(4),BINARY(0));
GOTO L491;

/*-----*/
/*                                          */
/*      TAB                                          */
/*                                          */
/*-----*/
L481: CALL SCOPY(ADDR(XD(2)),ADDR(XD) );
XPX=0;
T2= '1'B;
CALL PARSE;
IF IERR ^= 0 THEN GOTO L9001;
L478: IF DTYPE ^= '1' THEN GOTO L479;
RSLT=XDEQ;
GOTO L480;
L479: CALL CVB(ADDR(XD),ADDR(RSLT));
L480: I=RSLT;
TEXT(1)='1';
CALL INOUT(TEXTADR,3,I);
GOTO L491;

/*-----*/
/*                                          */
/*      EVALUATE VARIABLE OR CONSTANT                                          */
/*                                          */
/*-----*/
L472: XPX=0;
T2= '1'B;
CALL PARSE;
IF IERR ^= 0 THEN GOTO L9001;
L490: IF DTYPE = '2' THEN GOTO L492;
REQ(1)=XD(1);
REQ(2)=XD(2);
REQ(3)=XD(3);
REQ(4)=XD(4);
CALL CVC(ADDR(XD),ADDR(RSLT));
L492: CALL INOUT(ADDR(XD),BINARY(4),BINARY(0));

/*-----*/
/*                                          */
/*      NEXT ARGUMENT?                                          */
/*                                          */
/*-----*/
L491: IF XPY > LEN(ADDR(XDSAVE)) THEN GOTO L240;

```

```

        IF XDSAVE(XPY) ^= ',' THEN GOTO L256; /* ERROR*/
        CALL SUBSTR(ADDR(XDSAVE(XPY+1)),ADDR(XD),1,-1);
        GOTO L471;

/*-----*/
/*                                          */
/*      GOTO                                          */
/*                                          */
/*-----*/
L500: ARGRET=L501;
      GOTO L5000; /* ARGSCAN*/

/*-----*/
/*                                          */
/*      BRANCH TO INTERNAL SUB TO EVALUATE ARGUMENT      */
/*                                          */
/*-----*/
L501: IORG=1;
      RETZ=L502;
      GOTO L5301; /* PROCESS ARGUMENTS*/
L502: IF IBR = 0 THEN GOTO L240;          /* CONTINUE - NO BRANCH*/
      IF FTMP(1) ^= AC(1) THEN      GSBX=0; /* CHAIN TOOK PLACE*/
      FORX=0;          /* DELETES ALL FOR'S*/
      PD1CUR=IBR; /* BRANCH ADDRESS*/
      GOTO L211;

/*-----*/
/*                                          */
/*      SET                                          */
/*                                          */
/*-----*/
L570: P1SVE=P1PX;
      T2= '1'B; /* YES MULTI ARG*/
      P1PX=P1PX+1;
      IF ZD(P1PX) ^= ':' THEN GOTO L575;
      LPOSTC=L575;
      GOTO L799; /* POST CONDITIONAL*/

/*-----*/
/*                                          */
/*      EXTRACT LEFT HAND SIDE OF STATEMENT (LHS)      */
/*                                          */
/*-----*/
L575: IF ZD(P1PX) ^= ' ' THEN GOTO L256;
      P1PX=P1PX+1;
L590: P1SVE=P1PX;
      I=0;
L600: P1PX=P1PX+1;
L610: IF P1PX > ZDLEN THEN GOTO L256; /* ERROR MESSAGE*/
      IF ZD(P1PX) ^= '"' THEN GOTO L611;
      CALL QSCAN(ADDR(ZD),P1PX,IERR); /* QUOTE SCANNER*/
      IF IERR ^= 0 THEN GOTO L9001; ELSE GOTO L600;
L611: IF ZD(P1PX) = '(' THEN I=I+1;
      IF ZD(P1PX) = ')' THEN I=I-1;
      IF ZD(P1PX) = '=' & I = 0 THEN GOTO L640; /* NOT = IN PARENS*/
L630: GOTO L600;

```

```

/*-----*/
/*
/*      PROCESS LHS DEFINITION
/*
/*-----*/
L640:  CALL SUBSTR(ADDR(ZD),ADDR(XD),P1SVE,P1PX-P1SVE);
L641:  XPX=0;
      TOPX=1;
      CALL PARSE;
      IF IERR = 0 THEN GOTO L9001;
L650:  CALL SCOPY(ADDR(V1D),ADDR(VD) );
/*-----*/
/*
/*      CHECK FOR LHS INDIRECTION
/*
/*-----*/
L660:  IF VD(1) = '@' THEN GOTO L740;
      ARGRET=L641;
      GOTO L670; /* PROCESS INDIRECTION*/
/*-----*/
/*
/*      EXTRACT AND PROCESS RHS
/*
/*-----*/
L740:  JPX=P1PX+1;
L750:  P1PX=P1PX+1;
      IF ZDLEN < P1PX THEN GOTO L770;
      IF ZD(P1PX) = ' ' THEN GOTO L760;
      CALL QSCAN(ADDR(ZD),P1PX,IERR);
      IF IERR = 0 THEN GOTO L9001;
      ELSE GOTO L750;
L760:  IF ZD(P1PX) = ' ' THEN GOTO L750;
L770:  CALL SUBSTR(ADDR(ZD),ADDR(XD),JPX,P1PX-JPX);
L780:  XPX=0;
      CALL PARSE;
      IF IERR = 0 THEN GOTO L9001;
      ITYPE=DTYPE;
      RETZ=L791;
      GOTO L800;
L791:  IF XPY > LEN(ADDR(XDSAVE)) THEN GOTO L240;
      IF XDSAVE(XPY) = ',' THEN GOTO L256; /* ERROR*/
      CALL SUBSTR(ADDR(XDSAVE(XPY+1)),ADDR(XD),1,-1);
      P1PX=INDEX(ADDR(ZD(P1SVE)),ADDR(XD),1); /* FIND POSITION IN ZD*/
      P1PX=P1PX+P1SVE-1;
      GOTO L590; /* AGAIN*/
/*-----*/
/*
/*      QUIT
/*
/*-----*/
L4000: P1PX=P1PX+1;
      IF P1PX > ZDLEN THEN GOTO L4005;

```

```

      IF ZD(P1PX) ^= ':' THEN GOTO L4005;
      LPOSTC=L4005;
      GOTO L799; /* POST CONDITIONAL*/

/*-----*/
/*
/*      QUIT CAN RESULT FROM FOR, DO OR END OF PROGRAM
/*      AN ARTIFICIAL QUIT IS GENERATED WHEN THERE IS NO MORE TEXT
/*      CHECK THE GOSUB STACK
/*      IF ITEM ONE OF STACK ELEMENT IS NEG, FOR LOOP CONTROL PRESENT*/
/*
/*-----*/
L4005: IF GSBX = 0 THEN GOTO L4010; /* END - NO ACTIVE GOSUBS*/
/*-----*/
/*
/*      TEST FOR FOR LOOP GOSUB MARKER - ELEMENT 1 LT 0
/*
/*-----*/
      IF GOSUB(1,GSBX) < 0 THEN GOTO L7019;
/*-----*/
/*
/*      GOSUB RETURN - RESTORE ENVIRONMENT TO CONDITIONS WHEN GOSUB
/*      EXECUTED - RELOAD PD1CUR, PD1OLD, XD (RESIDUE ONLY), ZD ETC.
/*      RETURN TO BRANCH ADDRESS SAVED.
/*
/*-----*/
      PD1CUR=GOSUB(1,GSBX); /* RESTORE POINTER*/
      XX=INDEX(ADDR(PD1(PD1CUR)),ADDR(XTAB),1);
      CALL SUBSTR(ADDR(PD1(PD1CUR)),ADDR(ZD),XX+1,-1);
      ZDLEN=LEN(ADDR(ZD));
      PD1OLD=PD1CUR;
      PD1CUR=PD1CUR+LEN(ADDR(PD1(PD1CUR)))+1;
      IF GOSUB(4,GSBX) ^= 0 THEN GOTO L4006;
      AB=0;
      XD(1)=AC(2); /* EMPTY*/
      GOTO L4007;
L4006: CALL SCOPY(ADDR(ZD(GOSUB(4,GSBX))),ADDR(XD)); /* COMMAND RES*/
L4007: M=GOSUB(5,GSBX); /* ORIGIN X COMMAND*/
      I=PD1LEN;
      PD1LEN=GOSUB(6,GSBX); /* ORIGINAL PD1LEN*/
/*-----*/
/*
/*      ERASE SUBROUTINE
/*
/*-----*/
      DO J=PD1LEN+1 TO I;
      AB=0;
      PD1(J)=AC(2);
      END; /* ERASE SUBROUTINE*/
      P1PX=GOSUB(2,GSBX); /* ORIGINAL POSITION*/
      GSUBL=GOSUBLAB(3,GSBX); /* RETURN POINT*/
      RETZ=GSUBL;
      GSBX=GSBX-1; /* FREE STACK*/

```

```

        GOTO RETZ; /* RETURN*/
/*-----*/
/*
/*      CHECK FOR "FOR" LOOP
/*
/*-----*/
L4010: IF FORX = 0 THEN GOTO L210; /* TERMINATE*/
      ELSE GOTO L7019; /* LOOP TERMINATION*/
/*-----*/
/*
/*      XECUTE
/*
/*-----*/
L8000: M=P1PX; /* ORIGIN OF X COMMAND*/
      ARGRET=L8001;
      GOTO L5000; /* ARGRET*/
/*-----*/
/*
/*      PARSE ARGUMENT
/*
/*-----*/
L8001: T2= '1'B; /* MULTI ARG YES*/
      XPX=0;
      CALL PARSE;
      IF DTYPE ^= '2' THEN GOTO L256; /* CHAR STRINGS ONLY*/
L8002: IF IERR ^= 0 THEN GOTO L9001;
      CALL SCOPY(ADDR(XD),ADDR(FTMP) );
/*-----*/
/*
/*      CHECK FOR ARGUMENT POST-CONDITIONAL
/*
/*-----*/
      IF XDSAVE(XPY) ^= ':' THEN GOTO L8033;
      CALL SCOPY(ADDR(XDSAVE(XPY+1)),ADDR(XD) );
      XPX=0;
      T2= '1'B;
      CALL PARSE;
      IF IERR ^= 0 THEN GOTO L9001;
L8032: IF DTYPE = '2' THEN GOTO L8003;
      RSLT=XDEQ;
      GOTO L8004;
L8003: CALL CVB(ADDR(XD),ADDR(RSLT));
L8004: K=RSLT;
      IF K ^= 0 THEN GOTO L8033;
      IF XPY > LEN(ADDR(XDSAVE)) THEN GOTO L240;
      IF XDSAVE(XPY) = ',' THEN GOTO L8034; ELSE GOTO L9106;
L8034: CALL SCOPY(ADDR(XDSAVE(XPY+1)),ADDR(XD) );
      GOTO L8001;
/*-----*/
/*
/*      MOVE LINE TO BE EXECUTED INTO CODE SPACE ABOVE
/*
/*      THIS PROGRAM - REMEMBER OLD EXTENT PD1LEN FOR
/*

```

```

/*      ERASING REFERENCES LATER      */
/*      */
/*-----*/
L8033: L=PD1LEN;
      DO I=1 TO LEN(ADDR(FTMP));
      PD1(PD1LEN+I)=FTMP(I);
      END;
      PD1(PD1LEN+1+LEN(ADDR(FTMP)))=AC(1); /* TERMINATOR*/
      PD1LEN=PD1LEN+1+LEN(ADDR(FTMP));
/*-----*/
/*      */
/*      PRESERVE COMMAND RESIDUE      */
/*      */
/*-----*/
      IF XPY > LEN(ADDR(XDSAVE)) THEN GOTO L8036;
      IF XDSAVE(XPY) = ',' THEN GOTO L8037;
      GOTO L9106;
L8037: CALL SCOPY(ADDR(XDSAVE(XPY+1)),ADDR(XD) );
      XPX=1;
      GOTO L8031;
L8036: AB=0;
      XD(1)=AC(2); /* EMPTY*/
      XPX=0;
/*-----*/
/*      */
/*      SET UP GOSUB STACK ENTRY      */
/*      */
/*-----*/
L8031: GSBX=GSBX+1;
      GOSUB(1,GSBX)=PD1OLD;
      GOSUB(2,GSBX)=P1PX;
/*-----*/
/*      */
/*      FIND XD IN ZD                  */
/*      */
/*-----*/
      I=0;
      IF LEN(ADDR(XD)) = 0 THEN GOTO L8006;
L8005: I=INDEX(ADDR(ZD(M)),ADDR(XD),I+1);
      IF M+I+LEN(ADDR(XD))-1 = P1PX THEN GOTO L8005;
L8006: GOSUB(4,GSBX)=I;
      GOSUB(5,GSBX)=M;
      GOSUB(6,GSBX)=L;
      RETZ=L8040;
      GSUBL=RETZ;
      GOSUBLAB(3,GSBX)=GSUBL; /* RETURN POINT*/
      PD1CUR=L+1;             /* START POINT*/
      XX=0;
      GOTO L212;              /* RECURSE*/
/*-----*/
/*      */
/*      RETURN POINT - ENVIRONMENT RESTORED      */

```

```

/*                                                    */
/*-----*/
L8040: IF LEN(ADDR(XD)) = 0 THEN GOTO L240;
      ELSE GOTO L8001; /* NEXT ARG*/
/*-----*/
/*                                                    */
/* FOR                                                    */
/*                                                    */
/*-----*/
L7000: L=P1PX;
/*-----*/
/*                                                    */
/* HOLD FOR LATER STACK ENTRY                          */
/*                                                    */
/*-----*/
      ARGRET=L7001;
      GOTO L5000; /* ARGSCAN*/
/*-----*/
/*                                                    */
/* FIND LOCAL VARIABLE                                  */
/*                                                    */
/*-----*/
L7001: M=P1PX;          /* HOLD FOR LATER STACK ENTRY */
      XPX=INDEX(ADDR(XD),ZPAD('='),1);
      IF XPX = 0 THEN GOTO L256; /* ERROR*/
      CALL SCOPY(ADDR(XD(XPX+1)),ADDR(FTMP) ); /* SAVE REMAINDER*/
      XD(XPX)=AC(1); /* STRING TERMINATOR FOR LOCAL VARIABLE*/
L7004: XPX=0;
      TOPX=1;          /* LHS SWITCH*/
      CALL PARSE;
      IF IERR ^= 0 THEN GOTO L9001;
/*-----*/
/*                                                    */
/* CHECK FOR INDIRECTION OF LOOP VARIABLE              */
/*                                                    */
/*-----*/
L7002: IF V1D(1) ^= '@' THEN GOTO L7010;
      CALL SCOPY(ADDR(V1D),ADDR(VD) );
      ARGRET=L7003;
      GOTO L670;
L7003: GOTO L7004;
/*-----*/
/* ALLOCATE FOR STACK; COPY LOOP VARIABLE INTO STACK  */
/*                                                    */
/*-----*/
L7010: FORX=FORX+1; /* STACK EXTENT*/
      CALL SCOPY(ADDR(V1D),ADDR(FORV(FORX,1))); /* LOCAL VARIABLE*/
      CALL SCOPY(ADDR(FTMP),ADDR(XD) ); /* REMAINDER OF COMMAND*/
/*-----*/
/* STACK SET UP                                         */

```

```

/*      FORIT - INCREMENT TEST FLAG                                */
/*      FORLT - LIMIT TEST FLAG                                    */
/*      GOSUB(,1) - FOR MARKER                                    */
/*      GOSUB(,3) - ORG OF FOR                                    */
/*      GOSUB(,4) - ORG ARGUMENTS OF FOR                        */
/*      -----*/
L6003: XPX=0;
      FORIT(FORX)= '0'B;
      FORLT(FORX)= '0'B;
      GSBX=GSBX+1;
      GOSUB(1,GSBX)=-1; /* FOR MARKER*/
      GOSUB(2,GSBX)=0;
      GOSUB(3,GSBX)=L;
      GOSUB(4,GSBX)=M;
/*      -----*/
/*      FIND INITIAL VALUE                                        */
/*      -----*/
      T2= '1'B; /* MULTI-ARG*/
      CALL PARSE;
      IF IERR ^= 0 THEN GOTO L9001;
/*      -----*/
/*      XD HAS INIT VAL ON RETURN - COPY TO FTMP                */
/*      -----*/
      IF DTYPE = '1' THEN CALL CVC(ADDR(XD),ADDR(XD));
      CALL SCOPY(ADDR(XD),ADDR(FTMP) );
/*      -----*/
/*      CHECK FOR MULTI-ARGUMENT, END OF COMMAND, OR INCR FOLLOWS */
/*      -----*/
      IF XDSAVE(XPY) = ',' THEN GOTO L7014; /* MULTI ARG*/
      IF XPY > LEN(ADDR(XDSAVE)) THEN GOTO L6000;
      IF XDSAVE(XPY) ^= ':' THEN GOTO L256; /* ERROR*/
/*      -----*/
/*      FIND INCREMENT                                           */
/*      -----*/
      CALL SCOPY(ADDR(XDSAVE(XPY+1)),ADDR(XD) );
      XPX=0;
      T2= '1'B; /* MULTI-ARG*/
      CALL PARSE;
      IF IERR ^= 0 THEN GOTO L9001;
/*      -----*/
/*      XD HAS INCRMNT ON RETURN - COPY TO STACK AND SET FLAG   */
/*      -----*/

```

```

/*-----*/
L7012: FORIT(FORX)= '1'B; /* INCR TEST*/
      IF DTYPE = '2' THEN GOTO L7025;
      RSLT=XDEQ;
      GOTO L7026;
L7025: CALL CVB(ADDR(XD),ADDR(RSLT)); /* INCREMENT*/
L7026: FORIN(FORX)=RSLT;
/*-----*/
/*
/*      CHECK FOR END OF COMMAND, MULTI-ARG OR LIMIT FOLLOWS
/*
/*
/*-----*/
      IF XDSAVE(XPY) = ',' THEN GOTO L7014;
      IF XPY > LEN(ADDR(XDSAVE)) THEN GOTO L6000;
      IF XDSAVE(XPY) != ':' THEN GOTO L256; /* BAD CHAR*/
/*-----*/
/*
/*      GET UPPER LIM
/*
/*
/*-----*/
      CALL SCOPY(ADDR(XDSAVE(XPY+1)),ADDR(XD) );
      XPX=0;
      T2= '1'B;
      CALL PARSE;
      IF IERR != 0 THEN
        GOTO L9001;
/*-----*/
/*
/*      XD HAS LIM - COPY TO STACK AND SET FLAG
/*
/*
/*-----*/
L7013: FORLT(FORX)= '1'B;
      IF DTYPE = '2' THEN GOTO L7027;
      RSLT=XDEQ;
      GOTO L7028;
L7027: CALL CVB(ADDR(XD),ADDR(RSLT));          /* LIMIT*/
L7028: FORLV(FORX)=RSLT;
/*-----*/
/*
/*      CHECK FOR END OF COMMAND OR MULTI ARG
/*
/*
/*-----*/
      IF XDSAVE(XPY) = ',' THEN GOTO L7014;
      IF XPY > LEN(ADDR(XDSAVE)) THEN GOTO L6000;
      ELSE GOTO L256;
/*-----*/
/*
/*      MULTI-ARGUMENT CASE - FIND POSITION OF WHERE
/*      WE ARE IN ZD AND SET K FOR RETORE OPERATION LATER
/*
/*
/*-----*/
L7014: I=L;

```

```

L6004: K=INDEX(ADDR(ZD),ZPAD(I),1);
      IF K+LEN(ADDR(XDSAVE(XPY))) = M THEN
      GOTO L6001;
      I=K+1;
      GOTO L6004;
L6001: GOSUB(2,GSBX)=K+1;
      GOSUB(5,GSBX)=LEN(ADDR(XDSAVE(XPY+1)));
/*-----*/
/*
/*      INITIALIZE LOOP VARIABLE
/*
/*-----*/
L6000: SYMFLG=0;
      ITYPE='2';
      CALL SYM(ADDR(FORV(FORX,1)),ADDR(FTMP),ITYPE);
/*-----*/
/*
/*      STORE VALUE
/*
/*-----*/
      IF SYMLEN <= PD1LEN THEN GOTO L9131;
/*-----*/
/*
/*      SAVE POINTER IN ZD OF NEXT COMMAND - CONTINUE EXECUTION
/*
/*-----*/
      FORNXT(FORX)=M;
      P1PX=M;
      GOTO L240;
/*-----*/
/*
/*      CONTINUE
/*      END OF LINE RETURN
/*
/*-----*/
L7015: IF FORIT(FORX) THEN GOTO L7016; /* INC OR LIMIT */
/*-----*/
/*
/*      CHECK FOR MULTIPLE ARG CASE HERE
/*
/*-----*/
L7019: IF GOSUB(2,GSBX) > 0 THEN
      GOTO L6002;
      FORX=FORX-1; /* ENTRY FOR QUIT */
      GSBX=GSBX-1;
      IF FORX > 0 & GOSUB(1,GSBX) = -1 THEN GOTO L7015;
      /* OUTER FOR */
      GOTO L211; /* NEXT LINE */
/*-----*/
/*
/*      RESTORE XD AND PARSE NEXT ARGUMENT
/*

```

```

/*-----*/
L6002: CALL SUBSTR(ADDR(ZD),ADDR(XD),GOSUB(2,GSBX),GOSUB(5,GSBX));
      L=GOSUB(3,GSBX);
      M=GOSUB(4,GSBX);
      GSBX=GSBX-1;
      GOTO L6003;

/*-----*/
/*                                     */
/*      INCREMENT                      */
/*                                     */
/*-----*/
L7016: SYMFLG=1;
      CALL SYM(ADDR(FORV(FORX,1)),ZPAD(''),ITYPE);
      IF ITYPE = '2' THEN
        GOTO L7030;
      REQ(1)=PD1(SYMADR);
      REQ(2)=PD1(SYMADR+1);
      REQ(3)=PD1(SYMADR+2);
      REQ(4)=PD1(SYMADR+3);
      GOTO L7031;
L7030: CALL CVB(ADDR(PD1(SYMADR)),ADDR(RSLT));
L7031: RSLT=RSLT+FORIN(FORX);
      SYMFLG=0;      /* STORE */
      ITYPE='1';
      CALL SYM(ADDR(FORV(FORX,1)),ADDR(RSLT),ITYPE);
      IF SYMLEN <= PDILEN THEN GOTO L9131;
      IF FORLT(FORX) THEN GOTO L7017; /*TEST LIMIT*/
L7020: P1PX=FORNXT(FORX); /* NEXT COMMAND */
      GOTO L240;

/*-----*/
/*                                     */
/*      LIMIT TEST                    */
/*                                     */
/*-----*/
L7017: IF FORIN(FORX) < 0 THEN
      GOTO L7018;
      IF RSLT > FORLV(FORX) THEN
        GOTO L7019; ELSE GOTO L7020;
L7018: IF RSLT < FORLV(FORX) THEN
      GOTO L7019; ELSE GOTO L7020;

/*-----*/
/*                                     */
/*      HALT/HANG*/                  */
/*                                     */
/*-----*/
L5100: ARGRET=L5101;
      GOTO L5000;      /* ARGSCAN */

/*-----*/
/*                                     */
/*      TEST FOR HALT OR HANG        */
/*                                     */
/*-----*/

```

```

L5101: IF XD(1) = 0 THEN GOTO L210; /* HALT */
L5102: XPX=0;
      T2= '1'B; /* MULTIARG */
      CALL PARSE;
      IF IERR = 0 THEN GOTO L9001;
      IF DTYPE = '2' THEN GOTO L5103;
      RSLT=XDEQ;
      GOTO L5107;
L5103: CALL CVB(ADDR(XD),ADDR(RSLT));
L5107: I=RSLT;
      IF I <= 0 THEN GOTO L5105;
      IF XWAIT(MTCUR) > 0 THEN GOTO L5108;
      XWAIT(MTCUR)=0;
L5108: /*IF SECNDS(XWAIT(MTCUR)) > I THEN GOTO L5105; */
      P1PX=P2PX;
      GOTO L240;
      /* RESET*/
L5105: XWAIT(MTCUR)=0;
      IF XPY > LEN(ADDR(XDSAVE)) THEN GOTO L240;
      /* CONTINUE*/
      IF XDSAVE(XPY) = ',' THEN GOTO L256;
L5104: CALL SUBSTR(ADDR(XDSAVE(XPY+1)),ADDR(XD),1,-1);
      GOTO L5102; /* NEXT ARG */

/*-----*/
/*
/*      ELSE
/*
/*-----*/
L5200: ARGRET=L5201;
      GOTO L5000;

/*-----*/
/*
/* ARGSCAN - EXCP:      PERMITS POST COND OF ELSE
/*
/*-----*/
L5201: IF XD(1) = AC(1) THEN
      GOTO L5202; ELSE GOTO L9105;
L5202: IF TPX = 0 THEN GOTO L240; /* CONTINUE*/
      P1PX=ZDLEN+1; /* MAX*/
      GOTO L240; /* CHECK FOR LOOPS OR NEXT LINE*/

/*-----*/
/*
/*      DO
/*
/*-----*/
L5300: M=P1PX;
      ARGRET=L5327;
      GOTO L5000;
L5327: IORG=PD1LEN+1;
      RETZ=L5326;
      GOTO L5301;
/* PROCESS ARGS*/

```

```

/*-----*/
/*
/*      RETURN POINT FROM MINI SUB
/*
/*-----*/
L5326: GSBX=GSBX+1;
      GOSUB(1,GSBX)=PD1OLD;
      GOSUB(2,GSBX)=P1PX;
      GSUBL=L5313;
      GOSUBLAB(3,GSBX)=GSUBL;
      I=0;
      IF XD(1) = AC(1) THEN
      GOTO L5314;
      IF XD(1) ~= ',' THEN GOTO L256;
L5379: CALL SCOPY(ADDR(XD(2)),ADDR(XD) );
L5360: I=INDEX(ADDR(ZD(M)),ADDR(XD),I+1);
      IF M+I+LEN(ADDR(XD))-1 ~= P1PX THEN
      GOTO L5360;
      GOSUB(4,GSBX)=I;
      GOTO L5315;
L5314: GOSUB(4,GSBX)=0;
L5315: PD1CUR=IBR;
      GOSUB(5,GSBX)=M;
      GOSUB(6,GSBX)=IORG-1;
      GOTO L211;

/*-----*/
/*
/*      RETURN POINT
/*
/*-----*/
L5313: IF XD(1) ~= AC(1) THEN GOTO L5327; ELSE GOTO L240;

/*-----*/
/*
/*      KILL
/*
/*-----*/
L5400: ARGRET=L5401;
      GOTO L5000;
L5401: IF XD(1) ~= AC(1) THEN GOTO L5403;

/*-----*/
/*
/*      KILL ALL
/*
/*-----*/
      SYMLen=4000;
      GOTO L240;
L5403: IF XD(1) = '(' THEN GOTO L5412;

/*-----*/
/*
/*      KILL SELECTED
/*
/*-----*/

```

```

L5405: TOPX=1;
/* LHS SWITCH*/
    XPX=0;
    CALL PARSE;
    IF DTYPE ^= '2' THEN GOTO L256;
L5441: IF IERR ^= 0 THEN
    GOTO L9001;
/*    ANS IN V1D ON RETURN*/
    IF V1D(1) ^= '@' THEN GOTO L5409;
/* KILL BY INDIRECTION*/
    CALL SCOPY(ADDR(V1D),ADDR(VD) );
    ARGRET=L5405;
/* PROCESS INDIRECTION*/
    GOTO L670;
/* PROCESS LHS INDIRECTION*/
/*-----*/
/*
/*      DELETE REF
/*
/*-----*/
L5409: IF V1D(1) = '-' THEN GOTO L5410;
/* KILL A GLOBAL*/
    SYMFLG=2;
/* DELETE*/
    CALL SYM(ADDR(V1D),ZPAD(''),ITYPE);
L5411: XPX=XPX+1;
    IF XPX >= LEN(ADDR(XD)) THEN GOTO L240;
    IF XD(XPX) ^= ',' THEN GOTO L256;
    XPX=XPX+1;
    CALL SUBSTR(ADDR(XD),ADDR(XD),XPX,-1);
    GOTO L5405;
/*-----*/
/*
/*      KILL A GLOBAL
/*
/*-----*/
L5410: CALL SCOPY(ADDR(V1D),ADDR(GD) );
    CALL GBLREF( '1'B );
    IF F = AC(1) THEN GOTO L5415;
    G(3)=3;
/* DELETE*/
    CALL GLOBAL;
    TMP1(2)=AC(1);
    IF F ^= AC(1) THEN GOTO L5411;
L5415: GOTO L9103;
/*-----*/
/*
/*      KILL EXCEPT
/*
/*-----*/
L5412: CALL SUBSTR(ADDR(XD),ADDR(XD),2,-1);
    I=LEN(ADDR(XD));

```

```

        IF XD(I) ^= ' ' THEN GOTO L256;
L5413: XD(I)=AC(1);
        FTMP(1)=AC(1);
/* EMPTY*/
L5414: TOPX=1;
/* LHS SWITCH*/
        XPX=0;
        CALL PARSE;
        IF IERR ^= 0 THEN GOTO L9001;
        IF VID(1) = '-' THEN GOTO L256;
/* KILL EXCPT NOT LEGAL FOR GLOBALS*/
        IF VID(1) ^= '@' THEN GOTO L5420;
        CALL SCOPY(ADDR(VID),ADDR(VD) );
        ARGRET=L5414;
        GOTO L670;
/* PROCESS LHS INDIRECTION*/
L5420: CALL CONCAT(ADDR(FTMP),ZPAD(' '),ADDR(FTMP));
/* ACCUMULATE NAMES NOT TO BE KILLED*/
        CALL CONCAT(ADDR(FTMP),ADDR(VID),ADDR(FTMP));
        IF XPX >= LEN(ADDR(XD)) THEN
            GOTO L5421;
        XPX=XPX+1;
        IF XD(XPX) ^= ',' THEN GOTO L256;
        XPX=XPX+1;
        CALL SUBSTR(ADDR(XD),ADDR(XD),XPX,-1);
        GOTO L5414;
/*-----*/
/*
/*      KILL LOOP
/*
/*-----*/
L5421: CALL KILL1(ADDR(TMP2));
        GOTO L240;
/*-----*/
/*
/*      CLOSE
/*
/*-----*/
L5500: ARGRET=L5501;
        GOTO L5000;
/* ARGSCAN*/
L5501: IF LEN(ADDR(XD)) = 0 THEN GOTO L256;
L5503: XPX=0;
        T2= '1'B;
        CALL PARSE;
        IF IERR ^= 0 THEN GOTO L9001;
L5502: IF DTYPE = '2' THEN GOTO L5505;
        RSLT=XDEQ;
        GOTO L5504;
L5505: CALL CVB(ADDR(XD),ADDR(RSLT));
L5504: I=RSLT;
        IF OPNFLG(I) ^= 0 THEN GOTO L5506;

```

```

        ELSE GOTO L9126;
L5506: IF OPNFLG(I) = 2 THEN ;
        OPNFLG(I)=0;
        IF XPY > LEN(ADDR(XDSAVE)) THEN GOTO L240;
        IF XDSAVE(XPY) ^= ',' THEN GOTO L256;
        CALL SUBSTR(ADDR(XDSAVE(XPY+1)),ADDR(XD),1,-1);
        GOTO L5503;
L5509: GOTO L9126;
/*-----*/
/*                                          */
/*      USE                                          */
/*                                          */
/*-----*/
L5600: ARGRET=L5601;
        GOTO L5000;
/*  ARGSCAN*/
L5601: IF XD(1) = AC(1) THEN GOTO L256;
        XPX=0;
        T2= '1'B;
        CALL PARSE;
        IF IERR ^= 0 THEN GOTO L9001;
L5602: IF DTYPE = '2' THEN GOTO L5603;
        RSLT=XDEQ;
        GOTO L5604;
L5603: CALL CVB(ADDR(XD),ADDR(RSLT));
L5604: IO=RSLT;
        IF XPY > LEN(ADDR(XDSAVE)) THEN GOTO L240;
        ELSE GOTO L256;
/*-----*/
/*                                          */
/*      REMOTE BLOCKS BASED ON ASSIGNED GOTO'S      */
/*                                          */
/*      POST CONDITIONALS REMOTE BLOCK              */
/*                                          */
/*-----*/
L799: P1PX=P1PX+1;
/*  PAST COLON*/
        POPX=P1PX;
        GOTO L803;
L792: POPX=POPX+1;
L803: IF POPX > ZDLEN THEN
        GOTO L802;
        IF ZD(POPX) ^= ''' THEN GOTO L793;
        CALL QSCAN(ADDR(ZD),POPX,IERR);
        IF IERR ^= 0 THEN GOTO L9001;
        ELSE GOTO L792;
L793: IF ZD(POPX) ^= ' ' THEN GOTO L792;
L802: XPX=0;
        CALL SUBSTR(ADDR(ZD),ADDR(XD),P1PX,POPX-P1PX);
        P1PX=POPX;
        CALL PARSE;
        IF IERR ^= 0 THEN GOTO L9001;

```

```

L794:  IF DTYPE = '2' THEN
        GOTO L804;
        RSLT=XDEQ;
        GOTO L805;
L804:  CALL CVB(ADDR(XD),ADDR(RSLT));
L805:  IF TRUNC(RSLT) ^= 0 THEN
        GOTO LPOSTC; /*RETURN TO EXECUTE COMMAND*/
L795:  P1PX=P1PX+1;
/* SCAN PAST COMMAND*/
L801:  IF P1PX > ZDLEN THEN
        GOTO L240;
/* NEXT COMMAND*/
        IF ZD(P1PX) ^= ' ' THEN GOTO L796;
        CALL QSCAN(ADDR(ZD),P1PX,IERR);
        IF IERR ^= 0 THEN GOTO L9001;
        ELSE GOTO L795;
L796:  IF ZD(P1PX) ^= ' ' THEN GOTO L795;
        ELSE GOTO L240;
/* NEXT COMMAND*/
/*-----*/
/*
/*      REMOTE BLOCK TO PROCESS LHS TYPE REFERENCES
/*
/*-----*/
L800:  IF VD(1) ^= ' ' THEN GOTO L850;
L810:  CALL SCOPY(ADDR(VD),ADDR(GD) );
        CALL GBLREF( '0'B );
        IF F = '0'B THEN GOTO L9118;
        IF DTYPE = '1' THEN GOTO L830;
L821:  CALL SCOPY(ADDR(XD),ADDR(PD) );
        G(2)=2;
        GOTO L840;
L830:  P=XDEQ;
        G(2)=1;
L840:  G(3)=1;
        CALL GLOBAL;
        TMP1(2)=AC(1);
/* GETS MESSED UP IN GLOBAL*/
        IF ERR = 0 THEN GOTO RETZ;
        ELSE GOTO L9128;
L850:  SYMFLG=0;
/* STORE*/
        CALL SYM(ADDR(VD),ADDR(XD),ITYPE);
        IF SYMLen <= PD1LEN THEN GOTO L9131;
        IF SYMFLG ^= 0 THEN GOTO RETZ;
        ELSE GOTO L9123;
/*-----*/
/*
/*      ARGSCAN REMOTE BLOCK
/*
/*-----*/
L5000: P1PX=P1PX+1;

```

```

        IF ZD(P1PX) ^= ':' THEN GOTO L5005;
        LPOSTC=L5005;
        GOTO L799;
/* POST CONDITIONAL*/
L5005: IF ZD(P1PX) = ' ' THEN GOTO L5009;
L5006: IF P1PX <= ZDLEN THEN
        GOTO L5021;
        XD(1)=AC(1);
        GOTO ARGRET;
L5021: GOTO L9106;
L5009: P1PX=P1PX+1;
        POPX=P1PX;
        GOTO L5010;
L5007: POPX=POPX+1;
L5010: IF POPX > ZDLEN THEN
        GOTO L5020;
/* EOL*/
        IF ZD(POPX) ^= ''' THEN GOTO L5015;
        CALL QSCAN(ADDR(ZD),POPX,IERR);
        IF IERR ^= 0 THEN GOTO L9001;
        ELSE GOTO L5007;
L5015: IF ZD(POPX) ^= ' ' THEN GOTO L5007;
L5020: CALL SUBSTR(ADDR(ZD),ADDR(XD),P1PX,POPX-P1PX);
        P1PX=POPX;
        GOTO ARGRET;

/*-----*/
/*
/*      REMOTE BLOCK TO PROCESS DO AND GOTO ARGS
/*
/*-----*/
L5301: XPX=0;
        J=0;
/* USED TO COUNT PARENS*/
/*-----*/
/*
/*      SCAN FOR FIRST ARGUMENT
/*      INDIVIDUAL ARGUMENTS MAY BE POST-CONDITIONALIZED
/*
/*-----*/
L5302: XPX=XPX+1;
        IF XPX > LEN(ADDR(XD)) THEN
        GOTO L5303;
        IF XD(XPX) ^= ''' THEN
        GOTO L5328;

/*-----*/
/*
/*      SCAN QUOTED STRING
/*
/*-----*/
        CALL QSCAN(ADDR(XD),XPX,IERR);
        IF IERR ^= 0 THEN GOTO L9001;
        ELSE GOTO L5302;

```

```

/*-----*/
/*
/*      CHECK FOR PARENED EXPRESSIONS -
/*      LOOK FOR COMMA, COLON OR END OF TEXT WHEN
/*      PAREN COUNT IS ZERO
/*-----*/
L5328: IF XD(XPX) = '(' THEN J=J+1;
      IF XD(XPX) = ')' THEN J=J-1;
      IF XD(XPX) = ',' & J = 0 THEN GOTO L5303;
      IF XD(XPX) = ':' & J = 0 THEN GOTO L5303;
      GOTO L5302;

/*-----*/
/*
/*      CHECK FOR INDIRECTION IN LEL FIELD, PARSE AND SUBSTITUTE
/*-----*/
L5303: IF XD(1) = '@' THEN GOTO L5351;
      T2= '1'B;
      XPX=0;
      TOPX=1; /* LHS TYPE REF*/
      CALL PARSE;
      IF IERR = 0 THEN
        GOTO L9001;
      CALL SCOPY(ADDR(V1D),ADDR(VD) );
      ARGRET=L5301;
      GOTO L670;

/*-----*/
/*
/*      COPY LEL TO DOSAVE (MAX 20 CHARS)
/*-----*/
L5351: L=0; /* OFFSET*/
      DOSAVE(1)=AC(1); /* STARTING LEL*/
      IF XD(1) = '-' THEN GOTO L5361; /*NO START LEL-FILE SPEC ONLY*/

/*-----*/
/*
/*      FIND END OF LEL FIELD
/*-----*/
      DO I=1 TO XPX-1; /* FIND DELIM AFTER LEL*/
      IF XD(I) >= 'A' & XD(I) <= 'I' THEN GOTO L5353;
      IF XD(I) >= 'J' & XD(I) <= 'R' THEN GOTO L5353;
      IF XD(I) >= 'S' & XD(I) <= 'Z' THEN GOTO L5353;
      IF XD(I) >= '0' & XD(I) <= '9' THEN GOTO L5353;
      GOTO L5354; /* END OF LAB*/
L5353: END;

/*-----*/
/*
/*      COPY LEL TO DOSAVE
/*-----*/
      CALL SUBSTR(ADDR(XD),ADDR(DOSAVE),1,XPX-1);

```

```

      CALL SCOPY(ADDR(XD(XPX)),ADDR(XD) );
/* REMOVE FRONT PART - LEAVE RESIDUE*/
      XPX=1;
      GOTO L5361; /* SIMPLE LEL*/
/*-----*/
/*
/*      LEL FOLLOWED BY SPECIAL CHARACTER
/*
/*-----*/
L5354: CALL SUBSTR(ADDR(XD),ADDR(DOSAVE),1,I-1);
      IF XD(I) ^= '+' THEN GOTO L5362; /* NOT OFFSET*/
/*-----*/
/*
/*      FIND END OF OFFSET
/*
/*-----*/
      DO J=I+1 TO XPX-1;
      IF XD(J) ^= '-' THEN GOTO L5355;
      /*      FILE SPECS MUST BE -@... OR -"...*/
      IF XD(J+1) = '@' | XD(J+1) = '"' THEN GOTO L5356;
L5355: END;
      K=1;
      GOTO L5357;
L5356: K=0;
      XD(J)=': ';
/*-----*/
/*
/*      PASS LABEL OFFSET EXPRESSION TO PARSER
/*
/*-----*/
L5357: T2= '1'B;
      XPX=I;
      CALL PARSE;
      IF IERR ^= 0 THEN
      GOTO L9001;
      IF DTYPE ^= '1' THEN
      GOTO L5358;
      RSLT=XDEQ;
      GOTO L5359;
L5358: CALL CVB(ADDR(XD),ADDR(RSLT));
L5359: L=RSLT;
      IF XPY > LEN(ADDR(XDSAVE)) THEN
      GOTO L5364;
      CALL SCOPY(ADDR(XDSAVE(XPY)),ADDR(XD) );
      IF K = 0 THEN XD(1)='-';
      XPX=1;
      GOTO L5361;
L5364: XD(1)=AC(1);
      XPX=0;
      GOTO L5361;
/*-----*/
/*

```

```

/*      MUST BE FILE SPEC FOLLOWING LEL                                */
/*                                                                 */
/*-----*/
L5362: IF XD(I) ^= '-' THEN GOTO L256;
L5363: CALL SCOPY(ADDR(XD(I)),ADDR(XD) );
/*-----*/
/*                                                                 */
/*      EXTRACT FILE SPEC IF ANY                                    */
/*                                                                 */
/*-----*/
L5361: CALL SCN1(J,ADDR(XD),ADDR(FTMP),IERR,XPX);
      IF IERR ^= 0 THEN
      GOTO L9001;
      GOTO L5332;
/*-----*/
/*                                                                 */
/*      SCAN FOR LEL                                              */
/*                                                                 */
/*-----*/
L5371: IF FTMP(1) = AC(1) THEN
      GOTO L5372;
/*-----*/
/*                                                                 */
/*      FILE LOAD PROCEEDS                                        */
/*                                                                 */
/*-----*/
      IF FTMP(1) ^= '@' THEN GOTO L5377;
/*-----*/
/*                                                                 */
/*      PROCESS INDIRECT FILE REFERENCE                          */
/*                                                                 */
/*-----*/
      CALL SCOPY(ADDR(XD),ADDR(VD) );
/*  HOLD AREA*/
      CALL SCOPY(ADDR(FTMP(2)),ADDR(XD) );
/*  REMOVE @ SIGN*/
      XPX=0;
      CALL PARSE;
      IF IERR ^= 0 THEN
      GOTO L9001;
      IF DTYPE = '2' THEN GOTO L5375;
      ELSE GOTO L9126;
L5375: CALL SCOPY(ADDR(XD),ADDR(FTMP) );
      CALL SCOPY(ADDR(VD),ADDR(XD) );
/*-----*/
/*                                                                 */
/*      LOAD SPECIFIED FILE - IORG SET BY GO OR DO HANDLER      */
/*                                                                 */
/*-----*/
L5377: CALL READF;
      LIMIT=PD1LEN;
/*-----*/

```

```

/*                                                    */
/*      IF DOSAVE NOT EMPTY, FIND LEL - FROM IORG      */
/*                                                    */
/*      OTHERWISE SET IBR TO IORG                      */
/*                                                    */
/*-----*/
L5372: IBR=IORG;
      IF FTMP(1) = AC(1) THEN
      IBR=1;
      IF DOSAVE(1) = AC(1) THEN
      GOTO L5369;
L5312: IF IBR >= PD1LEN THEN
      GOTO L5331;
      I=INDEX(ADDR(PD1(IBR)),ADDR(XTAB),1);
      IF I > 1 THEN
      GOTO L5311;
L5333: IBR=IBR+LEN(ADDR(PD1(IBR)))+1;
      GOTO L5312;
L5311: CALL SUBSTR(ADDR(PD1(IBR)),ADDR(TMP3),1,I-1);
      IF ISCOMP(ADDR(TMP3),ADDR(DOSAVE)) = 0 THEN
      GOTO L5333; ELSE GOTO L5369;
L5331: GOTO L9108;
/*-----*/
/*                                                    */
/*      PROCESS OFFSET IF PRESENT                      */
/*                                                    */
/*-----*/
L5369: IF L = 0 THEN
      GOTO RETZ;
      DO I=1 TO MAX(1,L);
      IBR=IBR+LEN(ADDR(PD1(IBR)))+1;
      IF IBR < PD1LEN THEN GOTO L5374;
      ELSE GOTO L9118;
L5374: END;
      GOTO RETZ;
/*-----*/
/*                                                    */
/*      PROCESS POST CONDITIONAL IF PRESENT (K IS RESULT) */
/*                                                    */
/*-----*/
L5332: K=1;
      XPX=0;
      IF XD(1) = ':' THEN GOTO L5316;
      T2= '1'B;
      XPX=1;
      CALL PARSE;
      IF IERR = 0 THEN
      GOTO L9001;
      IF DTYPE = '1' THEN
      GOTO L5317;
      RSLT=XDEQ;
      GOTO L5321;
L5317: CALL CVB(ADDR(XD),ADDR(RSLT));

```

```

L5321: K=RSLT;
/*-----*/
/*
/*      COPY RESIDUE OF LINE INTO XD IF ANY
/*
/*-----*/
      XPX=0;
      XD(1)=AC(1);
      IF XPY > LEN(ADDR(XDSAVE)) THEN
      GOTO L5316;
      IF XDSAVE(XPY) = ',' THEN GOTO L5323;
      ELSE GOTO L9106;
L5323: CALL SCOPY(ADDR(XDSAVE(XPY)),ADDR(XD) );
L5316: IF K = 1 THEN
      GOTO L5371;
      IBR=0;
      IF XD(1) = AC(1) THEN
      GOTO RETZ;
      IF XD(1) = ',' THEN GOTO L256;
L5370: CALL SCOPY(ADDR(XD(2)),ADDR(XD) );
      GOTO L5301;
/*-----*/
/*
/*      PROCESS INDIRECTION REMOTE BLOCK
/*
/*-----*/
L670:  CALL SUBSTR(ADDR(VD),ADDR(VD),2,-1);
      IF VD(1) = '-' THEN GOTO L710;
L680:  SYMFLG=1;
/* RETRIEVE*/
      CALL SYM(ADDR(VD),ZPAD(''),ITYPE2);
      IF ITYPE2 = 2 THEN
      GOTO L690;
      IF SYMFLG = 0 THEN
      GOTO L700;
L690:  GOTO L9117;
L700:  CALL CONCAT(ADDR(PD1(SYMAADR)),ADDR(XD(XPX+1)),ADDR(TMP2));
      CALL SCOPY(ADDR(TMP2),ADDR(XD) );
      GOTO ARGRET;
L710:  CALL SCOPY(ADDR(VD),ADDR(GD) );
      CALL GBLREF( '1'B );
      IF F = 0 THEN
      GOTO L711;
      G(3)=0;
      CALL GLOBAL;
      TMP1(2)=AC(1);
/* GETS MESSE UP IN GLOBAL*/
      IF F = AC(1) THEN
      GOTO L720;
L711:  GOTO L9103;
L720:  IF G(2) = 2 THEN
      GOTO L690;

```

```

L730:  CALL CONCAT(ADDR(PD),ADDR(XD(XPX+1)),ADDR(TMP2));
       CALL SCOPY(ADDR(TMP2),ADDR(XD) );
       GOTO ARGRET;

/*-----*/
/*
/*      ERROR
/*
/*-----*/
L9001: ERNBR=IERR;
L9000: CALL ERRMOD(ERNBR,ZD,1);
       GOTO L1;
L9002: CALL ERRMOD(BINARY(29),ZD,1); /*  CTRL-C FOR FORTRAN */
       GOTO L1;
L9103: ERNBR=3;
       GOTO L9000;
L9105: ERNBR=5;
       GOTO L9000;
L9106: ERNBR=6;
       GOTO L9000;
L9108: ERNBR=8;
       GOTO L9000;
L9113: ERNBR=13;
       GOTO L9000;
L9114: ERNBR=14;
       GOTO L9000;
L9117: ERNBR=17;
       GOTO L9000;
L9118: ERNBR=18;
       GOTO L9000;
L9119: ERNBR=19;
       GOTO L9000;
L9121: ERNBR=21;
       GOTO L9000;
L9123: ERNBR=23;
       GOTO L9000;
L9126: ERNBR=26;
       GOTO L9000;
L9128: ERNBR=28;
       GOTO L9000;
L9131: GOTO L9000;

```

PARSE: PROCEDURE;

```

/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                        */
/*      KEVIN C. O'KANE                             */
/*      DEPARTMENT OF COMPUTER SCIENCE              */
/*      THE UNIVERSITY OF TENNESSEE                  */
/*      KNOXVILLE, TENNESSEE 37916                 */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                  */
/*      JANUARY 1981                                */
/*      IBM PLI VERSION                              */
/*-----*/
/*-----*/
/*      MAIN EXPRESSION HANDLER                      */
/*      IF T2 IS SET, A COMMA N XD WILL CAUSE A RETURN TO CALLER */
/*      WITH ANSWER ON STOP OF STACK (MULTI ARGUMENT SWITCH).    */
/*      XPX WILL POINT TO THE COMMA.                 */
/*      0=(;1=/;2=*;3=-;4=+;5=);6=(;7=_;8==;9=>;11=<;12=' */
/*      13=\;14=#;15=[;16=];17=?;18='';19='>;20='<;21='[';22=']';23='?' */
/*      24=&;25='!';26='&;27='! */
/*      CODE 251 MEANS PROCESSED OPEN PAREN IN FCN/ARRAY REF      */
/*      CODE 252 MEANS PROCESSED CLOSE PAREN IN FCN/ARRAY REF    */
/*      CODE 253 MEANS PROCESSED COMMA IN FCN/ARRAY REF          */
/*      CODE 254 MEANS PROCESSED COLON IN $$ FCN                  */
/*      CODE 255 MEANS PROCESSED QUOTE (") IN PATTERN MATCH      */
/*-----*/
/*      LOGICAL*1 COD251(2) | PROCESSED OPEN                */
/*      LOGICAL*1 COD252(2) | PROCESSED CLOSE                */
/*      LOGICAL*1 COD253(2) | PROCESSED COMMA                */
/*      LOGICAL*1 COD254(2) | PROCESSED COLON                */
/*      LOGICAL*1 COD255(2) | PROCESSED QUOTE                */
/*      LOGICAL*1 XTAB(2)  | TAB CHARACTER                   */
/*-----*/
/*      FILES                                              */
/*-----*/
/*      GLOBAL COMMUNICATION AREA                        */
/*-----*/
/*      STACK                                              */
/*-----*/
/*      LOGICAL*1 STKTYP(10) | DATA TYPE OF STACK ELEMENT    */
/*      LOGICAL*1 S1D(4000)  | STRING VALUE OF STACK ELEMENT  */
/*      EQUIVALENCE (S1D,PD1) | OVERLAY - STACK IS ABOVE PGM AREA */
/*      REAL RSTK(10)        | REAL VALUE STACK              */
/*      INTEGER SDLIM         | START OF TOP STRING           */
/*      LOGICAL*1 S1P(10)     | STACK ELEMENT TYPE CODE       */
/*      INTEGER SPX           | EXTENT OF STACK               */
/*-----*/

```

```

/*      PARAMETERS TO SUBROUTINE PATRN      */
/*      PARAMETERS TO SUBROUTINE PARSE      */
/*      COMMON UTILITY ARRAYS               */
/*      SYSTEM COMMON AREA                  */
/*      LOGICAL*1 XXX8(282),XX9(6) |PLACE HOLDERS      */
/*-----*/
/*      INTERMEDIATE CHARACTER AND NUMERIC RESULT REGISTERS      */
/*-----*/
/*      REAL AA          |LEFT ARGUMENT      */
/*      REAL BB          |RIGHT ARGUMENT     */
/*      REAL RSLT        |RESULT ARGUMENT    */
/*      LOGICAL*1 BBTYP,ADTYP |DATA TYPE FLAGS      */
/*      INTEGER IORG     |ORIGIN CURRENT ROUTINE      */
/*-----*/
/*      MASTER CHARACTER SET STRING      */
/*-----*/
/*      LOGICAL*1 NXTCHR     |NEXT CHARACTER      */
/*-----*/
/*      INITIALIZATION      */
/*          SET STACK EMPTY      */
/*          SET RETURN STACK EMPTY      */
/*-----*/

      DECLARE 1 CODESCO EXTERNAL,
              2 DUMITEM FLOAT BINARY(53),
              2 COD251( 2) CHAR(1),
              2 COD252( 2) CHAR(1),
              2 COD253( 2) CHAR(1),
              2 COD254( 2) CHAR(1),
              2 COD255( 2) CHAR(1),
              2 XTAB( 2) CHAR(1),
              2 DUMITEM4(2) FIXED BINARY(31);

      DECLARE 1 IOCOMMO EXTERNAL,
              2 DUMITEM FLOAT BINARY(53),
              2 ERR FIXED BINARY(31),
              2 G55 FIXED BINARY(31),
              2 G54 FIXED BINARY(31),
              2 D2 CHAR(1),
              2 F CHAR(1);

      DECLARE 1 PTRNCOM EXTERNAL,
              2 DUMITEM FLOAT BINARY(53),
              2 BD(256) CHAR(1),
              2 IERR1 FIXED BINARY(31);

      DECLARE 1 PRSECOM EXTERNAL,
              2 DUMITEM FLOAT BINARY(53),
              2 IORG FIXED BINARY(31),
              2 XPX FIXED BINARY(31),
              2 XPY FIXED BINARY(31),
              2 TOPX FIXED BINARY(31),
              2 T2 CHAR(1),
              2 V1D(257) CHAR(1),
              2 XD(256) CHAR(1),
              2 XDSAVE(256) CHAR(1),

```

```

      2 IERR FIXED BINARY(31),
      2 DTYPE CHAR(1),
      2 DUMITEM2( 1)FLOAT BINARY;
DECLARE TMP1( 2) CHAR(1),
      TMP2(256) CHAR(1);
DECLARE AC(2) CHAR(1) DEF AB,
      AB FIXED BIN(15,0);
DECLARE 1 UNLCM EXTERNAL,
      2 DUMITEM FLOAT BINARY(53),
      2 SYMLen FIXED BINARY(31),
      2 SYMADR FIXED BINARY(31),
      2 SYMFLG FIXED BINARY(15,0),
      2 GD(257) CHAR(1),
      2 G(15) FIXED BINARY(31),
      2 P FLOAT BINARY,
      2 PD(257) CHAR(1),
      2 VD(256) CHAR(1),
      2 DOSAVE(21) CHAR(1),
      2 XXX8(5) CHAR(1),
      2 DATRIB CHAR(1),
      2 DVAR FIXED BINARY(31),
      2 XX9(6) CHAR(1),
      2 PD1LEN FIXED BINARY(31),
      2 PD1CUR FIXED BINARY(31),
      2 TRACE FIXED BINARY(31),
      2 ZD(256) CHAR(1),
      2 PD1(4000) CHAR(1),
      2 FORV(5,10) CHAR(1),
      2 FORIT(5) FIXED BINARY(1),
      2 FORLT(5) FIXED BINARY(1),
      2 FTMP(256) CHAR(1),
      2 FORIN(5) FLOAT BINARY,
      2 FORLV(5) FLOAT BINARY,
      2 FORNXT(5) FIXED BINARY(31),
      2 FORX FIXED BINARY(31),
      2 GOSUB(6,5) FIXED BINARY(31),
      2 GSBX FIXED BINARY(31);
DECLARE IRETRN LABEL STATIC,
      WHOOPS FLOAT BINARY STATIC,
      STKTYP( 10) FIXED BIN(15,0) STATIC,
      NXTCHR(2) CHAR(1) STATIC,
      LHSSW FIXED BINARY(31) STATIC,
      ITYPE CHAR(1),
      ADTYP FIXED BIN(15,0) STATIC,
      BBTYP FIXED BIN(15,0) STATIC,
      SDLIM FIXED BINARY(31) STATIC,
      ERNBR FIXED BINARY(31) STATIC,
      OPEN FLOAT BINARY STATIC,
      FREE FLOAT BINARY STATIC,
      SDLI FLOAT BINARY STATIC;
DECLARE BBEQ( 4) CHAR(1) DEF BB,
      ADEQ( 4) CHAR(1) DEF AA,

```

```

        CSTR( 22) CHARACTER(1)
STATIC INITIAL('"'',':',251,'','&','!','(','+', '-','*', '/', '_'
, '=' , '>' , '<' , '\' , '#' , '[' , ']' , '?' , '!' , '!' , '!' , '!' )' ) ,
        CODE( 8) FIXED BIN(15,0) STATIC INITIAL(
        18,19,20,21,22,23,26,27),
        RSTK( 10) FLOAT BINARY STATIC,
        RSLT FLOAT BINARY STATIC,
        XDEQ FLOAT BIN DEF XD,
        T1PX FIXED BINARY(31) STATIC,
        T2PX FIXED BINARY(31) STATIC;
DECLARE RETZ LABEL;
DECLARE RETX FIXED BINARY(31) STATIC,
        VAL FLOAT BINARY STATIC,
        JPX FIXED BINARY(31) STATIC,
        ADX FLOAT BINARY STATIC,
        S1D( 4000) CHAR(1),
        S1P( 10) FIXED BIN(15,0) STATIC,
        SPX FIXED BINARY(31) STATIC,
        RET( 10) LABEL STATIC,
        IB FIXED BINARY(31) STATIC,
        $N FLOAT BINARY STATIC,
        BRANCH03(22)LABEL INITIAL(L1280,
        L921,L940,L960,L1000,L1000,L1000,
        L1000,L1000,L1000,L1000,L1000,
        L1000,L1000,L1000,L1000,L1000,
        L1000,L1000,L1000,L1000,L1100),
        BRANCH02( 27)LABEL INITIAL(L2061,L2061,L2061,
        L2061,L2070,L2070,L2063,L2062,
        L2061,L2070,L2061,L2070,L2061,
        L2061,L2063,L2063,L2063,L2063,
        L2061,L2061,L2063,L2063,L2063,
        L2061,L2061,L2061,L2061),
        BRANCH01(27)LABEL INITIAL(L2080,L2090,L2100,
        L2110,L2070,L2070,L2120,L2130,
        L2150,L2070,L2200,L2070,L2500,
        L2502,L2504,L2505,L2506,L2130,
        L2516,L2517,L2518,L2519,L2520,
        L2523,L2525,L2527,L2529),
        AA FLOAT BINARY;
DECLARE BB FLOAT BINARY,
        K FIXED BINARY(31) STATIC,
        V FLOAT BINARY STATIC,
        J FIXED BINARY(31) STATIC,
        I FIXED BINARY(31) STATIC;
DECLARE TEST FIXED BINARY(31) STATIC;
CSTR(3)=COD251(1);
AB=0;
SPX=0 ;
DTYPE='2' ;
RETX=0 ;
NXTCHR(2)=AC(2);
TMP1(2)=AC(2);

```

```

T2PX=0;
S1D(PD1LEN+2)=AC(2) ;
S1D(PD1LEN+3)=AC(2) ;
SDLIM=PD1LEN+3 ;

/*-----*/
/*
/*      MAIN RECURSIVE INTERNAL ENTRY POINT
/*      PUSH STACK FOUNDATION CODE ONTO STACK
/*
/*-----*/
L900:  SPX=SPX+1 ;
      S1P(SPX)=10 ;
      STKTYP(SPX)=2; /*CHARACTERTYPE */

/*-----*/
/*
/*      INCREMENT TEXT POINTER AND TEST FOR END
/*
/*-----*/
L920:
      XPX=XPX+1 ;
      TEST=LEN(ADDR(XD));
      IF XPX > (LEN(ADDR(XD))) THEN
      GOTO L2430;
      NXTCHR(1)=XD(XPX) ;
      IF NXTCHR(1) >= 'A' & NXTCHR(1)<= 'I' THEN GOTO L1280;
      IF NXTCHR(1) >= 'J' & NXTCHR(1)<= 'R' THEN GOTO L1280;
      IF NXTCHR(1) >= 'S' & NXTCHR(1)<= 'Z' THEN GOTO L1280;
      IF NXTCHR(1) >= '0' & NXTCHR(1)<= '9' THEN GOTO L1280;
      DO I=1 TO 22 ;
      IF CSTR(I) = NXTCHR(1) THEN
      IF I<=22 & I > 0 THEN GOTO BRANCH03(I);
L919:  END;
      GOTO L1280;

/*-----*/
/*
/*      TEST FOR COLON AND IF STACK ELEMENT UNDER TOP INDICATES
/*      THE $SELECT FUNCTION - IF SO, CONCATENATE STACK TOP
/*      TO ELEMENT BENEATH STACK TOP SEPARATED BY COD254 (:)
/*      FREE STACK TOP
/*
/*-----*/
L921:  I=SDLIM ;
      IF STKTYP(SPX) = 1 THEN
      GOTO L926;
L923:  I=I-1 ;
      IF S1D(I-1) ^= AC(2) THEN
      GOTO L923;
L926:  IF S1D(I) ^= '$' | S1D(I+1) ^= 'S' THEN GOTO L2430;
      IF STKTYP(SPX) = 2 THEN GOTO L924;
      I=LEN(ADDR(S1D(SDLIM))) ;
      CALL CVC(ADDR(S1D(SDLIM+I)),ADDR(RSTK(SPX))) ;
      GOTO L925;

```

```

L924: CALL DSHIFT(ADDR(S1D),SDLIM) ;
L925: CALL CONCAT(ADDR(S1D(SDLIM)),ADDR(COD254),ADDR(S1D(SDLIM))) ;
      SPX=SPX-1 ;
      GOTO L920;

/*-----*/
/*
/* CHECK FOR CODED OPEN PAREN - INDICATES AN INTERNAL CALL
/* PUSH FCN/ARRAY OPEN PAREN MARKER ONTO STACK; CONCATENATE
/* INITIAL PORTION OF ARRAY DEFINITION STRING, CODED OPEN
/* PAREN MARKER ONTO STACK
/*
/*-----*/
L940: SPX=SPX+1 ;
      I=LEN(ADDR(S1D(SDLIM))) ;
      SDLIM=SDLIM+I+1 ;
      IF SDLIM >= SYMLen THEN
      GOTO L3;
      CALL CONCAT(ADDR(V1D),ADDR(COD251),ADDR(S1D(SDLIM))) ;
      STKTYP(SPX)=2 ;
      S1P(SPX)=6; /*FCN/ARRAYCODE */
      GOTO L920;

/*-----*/
/*
/* CHECK FOR COMMA
/* CHECK IF FCN/ARRAY MARKER ON STACK ELEMENT UNDER TOP
/* IF SO, CONCATENATE COMMA AND STACK TOP ONTO STACK
/* ELEMENT UNDER TOP.
/* IF T2 SWITCH SET (MULTI-ARG SWITCH) EXIT
/*
/*-----*/
L960: IF S1P(SPX-1) = 6 THEN
      GOTO L980;
      IF T2 THEN GOTO L2430;
      ERNBR=16 ;
      GOTO L1;

/*-----*/
/*
/* CONCATENATE STACK TOP AND CODED COMMA ONTO ELEMENT UNDER TOP
/* IN FCN/ARRAY DEFINITION
/*
/*-----*/
L980: IF STKTYP(SPX) = 2 THEN
      GOTO L981;
      I=LEN(ADDR(S1D(SDLIM))) ;
      CALL CVC(ADDR(S1D(SDLIM+I)),ADDR(RSTK(SPX))) ;
      GOTO L982;
L981: CALL DSHIFT(ADDR(S1D),SDLIM) ;
L982: CALL CONCAT(ADDR(S1D(SDLIM)),ADDR(COD253),ADDR(S1D(SDLIM))) ;
      SPX=SPX-1 ;
      GOTO L920;

/*-----*/
/*

```

```

/*      CHECK FOR DELIMITERS IN LIST      */
/*      IF DELIMITER, ALLOCATE STACK AND PLACE DELIMITER CODE IN      */
/*      STACK PRECEDENCE OPEN PARENS HANDLED HERE      */
/*-----*/
L1000: SPX=SPX+1 ;
      IF NXTCHR(1)  = ' ' THEN GOTO L1040;
      TMP1(1)=XD(XPX+1) ;
      I=INDEX(ZPAD('=><[]?&!'),ADDR(TMP1),1);
      IF I  = 0 THEN  GOTO L1021;
      ELSE GOTO L1040;
L1021: XPX=XPX+1 ;
      S1P(SPX)=CODE(I) ;
      GOTO L920;
/*      INSERT CODE*/
L1040: S1P(SPX)=INDEX(ZPAD('(*-+ _=> <'\'# ?  []  &!'),
      ADDR(NXTCHR(1)),1)-1;
      IF S1P(SPX) >= 0 THEN GOTO L920;
      ERNBR=12 ;
      GOTO L1;

/*-----*/
/*      */
/*      CHECK FOR CLOSE      */
/*      STACK ELEMENT BENEATH TOP MUST CONTAIN OPEN PAREN MARK      */
/*      IF PRECEDENCE OPEN PAREN BENEATH, EXTRACT VALUE      */
/*      IF FCN/ARRAY OPEN PAREN, COPY RESULT AND RETURN      */
/*-----*/
L1100: IF S1P(SPX-1) = 0 THEN GOTO L1180;
      IF S1P(SPX-1) = 6 THEN GOTO L1150;
      ERNBR=16 ;
      GOTO L1;

/*-----*/
/*      */
/*      FCN/ARRAY RETURN SECTION      */
/*-----*/
L1150: IF STKTYP(SPX) = 2 THEN
      GOTO L1151;
      I=LEN(ADDR(S1D(SDLIM))) ;
      CALL CVC(ADDR(S1D(SDLIM+I)),ADDR(RSTK(SPX))) ;
      GOTO L1152;
L1151: CALL DSHIFT(ADDR(S1D),SDLIM) ;
L1152: CALL CONCAT(ADDR(S1D(SDLIM)),ADDR(COD252),ADDR(S1D(SDLIM))) ;
L1160: SPX=SPX-1 ;
      RETZ=RET(RETZ) ;
      IRETRN=RETZ;
      RETX=RETX-1 ;
      GOTO IRETRN;

/*-----*/
/*      */
/*      PRECEDENCE CLOSE PAREN      */

```

```

/*      ITEM ON STACK TOP MUST BE VALUE (CODE=5)      */
/*      */
/*-----*/
L1180: IF S1P(SPX) = 5 THEN
      GOTO L1220;
      ERNBR=12 ;
      GOTO L1;
/*      EXTRACT VALUE FROM STACK TOP*/
L1220: IF STKTYP(SPX) = 2 THEN
      GOTO L1221;
      BB=RSTK(SPX) ;
      BBTYP=1 ;
      GOTO L1240;
L1221: CALL SCOPY(ADDR(S1D(SDLIM)),ADDR(BD) ) ;
L1222: SDLIM=SDLIM-1 ;
      IF S1D(SDLIM-1) -= AC(2) THEN
      GOTO L1222;
      BBTYP=2 ;
L1240: SPX=SPX-2; /*ERASE TOP2: VAL&OPEN */
      GOTO L2280;
/*-----*/
/*      */
/*      SCAN FOR OPERAND      */
/*      CHECK FOR PATTERN OPERATION AND DECODE RIGHT ARG SEPARATELY */
/*      */
/*-----*/
/*-----*/
/*      */
/*      PATTERN CASE      */
/*      */
/*-----*/
L1280: BBTYP=2 ;
      BD(1)=AC(2) ;
      CALL PATRN1(BD,XD,JPX,ERNBR,XPX) ;
      IF ERNBR -= 0 THEN GOTO L1;
      ELSE GOTO L1820;
/*-----*/
/*      */
/*      BRANCH IF NUMERIC      */
/*      */
/*-----*/
L1289: IF (XD(XPX) >= '0' & XD(XPX) <= '9') | XD(XPX) = '.' THEN
      GOTO L1740; /*NUMERIC*/
      BBTYP=2; /*CHARACTER */
/*-----*/
/*      */
/*      CHECK FOR QUOTED LITERAL      */
/*      */
/*-----*/
      IF XD(XPX) -= ''' THEN GOTO L1371;
/*      QUOTED STRING SCAN*/
      I=LEN(ADDR(XD)) ;

```

```

        JPX=0 ;
L1320: XPX=XPX+1 ;
        IF XPX <= I THEN
            GOTO L1340;
        ERNBR=2 ;
        GOTO L1;
L1340: IF XD(XPX) = '' THEN GOTO L1341;
        JPX=JPX+1 ;
        BD(JPX)=XD(XPX) ;
        GOTO L1320;
L1341: IF XD(XPX+1) != '' THEN GOTO L1342;
        JPX=JPX+1 ;
        BD(JPX)='';
        XPX=XPX+1 ;
        GOTO L1320;
L1342: JPX=JPX+1 ;
        BD(JPX)=AC(2) ;
L1360: GOTO L1820;
/*-----*/
/*                                                    */
/*      SCAN TO DELIMITER FOR VARIABLE NAME          */
/*                                                    */
/*-----*/
L1370: XPX=XPX+1 ;
L1371: IF XPX > LEN(ADDR(XD)) THEN
    GOTO L1390;
    TMP1(1)=XD(XPX) ;
    TMP1(2)=AC(2);
    IF INDEX(ZPAD(',-/*_)=><:[ ]?\'#\&!'),
        ADDR(TMP1),1) != 0 THEN GOTO L1390;
    IF XD(XPX) = '-' & XPX > JPX+1 THEN GOTO L1390;
    IF XD(XPX) = '-' & XPX = JPX+1 & XD(JPX) != '@' THEN
        GOTO L1390;
L1380: IF XD(XPX) = '(' THEN GOTO L1420;
        ELSE GOTO L1370;
/*-----*/
/*                                                    */
/*      EXTRACT VARIABLE NAME - CONTAINS NO PARENS    */
/*                                                    */
/*-----*/
L1390: CALL SUBSTR(ADDR(XD),ADDR(V1D),JPX,XPX-JPX) ;
        XPX=XPX-1 ;
        GOTO L1460;
/*-----*/
/*                                                    */
/*      PROCESS OPEN PAREN IN FCN/ARRAY REFERENCE    */
/*      CHANGE ( TO CODE 251                          */
/*      COPY STRING TO V1D (HEADER OF FCN/ARRAY REF)  */
/*                                                    */
/*-----*/
L1420: XD(XPX)=COD251(1) ;
L1430: CALL SUBSTR(ADDR(XD),ADDR(V1D),JPX,XPX-JPX) ;

```

```

        XPX=XPX-1 ;
/*      CALL EXPRESSION HANDLER*/
L1440: RETX=RETX+1 ;
        RETZ=L1450;
        RET(RETX)=RETZ ;
        GOTO L900;
/*      COPY ANSWER FROM STACK*/
/*      ANSWER CONTAINS PROCESSED ARGUMENTS FOR ARRAY OR FCN REF.*/
L1450: CALL SCOPY(ADDR(S1D(SDLIM)),ADDR(V1D) ) ;
L1451: SDLIM=SDLIM-1 ;
        IF S1D(SDLIM-1) ^= AC(2) THEN
            GOTO L1451;
        SPX=SPX-2 ;

/*-----*/
/*
/*      V1D NOW CONTAINS VARIABLE NAME, FCN OR ARRAY REF
/*      INDIRECTION AND GLOBALS STILL PRESENT
/*      FCN AND ARRAY REFS HAVE THEIR ARGUMENTS TRANSLATED TO TEXT
/*      TOPX IS SWITCH TO INDICATE THIS IS A LEFT HAND SIDE CALL
/*      STACK MUST HAVE ONLY ONE ELEMENT AT THIS POINT
/*      RETURN TO CALLER (SET AND READ)
/*
/*-----*/
L1460: IF TOPX ^= 1 | SPX ^= 1 THEN
        GOTO L1480;
        SPX=SPX-1 ;
        TOPX=0; /*LHSSW */
        RETURN;

/*-----*/
/*
/*      CHECK FOR GLOBAL VARIABLE
/*
/*-----*/
L1480: IF V1D(1) ^= '-' THEN GOTO L1570;
L1490: CALL SCOPY(ADDR(V1D),ADDR(GD) ) ;
        CALL GBLREF( '1'B ) ;
        DATRIB=0 ;
        IF F = 0 THEN
            GOTO L1492;
        G(3)=0 ;
        IF S1D(SDLIM) ^= '$' | T1PX = 1 THEN GOTO L1491;
        IF S1D(SDLIM+1) = 'N' THEN G(3)=2;
        IF S1D(SDLIM+1) = 'D' THEN G(3)=4;
/*      CALL GLOBAL FILE HANDLER*/
L1491: CALL GLOBAL;
        TMP1(2)=AC(2); /*GETSMESSEDUPINGLOBAL */
/*      CHECK FOR $DATA FUNCTION*/
L1492: IF S1D(SDLIM) ^= '$' | T1PX = '1'
        | S1D(SDLIM+1) ^= 'D' THEN GOTO L1510;
        BBTYP=1 ;
        BB=DATRIB; /*$DATA CODE RETURNED IN DATR IB */
        GOTO L1820;

```

```

/*      NORMAL GLOBAL*/
L1510: IF F  = 0 THEN
      GOTO L1520;
L1511: ERNBR=3 ;
      GOTO L1;

/*-----*/
/*
/*      CHECK INDIRECTION SWITCH T1PX
/*      IF INDIRECTION, SUBSTITUTE VALUE OF GLOBAL
/*
/*-----*/
L1520: IF T1PX  = 1 THEN
      GOTO L1550;
      IF G(2) = 2 THEN
      GOTO L1540;
      ERNBR=16 ;
      GOTO L1;
L1540: CALL SUBSTR(ADDR(XD),ADDR(TMP2),XPX+1,-1); /*RESIDUE OF LINE */
      CALL CONCAT(ADDR(PD),ADDR(TMP2),ADDR(XD));
          /*INSERT PD AT FRONT OF LINE */
      XPX=0 ;
      T1PX=0 ;
      GOTO L920;

/*-----*/
/*
/*      EXTRACT GLOBAL VALUE
/*
/*-----*/
L1550: IF G(2)  = 2 THEN
      GOTO L1560;
      CALL SCOPY(ADDR(PD),ADDR(BD) ) ;
      BBTYP=2 ;
      GOTO L1820;
L1560: IF G(2)  = 1 THEN
      GOTO L1511;
      BB=P ;
      BBTYP=1 ;
      GOTO L1820;

/*-----*/
/*
/*      BUILT-IN VARIABLES
/*
/*-----*/
L1570: IF V1D(1)  = '$' THEN GOTO L1580;
      IF V1D(2) = 'Z' THEN GOTO L1571;
      IF V1D(2) = 'R' THEN GOTO L1571;
      IF INDEX(ADDR(V1D),ADDR(COD251),1)  = 0 THEN
      GOTO L1626;
      TMP1(1)=V1D(2) ;
      I=INDEX(ZPAD('ZHTJSXYI'),ADDR(TMP1),1);
      IF I  = 0 THEN
      GOTO L1626;

```

```

L1571: IF ISCOMP(ADDR(V1D),ZPAD('$ZN')) .NE. 0 THEN GOTO L1572;
      CALL GBLREF(BINARY(2)) ;
      CALL SCOPY(ADDR(GD),ADDR(BD)) ;
      BBTYP=2 ;
      GOTO L1820;
L1572: CALL SVAR(IERR,V1D,BD,PD1LEN,SYMLEN) ;
      IF IERR .NE. 0 THEN
        GOTO L2;
      BBTYP=2 ;
      GOTO L1820;

/*-----*/
/*                                          */
/*      CHECK FOR FUNCTION                  */
/*                                          */
/*-----*/
L1626: IF V1D(2) .NE. 'D' & V1D(2) .NE. 'N' THEN GOTO L1627;
      I=INDEX(ADDR(V1D),ADDR(COD251),1) ;
      V1D(LEN(ADDR(V1D)))=AC(2) ;
      CALL SCOPY(ADDR(V1D(I+1)),ADDR(BD) ) ;
      BBTYP=2 ;
      GOTO L1820;
L1627: CALL FCN(V1D,BD,IERR,PD1,PD1LEN) ;
      BBTYP=2 ;
      IF IERR = 0 THEN GOTO L1820;
      ELSE GOTO L2;

/*-----*/
/*                                          */
/*      CHECK FOR INDIRECTION              */
/*      T1PX IS INDIRECTION SWITCH         */
/*      WHEN T1PX IS SET THE VALUE FOR A VARIABLE OR GLOBAL */
/*      WILL BE SUBSTITUTED INTO THE LINE. */
/*                                          */
/*-----*/
L1580: T1PX=0 ;
      IF V1D(1) .NE. '@' THEN GOTO L1610;
L1590: CALL SUBSTR(ADDR(V1D),ADDR(V1D),2,-1) ;
      T1PX=1 ;
      T2PX=1;
L1600: IF V1D(1) = '-' THEN GOTO L1490;

/*-----*/
/*                                          */
/*      SYMBOL TABLE LOOK-UP - $NEXT      */
/*                                          */
/*-----*/
L1610: IF S1D(SDLIM) .NE. '$' | S1D(SDLIM+1) .NE. 'N' THEN GOTO L1632;
      SYMFLG=3; /*$N */
      CALL SYM(ADDR(V1D),ADDR(BD),ITYPE) ;
      BBTYP=2 ;
      IF SYMFLG .NE. 0 THEN
        GOTO L1820;
      ERNBR=27 ;
      GOTO L1;

```

```

/*-----*/
/*
/*      RETRIEVE LOOK-UP
/*
/*-----*/
L1632: SYMFLG=1 ;
      CALL SYM(ADDR(V1D),ZPAD(''),ITYPE) ;
      IF S1D(SDLIM) ^= '$' THEN GOTO L1613; /*NOT FCN EXCEPTION*/
/*-----*/
/*
/*      CHECK FOR $NEXT
/*
/*-----*/
      IF S1D(SDLIM+1) ^= 'D' THEN GOTO L1611;
      BBTYP=1 ;
      BB=0.0E0 ;
      IF SYMFLG ^= 0 THEN
      BB=1.0E0 ;
      GOTO L1820;
/*-----*/
/*
/*      CHECK FOR $TEXT
/*
/*-----*/
L1611: IF S1D(SDLIM+1) ^= 'T' THEN GOTO L1613;
L1631: J=1 ;
L5312: IF J >= PD1LEN THEN
      GOTO L5331;
      I=INDEX(ADDR(PD1(J)),ADDR(XTAB),1) ;
      IF I > 1 THEN
      GOTO L5311;
L5330: J=J+LEN(ADDR(PD1(J)))+1 ;
      GOTO L5312;
L5311: CALL SUBSTR(ADDR(PD1(J)),ADDR(TMP3),1,I-1) ;
      IF ISCOMP(ADDR(V1D),ADDR(TMP3)) ^= 0 THEN
      GOTO L5330;
      GOTO L1615;
L5331: K=-32000; /*DOESNOTEXIST */
      GOTO L1617;
/*      FIND LINE NUMBER*/
L1615: I=J ;
      J=1 ;
      K=1 ;
L1618: IF J = I THEN
      GOTO L1617;
      J=J+LEN(ADDR(PD1(J)))+1 ;
      IF J = 0 | J > PD1LEN THEN
      GOTO L5331;
      K=K+1; /*LINECOUNT */
      GOTO L1618;
L1617: RSLT=K ;
      CALL CVC(ADDR(BD),ADDR(RSLT)) ;

```

```

        BBTYP=2 ;
        GOTO L1820;
/*-----*/
/*                                          */
/*      CHECK FOR VARIABLE NOT FOUND      */
/*                                          */
/*-----*/
L1613: IF SYMFLG  ^= 0 THEN
        GOTO L1690;
        ERNBR=17 ;
        GOTO L1;
/*-----*/
/*                                          */
/*      CHECK FOR INDIRECTION SWITCH      */
/*                                          */
/*-----*/
L1690: IF T1PX  ^= 1 THEN
        GOTO L1700;
        T1PX=0 ;
        CALL SUBSTR(ADDR(XD),ADDR(TMP2),XPX+1,-1) ;
        IF ITYPE = '2' THEN
            GOTO L1691;
        CALL CVC(ADDR(TMP3),ADDR(PD1(SYMADR))) ;
        CALL CONCAT(ADDR(TMP3),ADDR(TMP2),ADDR(XD)) ;
        XPX=0 ;
        GOTO L920;
L1691: CALL CONCAT(ADDR(PD1(SYMADR)),ADDR(TMP2),ADDR(XD)) ;
        XPX=0 ;
        GOTO L920;
/*-----*/
/*      EXTRACT VALUE FROM SYMBOL TABLE  */
/*-----*/
L1700: T1PX=0 ;
        IF ITYPE = '1' THEN
            GOTO L1701;
        CALL SCOPY(ADDR(PD1(SYMADR)),ADDR(BD) ) ;
        BBTYP=2 ;
        GOTO L1820;
L1701: BBEQ(1)=PD1(SYMADR) ;
        BBEQ(2)=PD1(SYMADR+1) ;
        BBEQ(3)=PD1(SYMADR+2) ;
        BBEQ(4)=PD1(SYMADR+3) ;
        BBTYP=1 ;
        GOTO L1820;
/*-----*/
/*                                          */
/*      PROCESS NUMERIC OPERAND           */
/*                                          */
/*-----*/
L1721: XPX=XPX+1 ;
        IF XPX > LEN(ADDR(XD)) THEN
            GOTO L1780;

```

```

L1740: IF (XD(XPX) >= '0' & XD(XPX) <= '9') |
        XD(XPX) = '.' THEN GOTO L1721;
/*      CHECK FOR VALID NUMBER*/
L1780: IF XPX-JPX >= 0 THEN
        GOTO L1800;
        ERNBR=10 ;
        GOTO L1;
/*      EXTRACT VALUE - CONVERT FROM CHAR,BIN*/
L1800: BBTYP=1 ;
        TMP1(1)=XD(XPX) ;
        XD(XPX)=AC(2) ;
        CALL CVB(ADDR(XD(JPX)),ADDR(BB)) ;
        XD(XPX)=TMP1(1) ;
        XPX=XPX-1 ;

/*-----*/
/*
/*      PROCESS VALUE IN BB OR BD
/*      PUSH ONTO STACK IF STACK TOP IS OPEN OR EMPTY
/*
/*-----*/
L1820: IF S1P(SPX) <= 10 & S1P(SPX) >= 0 & S1P(SPX) <= 6 & T2PX <= 1
        THEN GOTO L1970;
        T2PX=0;
        SPX=SPX+1 ;
        STKTYP(SPX)=BBTYP ;
        IF BBTYP = 1 THEN
                GOTO L1901;
        I=LEN(ADDR(S1D(SDLIM))) ;
        SDLIM=SDLIM+I+1 ;
        IF SDLIM >= SYMLEN THEN
                GOTO L3;
        CALL SCOPY(ADDR(BD),ADDR(S1D(SDLIM)) ) ;
        GOTO L1920;
L1901: RSTK(SPX)=BB ;
L1920: S1P(SPX)=5; /*CODE FOR VALUE */
        GOTO L920;

/*-----*/
/*
/*      PROCESS EXPRESSION - CHECK FOR OPERATOR ON TOP OF STACK
/*
/*-----*/
L1970: IF S1P(SPX) > 0 & S1P(SPX) < 5 THEN
        GOTO L2020;
        IF S1P(SPX) >= 6 THEN
                GOTO L2020;
L2000: ERNBR=11 ;
        GOTO L1;

/*-----*/
/*
/*      CHECK FOR NUMBER UNDER OPERATOR
/*      IF NOT, CHECK FOR UNARY OPERATOR
/*
/*-----*/

```

```

/*-----*/
L2020: IF S1P(SPX-1) = 5 THEN
      GOTO L2040;
      IF S1P(SPX) = 3 THEN
      GOTO L2021;
      IF S1P(SPX) = 4 THEN
      GOTO L2025;
      IF S1P(SPX) = 12 THEN
      GOTO L2023;
      ERNBR=1 ;
      GOTO L1;

/*-----*/
/*                                           */
/*      UNARY MINUS                          */
/*                                           */
/*-----*/
L2021: IF BBTYP = 2 THEN
      CALL CVB(ADDR(BD),ADDR(BB)) ;
      BB=-BB ;
      BBTYP=1 ;
      GOTO L2022;

/*-----*/
/*                                           */
/*      NOT                                  */
/*                                           */
/*-----*/
L2023: IF BBTYP = 2 THEN
      CALL CVB(ADDR(BD),ADDR(BB)) ;
      BBTYP=1 ;
      IF BB ^= 0 THEN
      GOTO L2024;
      BB=1 ;
      GOTO L2022;
L2024: BB=0 ;
      GOTO L2022;

/*-----*/
/*                                           */
/*      PLUS                                */
/*                                           */
/*-----*/
L2025: IF BBTYP = 2 THEN
      CALL CVB(ADDR(BD),ADDR(BB)) ;
L2022: SPX=SPX-1; /*FREE */
      BB=BB ;
      BBTYP=1 ;
      GOTO L2280;

/*-----*/
/*                                           */
/*      EXTRACT VALUE UNDER OPERATOR        */
/*                                           */
/*-----*/
L2040: IF STKTYP(SPX-1) = 1 THEN

```

```

        GOTO L2041;
        ADX=SDLIM ;
/*      CALL SCOPY (S1D(SDLIM),AD)*/
L2042:  SDLIM=SDLIM-1 ;
        IF S1D(SDLIM-1) -= AC(2) THEN
            GOTO L2042;
        ADTYP=2 ;
        GOTO L2060;
L2041:  ADTYP=1 ;
        AA=RSTK(SPX-1) ;
/*-----*/
/*                                           */
/*      BRANCH DEPENDING UPON OPERATOR      */
/*                                           */
/*-----*/
L2060:  IF S1P(SPX)<=27 &S1P(SPX) > 0 THEN GOTO BRANCH02(S1P(SPX));
L2061:  IF BBTYP = 2 THEN
        CALL CVB(ADDR(BD),ADDR(BB)) ;
        IF ADTYP = 2 THEN
            CALL CVB(ADDR(S1D(ADX)),ADDR(AA)) ;
            BBTYP=1 ;
            ADTYP=1 ;
            GOTO L2062;
L2063:  IF BBTYP = 1 THEN
        CALL CVC(ADDR(BD),ADDR(BB)) ;
        IF ADTYP = 1 THEN
            CALL CVC(ADDR(S1D(ADX)),ADDR(AA)) ;
L2062:  IF S1P(SPX)<=27 &S1P(SPX) > 0 THEN GOTO BRANCH01(S1P(SPX));
L2070:  ERNBR=12 ;
        GOTO L1;
/*-----*/
/*                                           */
/*      DIVIDE OPERATION                      */
/*                                           */
/*-----*/
L2080:  IF BB -= 0 THEN
        GOTO L2081;
        ERNBR=15 ;
        GOTO L1;
L2081:  BB=AA/BB ;
L2082:  BBTYP=1 ;
        GOTO L2260;
/*-----*/
/*                                           */
/*      MULTIPLY OPERATION                   */
/*                                           */
/*-----*/
L2090:  BB=AA*BB ;
        GOTO L2082;
/*-----*/
/*                                           */
/*      SUBTRACT                            */
/*                                           */

```

```

/*                                                                    */
/*-----*/
L2100: BB=AA-BB ;
      GOTO L2082;
/*-----*/
/*                                                                    */
/*      ADDITION                                                         */
/*                                                                    */
/*-----*/
L2110: BB=AA+BB ;
      GOTO L2082;
/*-----*/
/*                                                                    */
/*      CONCATENATE                                                      */
/*                                                                    */
/*-----*/
L2120: BBTYP=2 ;
      CALL CONCAT(ADDR(S1D(ADX)),ADDR(BD),ADDR(TMP2)) ;
      CALL SCOPY(ADDR(TMP2),ADDR(BD) ) ;
      GOTO L2260;
/*-----*/
/*                                                                    */
/*      EQUALS RELATIONAL (8)                                           */
/*      NOT EQUALS (18)                                                */
/*                                                                    */
/*-----*/
L2130: IF DTYPE = '1' THEN
      GOTO L2140;
      IF BBTYP = 1 & ADTYP = 1 THEN
      GOTO L2140;
      IF BBTYP = 1 THEN
      CALL CVC(ADDR(BD),ADDR(BB)) ;
      BB=0.OEO ;
      IF ADTYP = 1 THEN
      CALL CVC(ADDR(S1D(ADX)),ADDR(AA)) ;
      IF S1P(SPX) = 18 THEN
      GOTO L3000;
      IF ISCOMP(ADDR(S1D(ADX)),ADDR(BD)) = 0 THEN
      BB=1.OEO ;
      GOTO L3001;
L3000: IF ISCOMP(ADDR(S1D(ADX)),ADDR(BD)) ^= 0 THEN
      BB=1.OEO ;
L3001: BBTYP=1 ;
      GOTO L2260;
/*-----*/
/*                                                                    */
/*      NUMERIC EQUALS/NOT EQUALS                                       */
/*                                                                    */
/*-----*/
L2140: IF ADTYP = 2 THEN
      CALL CVB(ADDR(S1D(ADX)),ADDR(AA)) ;
      IF BBTYP = 2 THEN

```

```

      CALL CVB(ADDR(BD),ADDR(BB)) ;
      BBTYP=1 ;
      IF S1P(SPX) = 18 THEN
      GOTO L3002;
      IF AA = BB THEN
      GOTO L2141;
      GOTO L3003;
L3002: IF AA = BB THEN
      GOTO L2141;
L3003: BB=0.0E0 ;
      GOTO L2260;
L2141: BB=1.0E0 ;
      GOTO L2260;

/*-----*/
/*
/*      A .GT. B RELATIONAL
/*
/*-----*/
L2150: BBTYP=1 ;
      IF AA > BB THEN
      GOTO L2151;
      BB=0.0E0 ;
      GOTO L2260;
L2151: BB=1.0E0 ;
      GOTO L2260;

/*-----*/
/*
/*      RELATIONAL .LT.
/*
/*-----*/
L2200: BBTYP=1 ;
      IF AA < BB THEN
      GOTO L2201;
      BB=0.0E0 ;
      GOTO L2260;
L2201: BB=1.0E0 ;
      GOTO L2260;

/*-----*/
/*
/*      INTEGER DIVISION
/*
/*-----*/
L2500: IF BB = 0 THEN
      GOTO L2501;
      ERNBR=12 ;
      GOTO L1;
L2501: BB=TRUNC(AA/BB) ;
      GOTO L2082;

/*-----*/
/*
/*      MODULO
/*
/*-----*/

```

```

/*-----*/
L2502: IF BB = 0 THEN
      GOTO L2503;
      ERNBR=12 ;
      GOTO L1;
L2503: BB=MOD(AA,BB) ;
      GOTO L2082;
/*-----*/
/*                                     */
/*      CONTAINS                       */
/*                                     */
/*-----*/
L2504: BB=0.0E0 ;
      BBTYP=1 ;
      IF INDEX(ADDR(S1D(ADX)),ADDR(BD),1) = 0 THEN
      GOTO L2260;
      BB=1.0E0 ;
      GOTO L2260;
/*-----*/
/*                                     */
/*      FOLLOWS                       */
/*                                     */
/*-----*/
L2505: BB=0.0E0 ;
      BBTYP=1 ;
      IF ISCOMP(ADDR(S1D(ADX)),ADDR(BD)) <= 0 THEN
      GOTO L2260;
      BB=1.0E0 ;
      GOTO L2260;
/*-----*/
/*                                     */
/*      PATTERN MATCH                 */
/*                                     */
/*-----*/
L2506: BBTYP=1 ;
      CALL PATRN(S1D(ADX)) ;
      IF IERR1 = 1 | IERR1 = 0 THEN
      GOTO L2507;
      ERNBR=16 ;
      GOTO L1;
L2507: IF IERR1 = 0 THEN
      BB=0.0E0 ;
      IF IERR1 = 1 THEN
      BB=1.0E0 ;
      GOTO L2260;
/*-----*/
/*                                     */
/*      NOT GREATER THAN              */
/*                                     */
/*-----*/
L2516: BBTYP=1 ;
      IF AA <= BB THEN

```

```

        GOTO L2751;
        BB=0.0E0 ;
        GOTO L2260;
L2751: BB=1.0E0 ;
        GOTO L2260;

/*-----*/
/*
/*      NOT LESS THAN (20)
/*
/*-----*/
L2517: BBTYP=1 ;
        IF AA >= BB THEN
        GOTO L2752;
        BB=0.0E0 ;
        GOTO L2260;
L2752: BB=1.0E0 ;
        GOTO L2260;

/*-----*/
/*
/*      NOT CONTAINED (21)
/*
/*-----*/
L2518: BBTYP=1 ;
        BB=0.0E0 ;
        IF INDEX(ADDR(S1D(ADX)),ADDR(BD),1) = 0 THEN
        BB=1.0E0 ;
        GOTO L2260;

/*-----*/
/*
/*      NOT FOLLOWS (22)
/*
/*-----*/
L2519: BBTYP=1 ;
        BB=0.0E0 ;
        IF ISCOMP(ADDR(S1D(ADX)),ADDR(BD)) > 0 THEN
        BB=1.0E0 ;
        GOTO L2260;

/*-----*/
/*
/*      NOT PATTERN (23)
/*
/*-----*/
L2520: CALL PATRN(S1D(ADX)) ;
        BBTYP=1 ;
        IF IERR1 = 1 | IERR1 = 0 THEN
        GOTO L2521;
        ERNBR=16 ;
        GOTO L1;
L2521: IF IERR1 = 0 THEN
        BB=1.0E0 ;
        IF IERR1 = 1 THEN
        BB=0.0E0 ;

```

```

      GOTO L2260;
/*-----*/
/*
/*      AND (24)
/*
/*-----*/
L2523: BBTYP=1 ;
      IF BB ^= 0 & AA ^= 0 THEN
      GOTO L2753;
      BB=0.0E0 ;
      GOTO L2260;
L2753: BB=1.0E0 ;
      GOTO L2260;
/*-----*/
/*
/*      OR (25)
/*
/*-----*/
L2525: BBTYP=1 ;
      IF AA ^= 0 | BB ^= 0 THEN
      GOTO L2754;
      BB=0.0E0 ;
      GOTO L2260;
L2754: BB=1.0E0 ;
      GOTO L2260;
/*-----*/
/*
/*      NOT AND (26)
/*
/*-----*/
L2527: BBTYP=1 ;
      IF AA ^= 0 & BB ^= 0 THEN
      GOTO L2755;
      BB=1.0E0 ;
      GOTO L2260;
L2755: BB=0.0E0 ;
      GOTO L2260;
/*-----*/
/*
/*      NOT OR (27)
/*
/*-----*/
L2529: BBTYP=1 ;
      IF AA ^= 0 | BB ^= 0 THEN
      GOTO L2756;
      BB=1.0E0 ;
      GOTO L2260;
L2756: BB=0.0E0 ;
      GOTO L2260;
/*-----*/
/*
/*      FREE TOP 2 STACK ELEMENTS - OPERATOR AND NUMBER
/*

```

```

/*      CHECK FOR OPEN PAREN - PUSH      */
/*      CHECK FOR NUMBER BENEATH - BAD   */
/*      OTHERWISE CONCLUDE ANOTHER OPERATOR IN STACK AND PROCESS */
/*-----*/
L2260: SPX=SPX-2 ;
L2280: IF S1P(SPX) = 0 | S1P(SPX) = 6 | S1P(SPX) = 10 THEN
      GOTO L2370;
L2330: IF S1P(SPX) = 5 THEN
      GOTO L2000;
L2350: IF BBTYP = 1 THEN
      GOTO L1970;
      BBTYP=2 ;
      GOTO L1970;
/*-----*/
/*      PUSH ANSWER      */
/*-----*/
L2370: SPX=SPX+1 ;
      IF BBTYP = 1 THEN
      GOTO L2380;
      I=LEN(ADDR(S1D(SDLIM))) ;
      SDLIM=SDLIM+I+1 ;
      IF SDLIM >= SYMLLEN THEN
      GOTO L3;
      CALL SCOPY(ADDR(BD),ADDR(S1D(SDLIM)) ) ;
      STKTYP(SPX)=2 ;
      GOTO L2390;
L2380: RSTK(SPX)=BB ;
      STKTYP(SPX)=1 ;
L2390: S1P(SPX)=5 ;
      GOTO L920;
/*-----*/
/*      EXIT SEQUENCE      */
/*-----*/
L2430: IF S1P(SPX-1) = 10 THEN
      GOTO L2440;
L2431: ERNBR=16 ;
      GOTO L1;
L2440: IF S1P(SPX) = 5 THEN
      GOTO L2431;
      CALL SCOPY(ADDR(XD),ADDR(XDSAVE) ) ;
      IF STKTYP(SPX) = 1 THEN
      GOTO L2441;
      CALL SCOPY(ADDR(S1D(SDLIM)),ADDR(XD) ) ;
      DTYPE='2' ;
      GOTO L2442;
L2441: XDEQ=RSTK(SPX) ;
      DTYPE='1' ;

```

L2442: XPY=XPX; /*WHOOOPS */

IERR=0 ;

RETURN;

/*-----*/

/* */

/* ERROR EXIT */

/* */

/*-----*/

L1: IERR=ERNBR ;

L2: RETURN;

L3: IERR=31 ;

RETURN;

END;

TRIM: PROC (INADR);

```

/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                        */
/*      KEVIN C. O'KANE                             */
/*      DEPARTMENT OF COMPUTER SCIENCE              */
/*      THE UNIVERSITY OF TENNESSEE                 */
/*      KNOXVILLE, TENNESSEE 37916                */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                  */
/*      JANUARY 1981                                */
/*      IBM PLI VERSION                             */
/*-----*/
/*-----*/
/*      THIS PROCEDURE DROPS TRAILING BLANKS FROM A STRING      */
/*-----*/
      DCL INADR POINTER;
      DCL IN (256) CHAR(1) BASED (INADR);
      DCL (I,K) FIXED BIN(15,0) STATIC;
      DCL BLANK CHAR(1) STATIC INIT (' ');
      DCL AB FIXED BIN(15,0) STATIC;
      DCL AC(2) CHAR(1) DEF AB;
      AB=0;
      K=LEN(ADDR(IN));
      I=K;
L10:  IF IN(I) ^= BLANK THEN RETURN;
      IN(I)=AC(2);
      I=I-1;
      IF I > 1 THEN GOTO L10;
      RETURN;
END TRIM;

```

MTPRNT: PROC (STRADR);

```

/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                        */
/*      KEVIN C. O'KANE                             */
/*      DEPARTMENT OF COMPUTER SCIENCE              */
/*      THE UNIVERSITY OF TENNESSEE                 */
/*      KNOXVILLE, TENNESSEE 37916                */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                  */
/*      JANUARY 1981                                */
/*      IBM  PLI VERSION                            */
/*-----*/
/*-----*/
/*      THIS PROCEDURE PRINTS A STRING              */
/*-----*/

```

```

DCL STRADR POINTER;
DCL STR(256) CHAR(1) BASED (STRADR);
DCL (I,J) FIXED BIN(15,0) STATIC;
DCL MTSTRG CHAR(255) VARYING INIT('');
I=LEN(STRADR);
DO J=1 TO I;
MTSTRG=MTSTRG||STR(J);
END;
PUT EDIT(MTSTRG) (A) SKIP ;
RETURN;
END MTPRNT;

```

STRPAD: PROC (INADR, LN);

```

/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                        */
/*      KEVIN C. O'KANE                             */
/*      DEPARTMENT OF COMPUTER SCIENCE              */
/*      THE UNIVERSITY OF TENNESSEE                 */
/*      KNOXVILLE, TENNESSEE 37916                */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                  */
/*      JANUARY 1981                                */
/*      IBM PLI VERSION                             */
/*-----*/
/*-----*/
/*      THIS PROCEDURE PADS A STRING WITH 'LN' BLANKS      */
/*-----*/
      DCL INADR POINTER;
      DCL IN(256) CHAR(1) BASED (INADR);
      DCL LN FIXED BIN(15,0);
      DCL BLANK CHAR(1) STATIC INITIAL (' ');
      DCL AB FIXED BIN(15,0) STATIC INIT(0);
      DCL AC (2) CHAR DEF AB;
      I=LEN(ADDR(IN));
      IF I > LN THEN RETURN;
      I=I+1;
      DO J=I TO LN;
      IN(J)=BLANK;
      END;

      IN(LN+1)=AC(2);
      RETURN;
      END;

```

ISCOMP: PROC(APTR,BPTR) RETURNS (FIXED BIN(15,0));

```

/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                        */
/*      KEVIN C. O'KANE                             */
/*      DEPARTMENT OF COMPUTER SCIENCE              */
/*      THE UNIVERSITY OF TENNESSEE                 */
/*      KNOXVILLE, TENNESSEE 37916                */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                  */
/*      JANUARY 1981                                */
/*      IBM  PLI VERSION                             */
/*-----*/
/*-----*/
/*      THIS PROCEDURE COMPARES TWO STRINGS. IF THE STRINGS      */
/*      ARE EQUAL IT RETURNS ZERO, IF 'A' IS GREATER THAN 'B'    */
/*      IT RETURNS 1. IF 'A' IS LESS THAN 'B' IT RETURNS -1.     */
/*-----*/

```

```

DCL APTR POINTER;
DCL BPTR POINTER;
DCL A (256) CHAR(1) BASED (APTR);
DCL B (256) CHAR(1) BASED (BPTR);
DCL (I,J,K) FIXED BIN(15,0) STATIC;
I=LEN(ADDR(A));
J=LEN(ADDR(B));
IF I <= J THEN GOTO L100;

DO K= 1 TO I;
IF A(K) = B(K) THEN GOTO L10;
IF A(K) > B(K) THEN RETURN(1);
IF A(K) < B(K) THEN RETURN (-1);
L10:  END;
RETURN(0);

L100: IF I < J THEN K=I;
      IF I > J THEN K=J;

DO L=1 TO K;
IF A(L) = B(L) THEN GOTO L110;
IF A(L) > B(L) THEN RETURN(1);
IF A(L) < B(L) THEN RETURN (-1);
L110: END;

IF I>J THEN RETURN(1);
IF I<J THEN RETURN (-1);
RETURN(0);
END ISCOMP;

```

```
CONCAT: PROC(LFTADR, RGTADR, OUTADR);
```

```

/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                        */
/*      KEVIN C. O'KANE                             */
/*      DEPARTMENT OF COMPUTER SCIENCE              */
/*      THE UNIVERSITY OF TENNESSEE                  */
/*      KNOXVILLE, TENNESSEE 37916                 */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                  */
/*      JANUARY 1981                                */
/*      IBM PLI VERSION                             */
/*-----*/
/*-----*/
/*      THIS PROCEDURE CONCATENATES TWO STRINGS      */
/*-----*/

```

```

DCL (LFTADR, OUTADR, RGTADR) POINTER;
DCL LFT (256) CHAR(1) BASED (LFTADR);
DCL RGT (256) CHAR(1) BASED (RGTADR);
DCL OUT (256) CHAR(1) BASED (OUTADR);
DCL (I, J, K) FIXED BIN(15, 0) STATIC;
DCL AB FIXED BIN(15, 0) STATIC INIT(0);
DCL AC(2) CHAR(1) DEF AB;

```

```

I=LEN(ADDR(LFT));
J=LEN(ADDR(RGT));
DO K=1 TO I;
OUT(K)=LFT(K);
END;
DO K=1 TO J;
OUT(I+K)=RGT(K);
END;
OUT(I+J+1)=AC(2);
RETURN;
END CONCAT;

```

INDEX: PROC (SOURCEADR,PATRNADR,STRT) RETURNS (FIXED BIN(15,0));

```

/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                      */
/*      KEVIN C. O'KANE                            */
/*      DEPARTMENT OF COMPUTER SCIENCE             */
/*      THE UNIVERSITY OF TENNESSEE                 */
/*      KNOXVILLE, TENNESSEE 37916                */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                  */
/*      JANUARY 1981                               */
/*      IBM  PLI VERSION                            */
/*-----*/
/*-----*/
/*      THIS PROCEDURE SEARCHES A  STRING FOR A GIVEN PATTERN  */
/*      STARTING AT POSITION 'STRT'. IF FOUND IT RETURNS THE     */
/*      CHARACTER POSITION.                                       */
/*-----*/

```

```

DCL SOURCEADR POINTER;
DCL PATRNADR POINTER;
DCL STRT FIXED BIN(15,0);
DCL SOURCE (256) CHAR(1) BASED (SOURCEADR);
DCL PATRN (256) CHAR(1) BASED (PATRNADR);
DCL (I,J,K,L) FIXED BIN(15,0) STATIC;

```

```

I=LEN(ADDR(PATRN));
J=LEN(ADDR(SOURCE));
IF STRT = 0 THEN STRT=1;
DO K=STRT TO J;
DO L=1 TO I;
IF PATRN(L) ^= SOURCE(K+L-1) THEN GOTO L10;
END;
RETURN(K);
L10:  END;
RETURN(0);
END INDEX;

```

SCOPY: PROC (INADR,OUTADR);

```

/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                        */
/*      KEVIN C. O'KANE                              */
/*      DEPARTMENT OF COMPUTER SCIENCE              */
/*      THE UNIVERSITY OF TENNESSEE                  */
/*      KNOXVILLE, TENNESSEE 37916                 */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                  */
/*      JANUARY 1981                                */
/*      IBM  PLI VERSION                             */
/*-----*/
/*-----*/
/*      THIS PROCEDURE COPIES A STRING               */
/*-----*/

```

```

DCL INADR POINTER;
DCL OUTADR POINTER;
DCL OUT (256) CHAR(1) BASED (OUTADR);
DCL IN (256) CHAR(1) BASED (INADR);
DCL I FIXED BIN(15,0) STATIC;
DCL AB FIXED BIN(15,0) STATIC INIT(0);
DCL AC(2) CHAR(1) DEF AB;

```

```

DO I=1 TO 256;
OUT(I)=IN(I);
IF IN(I) = AC(2) THEN RETURN;
END;
OUT(256)=AC(1);
RETURN;
END SCOPY;

```

```
LEN:  PROC (INADR) RETURNS (FIXED BIN(15,0));
```

```
/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                        */
/*      KEVIN C. O'KANE                             */
/*      DEPARTMENT OF COMPUTER SCIENCE              */
/*      THE UNIVERSITY OF TENNESSEE                 */
/*      KNOXVILLE, TENNESSEE 37916                */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                  */
/*      JANUARY 1981                                */
/*      IBM  PLI VERSION                             */
/*-----*/
/*-----*/
/*      THIS PROCEDURE RETURNS THE LENGTH OF A STRING      */
/*-----*/
```

```
DCL INADR POINTER;
DCL IN (256) CHAR(1) BASED (INADR);
DCL AB FIXED BIN(15,0) STATIC INIT(0);
DCL I FIXED BIN(15,0);
DCL AC(2) CHAR(1) DEF AB;
DCL TEMP CHAR(1);
```

```
DO I=1 TO 256;
TEMP=IN(I);
IF IN(I) = AC(2) THEN RETURN(I-1);
END;
```

```
RETURN(255);
END LEN;
```

SUBSTR: PROC (INADR,OUTADR,START,LEN);

```

/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                        */
/*      KEVIN C. O'KANE                             */
/*      DEPARTMENT OF COMPUTER SCIENCE              */
/*      THE UNIVERSITY OF TENNESSEE                 */
/*      KNOXVILLE, TENNESSEE 37916                */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                  */
/*      JANUARY 1981                                */
/*      IBM  PLI VERSION                             */
/*-----*/
/*-----*/
/*      THIS PROCEDURE CREATES A SUBSTRING OF LENGTH 'LEN'      */
/*      STARTING FROM POSITION 'START'.                          */
/*-----*/

```

```

DCL (INADR,OUTADR) POINTER;
DCL IN (256) CHAR(1) BASED (INADR);
DCL OUT (256) CHAR(1) BASED (OUTADR);
DCL (LEN,STRT) FIXED BIN(31);
DCL (I,J,K) FIXED BIN(31) STATIC;
DCL AB FIXED BIN(15,0) STATIC INIT(0);
DCL AC(2) CHAR(1) DEF AB;
DCL START FIXED BINARY(31) ;

```

```

IF LEN < 0 THEN GOTO L100;
IF LEN > 0 THEN GOTO L10;
OUT(1)=AC(2);
RETURN;

```

```

L10:  K=START+LEN+1;
      DO I=START TO K;
      OUT(I-START+1)=IN(I);
      END;

```

```

      OUT(LEN+1)=AC(2);
      RETURN;

```

```

L100: DO I=START TO 255;
      OUT(I-START+1)=IN(I);
      IF IN(I) = AC(2) THEN RETURN;
      END;

```

```

      OUT(256)=AC(2);
      RETURN;
      END SUBSTR;

```

```
MTOUT: PROC (CHAN,CHARADR) RETURNS (FIXED BIN(15,0));
```

```

/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                      */
/*      KEVIN C. O'KANE                            */
/*      DEPARTMENT OF COMPUTER SCIENCE             */
/*      THE UNIVERSITY OF TENNESSEE                 */
/*      KNOXVILLE, TENNESSEE 37916                */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                  */
/*      JANUARY 1981                                */
/*      IBM  PLI VERSION                            */
/*-----*/
/*-----*/
/*      THIS PROCEDURE WAS INCLUDED FOR COMPATIBILITY WITH    */
/*      PDP11 VERSION OF MUMPS.                               */
/*-----*/

```

```

DCL CHAN FIXED BIN(15,0);
DCL CHARADR POINTER;
DCL CHAR CHAR(1) BASED (CHARADR);
PUT EDIT (CHAR) (A(1));
RETURN(0);
END MTOUT;

```

```
MTIN:  PROC (CHAN,CHARADR) RETURNS(FIXED BIN(15,0));
```

```
/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                        */
/*      KEVIN C. O'KANE                             */
/*      DEPARTMENT OF COMPUTER SCIENCE              */
/*      THE UNIVERSITY OF TENNESSEE                  */
/*      KNOXVILLE, TENNESSEE 37916                 */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                  */
/*      JANUARY 1981                                */
/*      IBM  PLI VERSION                             */
/*-----*/
/*-----*/
/*      THIS PROCEDURE WAS INCLUDED FOR COMPATIBILITY WITH THE  */
/*      PDP11 VERSION OF MUMPS.                                */
/*-----*/
/*-----*/
```

```
DCL CHAN FIXED BIN(15,0);
DCL CHARADR POINTER;
DCL CHAR CHAR(1) BASED (CHARADR);
GET EDIT (CHAR) (A(1));
RETURN(0);
END MTIN;
```

```
ERRMOD: PROC(I,STR,L);
```

```

/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                        */
/*      KEVIN C. O'KANE                             */
/*      DEPARTMENT OF COMPUTER SCIENCE              */
/*      THE UNIVERSITY OF TENNESSEE                 */
/*      KNOXVILLE, TENNESSEE 37916                */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                  */
/*      JANUARY 1981                                */
/*      IBM PLI VERSION                             */
/*-----*/
/*-----*/
/*      THIS PROCEDURE RETURNS AN ERROR NUMBER AND ERROR      */
/*      MESSAGE.                                              */
/*-----*/
      DCL (I,L,J,K) FIXED BIN(15,0);
      DCL STR (256) CHAR(1);
      DCL PTR POINTER;
      PTR=ADDR(STR);
      PUT SKIP(2);
      PUT EDIT ('ERR ',I) (A,F(4));
      J=LEN(PTR);
      DO K=L TO J;
      PUT EDIT (STR(K)) (A(1));
      END;
      END ERRMOD;

```

CVB: PROCEDURE(ADADR,AXLADR);

```

/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                      */
/*      KEVIN C. O'KANE                            */
/*      DEPARTMENT OF COMPUTER SCIENCE             */
/*      THE UNIVERSITY OF TENNESSEE                */
/*      KNOXVILLE, TENNESSEE 37916               */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                 */
/*      JANUARY 1981                               */
/*      IBM PLI VERSION                            */
/*-----*/
/*-----*/
/*      CONVERT TO BINARY                          */
/*      CONVERTS STRING IN AD TO REAL BINARY IN AXL. */
/*      USES MUMPS STRING TO NUMERIC CONVERSION RULES. */
/*      MAXIMUM OF 20 CHARACTERS ON NUMERIC INPUT    */
/*-----*/

DCL ADADR POINTER;
DCL AXLADR POINTER;
DECLARE TMP(21) CHAR(1);
DECLARE TMPSTR CHAR(21) DEF TMP;
DCL AB FIXED BIN(15,0) STATIC INIT(0);
DCL AC(2) CHAR(1) DEF AB;
DECLARE DUMBASE(4)CHAR(1);
DECLARE SIGN CHAR(1) STATIC,
      AXT( 4) CHAR(1) DEF AX,
      AXL(256) BASED (AXLADR) CHAR(1),
      DP CHAR(1) STATIC,
      AX FLOAT BINARY,
      AD( 256) BASED (ADADR) CHAR(1),
      I FIXED BINARY(31) STATIC;
IF AD(1) = AC(2) THEN GOTO L214;
L100: IF AD(1) >= '0' & AD(1) <= '9' THEN GOTO L200;
      IF AD(1) = '-' | AD(1) = '.' | AD(1) = '+' THEN GOTO L200;
      GOTO L214;
L200: IF LEN(ADDR(AD)) > 20 THEN
      GOTO L214;
L201: CALL SCOPY(ADDR(AD),ADDR(TMP)) ;
L101: IF TMP(1) <= ' ' THEN
      GOTO L202;
      CALL SCOPY(ADDR(TMP(2)),ADDR(TMP)) ;
      GOTO L101;
L202: IF TMP(1) = '-' & TMP(2) = '-' THEN GOTO L203;
      GOTO L204;
L203: CALL SUBSTR(ADDR(TMP),ADDR(TMP),2,-1) ;
      IF LEN(ADDR(TMP)) > 2 THEN
      GOTO L202;

```

```

L204:  IF TMP(1) = '+' & TMP(2) = '+' THEN GOTO L205;
      GOTO L206;
L205:  CALL SUBSTR(ADDR(TMP),ADDR(TMP),2,-1) ;
      IF LEN(ADDR(TMP)) > 2 THEN
      GOTO L204;
L206:  I=0 ;
      DP= '0'B ;
      SIGN= '0'B ;
L207:  I=I+1 ;
      IF TMP(I) >= '0' & TMP(I) <= '9' THEN GOTO L207;
      IF TMP(I) ^= '.' THEN GOTO L208;
      IF DP THEN GOTO L210;
      DP= '1'B ;
      GOTO L207;
L208:  IF TMP(I) = '-' | TMP(I) = '+' THEN GOTO L209;
      GOTO L210;
L209:  IF SIGN THEN GOTO L211;
      SIGN= '1'B ;
      GOTO L207;
L211:  IF I ^= 2 THEN GOTO L210;
      ELSE GOTO L214;
L210:  I=I-1 ;
      IF I <= LEN(ADDR(TMP)) THEN GOTO L212;
      ELSE GOTO L214;
L212:  IF I > 0 THEN GOTO L213;
      ELSE GOTO L214;
L213:  CALL SUBSTR(ADDR(TMP),ADDR(TMP),1,I) ;
      IF ~ DP THEN
      CALL CONCAT(ADDR(TMP),ZPAD('.',),ADDR(TMP)) ;
      CALL STRPAD(ADDR(TMP),20) ;
      GET STRING(TMPSTR) EDIT (AX) (E(15,7));
/* WAS DECODE (20,1000,TMP,ERR=214) AX*/
L300:  AXL(1)=AXT(1) ;
      AXL(2)=AXT(2) ;
      AXL(3)=AXT(3) ;
      AXL(4)=AXT(4) ;
      RETURN;
L214:  AX=0.0E0 ;
      GOTO L300;
      END;

```

CVC: PROCEDURE(ADADR,AXLADR);

```

/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                        */
/*      KEVIN C. O'KANE                             */
/*      DEPARTMENT OF COMPUTER SCIENCE              */
/*      THE UNIVERSITY OF TENNESSEE                 */
/*      KNOXVILLE, TENNESSEE 37916                */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                  */
/*      JANUARY 1981                                */
/*      IBM  PLI VERSION                             */
/*-----*/

```

```

/*-----*/
/*      CONVERT TO CHARACTER                        */
/*      CONVERTS REAL TO CHARACTER STRING. ADJUSTS AS APPROPRIATE */
/*      THE INTERNAL FORMATS BY DYNAMICALLY INSPECTING THE      */
/*      NUMBER TO BE CONVERTED USING A LOG BASE 10 FUNCTION.    */
/*      INPUT IS AXL (REAL) AND OUTPUT IS AD. RESULT IS         */
/*      20 OR FEWER CHARACTERS (MODULE TRIMS BLANKS AND DROPS   */
/*      TRAILING UNNEEDED ZEROS. USELESS DECIMAL POINTS ALSO  */
/*      DROPPED.)                                              */
/*-----*/

```

```

DCL ADADR POINTER;
DCL AXLADR POINTER;
DECLARE TMP( 21) CHAR(1);
DECLARE TMPSTR CHAR(21) DEF TMP;
DECLARE DUMBASE( 4)CHAR(1);
DECLARE AXT( 4) CHAR(1) DEF AX,
      AXL( 4)  BASED(AXLADR) CHAR(1);
DCL AB FIXED BIN(15,0) STATIC INIT(0);
DCL AC(2) CHAR(1) DEF AB;
DECLARE AX FLOAT BINARY,
      AD(256) BASED (ADADR) CHAR(1),
      J FIXED BINARY(31) STATIC,
      I FIXED BINARY(31)STATIC;
DCL NUMBERS CHAR(10) INIT('0123456789');

AXT(1)=AXL(1) ;
AXT(2)=AXL(2) ;
AXT(3)=AXL(3) ;
AXT(4)=AXL(4) ;
IF AX ^= 0.0E0 THEN GOTO L203;
AD(1)='0';
AD(2)=AC(2) ;
RETURN;
L203: PUT STRING(TMPSTR) EDIT(AX) (E(15,7));

```

```

    TMP(16)=AC(2) ;
    I=INDEX(ADDR(NUMBERS),ADDR(TMP(15)),1);
    TMP(15)=AC(2) ;
    J=INDEX(ADDR(NUMBERS),ADDR(TMP(14)),1);
    I=(J-1)*10+I-1 ;
    TMP(12)=AC(2) ;
    IF TMP(13) = '-' THEN I=-I;
    IF I > 6 THEN
        GOTO L102;
    IF I < 1 THEN
        GOTO L104;
    CALL SUBSTR(ADDR(TMP),ADDR(AD),3,1);
    CALL SUBSTR(ADDR(TMP),ADDR(AD(2)),5,I);
    AD(I+2)='.';
    CALL SCOPY(ADDR(TMP(5+I)),ADDR(AD(I+3))) ;
    J=9 ;
L107: IF AD(J) = '0' THEN GOTO L150;
    AD(J)=AC(2) ;
    J=J-1 ;
    GOTO L107;
L102: CALL SUBSTR(ADDR(TMP),ADDR(AD),3,1);
    CALL SUBSTR(ADDR(TMP),ADDR(AD(2)),5,7) ;
    IF I = 7 THEN
        GOTO L150;
    CALL CONCAT(ADDR(AD),ZPAD('.'),ADDR(AD)) ;
    DO J=8 TO MAX(8,I);
    CALL CONCAT(ADDR(AD),ZPAD('0'),ADDR(AD)) ;
L103: END;
    GOTO L150;
L104: IF I = 0 THEN
    GOTO L105;
    CALL SUBSTR(ADDR(TMP),ADDR(AD),3,9) ;
    J=9;
    GOTO L107;
L105: I=-I;
    CALL SUBSTR(ADDR(TMP),ADDR(AD(1+I)),3,1) ;
    CALL SUBSTR(ADDR(TMP),ADDR(AD(2+I)),5,7);
    AD(1)='.';
    DO J=2 TO I;
    AD(J)='0';
    END;
    J=LEN(ADDR(AD)) ;
    GOTO L107;
L150: IF AD(LEN(ADDR(AD))) = '.' THEN AD(LEN(ADDR(AD)))=AC(2);
    IF TMP(2) = '-' THEN GOTO L151;
    CALL CONCAT(ZPAD('-'),ADDR(AD),ADDR(TMP)) ;
    CALL SCOPY(ADDR(TMP),ADDR(AD)) ;
L151: IF AX < 9999999. THEN RETURN;
    DO I=8 TO LEN(ADDR(AD));
    AD(I)='0';
    END;
    RETURN;

```

END;

LPAD: PROCEDURE (STR,L);

```

/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                        */
/*      KEVIN C. O'KANE                             */
/*      DEPARTMENT OF COMPUTER SCIENCE              */
/*      THE UNIVERSITY OF TENNESSEE                 */
/*      KNOXVILLE, TENNESSEE 37916                */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                  */
/*      JANUARY 1981                                */
/*      IBM  PLI VERSION                             */
/*-----*/
/*-----*/
/*      PAD TO THE LEFT WITH BLANKS TO MAKE STRING OF */
/*      LENGTH L                                       */
/*-----*/

      DECLARE STR( 256) CHAR(1),
              PTR POINTER,
              J FIXED BINARY(31) STATIC,
              K FIXED BINARY(31) STATIC,
              I FIXED BINARY(31) STATIC,
              L FIXED BINARY(31);

      PTR=ADDR(STR);
      J=LEN(PTR) ;
      K=L-J ;
      IF K <= 0 THEN
      RETURN;
      DO I=J TO 1 BY -1;
L10:  STR(I+K)=STR(I) ;
      END;
      STR(J+K+1)=0 ;
      DO I=1 TO MAX(1,K);
      STR(I)= ' ' ;
      END;
      RETURN;
      END;

```

INOUT: PROCEDURE (TEXTADR,CMND,COL);

```

/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                        */
/*      KEVIN C. O'KANE                             */
/*      DEPARTMENT OF COMPUTER SCIENCE              */
/*      THE UNIVERSITY OF TENNESSEE                  */
/*      KNOXVILLE, TENNESSEE 37916                 */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                  */
/*      JANUARY 1981                                */
/*      IBM  PLI VERSION                             */
/*-----*/
/*-----*/
/*      THIS IS THE MAIN TERMINAL INPUT..OUTPUT ROUTINE.      */
/*-----*/

```

```

DECLARE TEXTADR POINTER;
DECLARE 1 FILESCO EXTERNAL,
        2 DUMITEM FLOAT BINARY(53),
        2 U8 CHAR(1),
        2 U9 CHAR(1),
        2 CLRGBL CHAR(1),
        2 TERM CHAR(1),
        2 G5RECS FIXED BINARY(31),
        2 G2RECS FIXED BINARY(31);

DECLARE 1 INOUTCC EXTERNAL,
        2 DUMITEM FLOAT BINARY(53),
        2 MTCUR FIXED BINARY(31);

DECLARE MTCUR FIXED BINARY(31) STATIC,
        TMP1( 2) CHAR(1) STATIC,
        CRLF( 4) CHAR(1) STATIC,
        CMND FIXED BINARY(31),
        TEXT( 256) CHAR(1) BASED(TEXTADR),
        PTR POINTER;
DECLARE OUTSTRG CHAR(255) VARYING EXTERNAL;

AB=0; TMP1(1)=AC(2); TMP1(2)=AC(2);
AB=12; CRLF(1)=AC(2);
AB=13; CRLF(2)=AC(2);
AB=0; CRLF(3)=AC(2); CRLF(4)=AC(2);
PTR=ADDR(TEXT);
DECLARE
        DEV FIXED BINARY(31) INITIAL(4),
        VER( 5) FIXED BINARY(31),
        HOR( 5) FIXED BINARY(31),
        COL FIXED BINARY(31),

```

```

      BRANCH01( 4) LABEL INITIAL(L10,L20,L30,L40);

DECLARE LF( 2) CHAR(1) STATIC,
        CR( 2) CHAR(1) STATIC,
        FF( 2) CHAR(1) STATIC,
        SP( 2) CHARACTER(1) STATIC,
        J FIXED BINARY(31) STATIC;

AB=10; LF(1)=AC(2); LF(2)=AC(1);
AB=13; CR(1)=AC(2); CR(2)=AC(1);
AB=12; FF(1)=AC(2); FF(2)=AC(1);
AB=0; SP(1)=' '; SP(2)=AC(2);
DECLARE DUMM003 CHARACTER( 1) INITIAL(' ');
DECLARE DUMM002 CHARACTER( 1) INITIAL('+');
DECLARE DUMM001 CHARACTER( 1) INITIAL('+');

/*-----*/
/*
/*      MAIN OUTPUT ROUTINE
/*
/*-----*/
L1:   IF CMND<=04 &CMND > 0 THEN GOTO BRANCH01(CMND);
/*-----*/
/*
/*      NEW PAGE
/*
/*-----*/
L10:  IF DEV ^= 5 THEN
      GOTO L11;
L12:  IF MTOUT(MTCUR-1,ADDR(FF)) ^= 0 THEN
      GOTO L12; ELSE GOTO L14;
L11:  PUT EDIT(OUTSTRG) (A);
      PUT PAGE;
      OUTSTRG=' ';
L14:  HOR(DEV)=1 ;
      VER(DEV)=0 ;
      RETURN;

/*-----*/
/*
/*      SKIP LINE
/*
/*-----*/
L20:  IF DEV ^= 5 THEN
      GOTO L21;
L22:  IF MTOUT(MTCUR-1,ADDR(CR)) ^= 0 THEN
      GOTO L22;
L23:  IF MTOUT(MTCUR-1,ADDR(LF)) ^= 0 THEN
      GOTO L23; ELSE GOTO L25;
L21:  PUT EDIT(OUTSTRG) (A) SKIP ;
      OUTSTRG=' ';
L25:  HOR(DEV)=0 ;
      VER(DEV)=VER(DEV)+1 ;
      RETURN;

```

```

/*-----*/
/*
/*      TAB
/*
/*-----*/
L30:  IF COL = HOR(DEV) THEN
      RETURN;
      IF COL > HOR(DEV) THEN
      GOTO L300;
      IF DEV ^= 5 THEN
      GOTO L37;
L36:  IF MTOUT(MTCUR-1,ADDR(CR)) ^= 0 THEN
      GOTO L36; ELSE GOTO L300;
      OUTSTRG=OUTSTRG||SP(1); /* OUTPUT A SPACE */
L37:  HOR(DEV)=1 ;
      VER(DEV)=VER(DEV)+1 ;
L33:  DO J=HOR(DEV) TO COL-1;
      OUTSTRG=OUTSTRG||SP(1); /* OUTPUT A SPACE */
      END;
      HOR(DEV)=COL ;
      RETURN;
L300: IF DEV ^= 5 THEN
      GOTO L33;
      DO J=HOR(DEV) TO COL-1;
L305: IF MTOUT(MTCUR-1,ZPAD(' ')) THEN GOTO L305;
      END;
      HOR(DEV)=COL ;
      RETURN;
/*-----*/
/*
/*      TEXT PRINT
/*
/*-----*/
L40:  IF DEV = 5 THEN
      GOTO L50;
      DO J=1 TO LEN(PTR);
      OUTSTRG=OUTSTRG||TEXT(J); /* BUILD OUTPUT STRING */
L402: HOR(DEV)=HOR(DEV)+1 ;
      END;
      RETURN;
L50:  DO J=1 TO LEN(PTR);
L52:  TMP1(1)=TEXT(J) ;
L54:  IF MTOUT(MTCUR-1,ADDR(TMP1)) ^= 0 THEN
      GOTO L54;
      HOR(DEV)=HOR(DEV)+1 ;
      END;
      RETURN;
      END;

```

SYM: PROCEDURE (APTR,BPTR,TYPE);

```

/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                        */
/*      KEVIN C. O'KANE                             */
/*      DEPARTMENT OF COMPUTER SCIENCE              */
/*      THE UNIVERSITY OF TENNESSEE                  */
/*      KNOXVILLE, TENNESSEE 37916                 */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                  */
/*      JANUARY 1981                                */
/*      IBM PLI VERSION                             */
/*-----*/
/*-----*/
/*      MAIN SYMBOL TABLE ROUTINE                  */
/*-----*/

```

```

DECLARE APTR POINTER;
DECLARE BPTR POINTER;
DECLARE TMP11(2) CHAR(1),
        TMP2( 256) CHAR(1),
        TMP3( 256) CHAR(1);

DECLARE 1 CODESCO EXTERNAL,
        2 DUMITEM FLOAT BINARY(53),
        2 COD251( 2) CHAR(1),
           2 COD252( 2) CHAR(1),
        2 COD253( 2) CHAR(1),
        2 COD254( 2) CHAR(1),
        2 COD255( 2) CHAR(1),
        2 XTAB( 2)CHAR(1),
        2 DUMITEM4(2) FIXED BINARY(31);

DECLARE 1 UNLCM EXTERNAL,
        2 DUMITEM FLOAT BINARY(53),
        2 SYMLEN
        FIXED BINARY(31),
        2 SYMADR FIXED BINARY(31),
        2 FLAG FIXED BINARY(15,0),
        2 GD(257) CHAR(1),
        2 G(15) FIXED BINARY(31),
        2 P FLOAT BINARY,
        2 PD(257) CHAR(1),
        2 VD(256) CHAR(1),
        2 DOSAVE(21) CHAR(1),
        2 XX8(5) CHAR(1),
        2 DATRIB CHAR(1),
        2 DVAR FIXED BINARY(31),
        2 XX9(6) CHAR(1),

```

```

2 PD1LEN FIXED BINARY(31),
2 PD1CUR FIXED BINARY(31),
2 TRACE FIXED BINARY(31),
2 ZD(256) CHAR(1),
2 PD1( 4000) CHAR(1),
2 FORV(5,10) CHAR(1),
2 FORIT(5) FIXED BINARY(1),
2 FORLT(5) FIXED BINARY(1),
2 FTMP(256) CHAR(1),
2 FORIN(5) FLOAT BINARY,
2 FORLV(5) FLOAT BINARY,
2 FORNXT(5) FIXED BINARY(31),
2 FORX FIXED BINARY(31),
2 GOSUB(6,5) FIXED BINARY(31),
2 GSBX FIXED BINARY(31);

```

```

DECLARE TMP1( 100) CHAR(1) STATIC,
        RSLT FLOAT BINARY STATIC,
        SYMX FIXED BINARY(31) STATIC,
        TYPE CHAR(1),
        II FIXED BINARY(31) STATIC,
        L FIXED BINARY(31) STATIC,
        M FIXED BINARY(31) STATIC,
        I FIXED BINARY(31) STATIC,
        J FIXED BINARY(31) STATIC,
        K FIXED BINARY(31) STATIC,
        B( 256) CHAR(1) BASED (BPTR),
        A( 256)CHAR(1) BASED(APTR);
DECLARE AC(2) CHAR DEF AB;
DECLARE AB FIXED BINARY(15,0);
AB=0;

```

```

/*-----*/
/*
/*      CODE 1 RETRIEVE
/*      CODE 0 STORE/CREATE
/*      CODE 2 DELETE EXPLICIT
/*      CODE 3 $N ON LAST ARGUMENT
/*      TYPE 1 - REAL
/*      TYPE 2 - CHAR
/*      TYPE IS IN FIRST BYTE OF NAME FIELD
/*-----*/
/*
/*      CODES
/*-----*/
/* LOGICAL*1 COD251(2) |OPEN
/* LOGICAL*1 COD252(2) |CLOSE
/* LOGICAL*1 COD253(2) |COMMA
/* LOGICAL*1 COD254(2) |COLON
/* LOGICAL*1 COD255(2) |QUOTE
/* LOGICAL*1 XTAB(2)   |TAB
/*
/*

```

```

/*-----*/
      IF FLAG = 3 THEN
      GOTO L300;
      IF FLAG = 1 THEN
      GOTO L100;
/*-----*/
/*
/*
/*      RETRIEVE
/*
/*
/*-----*/
      I=SYMLEN ;
L10:  IF I < 4000 THEN
      GOTO L20;
L200: FLAG=0 ;
      RETURN;
L20:  IF ISCOMP(ADDR(PD1(I+1)),ADDR(A)) = 0 THEN
      GOTO L30;
      J=I ;
      I=I+LEN(ADDR(PD1(I)))+1 ;
      IF PD1(J) ='1' THEN
      GOTO L21;
      I=I+LEN(ADDR(PD1(I)))+1 ;
      GOTO L10;
L21:  I=I+5 ;
      GOTO L10;
L30:  TYPE=PD1(I) ;
      SYMADR=I+LEN(ADDR(PD1(I)))+1 ;
      RETURN;
/*-----*/
/*
/*
/*      ENTER
/*
/*
/*-----*/
L100: I=SYMLEN ;
L110: IF I < 4000 THEN
      GOTO L120;
      IF FLAG = 2 | FLAG = 4 THEN
      RETURN;
      GOTO L150;
L120: IF ISCOMP(ADDR(PD1(I+1)),ADDR(A)) = 0 THEN GOTO L130;
      IF FLAG = 2 THEN
      GOTO L201;
      IF INDEX(ADDR(A),ADDR(COD251),1) = 0 THEN
      GOTO L201;
      IF A(LEN(ADDR(A))) = COD252(1) THEN
      A(LEN(ADDR(A)))=AC(1); /*REMOVECLOSE */
      II=INDEX(ADDR(PD1(I+1)),ADDR(A),1); /*SUBSTRING? */
      IF II = 1 THEN
      GOTO L201;
/*      CHECK THAT SUBXTR ENDS WITH COMMA OR CLOSE*/
      IF PD1(I+1+LEN(ADDR(A))) = COD253(1)
        & PD1(I+1+LEN(ADDR(A))) = COD252(1) THEN

```

```

        GOTO L201;
        ELSE GOTO L130;
L201:   J=I ;
        I=I+LEN(ADDR(PD1(I)))+1 ;
        IF PD1(J) ='1' THEN
            GOTO L121;
        I=I+LEN(ADDR(PD1(I)))+1 ;
        GOTO L110;
L121:   I=I+5 ;
        GOTO L110;
L130:   J=LEN(ADDR(PD1(I)))+1+I ;
        IF FLAG = 2 | FLAG = 4 THEN
            GOTO L134;
        IF PD1(I) = '1' & (TYPE = '1' | LEN(BPTR) = 4) THEN
            GOTO L131;
        IF PD1(I) ='1' THEN
            GOTO L135;
        IF LEN(ADDR(PD1(J))) = LEN(BPTR) & TYPE = '2' THEN
            GOTO L131;
        IF LEN(ADDR(PD1(J))) = 4 & TYPE = '1' THEN
            GOTO L131;
/*-----*/
/*                                           */
/*      DELETE AND COMPRESS                      */
/*                                           */
/*-----*/
L134:   IF PD1(I) ='1' THEN
        GOTO L135;
        J=J+LEN(ADDR(PD1(J)))+1 ;
        GOTO L136;
L135:   J=J+5 ;
L136:   L=I ;
        DO K=J-1 TO SYMLen+(J-I) BY -1;
        L=L-1 ;
        PD1(K)=PD1(L) ;
        END;
        SYMLen=SYMLen+(J-I) ;
        GOTO L150;
L131:   IF TYPE = '1' THEN
        GOTO L141;
        CALL SCOPY(BPTR,ADDR(PD1(J))) ;
        GOTO L142;
L141:   PD1(J)=B(1) ;
        PD1(J+1)=B(2) ;
        PD1(J+2)=B(3) ;
        PD1(J+3)=B(4) ;
        PD1(J+4)=AC(1) ;
L142:   SYMADR=J ;
        PD1(I)=TYPE ;
L160:   FLAG=1 ;
        RETURN;
L150:   IF FLAG = 0 THEN

```

```

      GOTO L100;
L151: SYMLen=SYMLen-2-LEN(APTR) ;
      IF TYPE = '2' THEN
      GOTO L501;
      SYMLen=SYMLen-5 ;
      GOTO L502;
L501: SYMLen=SYMLen-LEN(BPTR)-1 ;
L502: SYMX=SYMLen ;
      PD1(SYMX)=TYPE ;
      SYMX=SYMX+1 ;
      CALL SCOPY(APTR,ADDR(PD1(SYMX))) ;
      SYMX=SYMX+LEN(APTR)+1 ;
      IF TYPE = '1' THEN
      GOTO L143;
      CALL SCOPY(BPTR,ADDR(PD1(SYMX))) ;
      SYMADR=SYMX ;
      GOTO L160;
L143: PD1(SYMX)=B(1) ;
      PD1(SYMX+1)=B(2) ;
      PD1(SYMX+2)=B(3) ;
      PD1(SYMX+3)=B(4) ;
      PD1(SYMX+4)=AC(1) ;
      SYMADR=SYMX ;
      GOTO L160;
/*-----*/
/*                                          */
/*      NEXT                                */
/*                                          */
/*-----*/
L300: I=LEN(APTR) ;
      IF A(I) = COD252(1) THEN
      GOTO L301;
      FLAG=0 ;
      RETURN;
L301: I=I-1 ;
      IF A(I) /= COD251(1) & A(I) /= COD253(1) THEN
      GOTO L301;
      CALL SUBSTR(APTR,ADDR(TMP2),I+1,LEN(APTR)-I-1) ;
      CALL CVB(ADDR(TMP2),ADDR(RSLT)) ;
      CALL SUBSTR(APTR,ADDR(TMP1),1,I) ;
      J=SYMLen ;
      L=RSLT ;
      M=-1 ;
      FLAG=0 ;
L303: IF J >= 4000 THEN
      GOTO L302;
      CALL SUBSTR(ADDR(PD1(J)),ADDR(TMP3),2,LEN(ADDR(TMP1))) ;
      IF ISCOMP(ADDR(TMP1),ADDR(TMP3)) = 0 THEN
      GOTO L304;
L305: K=J ;
      J=J+LEN(ADDR(PD1(J)))+1 ;
      IF PD1(K) ='1' THEN

```

```

        GOTO L320;
        J=J+LEN(ADDR(PD1(J)))+1 ;
        GOTO L303;
L320:   J=J+5 ;
        GOTO L303;
L304:   K=INDEX(ADDR(PD1(J+1)),ADDR(COD253),LEN(ADDR(TMP3))+1);
        IF K = 0 THEN
            K=INDEX(ADDR(PD1(J+1)),ADDR(COD252),LEN(ADDR(TMP3))+1);
            CALL SUBSTR(ADDR(PD1(J+1)),ADDR(TMP2),LEN(ADDR(TMP3))+1,K) ;
            CALL CVB(ADDR(TMP2),ADDR(RSLT)) ;
            IF RSLT <= L THEN
                GOTO L305;
            IF M = -1 THEN
                M=RSLT ;
                IF RSLT < M THEN
                    M=RSLT ;
                    FLAG=1 ;
                    GOTO L305;
L302:   IF FLAG = 0 THEN
            GOTO L306;
            CALL SCOPY(ZPAD('-1'),BPTR) ;
            FLAG=1 ;
            TYPE='2' ;
            RETURN;
L306:   RSLT=M ;
            CALL CVC(BPTR,ADDR(RSLT)) ;
            FLAG=1 ;
            TYPE='2' ;
            RETURN;
        END;

```

```
GBLOPN: PROC;  
    RETURN; /* DUMMY */  
END GBLOPN;
```

ZPAD: PROC (IN) RETURNS (POINTER);

```

/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                      */
/*      KEVIN C. O'KANE                            */
/*      DEPARTMENT OF COMPUTER SCIENCE             */
/*      THE UNIVERSITY OF TENNESSEE                */
/*      KNOXVILLE, TENNESSEE 37916               */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                 */
/*      JANUARY 1981                               */
/*      IBM PLI VERSION                            */
/*-----*/
/*-----*/
/*      THIS PROCEDURE RETURNS THE ADDRESS OF A STRING CREATED */
/*      FROM A SET OF INPUT CHARACTERS.                 */
/*-----*/
      DCL IN CHAR(*) VARYING;
      DCL AB FIXED BIN(15,0);
      DCL AC(2) CHAR(1) DEF AB;
      DCL K FIXED BIN;
      DCL SUBSTR BUILTIN;
      AB=0;
      DO K=1 TO LENGTH(IN);
      ZPDTMP(K)=SUBSTR(IN,K,1);
      END;
      ZPDTMP(LENGTH(IN)+1)=AC(1);
      RETURN (ADDR(ZPDTMP));
      END ZPAD;

```

DSHIFT: PROC (S1DADR,SDLIM);

```

/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                        */
/*      KEVIN C. O'KANE                             */
/*      DEPARTMENT OF COMPUTER SCIENCE              */
/*      THE UNIVERSITY OF TENNESSEE                 */
/*      KNOXVILLE, TENNESSEE 37916                */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                  */
/*      JANUARY 1981                                */
/*      IBM PLI VERSION                             */
/*-----*/
/*-----*/
/*      THIS PROCEDURE SHIFTS A STRING              */
/*-----*/

```

```

      DCL S1DADR POINTER;
      DCL SDLIM FIXED BIN(31,0);
      DCL S1D (4000) CHAR(1) BASED (S1DADR);
      DCL AB FIXED BIN(15,0);
      DCL AC(2) CHAR(1) DEF AB;
      DCL I FIXED BIN(31,0);
      DCL (P1,P2) POINTER;
      DCL BP1 FIXED BIN(31,0) DEF P1;
      DCL BP2 FIXED BIN(31,0) DEF P2;
      AB=0;
      SDLIM=SDLIM-1;
      P1=S1DADR;
      BP1=BP1+SDLIM;
      P2=S1DADR;
      BP2=BP2+SDLIM-1;
      CALL SCOPY(P1,P2);
      DO I= SDLIM-1 TO 1 BY -1;
      IF S1D(I) = AC(2) THEN GOTO L20;
      END;
L20:  SDLIM=I+1;
      RETURN;
      END DSHIFT;

```

QSCAN: PROC (STRADR,COUNT,ERR);

```

/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                        */
/*      KEVIN C. O'KANE                              */
/*      DEPARTMENT OF COMPUTER SCIENCE              */
/*      THE UNIVERSITY OF TENNESSEE                  */
/*      KNOXVILLE, TENNESSEE 37916                 */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                   */
/*      JANUARY 1981                                */
/*      IBM  PLI VERSION                             */
/*-----*/
/*-----*/
/*      PROCEDURE TO SCAN FOR QUOTED STRINGS          */
/*-----*/
      DCL STRADR POINTER;
      DCL STRING(256) CHAR(1) BASED (STRADR);
      DCL ERR FIXED BIN(31,0);
      DCL COUNT FIXED BIN(31,0);
      ERR=0;
L10:  COUNT=COUNT+1;
      IF COUNT<=LEN(ADDR(STRING)) THEN GOTO L20;
      ERR=2;
      RETURN;
L20:  IF STRING(COUNT)~='"' THEN GOTO L10;
      IF STRING(COUNT+1)~='"' THEN RETURN;
      COUNT=COUNT+1;
      GOTO L10;
      END;

```

SCN1: PROCEDURE (J,XDADR,FTMPADR,IERR,XPX);

```

/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                        */
/*      KEVIN C. O'KANE                             */
/*      DEPARTMENT OF COMPUTER SCIENCE              */
/*      THE UNIVERSITY OF TENNESSEE                 */
/*      KNOXVILLE, TENNESSEE 37916                */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                  */
/*      JANUARY 1981                                */
/*      IBM PLI VERSION                             */
/*-----*/
/*-----*/
/*      THIS PROCEDURE EXTRACTS A FILE NAME          */
/*-----*/

DCL (XDADR,FTMPADR) POINTER;
DCL IERR FIXED BIN(31),
    (I,J) FIXED BIN(31),
    XPX FIXED BIN(31);
DCL FTMP(256) CHAR(1) BASED (FTMPADR);
DCL XD(256) CHAR(1) BASED (XDADR);
DCL AC(2) CHAR DEF AB;
DCL AB FIXED BIN(15,0);
AB=0;

/*-----*/
/*      INDIRECT REFERENCE FILE SPEC                */
/*-----*/
/*      EXTRACT FILE SPEC IF ANY                    */
/*-----*/
IERR=0;
L5361: IF XD(1) = '-' THEN GOTO L5373;
/*-----*/
/*      NO FILE SPECIFIED                           */
/*-----*/
FTMP(1)=AC(1) ;
RETURN;

/*-----*/
/*      FILE SPECIFIED - CHECK IF LITERAL           */
/*-----*/
L5373: IF XD(2) ~= ''' THEN GOTO L5376;
CALL SCOPY(ADDR(XD(3)),ADDR(XD)) ;
XPX=0 ;
CALL QSCAN(ADDR(XD),XPX,IERR) ;

```

```

IF(IERR  /= 0)THEN
RETURN;
CALL SCOPY(ADDR(XD(XPX+1)),ADDR(XD(XPX))) ;
CALL SUBSTR(ADDR(XD),ADDR(FTMP),1,XPX-1) ;
IF(XPX  >= LEN(ADDR(XD)))THEN
GOTO L5368;
CALL SCOPY(ADDR(XD(XPX)),ADDR(XD)) ;
RETURN;
L5376: J=0 ;
DO I = 2 TO LEN(ADDR(XD));
IF XD(I) >= 'A' & XD(I) <= 'I' THEN GOTO L5365;
IF XD(I) >= 'J' & XD(I) <= 'R' THEN GOTO L5365;
IF XD(I) >= 'S' & XD(I) <= 'Z' THEN GOTO L5363;
ELSE IF XD(I) = '(' THEN J=J+1;
ELSE IF XD(I) = ')' THEN J=J-1;
ELSE IF XD(I) = ':' & J=0 THEN GOTO L5365;
ELSE IF XD(I) = ',' & J=0 THEN GOTO L5365;
ELSE;
END;
L5365: CALL SCOPY(ADDR(XD(2)),ADDR(FTMP)) ;
GOTO L5367;
L5366: CALL SUBSTR(ADDR(XD),ADDR(FTMP),BINARY(2),I-2) ;
L5367: IF(I  >= LEN(ADDR(XD)))THEN
GOTO L5368;
CALL SCOPY(ADDR(XD(I)),ADDR(XD)) ;
RETURN;
L5368: XD(1)=AC(1) ;
RETURN;
END;

```

INPUT: PROCEDURE ;

```

/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                      */
/*      KEVIN C. O'KANE                            */
/*      DEPARTMENT OF COMPUTER SCIENCE             */
/*      THE UNIVERSITY OF TENNESSEE                 */
/*      KNOXVILLE, TENNESSEE 37916                */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                  */
/*      JANUARY 1981                               */
/*      IBM  PLI VERSION                           */
/*-----*/
/*-----*/
/*      THIS PROCEDURE IS AN EDITOR TO LOAD AND CREATE PROGRAMS  */
/*-----*/

```

```

DECLARE 1 UNLCM EXTERNAL,
        2 DUMITEM FLOAT BINARY(53),
        2 SYMLen FIXED BINARY(31),
        2 SYMADR FIXED BINARY(31),
        2 SYMFLG FIXED BINARY(15,0),
        2 GD(257) CHAR(1),
        2 G(15) FIXED BINARY(31),
        2 P FLOAT BINARY,
        2 PD(257) CHAR(1),
        2 VD(256) CHAR(1),
        2 DOSAVE(21) CHAR(1),
        2 XXX8(5) CHAR(1),
        2 DATRIB CHAR(1),
        2 DVAR FIXED BINARY(31),
        2 XX9(6) CHAR(1),
        2 PD1LEN FIXED BINARY(31),
        2 PD1CUR FIXED BINARY(31),
        2 TRACE FIXED BINARY(31),
        2 ZD(256) CHAR(1),
        2 PD1(4000) CHAR(1),
        2 FORV(5,10) CHAR(1),
        2 FORIT(5) FIXED BINARY(1),
        2 FORLT(5) FIXED BINARY(1),
        2 FTMP(256) CHAR(1),
        2 FORIN(5) FLOAT BINARY,
        2 FORLV(5) FLOAT BINARY,
        2 FORNXT(5) FIXED BINARY(31),
        2 FORX FIXED BINARY(31),
        2 GOSUB(6,5) FIXED BINARY(31),
        2 GSBX FIXED BINARY(31);

```

```

DECLARE 1 CODESCO EXTERNAL,
        2 DUMITEM FLOAT BINARY(53),
        2 COD251( 2) CHAR(1),

```

```

      2 COD252( 2) CHAR(1),
      2 COD253( 2) CHAR(1),
      2 COD254( 2) CHAR(1),
      2 COD255( 2) CHAR(1),
      2 XTAB( 2) CHAR(1),
      2 DUMITEM4(2) FIXED BINARY(31);
DECLARE 1 IOCOMMO EXTERNAL,
      2 DUMITEM FLOAT BINARY(53),
      2 ERR FIXED BINARY(31),
      2 G55 FIXED BINARY(31),
      2 G54 FIXED BINARY(31),
      2 D2 CHAR(1),
      2 F CHAR(1);
DECLARE ICHAN FIXED BINARY(31) STATIC,
      TYPE FIXED BINARY(31) STATIC,
      NAME FIXED BINARY(31) STATIC,
      DISP FIXED BINARY(31) STATIC,
      UNIT FIXED BINARY(31) STATIC,
      IREG FIXED BINARY(31) STATIC;
DECLARE TMP1( 2) CHAR(1) STATIC ,
      ZZD FIXED BINARY STATIC,
      ZZ FIXED BINARY STATIC,
      D FIXED BINARY(31) STATIC,
      J FIXED BINARY(31) STATIC,
      K FIXED BINARY(31) STATIC;
DECLARE I FIXED BINARY(31) STATIC,P FLOAT BINARY STATIC,
      AB FIXED BIN(15,0) STATIC,
      AC(2) CHAR(1) DEF AB;
DECLARE FNAME CHAR(8) STATIC,
      CARD CHAR(255) VARYING STATIC;
DECLARE DATA FILE RECORD EXTERNAL,
      OUT FILE RECORD EXTERNAL;
AB=0;
L10:  TMP1(2)=AC(2) ;
      IREG=0 ;
/*-----*/
/*
/*      CHECK FOR FIRST PART BEING NUMERIC - IF SO, INTERPRET
/*      NUMERIC ARGUMENT
/*
/*-----*/
      IF ZD(1) = '+' | ZD(1) = '-' THEN GOTO L110;
      IF ZD(1) < '0' | ZD(1) > '9' THEN GOTO L100;
L110:  CALL CVB(ADDR(ZD),ADDR(P)) ;
      IREG=P ;
      I=0 ;
      IF IREG = 0 THEN IREG=1;
      IF IREG = 1 & ZD(1) = '-' THEN IREG = -1;
L101:  I=I+1 ;
      IF I > LEN(ADDR(ZD)) THEN GOTO L2510;
      IF ZD(I) >= '0' & ZD(I) <= '9' THEN GOTO L101;
      CALL SUBSTR(ADDR(ZD),ADDR(ZD),I,-1) ;

```

```

/*-----*/
/*
/*      ZLIST
/*
/*-----*/
L100:  IF ZD(1) ^= 'Z' THEN GOTO L2518;
      CALL SCOPY(ADDR(ZD(2)),ADDR(ZD)) ;
      IF ZD(1) ^= 'L' THEN GOTO L2512;
      IF LEN(ADDR(ZD)) ^= 1 THEN GOTO L2512;
      IF IREG ^= 0 THEN GOTO L2513;
L1401: I=24 ;
L2516: IF I >= LIMIT THEN GOTO L2510;
      CALL MTPRNT(ADDR(PD1(I))) ;
      I=I+LEN(ADDR(PD1(I)))+1 ;
      GOTO L2516;
L2513: IF IREG < 0 THEN GOTO L3017;
      J=EDIT ;
L3020: DO I=1 TO MAX(1,IREG);
      IF J >= LIMIT THEN GOTO L2510;
      CALL MTPRNT(ADDR(PD1(J))) ;
      J=J+LEN(ADDR(PD1(J)))+1 ;
      END;
      GOTO L2510;
L3017: J=EDIT ;
      IREG=-IREG ;
      DO I = 1 TO (IREG+1);
L3021: J=J-1 ;
      IF J <= 24 THEN GOTO L3020;
      IF PD1(J) = AC(2) THEN GOTO L3019;
      GOTO L3021;
L3019:
      END;
      J=J+1 ;
      GOTO L3020;
/*-----*/
/*
/*      ZRUN
/*
/*-----*/
L2512: IF ZD(1) ^= 'R' THEN GOTO L2517;
      IF ZD(2) ^= 'U' THEN GOTO L2517;
      IF ZD(3) ^= 'N' THEN GOTO L2517;
      PD1CUR=1 ;
      IF PD1LEN > 23 THEN PD1CUR=24 ;
      PD1LEN=LIMIT ;
      PD1(LIMIT+1)=AC(2) ;
      GSBX=0 ;
      FORX=0 ;
      RETURN;
/*-----*/
/*
/*      CLEAR
/*
/*-----*/

```

```

/*                                                                 */
/*-----*/
L2517:  IF ZD(1) ^= 'C' THEN
        GOTO L6000;
        IF LEN(ADDR(ZD)) ^= 1 THEN
        GOTO L200;
        IGETL=0 ;
        IGET=0 ;
        PD1CUR=1 ;
        PD1LEN=24 ;
        LIMIT=24 ;
        GSBX=0 ;
        FORX=0 ;
        EDIT=24 ;
        GOTO L2510;

/*-----*/
/*                                                                 */
/*          ZT - SET TIME SLICE                                   */
/*                                                                 */
/*-----*/
L6000:  IF ZD(1) ^= 'T' THEN GOTO L200;
        TSLICE=IREG ;
        GOTO L2510;

/*-----*/
/*                                                                 */
/*          CHANGE Z=                                             */
/*                                                                 */
/*-----*/
L200:   IF ZD(1) ^= '=' THEN GOTO L2523;
        IF IGET = 0 THEN GOTO L204;
        IF IGET < EDIT THEN GOTO L204;
        IF IGET > EDIT+LEN(ADDR(PD1(EDIT))) THEN GOTO L204;
        IF IREG < 0 THEN GOTO L204;
        IF LEN(ADDR(ZD(2))) ^= IGETL THEN GOTO L202;
        DO I= IGET TO IGET+IGETL-1;
L203:   PD1(I)=ZD(I-IGET+2) ;
        END;
        GOTO L204;
L202:   IF LEN(ADDR(ZD(2))) > IGETL THEN GOTO L205;
        DO I= IGET TO IGET+LEN(ADDR(ZD(2)))-1;
L206:   PD1(I)=ZD(I-IGET+2) ;
        END;
        DO I= IGET+IGETL TO LIMIT;
L207:   PD1(I-IGETL+LEN(ADDR(ZD(2))))=PD1(I) ;
        END;
        LIMIT=LIMIT-IGETL+LEN(ADDR(ZD(2))) ;
        PD1LEN=LIMIT ;
        IGETL=LEN(ADDR(ZD(2))) ;
        GOTO L204;
L205:   DO I=LIMIT+(LEN(ADDR(ZD(2)))-IGETL) TO IGET+IGETL BY -1;
L208:   PD1(I)=PD1(I-(LEN(ADDR(ZD(2)))-IGETL)) ;
        END;

```

```

DO I= IGET TO IGET+LEN(ADDR(ZD(2)))-1;
L209: PD1(I)=ZD(I-IGET+2) ;
      END;
      LIMIT=LIMIT+(LEN(ADDR(ZD(2)))-IGETL) ;
      PD1LEN=LIMIT ;
      PD1(PD1LEN)=AC(2) ;
      IGETL=LEN(ADDR(ZD(2))) ;
      GSBX=0 ;
      FORX=0 ;
      PD1CUR=1 ;
      GOTO L204;

/*-----*/
/*                                           */
/*          ZVERIFY                          */
/*                                           */
/*-----*/
L2523: IF ZD(1) ^= 'V' THEN GOTO L2526;
L204:  IF EDIT >= LIMIT THEN GOTO L2510;
      CALL MTPRNT(ADDR(PD1(EDIT))) ;
      GOTO L2510;

/*-----*/
/*                                           */
/*          ZADVANCE                        */
/*                                           */
/*-----*/
L2526: IF ZD(1) ^= 'A' THEN GOTO L2527;
      IF IREG ^= 0 THEN GOTO L3007;
      IF EDIT >= LIMIT THEN GOTO L2510;
      DO I=EDIT TO MAX(EDIT,LIMIT);
      IF PD1(I) = AC(2) THEN GOTO L2529;
L2528: END;
L2529: EDIT=I+1 ;
      GOTO L2510;
L3007: IF IREG < 0 THEN GOTO L3022;
      DO I=1 TO MAX(1,IREG);
      IF EDIT < LIMIT THEN GOTO L3024;
      EDIT=LIMIT+1 ;
      GOTO L2510;
L3024: EDIT=EDIT+LEN(ADDR(PD1(EDIT)))+1 ;
L3023: END;
      GOTO L2510;
L3022: IREG=-IREG ;
      DO I= 1 TO IREG+1;
L3027: EDIT=EDIT-1 ;
      IF EDIT > 24 THEN GOTO L3026;
      EDIT=24 ;
      GOTO L2510;
L3026: IF PD1(EDIT) ^= AC(2) THEN GOTO L3027;
L3025: END;
      EDIT=EDIT+1 ;
      GOTO L2510;

/*-----*/

```

```

/*                                                    */
/*          ZBEGIN                                     */
/*                                                    */
/*-----*/
L2527:  IF ZD(1) -= 'B' THEN GOTO L2531;
        EDIT=24 ;
        GOTO L2510;

/*-----*/
/*                                                    */
/*          KILL                                       */
/*                                                    */
/*-----*/
L2531:  IF ZD(1) -= 'K' THEN GOTO L2538;
        IF LEN(ADDR(ZD)) -= 1 THEN GOTO L2538;
        IF EDIT >= LIMIT THEN GOTO L2510;
        I=LEN(ADDR(PD1(EDIT)))+1 ;
        DO J= EDIT+I TO LIMIT;
L2539:  PD1(J-I)=PD1(J) ;
        END;
        LIMIT=LIMIT-I ;
        PD1LEN=LIMIT ;
        PD1(LIMIT)=AC(2) ;
        GSBX=0 ;
        FORX=0 ;
        PD1CUR=1 ;
        GOTO L2510;

/*-----*/
/*                                                    */
/*          ZFIND                                     */
/*                                                    */
/*-----*/
L2538:  IF ZD(1) -= 'F' THEN GOTO L3005;
L3003:  IF EDIT >= LIMIT THEN GOTO L3002;
        IF INDEX(ADDR(PD1(EDIT)),ADDR(ZD(2)),1) = 0 THEN GOTO L201;
        IGET=EDIT+INDEX(ADDR(PD1(EDIT)),ADDR(ZD(2)),1)-1 ;
        IGETL=LEN(ADDR(ZD(2))) ;
        CALL MTPRNT(ADDR(PD1(EDIT))) ;
        GOTO L2510;
L201:  EDIT=EDIT+LEN(ADDR(PD1(EDIT)))+1 ;
        IF EDIT >= LIMIT THEN GOTO L3002;
        GOTO L3003;
        PRNTSTR=' NOT FOUND';
L3002:  CALL MTPRNT(ADDR(PRNTSTR)) ;
        GOTO L2510;

/*-----*/
/*                                                    */
/*          ZSAVE                                       */
/*                                                    */
/*-----*/
L3005:  IF ZD(1) -= 'S' THEN GOTO L3028;
        IF ZD(2) -= 'A' THEN GOTO L3028;
        IF ZD(3) -= 'V' | ZD(4) -= 'E' THEN GOTO L3028;

```

```

IF ZD(5) ^= ' ' THEN GOTO L3028;
IF LEN(ADDR(ZD)) = 5 THEN GOTO L3032;
CALL WRITEF;
GOTO L2510;
PRNTSTR=' > FILE ERROR';
L3032: CALL MTPRNT(ADDR(PRNTSTR)) ;
GOTO L2510;

/*-----*/
/*                                          */
/*          ZLOAD                          */
/*                                          */
/*-----*/
L3028: IF ZD(1) ^= 'L' THEN GOTO L2518;
      IF ZD(2) ^= 'O' THEN GOTO L2518;
      IF ZD(3) ^= 'A' | ZD(4) ^= 'D' THEN GOTO L3028;
      IF ZD(5) ^= ' ' THEN GOTO L2518;
      IF LEN(ADDR(ZD)) = 5 THEN GOTO L3032;
      CALL SUBSTR(ADDR(ZD),ADDR(ZD),BINARY(6),-1) ;
      PD1LEN=24 ;
      CALL READF;
      LIMIT=PD1LEN ;
      GSBX=0 ;
      FORX=0 ;
      PD1CUR=1 ;
      EDIT=24 ;
      IGET=0 ;
      IGETL=0 ;
      GOTO L2510;

/*-----*/
/*                                          */
/*          INSERT                         */
/*                                          */
/*-----*/
L2518: IF ZD(1) ^= '/' THEN GOTO L3029;
      IF EDIT ^= LIMIT+1 THEN GOTO L2532;
L2521: DO I= 2 TO LEN(ADDR(ZD));
      LIMIT=LIMIT+1 ;
L2520: PD1(LIMIT)=ZD(I) ;
      END;
      LIMIT=LIMIT+1 ;
      PD1LEN=LIMIT ;
      PD1(PD1LEN)=AC(2) ;
      EDIT=PD1LEN+1 ;
      GOTO L2510;
L2532: J=LEN(ADDR(ZD(2)))+1 ;
      DO I= 1 TO LIMIT-EDIT+1;
L2533: PD1(LIMIT+J-(I-1))=PD1(LIMIT-(I-1)) ;
      END;
      LIMIT=LIMIT+J ;
      PD1LEN=LIMIT ;
      DO I= 2 TO LEN(ADDR(ZD));
      PD1(EDIT)=ZD(I) ;

```

```

L2534:  EDIT=EDIT+1 ;
        END;
        PD1(EDIT)=AC(2) ;
        EDIT=EDIT+1 ;
        GSBX=0 ;
        FORX=0 ;
        PD1CUR=1 ;
        GOTO L2510;

/*-----*/
/*                                          */
/*          DEFAULT                      */
/*                                          */
/*-----*/
L3029:  IF ZD(1) /= XTAB(1) THEN GOTO L9999;
        DO I= LEN(ADDR(ZD))+2 TO 2 BY -1;
L3006:  ZD(I)=ZD(I-1) ;
        END;
        ZD(1)='/' ;
        GOTO L2518;

/*-----*/
/*                                          */
/*          DIRECT MODE                  */
/*                                          */
/*-----*/
L2510:  ERR=0 ;
        RETURN;

/*-----*/
/*                                          */
/*          ERROR                        */
/*                                          */
/*-----*/
L9999:  ERR=1 ;
        RETURN;
        END;

```

KILL1: PROCEDURE (TMP2PTR);

```

/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                      */
/*      KEVIN C. O'KANE                            */
/*      DEPARTMENT OF COMPUTER SCIENCE             */
/*      THE UNIVERSITY OF TENNESSEE                */
/*      KNOXVILLE, TENNESSEE 37916               */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                 */
/*      JANUARY 1981                               */
/*      IBM PLI VERSION                            */
/*-----*/
/*-----*/
/*      KILL LOOP                                  */
/*      USED BY FORM OF KILL COMMAND WHICH KILLS ALL EXCEPT A */
/*      CERTAIN LIST.                               */
/*      THE "NO-KILL" LIST IS IN FTMP              */
/*-----*/

```

```

DECLARE 1 CODESCO EXTERNAL,
        2 DUMITEM FLOAT BINARY(53),
        2 COD251( 2) CHAR(1),
        2 COD252( 2) CHAR(1),
        2 COD253( 2) CHAR(1),
        2 COD254( 2) CHAR(1),
        2 COD255( 2) CHAR(1),
        2 XTAB( 2) CHAR(1),
        2 DUMITEM4(2) FIXED BINARY(31);

```

```

DECLARE 1 PRSECOM EXTERNAL,
        2 DUMITEM FLOAT BINARY(53),
        2 IORG FIXED BINARY(31),
        2 XPX FIXED BINARY(31),
        2 XPY FIXED BINARY(31),
        2 TOPX FIXED BINARY(31),
        2 T2 CHAR(1),
        2 VID(257) CHAR(1),
        2 XD(256) CHAR(1),
        2 XDSAVE(256) CHAR(1),
        2 IERR FIXED BINARY(31),
        2 DTYPE CHAR(1),
        2 DUMITEM2( 1)FLOAT BINARY;

```

```

DECLARE 1 UNLCM EXTERNAL,
        2 DUMITEM FLOAT BINARY(53),
        2 SYMLen FIXED BINARY(31),
        2 SYMADR FIXED BINARY(31),
        2 SYMFLG FIXED BINARY(15,0),
        2 GD(257) CHAR(1),
        2 G(15) FIXED BINARY(31),
        2 P FLOAT BINARY,

```

```

2 PD(257) CHAR(1),
2 VD(256) CHAR(1),
2 DOSAVE(21) CHAR(1),
2 XXX8(5) CHAR(1),
2 DATRIB CHAR(1),
2 DVAR FIXED BINARY(31),
2 XX9(6) CHAR(1),
2 PD1LEN FIXED BINARY(31),
2 PD1CUR FIXED BINARY(31),
2 TRACE FIXED BINARY(31),
2 ZD(256) CHAR(1),
2 PD1(4000) CHAR(1),
2 FORV(5,10) CHAR(1),
2 FORIT(5) FIXED BINARY(1),
2 FORLT(5) FIXED BINARY(1),
2 FTMP(256) CHAR(1),
2 FORIN(5) FLOAT BINARY,
2 FORLV(5) FLOAT BINARY,
2 FORNXT(5) FIXED BINARY(31),
2 FORX FIXED BINARY(31),
2 GOSUB(6,5) FIXED BINARY(31),
2 GSBX FIXED BINARY(31);
DECLARE TMP2PTR POINTER;
DECLARE TMP2(256) CHAR(1) BASED (TMP2PTR),
      J FIXED BINARY(31) STATIC,
      K FIXED BINARY(31) STATIC,
      I FIXED BINARY(31) STATIC,
      IS FIXED BINARY(31) STATIC,
      IIS FIXED BINARY(31) STATIC;
      IS=1;
L5421: CALL CONCAT(ADDR(FTMP),ZPAD(' '),ADDR(FTMP)) ;
      I=SYMLEN ;
L5422: J=1 ;
      IF I < 4000 THEN GOTO L5423;
      RETURN;
L5423: IF INDEX(ADDR(FTMP),ADDR(PD1(I+1)),J) = 0 THEN GOTO L5424;
      J=INDEX(ADDR(FTMP),ADDR(PD1(I+1)),IS);
      IF FTMP(J-1) = ' ' | J = 1 THEN GOTO L5425;
      J=INDEX(ADDR(FTMP),ZPAD(' '),J) ;
      GOTO L5423;
L5425: K=INDEX(ADDR(FTMP(J)),ZPAD(' '),IS) ;
      IIS=K-1;
      CALL SUBSTR(ADDR(FTMP(J)),ADDR(TMP2),IS,IIS);
      IF ISCOMP(ADDR(TMP2),ADDR(PD1(I+1))) = 0 THEN GOTO L5424;
L5427: DTYPE=2 ;
      IF PD1(I) = '1' THEN DTYPE=1 ;
      I=LEN(ADDR(PD1(I)))+1+I ;
      IF DTYPE = 2 THEN GOTO L5443;
      I=I+4+1 ;
      GOTO L5422;
L5443: I=LEN(ADDR(PD1(I)))+1+I ;
      GOTO L5422;

```

```

L5424:  IF INDEX(ADDR(PD1(I+1)),ADDR(COD251),IS) = 0
        THEN GOTO L5426;
        K=INDEX(ADDR(PD1(I+1)),ADDR(COD251),IS) ;
        IIS=K-1;
        CALL SUBSTR(ADDR(PD1(I+1)),ADDR(TMP2),1,K-1);
        IF INDEX(ADDR(FTMP),ADDR(TMP2),J) = 0 THEN GOTO L5426;
        K=INDEX(ADDR(FTMP),ADDR(TMP2),J) ;
        IF FTMP(K-1) ^= ' ' THEN GOTO L5426;
        IF LEN(ADDR(TMP2)) ^= (INDEX(ADDR(FTMP(K)),ZPAD(' '),IS)-1)
            THEN GOTO L5426;
        GOTO L5427;
L5426:  CALL SCOPY(ADDR(PD1(I+1)),ADDR(TMP2)) ;
        SYMFLG=4 ;
        CALL SYM(ADDR(TMP2),ZPAD(' '),ITYPE);
        GOTO L5421;
        END;

```

READF: PROCEDURE ;

```

/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                        */
/*      KEVIN C. O'KANE                             */
/*      DEPARTMENT OF COMPUTER SCIENCE              */
/*      THE UNIVERSITY OF TENNESSEE                  */
/*      KNOXVILLE, TENNESSEE 37916                 */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                  */
/*      JANUARY 1981                                */
/*      IBM PLI VERSION                             */
/*-----*/
/*-----*/
/*      THIS PROCEDURE READS FROM A FILE            */
/*-----*/
      DECLARE 1 UNLCM EXTERNAL,
              2 DUMITEM FLOAT BINARY(53),
              2 SYMLEN FIXED BINARY(31),
              2 SYMADR FIXED BINARY(31),
              2 SYMFLG FIXED BINARY(15,0),
              2 GD(257) CHAR(1),
              2 G(15) FIXED BINARY(31),
              2 P FLOAT BINARY,
              2 PD(257) CHAR(1),
              2 VD(256) CHAR(1),
              2 DOSAVE(21) CHAR(1),
              2 XXX8(5) CHAR(1),
              2 DATRIB CHAR(1),
              2 DVAR FIXED BINARY(31),
              2 XX9(6) CHAR(1),
              2 PD1LEN FIXED BINARY(31),
              2 PD1CUR FIXED BINARY(31),
              2 TRACE FIXED BINARY(31),
              2 ZD(256) CHAR(1),
              2 PD1(4000) CHAR(1),
              2 FORV(5,10) CHAR(1),
              2 FORIT(5) FIXED BINARY(1),
              2 FORLT(5) FIXED BINARY(1),
              2 FTMP(256) CHAR(1),
              2 FORIN(5) FLOAT BINARY,
              2 FORLV(5) FLOAT BINARY,
              2 FORNXT(5) FIXED BINARY(31),
              2 FORX FIXED BINARY(31),
              2 GOSUB(6,5) FIXED BINARY(31),
              2 GSBX FIXED BINARY(31);
      DECLARE K FIXED BINARY(31) STATIC,
              J FIXED BINARY(31) STATIC,
              I FIXED BINARY(31) STATIC,

```

```

        FNAME CHAR(8) STATIC,
        CARD CHAR(255) VARYING STATIC;
DECLARE DATA FILE RECORD EXTERNAL,
        (SUBSTR,LENGTH) BUILTIN,
        AB FIXED BIN(15,0) STATIC,
        AC(2) CHAR(1) DEF AB;

AB=0;
J=24;
FNAME=' ';
DO I = 1 TO 8;
    IF ZD(I) = AC(2) THEN LEAVE;
    SUBSTR(FNAME,I,1)=SUBSTR(ZD(I),1,1);
END;
OPEN FILE(DATA) TITLE(FNAME) INPUT;
ON ENDFILE(DATA) GOTO CLSE;
NXT:  READ FILE(DATA) INTO (CARD);
      I=LENGTH(CARD);
      DO K = J TO J+I-1;
          PD1(K) = ' ';
          SUBSTR(PD1(K),1,1)=SUBSTR(CARD,K-J+1,1);
      END;
      PD1(J+I) = AC(2);
      J = J+I+1;
      GOTO NXT;
CLSE:  CLOSE FILE(DATA);
      PD1LEN=J;
      RETURN;
      END READF;

```

WRITEF: PROCEDURE ;

```

/*-----*/
/*      MUMPS INTERPRETER - MULTI-USER SYSTEM      */
/*      COPYRIGHT (C) 1981 BY                      */
/*      KEVIN C. O'KANE                          */
/*      DEPARTMENT OF COMPUTER SCIENCE            */
/*      THE UNIVERSITY OF TENNESSEE                */
/*      KNOXVILLE, TENNESSEE 37916              */
/*      ALL RIGHTS RESERVED - DUPLICATION PERMITTED WITH INCLUSION*/
/*      OF THIS COPYRIGHT NOTICE.                */
/*      JANUARY 1981                             */
/*      IBM  PLI VERSION                          */
/*-----*/
/*-----*/
/*      THIS PROCEDURE WRITES TO A FILE            */
/*-----*/
DECLARE 1 UNLCM EXTERNAL,
        2 DUMITEM FLOAT BINARY(53),
        2 SYMLen FIXED BINARY(31),
        2 SYMADR FIXED BINARY(31),
        2 SYMFLG FIXED BINARY(15,0),
        2 GD(257) CHAR(1),
        2 G(15) FIXED BINARY(31),
        2 P FLOAT BINARY,
        2 PD(257) CHAR(1),
        2 VD(256) CHAR(1),
        2 DOSAVE(21) CHAR(1),
        2 XXX8(5) CHAR(1),
        2 DATRIB CHAR(1),
        2 DVAR FIXED BINARY(31),
        2 XX9(6) CHAR(1),
        2 PD1LEN FIXED BINARY(31),
        2 PD1CUR FIXED BINARY(31),
        2 TRACE FIXED BINARY(31),
        2 ZD(256) CHAR(1),
        2 PD1(4000) CHAR(1),
        2 FORV(5,10) CHAR(1),
        2 FORIT(5) FIXED BINARY(1),
        2 FORLT(5) FIXED BINARY(1),
        2 FTMP(256) CHAR(1),
        2 FORIN(5) FLOAT BINARY,
        2 FORLV(5) FLOAT BINARY,
        2 FORNXT(5) FIXED BINARY(31),
        2 FORX FIXED BINARY(31),
        2 GOSUB(6,5) FIXED BINARY(31),
        2 GSBX FIXED BINARY(31);
DECLARE K FIXED BINARY(31) STATIC,
        J FIXED BINARY(31) STATIC,
        I FIXED BINARY(31) STATIC,

```

```

        FNAME CHAR(8) STATIC,
        CARD CHAR(255) VARYING STATIC;
DECLARE OUT FILE RECORD EXTERNAL,
        (SUBSTR,LENGTH) BUILTIN,
        AB FIXED BIN(15,0) STATIC,
        AC(2) CHAR(1) DEF AB;

AB=0;
FNAME=' ';
DO I = 1 TO 8;
    IF ZD(I+5) = AC(2) THEN LEAVE;
    SUBSTR(FNAME,I,1)=SUBSTR(ZD(I+5),1,1);
END;
OPEN FILE(OUT) TITLE(FNAME) OUTPUT;
I=24;
CARD='';
K=0;
DO J=I TO LIMIT-1;
    IF PD1(J) = AC(2) THEN DO;
        WRITE FILE(OUT) FROM(CARD);
        CARD='';
        K=0;
        IF J=LIMIT-1 THEN LEAVE;
    END;
    K=K+1;
    CARD=CARD||SUBSTR(PD1(J),1,1);
END;
IF K>0 THEN WRITE FILE(OUT) FROM(CARD);
CLOSE FILE(OUT);
EXIT: RETURN;
END WRITEF;
L9999 END MUMPS;

```

VITA

Jean K. Gaffney was born in Oxfordshire, England on March 30, 1941. She was educated at the Ursuline High School, Essex, England. She attended the Northampton College of Advanced Technology, London for three years on a part time basis and was awarded a certificate in Numerical Analysis, Computing and Statistics. She has twenty three years experience in the computing field and is currently employed as a Senior Systems Programmer Analyst at the University of Tennessee Computing Center.