June, 1973

To the Graduate Council:

I am submitting herewith a thesis written by Eric Allen Ferguson entitled "TWLVE - Two Way Lists with Variable Length Elements." I recommend that it be accepted for nine quarter hours of credit in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Ronald P. Loiacono
Major Professor

We have read this thesis
and recommend its acceptance:

Harvey P Carter

Robert M. Aiken

Accepted for the Council:

Hilton A. Smith
Vice Chancellor for
Graduate Studies and Research

TWLVE - TWO WAY LISTS WITH VARIABLE LENGTH ELEMENTS

A Thesis

Presented to

the Graduate Council cf

The University of Tennessee

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Eric Allen Ferguson

August 1973

ACKNOWLEDGMENTS

For their guidance and support, the author expresses his appreciation to the members of his committee: Dr. Ronald P. Leinius, Dr. Harvey P. Carter and Dr. Robert M. Aiken. The author is indebted to the Oak Ridge National Laboratory for assembling and publishing this thesis. Oak Ridge National Laboratory is operated by Union Carbide Corporation for the Atomic Energy Commission.

The author would also like to acknowledge the use of the IBM/360 and the PDP/10 in the preparation of this text. The text was created on the PDP/10 using TECO and BOXOFF and transferred to the IBM/360 using RUNOVR, where it was printed using a TN print train. These two computers are part of the computer center at Oak Ridge National Laboratory.

1101035

ABSTRACT

The purpose of this thesis was to develop a package of computer subprograms which can be used by an IBM/360 FORTRAN programmer to perform basic list processing operations. The subprograms are written using IBM/360 Assembler language and FORTRAN.

An introduction to list processing is given including a discussion of lists, list structures and brief descriptions of some existing list processing languages. Next, TWLVE is described. There are sections which describe how memory is managed by TWLVE, how to build lists and list structures using TWLVE routines, how to traverse TWLVE lists and list structures, how the user can insert and delete elements of TWLVE lists and how TWLVE lists can be deleted.

The calling sequence for all TWLVE routines is given as well as the program logic for all routines. The Appendixes contain error descriptions, TWLVE element descriptions, flow charts, program listings and a FORTRAN program which uses TWLVE to solve a sample problem.

TABLE OF CONTENTS

CHAPTER		PAGE
INTRODUCTION		1
I.	LIST PROCESSING	5
	Lists	6
	List Structures	11
	Brief Summary of Some List Processing Languages	14
	IPL	15
	LISP	16
	SLIP	17
II.	TWLVE	20
	General Discussion	20
	Guide for Using TWLVE	30
	Creating lists and list structures	31
	Traversing lists and list structures	33
	Inserting and deleting elements and deleting lists	36
	Trace Feature	39
	Programming Considerations	41
	Routine names	41
	Modifying COMMON	42
	General hints	42
	System Considerations	44
	Operating system	44

CHAPTER PAGE

Region size. 45
Execution time estimates 45
Control cards. 45
System errors. 46
Summary. 46
III. ROUTINE DESCRIPTIONS 50
Maintaining the List of Available Space. 50
Initializing the list of available space . . . 52
Releasing the space allocated for lists. . . . 53
Obtaining a block of memory from the LAS . . . 53
Connecting the LAS fragments 57
Returning list space to the LAS. 59
Returning element space to the LAS 59
Interrogating the LAS. 60
Defining and Building Two-way Lists. 61
Defining a list. 62
Inserting an element into a list 65
Inserting a LISTR into a list. 67
Deleting an element from a list. 70
Deleting a list. 71
Traversing Lists and Fetching Control Information 72
Traversing a list structure. 73
Traversing a list. 75
Fetching the address of a LCB. 76

CHAPTER PAGE

Fetching the contents of a pcinter 77
Fetching the ID of an element. 77
Fetching the length of an element's user data field. 78
Fetching the address of an element's user data field. 78
Logical Operations 78
AND operation. 79
Inclusive OR operation 80
Exclusive OR operation 80
Testing under mask 80
Logical comparing. 81
Moving data. 82
Loading the address of a variable. 82
Push Down Stores 83
Initializing memory for push down stores . . . 84
Defining a push down store 85
Deleting a push down store 87
Pushing an element into a push down store. . . 88
Popping an element from a push down store. . . 90
Releasing memory allocated for push down stores 91
Examining push down stores 92

CHAPTER PAGE

BIBLIOGRAPHY 95

APPENDIXES 97

A. ERROR MESSAGES 98

B. ELEMENT DESCRIPTIONS 99

C. FLOW CHARTS. 104

D. PROGRAM LISTINGS 132

E. TWLVE EXAMPLE. 177

VITA 184

INTRODUCTION

TWLVE is a list processing system which gives an IBM/360 FORTRAN programmer the ability to perform list processing functions. The purpose for implementing this system was to provide the computer science student with a useful tool both at the user level and at the implementor level and to provide the applications programmer with a set of subroutines which can aid in information processing.

Lists are used to represent structural relationships. List processing systems are designed to create and operate with list structures. List structures offer important advantages over other types of information structures (i.e. arrays) in many types of computer applications. Some examples of these applications are sparse matrix storage, storage of polynomial equations, discrete simulation and table building.

A sparse matrix is a matrix of large order in which most of the elements are zero. If an array structure is used to store a sparse matrix then N (the number of rows) times M (the number of columns) elements are required. When representing a sparse matrix, using list structures, only the non-zero values require an element; however, each element uses several pointers. Assuming that each pointer takes the same amount of core as a matrix value, then the storage required, when using list structures, is more

economical if J (the number of pointers plus one) times I (the number of non-zero matrix values) is less than N times M.

Polynomials can be represented in core most naturally by using list structures. This is because polynomials grow to unpredictable sizes and because many polynomials may need to be stored in memory at the same time. Typically, elements used to represent polynomial terms have a field for the coefficient of the term, fields for the exponents of all variables and a link pointing to the next term.

Both discrete simulation and table building applications often require that entries be in some systematic order and that entries be periodically inserted and deleted. If the desired data is in memory using array structures then the arrays must be reorganized any time an element is inserted or deleted. If the data is stored using list structures then the links of only two elements (if the lists are two-way) must be changed when an element is inserted or deleted.

The information presented in this thesis can be useful to the computer science student, as a user of the TWLVE system, in several computer science areas. In an operating systems course, TWLVE can be used for such purposes as dynamically building control blocks and dynamically allocating buffers. In an information retreival course, TWLVE can be used for storing structured data. In a

compiler design course, TWLVE can be used to build symbol tables and to facilitate the parsing of expressions and statements. At the implementor level, the computer science student could add "higher level" modules to the TWLVE system. For example, as a programming project a general routine to read lists into memory and tc write lists from memory or a routine to search for a string of data could be undertaken. Also, at the implementor level, the computer science student could modify various list processing techniques used by TWLVE (available space management, etc.) and then attempt to find which techniques are better for various applications.

The material in this thesis is organized in the following manner. Chapter I contains a general introduction to list processing. It is recommended reading for those who are not familiar with list processing terminology. There are sections in Chapter II which should be of primary interest to the TWLVE user. These sections include a guide for using TWLVE, a discussion of the TWLVE trace feature, programming considerations, system considerations and the TWLVE calling sequences which are summarized in Chapter II and completely described in Chapter III. The TWLVE user should also be interested in the FORTRAN main program, described in Appendix E, which uses TWLVE to solve a sample problem. The detailed information given in Chapter III is provided, along with the flow charts in Appendix C and the

program listings in Appendix D, for those who are interested in the implementation of TWLVE.

There are two subsets of the TWLVE system which can be used for non-list-processing purposes. One subset is the routine which is used to perform logical operations and the other is the routine which is used to create and operate on push down stores. The examples in this publication use these two routines for list processing purposes, but either may be used independently.

CHAPTER I

LIST PROCESSING

Array structures and list structures represent commonly used methods for storing multi-item data in main memory. Arrays are made up of elements which are stored in consecutive locations of memory. This means that the elements of an array can be indexed by computing each address in the following method.

$$A(I) = B + I * L ,$$

where:

A(I) is the address of the Ith element in the array,

B is the address of the 0th element in the array (base address),

I indicates which element of the array is to be referenced, and

L is the length (in address units, i.e. bytes, words, etc.) of each element.

A major advantage of using an array for data storage is the speed with which elements of the array can be addressed. A primary disadvantage is that all of the elements in an array must be of the same length.

A list, on the other hand, may or may not have its elements stored in consecutive locations. The elements of a

list are linked together by the use of pointers. One advantage of using lists is that elements can be variable length. Another advantage is that it is easy to insert or delete an element. This is because only the pointers have to be changed whenever an element is inserted into a list or when an element is deleted from a list. A disadvantage is that it takes I steps to find the Ith element of a list.

A list can be classified as a simple list or as a list structure. The remainder of this chapter contains a general discussion of simple lists, followed by a discussion of list structures and finally, a brief summary of some pre-existing list processing languages.

I. LISTS

Definition 1.1: An element is a piece of data which can be divided into subelements and which may be fixed length or variable length. An element is stored in consecutive locations in memory. The length of an element can be expressed in bits, bytes, words, or decimal digits. Fixed length means that all elements have the same length and variable length means that all elements do not have the same length. Elements are sometimes called "items", "nodes", "records", "entities", "beads" or "packets".

Definition 1.2: A field is a subelement. Fields are used for storing parts of the data as well as for pointers and as flags. Fields can be used for storing characters ,

fixed point numbers (binary), floating point numbers, decimal constants, etc.

Definition 1.3: A pointer is a field in an element which contains a reference to another element. The reference may be a relative address or an absolute address. Pointers are often called "links".

Definition 1.4: A flag is a field in an element which is used to indicate a special condition. A flag can be one bit in length in which case it is said to be set or not set (on or off, 1 or 0, etc.).

Definition 1.5: A simple list (usually simply called list) is a set of elements which are linked together, in some way, by the use of pointers. Note that a list may or may not be stored in consecutive memory locations, so that it takes I steps to find the Ith element in the list, instead of one step, as it takes to find the Ith element of an array.

Definition 1.6: The head of a list is the first element in the list. In some cases the head of a list will have a label (name). Usually, a flag is set to indicate which element is the head of a list.

Definition 1.7: The null pointer is used to indicate that an element is the last element in a list and therefore doesn't point to another element. The symbols $$$$ are used

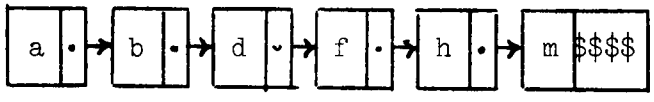
in this paper to represent a null pointer.

Definition 1.8: An empty list is a list that contains no elements.

A simple list can be denoted by the following:

A: (a, b, d, f, h, m) .

This is a list with six elements; the first element is called a, the second element is called b, the last element is called m and the name of the list is A. This same list can be graphically shown as:



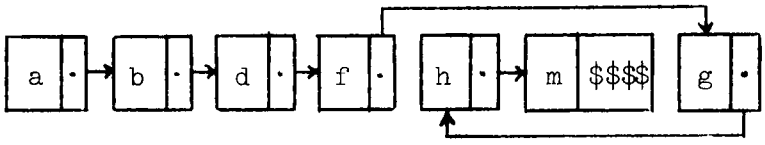
Some of the operations which might need to be performed on simple lists are (1):

1) find the Ith element of the list,
2) insert a new element into the list,
3) delete an element from the list,
4) concatenate lists,
5) divide a list into two or more lists,
6) copy a list, and
7) count the number of elements in a list.

For example, the element g is inserted into list A after the element f. The result is

A': (a, b, d, f, g, h, m) .

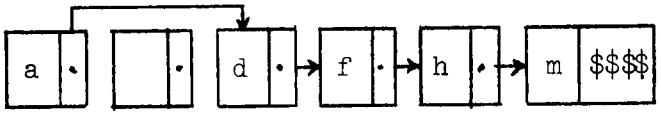
or graphically,



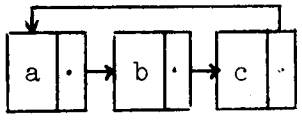
Now, suppose the element b is deleted from the list called A. The result is now

A': (a, d, f, h, m) ,

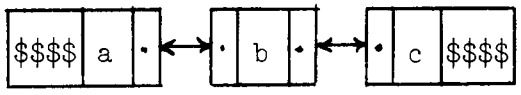
or graphically,



The lists described above are one-way linear lists, other types of simple lists are possible. One of these is a one-way circular list, which is a list whose last element points to the first element of the list.



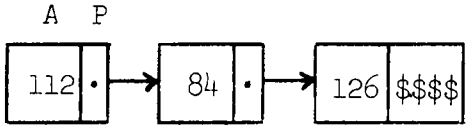
Another type of simple list is a two-way linear list in which each element has two pointers: one pointing to the element preceding and one pointing to the element following.



A final type of simple list is a list with multiple pointers or links.

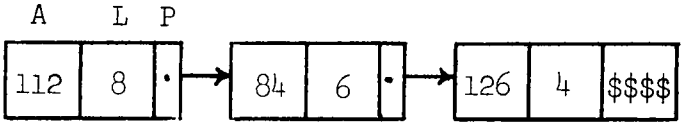
When using lists for storing items in the computer, one

very important consideration is how to keep track of space which is available for storage. One way to do this is to keep a list of available space (or empty space). When an item is to be added to a list, which consists of fixed length elements, the first element of the list of available space is checked to see what storage location(s) is empty. That element is then deleted from the list of available space and is added to the list. Conversly, when an element is deleted from a list, that element is returned to the list of available space. Graphically, a list of available space, when the elements of the list are fixed length, can be sketched as follows.



Where the first field (call it the A field) contains the address of the location that is available and the second field (call it the P field) contains a pointer to the next element of the list.

If elements of the list are variable length, then the list of available space elements should contain an additional field (call it the L field) which tells how many consecutive words (or bytes, bits, decimal digits, characters, etc.) are available beginning at the location specified in the A field.



When deciding where to store an item, which is added to a list with variable length elements, there are two alternatives: the "First Fit" and the "Best Fit". First, the list of available space could be searched until the first element is found whose L field indicates that enough consecutive memory is available to add the item to the list. If the item requires less memory than is available then the element will remain in the list of available space, but with the contents of the L field reduced by the length of the item and with the contents of the A field shifted appropriately. The second alternative is to try to find the best match by looking through the entire list of available space.

II. LIST STRUCTURES

Definition 1.9: A list structure is a set of elements (entries) that are linked together and whose elements can point to another list. Thus, a list structure can contain (be made up of) elements and lists. A simple list is a list structure whose entries are all elements.

Defintion 1.10: A sublist is a list which is contained in another list. The sublist of a list may have sublists which in turn may have sublists and so on.

List structures can be denoted by the following, sometimes called S-expressions (2):

A: (a, b, D, e, G) .

where, A is the name of the list, the entries a, b, and e are elements, and the entries D and G are sublists. Note, the convention used in this paper is that lower case letters are used as names for elements and capital letters are used as names for lists. If the lists D and G are made up of the following entries:

D: (i, j) ,

G: (m, H, n) ,

and,

H: (o, p, q) ,

then equivalent S-expressions for A are:

A: (a, b, (i, j), e, (m, H, n)) ,

and

A: (a, b, (i, j), e, (m, (o, p, q), n)) ,

or

A: (a, b, D: (i, j), e, G: (m, H, n)) ,

and

A: (a, b, D: (i, j), e, G: (m, H: (o, p, q), n)) .

There is a very important difference when returning storage to the list of available space for list structures rather than for simple lists. Since a list may be contained as a sublist in more than one list, there must be a way to decide when to return space occupied by a list to the list

of available space. For example, let A and G be defined as:

A: (a, b, D) ,

and

G: (e, f, D) .

If list A is deleted, the space occupied by D can't be returned to the list of available space because D is also an entry in list G.

There are two methods which can be used for maintaining the list of available space for list structures. In either case it is the user's responsibility to issue delete commands and it is the list processing system's responsibility to decide which memory locations are available. One method is to use reference counters. A field in the head element of a list is used as a counter. Any time the list is used as a sublist the counter is increased by one. If the list (used as a sublist) is deleted from a list, then the counter is decreased by one. If the counter reaches zero then the space occupied by the list is returned to the list of available space. A second method is by the so called "garbage collection" technique. This technique makes use of a flag which is set if the entry is being used. When more space is needed, the "garbage collector" creates a new list of available space by adding elements to the list of available space, if the flag, at that entry, is not set. Note that it may be necessary to traverse all lists to be sure that an entry is not used.

The interested reader is referred to Knuth (1).

III. BRIEF SUMMARY OF SOME LIST PROCESSING LANGUAGES

Programming languages which are used primarily for the manipulation of lists are known as list processing languages. These programming languages may be independent computer languages or extensions of existing languages (such as FORTRAN). Two well known examples of the first type are IPL (Information Processing Language) developed by A. Newell, C. Shaw, and H. Simon (6) and LISP designed by J. McCarthy (1,5). An example of a list processing language which is a set of FORTRAN subroutines is SLIP which was developed by J. Weizenbaum (4). The earliest of these list processing languages, IPL, was developed near the middle 1950's. Since then new versions have been designed. The first list processing languages used only one computer word for storing elements of a list. Later, the idea of using several consecutive words for one element was introduced. The traversing of lists also evolved over a period of time. At first lists were linked by the use of a single pointer. Later, two-way linked lists, circular lists, as well as other multi-linked structures were introduced. The remainder of this chapter contains a brief summary of some list processing languages. If a more complete summary and/or comparison of these languages is desired, the article by Bobrow and Raphael (5) should be read.

IPL

In IPL both the program and the data are lists. The elements in data lists are divided into three fields: the name field, the symbol (data) field and the pointer field. The elements in instruction lists have two additional fields: the PQ field, which describes the operation to be performed by the instruction, and the comments field. An instruction list is made up of elements which contain instructions for input, output, and list manipulating. Input may consist of the name of a data list or the name of a location in memory that contains the name of the data list. To help facilitate the processing of lists, many routines are available for list operations that are used often (such as inserting elements into a list or deleting an element from a list).

List structures, in IPL, are defined by the programmer as a list of symbols. All symbols are regional or local. Regional symbols are used as the name of a list or of an element in a list. Local symbols are used for naming sublists which are common only to the list structure in which they are contained. The usual IPL internal data structure consists of one pointer which links to the next element in the list. The elements of IPL are fixed length.

In IPL the programmer is responsible for deleting elements or lists which are not needed, and the failure to do so is a programming error that results in the use of

unneeded memory. IPL has provisions for storing data lists and program lists on auxiliary storage and then to read the information when necessary. Other features of IPL are that recursive operations are definable and usable, a program can modify itself, no mnemonic symbols are available, and the arithmetic facility is poor because of awkward representations and slow interpretive execution (5).

LISP

Elements in LISP are called atomic symbols or atoms. The programmer has the ability to link atomic symbols as binary trees or as lists. Elements are fixed length and can store up to 30 characters. The list notation of LISP can be used to represent atoms or lists. An example is

(building, (house, store, (big, small))) ,

where the list is made up of two entries; the first is an atom and the second is a sublist which contains a sublist. Sublists are not named. In LISP, the atom NIL is used as a null pointer. Internally, the representation used by LISP is similiar to that used by IPL. Two fields are used as pointers. Atomic elements are marked by a flag.

LISP uses a complicated system for keeping track of which locations in memory have been used. The programmer is not responsible for returning deleted elements and/or lists to the list of available space. LISP uses a "garbage collector" to create a new list of available space when the old list of available space is empty. This "garbage

collection" is time consuming but it does free the programmer from the bookkeeping which would otherwise be needed. LISP has no provisions for using auxiliary storage.

LISP programs consist of functions, rather than sequences of instructions. Arguments are given to the functions which in turn construct data structure values. There are three ways functions may be combined: composition, conditional expressions, and recursion. The value of a function may itself be an argument so that it must be evaluated before the main function can be evaluated. This nesting of functions is called composition. Conditional expressions (similar to ALGOL) as well as recursion may also be used.

LISP programs can modify themselves and mnemonic symbols can be used. Arithmetic is performed after expressions have been converted to polish notation. One final feature of LISP is the language's trace feature which is a useful debugging tool.

SLIP

Elements in SLIP are made up of two computer words and contain two pointers. The first pointer is called the forward link and the second pointer is called the backward link. Lists in SLIP are symmetric because any operation that can be carried out on top of the list can also be carried out on the bottom of the list. The first word of the element is divided into three fields: the identifier

field (ID), the left link (LNKL) and the right link (LNKR). The second word of a field is usually used for storing the data. The FORTRAN callable subroutines which are used for manipulating SLIP elements are called primitives. The primitives are written in machine language and are the only machine dependent portions of SLIP. Primitives handle the following tasks.

1) Extract the ID portion of an element.
2) Extract the machine address stored in LNKL or in LNKR.
3) Extract the data (second word) in an element.
4) Find the machine address of an element.
5) Directly set the values of the fields in an element.
6) Indirectly set the values of the fields in an element.
7) Store a full length word of data in an element

An example of the use of a primitive is (4):

```
CALL SETIND(I,L,R,CELL) ,
```

which stores I in the ID field of the element named CELL, L in the LNKL field of that element and R in the LNKR field.

The programmer must write DIMENSION statements in the FORTRAN calling program to reserve space for lists. In particular, a singly dimensioned array must be set up for the list of available space. All elements of other lists

will be taken from this list of available space and returned to it at the appropriate time. SLIP lists are characterized by a set of elements such that the LNKR field of each element points to the next-to-the-right element and the LNKL field of each element points to the next-to-the-left element.

SLIP allows lists to be sublists in other lists. It uses a reference counter in the HEADER (head element) of a list to determine when the space occupied by the list can be returned to the list of available space. A very nice feature of SLIP is that, since the lists are linked in both directions, lists can be returned to the list of available space by simply returning the HEADER (it points to the first and last elements of the list). Recursion is also possible. The routines which handle recursion are written in machine language because, essentially, they must save the normal return address generated by FORTRAN. Other things to note about SLIP are the following.

1) Data may be written to and read from auxiliary storage.
2) The form of the program is that of FORTRAN with pseudo-functions.
3) Since SLIP is a set of FORTRAN subroutines the arithmetic features of FORTRAN are available.

CHAPTER II

TWLVE

I. GENERAL DISCUSSION

TWLVE is a package of IBM/360 assembly language and FORTRAN routines which give a FORTRAN programmer the ability to do list processing. TWLVE lists are two-way linear lists with variable length elements. An independent subset of TWLVE creates and performs operations upon pushdown stores (PDS). Many of the features of SLIP (4) are incorporated into TWLVE. The remainder of this section gives a general discussion of TWLVE. Other sections in this chapter give a guide for using TWLVE, a description of the trace feature available in TWLVE, programming considerations, system considerations, and a summary. Chapter III contains detailed descriptions of the TWLVE routines. Flow charts for most of TWLVE are given in Appendix C, and the coding for all routines is contained in Appendix D. Appendix A contains a description of the error messages and Appendix B gives a description of all list elements and all PDS elements. Finally, Appendix E has a FORTRAN program which uses TWLVE routines to solve a sample problem.

There are three types of elements in TWLVE lists: LCB, simple element, and LISTR. Each element contains the standard control fields which are: an identification field

(ID), a length field (LEN), a forward pointer (P1), and a backward pointer (P2). ID identifies the type of element, LEN contains the element length, P1 points to the next element in the list and P2 contains a pointer which links to the previous element. A list control block (LCB) serves as the head element for a list. In addition to the standard control fields, a LCB has a NAME field, a MARK field, a COUNT field, a P4 field and a P5 field. NAME contains the list name. MARK is used to indicate that a list can, or can't, be deleted automatically. COUNT contains a count of the number of times that the list called NAME is used as a sublist. P4 contains a pointer that links to the previous LCB and P5 cotains a pointer that links to the next LCB. The list control blocks are linked with the list names in ascending collating sequence. A simple list element (sometimes called element or list element) contains, in addition to the standard control fields, a user data field. The user data field may be used to store any type of data. The length of the data may be different for each element; but, all lengths should be an integer multiple of four bytes. A LISTR, in addition to the standard control fields, contains a field which is used to link to a LCB (P3). Thus, a LISTR is used to enable a list to contain a sublist. A LISTR may optionally contain a user data field. Figure 2.1 gives a diagram of the list X,

X: (a, b, Y, c) .

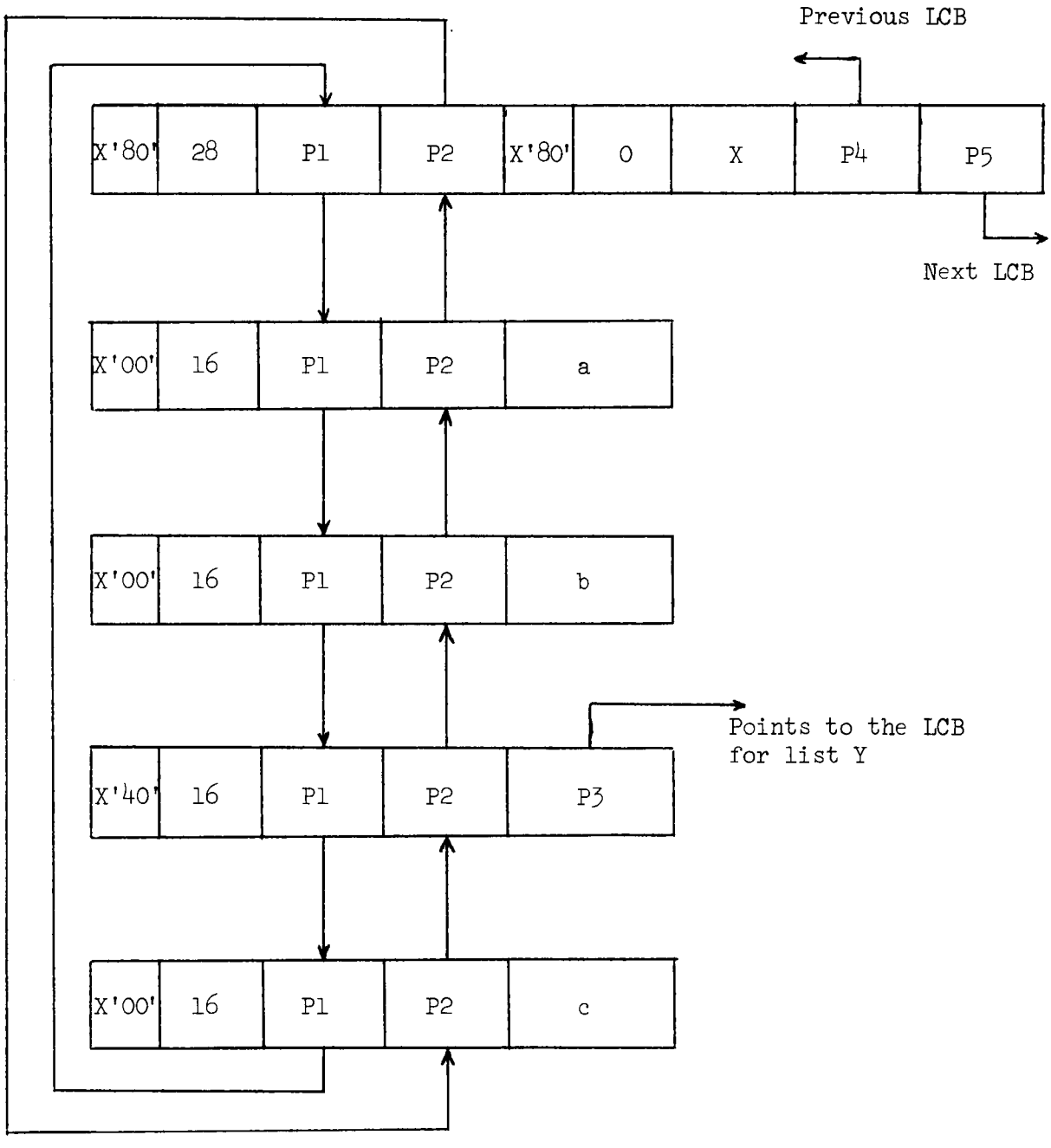


Figure 2.1. Diagram of the list X:(a,b,Y,c).

Note that the pointers always contain the address of the first byte of the element pointed to. It is assumed that the list X was defined with IMARK = 1 and that it is not used as a sublist.

Pushdown stores are made up of two types of elements. A PDS name element has the following fields: NAME, P1 and P2. NAME contains the name of the PDS, P1 links to the next PDS name element and P2 points to the top element in the PDS called NAME. A simple PDS element (sometimes called a PDS element) contains a P1 field and a user data field. The P1 field contains a pointer which links to the next lower element in the PDS. The user data field contains a word (four bytes) of data. An associated PDS, called NAME, is created any time a list, called NAME, is defined. The push down stores are used, by TWLVE, when list structures are traversed.

TWLVE uses three lists of available space. One list of available space is used for the PDS elements (PLAS), one list of available space is used for the PDS name elements (NLAS) and one list of available space is used for all of the list elements (simply called LAS). Since the NLAS and PLAS both consist of fixed length elements, maintaining them is fairly easy. Any time a PDS name element is created the NLAS is examined to make sure it is not empty. If not, the first element in NLAS is released. Any time a PDS name element is deleted, the space occupied by the name element is chained to the end of the NLAS. Simple PDS elements are handled the same way using the PLAS. In any case, if the list of available space is empty then the user is notified by examining a variable returned, by TWLVE, to the calling

routine. Maintaining the LAS is more complex.

There are two types of elements which may be contained in the LAS for list elements. A simple LAS element contains an ID field, a LEN field and a P1 field. The ID field identifies the type of element, the LEN field contains the length of the element and the P1 field points to next element in the LAS. A LAS LISTR contains, in addition to the ID, LEN and P1 fields, a field called P3. This field points to a LCB.

When attempting to allocate space from the LAS, the list is searched (beginning at the first LAS element) until the required length, L, is less than, or equal to LEN of the LAS element being examined. If L is less than LEN then the element is left in the LAS but whith it's address (contained in P1 of the previous LAS element) increased by L and it's LEN decreased by L. If L is equal to LEN then the element is deleted from the LAS. When a list, or an element in a list, is deleted the occupied space is returned by chaining the list (or element) onto the end of the LAS. IDs of LCBs are changed so that they are inserted into the LAS as simple LAS elements. LISTRs are inserted into the LAS as LAS LISTRs.

When a LISTR is encountered in the LAS, the LCB pointed to by P3 of the LISTR is examined and the ID of the LISTR is changed to indicate that the element is a simple LAS element (by doing this, a LISTR can only cause the LCB pointed to by

P3 to be examined once). When examining the LCB, MARK is examined first to see if the list can be deleted automatically. If not, nothing is done; if so, the count field of the LCB is decreased by one. If the count is less than or equal to zero, the list is deleted and the space occupied by the list is returned to the LAS. For example, suppose the required length, L, is equal to 80 and the LAS is as shown in Figure 2.2 (A). After obtaining the required space, the LAS is as shown in Figure 2.2 (B).

Sometimes it will be necessary to connect LAS elements (fragments) to meet the space needs. This is done by computing a searched for address (equal to the address of a LAS element plus the length of that element) and then searching the LAS to see if any other LAS elements begin at this address. The process is continued until all combinations have been tried. For example, figure 2.3 (A) shows the LAS before attempting to combine any LAS fragments. After all connections have been made to the LAS element which began at address 100, the LAS is as shown in Figure 2.3 (B). No connections are possible with the LAS element which begins at address 150, but after all connections have been made to the element which begins at address 80, the LAS is as shown in Figure 2.3 (C). Note that LAS LISTR elements encountered in the process of connecting the LAS fragments are treated the same as when space is being obtained from the LAS (i.e. the list pointed

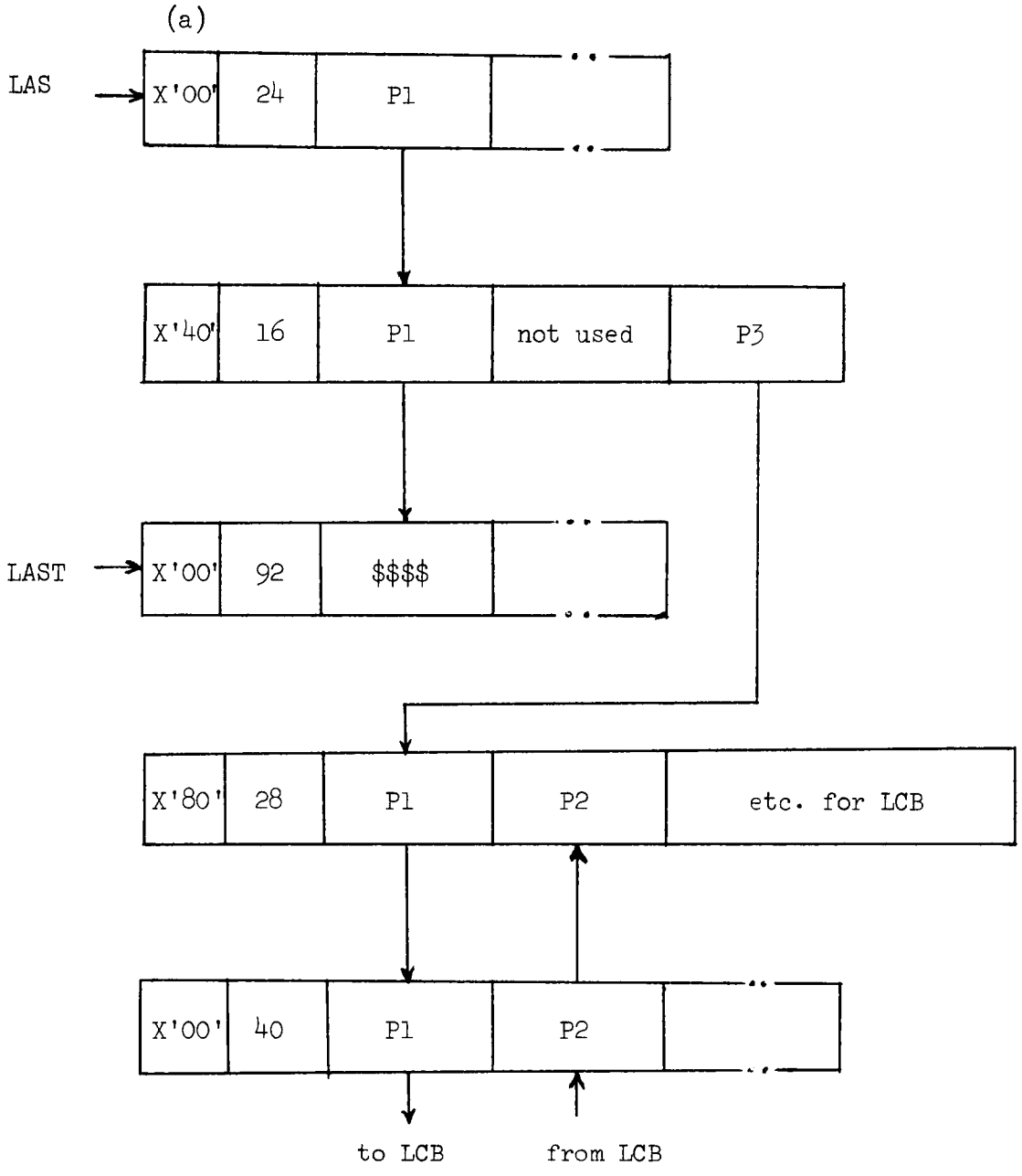


Figure 2.2. LAS (a) before obtaining requested space (80 bytes) and (b) after obtaining requested space.

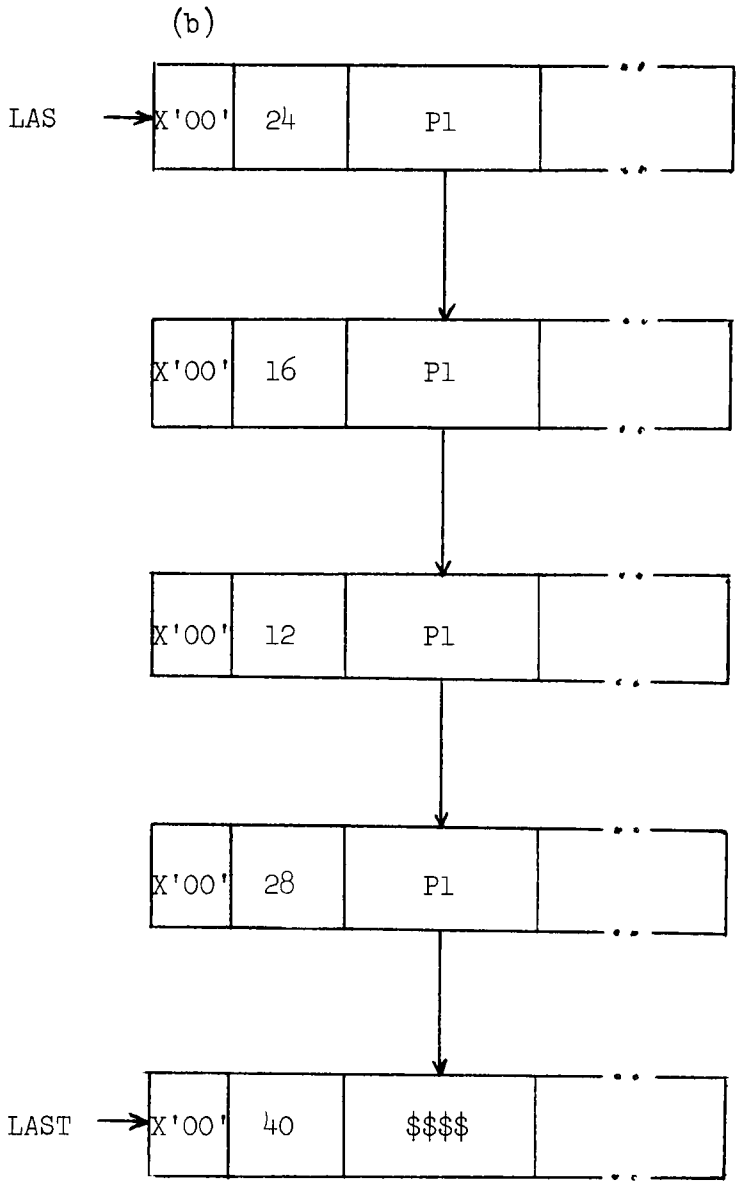


Figure 2.2 (Continued).

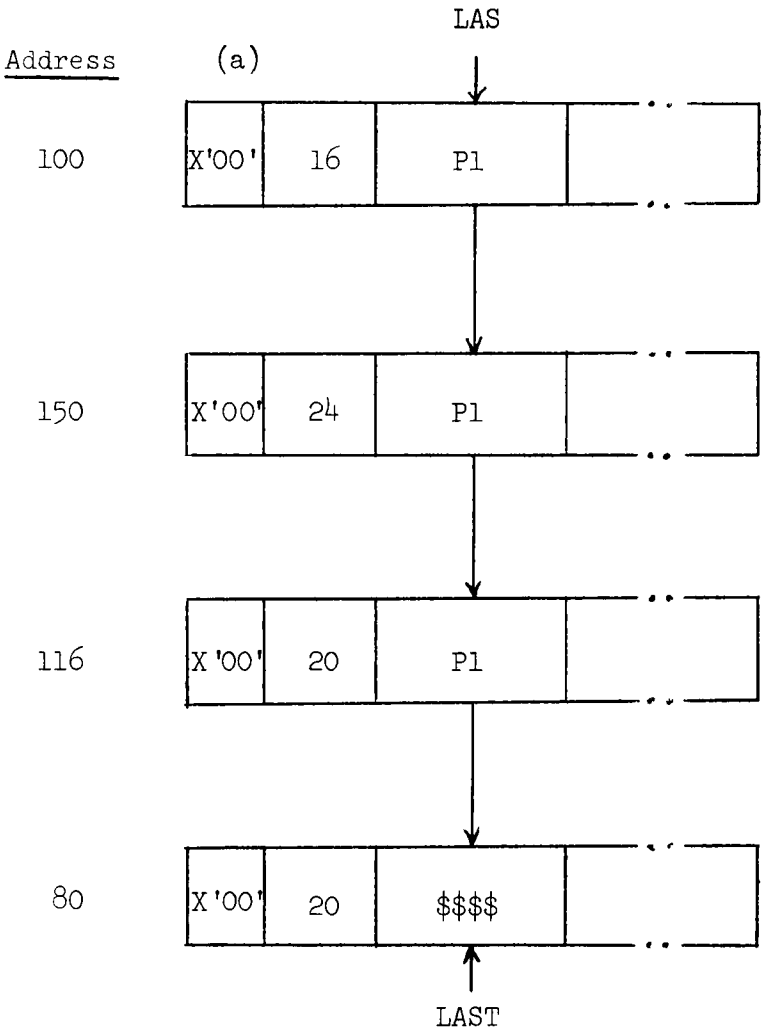
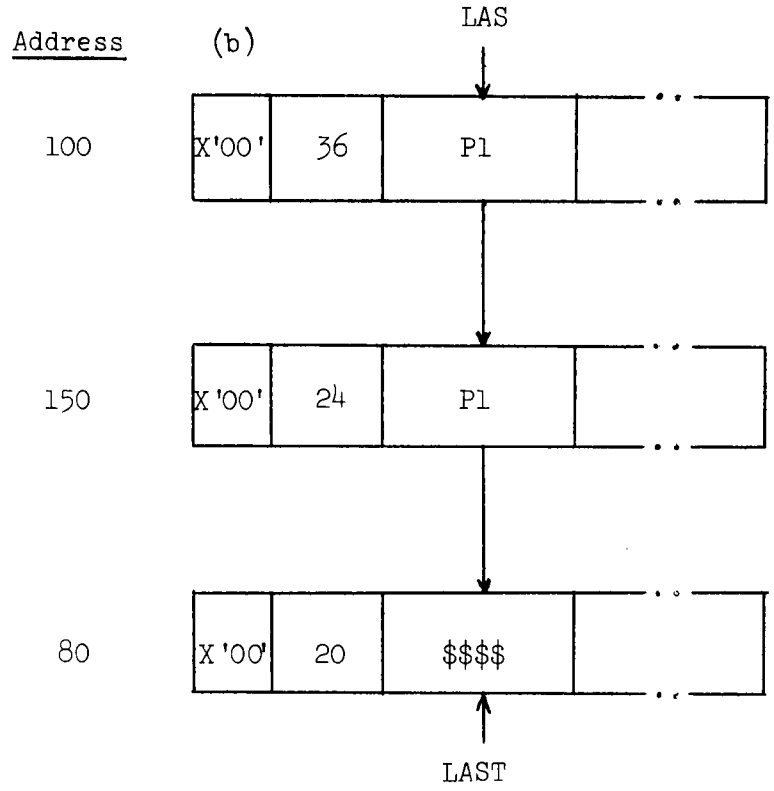


Figure 2.3. Connecting LAS fragments (a) before attempting any connections, (b) after all connections have been made to the element located at address 100, and (c) after all possible connections have been made.



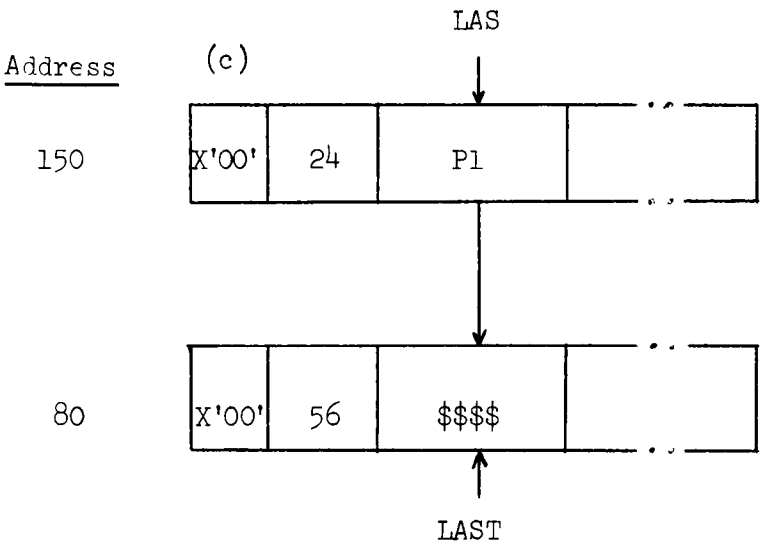


Figure 2.3 (Continued).

to by P3 of the LISTR is examined for automatic deletion and the ID of the LISTR is changed).

II. GUIDE FOR USING TWLVE

This section contains examples of FORTRAN statements which can be used to perform certain list processing functions (i.e. building lists, traversing lists, inserting elements into a list, etc.). This section is not intended to give a complete description of any TWLVE routine or calling sequence. For complete descriptions Chapter III should be consulted. Also, the coding in this section is not always complete in that FORMATS are never included and some branch addresses (those that an error condition causes a branch to) are not included.

For a sample problem, the following is assumed. The list element user data fields have three subfields: county population (ICP), county name length (ICNL), and county name (NAMEC). ICP and ICNL are fixed length, INTEGER*4 fields. NAMEC is variable length, but it is assumed that it will never exceed 30 bytes in length. Counties are grouped into three possible levels: district, region, and area. Elements will be represented by lower case letters (i.e. "a" for the element with its NAMEC field equal to Amos county, "b" for the element whith its NAMEC field equal to Baker county, etc.). Lists will be represented by upper case letters (i.e. "X" for area one, "Y" for region one, and "Z" for

district one). It is also assumed that 2000 bytes will be adequate for building the lists.

Creating Lists and List Structures

The following set of FORTRAN statements can be used to create list structures with elements as described above.

```
      .
      LOGICAL*1 NAMEC(30)
      COMMON /ELEM/ ICP,ICNL,NAMEC
      .
      CALL INITLP(2000)
      READ (5,1000) NLIST
      DO 10 I=1,NLIST
      READ (5,1001) LNAME
      CALL DEFNLP(LNAME,1,IADD,IFLAG)
      IF(IFLAG .NE.  0) GO TO 100
   12 READ (5,1002) ICP,ICNL,(NAMEC(N),N=1,ICNL)
      IF(ICP) 10,20,30
   20 CALL INSLLP(1,IADD,NAMEC,IFLAG)
      IF(IFLAG .NE.  0) GO TO 101
      GO TO 12
   30 CALL INSELP(1,IADD,ICNL+8,0,ICNL+8,ICP,IFLAG)
      IF(IFLAG .NE.  0) GO TO 102
      GO TO 12
   10 CONTINUE
      .
```

There are several things that should be noted about the

above statements. First of all, the list space is initialized to a size of 2000 bytes by the call to INITLP. INITLP must be called before lists can be defined and built. The call to INITLP also initializes the PDS lists of available space. Another thing to note is the use of IFLAG in the sample coding. Many of the TWLVE routines return a value in a variable (IFLAG in the above coding) which is used to indicate the success or failure of the intended operation. This variable should always be tested, upon return from the TWLVE routine, and appropriate action taken if the variable (IFLAG) indicates failure of the intended operation. In the above coding, statements 100, 101 or 102 will be branched to in case of a failure to define a list, insert a LISTR (reference to a sublist), or insert a list element.

To create the following list structure:

X: (a, b, Y, c) ,

Y: (e, f, Z) ,

Z: (g, h) ,

the input data should be: ("indicates a comment")

```
        3 "number of lists"
     D001 "Z"
     1300 4 GOOD "g"
      811 5 HOBBS "h"
       -1
     R001 "Y"
```

```
2500 4 EAST "e"
1819 4 FISK "f"
   0 4 D001 "LISTR to Z"
  -1
A001 "X"
6300 4 AMOS "a"
5010 5 BAKER "b"
   0 4 R001 "LISTR to Y"
7002 6 CRATER "c"
  -1
```

Traversing lists and list structures

The total population (ITPOP) of the area (A001) can be computed by traversing the list structure X and summing the populations. The following set of FORTRAN statements will do this.

```
      .
      ITPOP=0
      JADD=IANMLP('A001')
      IF(JADD .EQ.  0) GO TO 200
50    CALL TRVLLP('A001',1,JADD,IFLAG)
      IF(IFLAG .EQ.  2) GO TO 40
      IF(IFLAG .NE.  0) GO TO 201
      CALL IMVCLP(ILALP(IS),0,IUADLP(JADD),0,4)
      ITPOP=ITPOP+IS
      GO TO 50
40    WRITE (6,2000) ITPOP
```

The function INAMLP returns the address of the list named A001 (X). If the list is not defined then the function returns a value of zero. The routine called TRVLLP is used for traversing the list structure. JADD, upon return, contains the address of the next element in the list structure. IFLAG is equal to zero if the traversal to the next element is successful and is set to two if the entire list has been traversed. The call to IMVCLP is used to move the population, ICP subfield of the element pointed to by JADD, into the variable called IS.

The associated PDS is used when traversing the list structure. If a LISTR is encountered, an address is pushed into the associated PDS. If a LCB is encountered, an address is popped from the associated PDS. An empty PDS indicates that the entire list structure has been traversed. For example, consider the list structure X. Assuming that JADD is initialized to the address of the LCB for the list called A001 and that the PDS called A001 is initially empty, then the following shows the element whose address is returned (the element is underlined) upon return from the calls to TRVLLP and the contents of the associated PDS after each call.

1) X: (a, b, (e, f, (g, h),), c) ,
   PDS (top to bottom): $$$$.
2) X: (a, b, (e, f, (g, h),), c) ,

PDS (top to bottom): $$$$.

3) X: (a, b, (e, f, (g, h),), c) ,

PDS (top to bottom): A(LINKR-Y), $$$$.

4) X: (a, b, (e, f, (g, h),), c) ,

PDS (top to bottom): A(LINKR-Y), $$$$.

5) X: (a, b, (e, f, (g, h),), c) ,

PDS (top to bottom): A(LINKR-Z), A(LINKR-Y),
$$$$.

6) X: (a, b, (e, f, (g, h),), c) ,

PDS (top to bottom): A(LINKR-Z), A(LINKR-Y),
$$$$.

7) X: (a, b, (e, f, (g, h),), c) ,

PDS (top to bottom): $$$$.

On each of the above calls, IFLAG is returned set equal to zero. The next call to TRVLLP causes IFLAG to be returned set to two. A(LINK-Y) is used to represent the address of the LINKR whose P3 field contains the address of the LCB for list Y.

The list structure can also be traversed and ITPOP computed using the following FORTRAN statements.

```
      .
      ITPOP=0
      JADD=INAMLP('A001')
      IF(JADD .EQ.  0) GO TO 200
  150 CALL TRVELP(1,JADD,IFLAG)
      IF(IFLAG .EQ.  2) GO TO 152
```

```
      IF(IFLAG .EQ.  1) GO TO 145
      CALL INVCLP(ILALP(IS),0,IUADLP(JADD),0,4)
      ITPOP=ITPOP+IS
      GO TO 150
  145 CALL PUSHPS('A001',JADD,IFLAG)
      IF(IFLAG .NE.  0) GO TO 202
      JADD=IPNTLP(JADD,3)
      GO TO 150
  152 CALL POPPS('A001',JADD,IFLAG)
      IF(IFLAG .EQ.  0) GO TO 150
      IF(IFLAG .EQ.  1) GO TO 203
      WRITE (6,2000) ITPOP
      .
```

The only assumption is that the PDS named A001 is empty before any of the above statements are executed.

Inserting and Deleting Elements and Deleting Lists

An element can be inserted into a list by calling INSELP. The following set of FORTRAN statements can be used to insert an element before the element, in list Z, which has its NAMEC field equal to NAMEB (read in). It is assumed that the lists are previously defined as above.

```
      .
      LOGICAL*1 NAMEB(30),NAMEC(30)
      COMMON /ELEM/ ICP,ICNL,NAMEC
      .
      READ (5,1003) ICP,ICNL,ICNB,(NAMEC(I),I=1,ICNL),
```

```
     1 (NAMEB(I),I=1,ICNB)
       KADD=ILALP(NAMEB)
       JADD=IANMLP('D001')
       IF(JADD .EQ.  0) GO TO 300
   301 CALL TRVELP(1,JADD,IFLAG)
       IF(IFLAG .NE.  0) GO TO 350
       IF(ICLCLP(KADD,0,IUADLP(JADD),8,ICNB) .EQ.  0)
      2 GO TO 302
       GO TO 301
   302 CALL INSELP(2,JADD,ICNL+8,0,ICNL+8,ICP,IFLAG)
       IF(IFLAG .NE.  0) GO TO 370
       .
```

If the data card read in is

```
1001 8 5 NUCOUNTYHOBBS "n" ,
```

then list Z is changed to

```
Z: (g, n, h) .
```

The same operation can be performed by starting the search at the end of the list. The only statement that needs to be changed is

```
301 CALL TRVELP(2,JADD,IFLAG) .
```

An element can be deleted from a list by calling DLTELP. The following set of FORTRAN statements can be used to delete an element from the list Z. The element which has a NAMEC field equal to NAMED (read in) is deleted. It is assumed that Z is defined as

```
Z: (g, n, h) ,
```

before the set of statements is executed.

```
      .
      LOGICAL*1 NAMED(30)
      .
      READ (5,1004) ICND,(NAMED(N),N=1,ICND)
      KADD=ILALP(NAMED)
      JADD=IANMLP('D001')
      IF(JADD .EQ.  0) GO TO 400
  401 CALL TRVELP(1,JADD,IFLAG)
      IF(IFLAG .NE.  0) GO TO 450
      IF(ICLCLP(KADD,0,IUADLP(JADD),8,ICND)
     1 .EQ.  0) GO TO 402
      GO TO 401
  402 CALL DLTELP(JADD)
      .
```

If the data card read in is

```
    5 HOBBS "h"
```

then list Z is changed to

```
      Z: (g, n) .
```

A list can be deleted by calling DLTLLP. The list X can be deleted by the following FORTRAN statement

```
      CALL DLTLLP('A001',IFLAG) .
```

It should be noted that the space occupied by list Y (a sublist to list X) will not automatically be deleted because Y was defined with IMARK set equal to one. The user is warned to not delete a sublist without first deleting the

LISTR, in the list, which points to the sublist. Doing this can cause unpredictable results when the list structure is traversed.

III. TRACE FEATURE

This section describes those calls (to TWLVE routines) which can be traced and how to turn various traces on and off. It also describes the output generated from the traces. All traces are printed on logical unit ITRACE which is defined in the COMMON COM$$$ (see sections called "PROGRAMMING CONSIDERATIONS" and "SYSTEM CONSIDERATIONS" for a complete description of the COMMON COM$$$).

Calls to INITLP, RLSELP, OBTNLP, CONFLP, RTNLLP and RTNELP can be traced by executing the following FORTRAN statement

```
CALL TONSLP .
```

The trace, for these calls, can be turned off by executing the statement

```
CALL TOFSLP .
```

When the trace is on, the FORTRAN routine MESSP$ is branched to as soon as control passes to one of the above entry points. The arguments which are passed to the subprograms are printed out using a Z format specification. For example, if INITLP is called and the trace is on, the following message is printed on logical unit ITRACE:

```
INITLP CALLED WITH NB= cccccccc
```

where cccccccc is the hexadecimal representation of the argument NB.

Calls to DEFNLP, INSELP, INSLLP, DLTELP and DLTLLP are traced if the FORTRAN statement

CALL TONDLP

is executed. The trace can be turned off by executing the statement

CALL TOFDLP .

The FORTRAN routine MESSP$ is branched to as soon as control is passed to any of the entry points, except DEFNLP. The trace print for DEFNLP is recorded immediately before returning to the control routine, so that the address of the inserted LCB (which is returned) can be printed. This causes the call to OBTNLP (possibly a call to CONFLP and another call to OBTNLP, as well) to be recorded (if a call has been made to TONSLP) and a call to DEFNPS to be recorded (if a call has been made to TRONPS, see below) before the trace print for the call to DEFNLP. As in the previous case, arguments that are passed to the subprograms are printed out using a Z format.

Calls to INITPS, DEFNPS, DELTPS, PUSHPS, POPPS and RLSEPS are traced if the FORTRAN statement

CALL TRONPS

is executed. The trace can be turned off by executing the statement

CALL TOFPPS .

The FORTRAN routine MESS$$ is branched to as soon as control is passed to any of the entry points. Arguments passed to the subprograms are printed out using a Z format. For example, if the following call is made

```
CALL PUSHPS('ABCD',20) ,
```

and if the trace is on then the following message is printed on logical unit ITRACE.

```
PUSHPS CALLED WITH NAME= C1C2C3C4 AND
IADD= 00000014 .
```

IV. PROGRAMMING CONSIDERATIONS

Routine Names

The following names are included as part of the TWLVE package. The user should not use any of these names for naming subroutines, functions or labelled COMMON blocks. The TWLVE routine names, in alphabetical order, are: COM$$$, CONFL$, CONFLP, DEFNL$, DEFNLP, DEFNP$, DEFNPS, DELTP$, DELTPS, DLINKLP, DLTEL$, DLTELP, DLTLL$, DLTLLP, DUMPSS, IANMLP, ICLCLP, IIDLP, ILALP, ILENLP, INVCLP, INCLP, INITL$, INITLP, INITP$, INITPS, INSEL$, INSELP, INSLL$, INSLLP, IOCLP, IPNTLP, ITMLP, IUADLP, IXCLP, LASLP$, LOPERLP, MESS$$, MESSP$, MODCOM, NEEPS, OBTNL$, OBTNLP, PEEPS, PLASLP, POPP$, POPPS, PUSHP$, PUSHPS, PUSHST, RLSEL$, RLSELP, RLSEP$, RLSEPS, RTNEL$, RTNELP, RTNLL$, RTNLLP, SPACELP, TOFDLP, TOFFPS, TOFSLP, TONDLP, TONSLP, TRAVSLP, TRONPS, TRVELP, TRVLLP.

Modifying COMMON

The labelled COMMON COM$$$ contains the variables ITRACE, IOUT and IPRINT. ITRACE is the logical unit used for printing trace information. IOUT is the logical unit used for printing error messages and for dumping the PDSs. IPRNT is used for controlling the PDS dumps (see section called "Examining Push Down Stores" in chapter three). The variables are initialized in a BLOCKDATA subprogram to the following: ITRACE is set to 40, IOUT is set to 51 and IPRNT is set to zero. To modify any of the variables in the COMMON the following call can be made

CALL MODCOM(I,IVALUE) .

If I is equal to one, ITRACE is set equal to IVALUE, if I is equal to two then IOUT is set equal to IVALUE and if I is equal to three then IPRNT is set equal to IVALUE. The contents of all of the variables in the COMMON COM$$$ can be printed out, using an I format, on logical unit IOUT by calling MODCOM with I equal to zero.

General Hints

To ensure user program compatibility with any changes in TWLVE, the user should use IANMLP, IPNTLP, IIDLP, ILENLP, and IUADLP to fetch control information rather than picking up the information using IMVCLP. For example, if one wanted to store P2 of the element pointed to by JADD (assume that the address of the element has been previously placed in JADD) into IP2 then

```
IP2=IPNTLP(JADD,2) ,
```

is preferable to

```
CALL IMVCLP(ILALP(IP2),0,JADD,8,4) .
```

It is recommended that user data fields be structured in COMMON if possible. If only one variable length subfield is to be combined with fixed length subfields, then it is suggested that the variable length subfield be placed after the fixed length subfields. For example, suppose the user data field of an element contains the fixed length subfields: IDATE, an INTEGER*4 subfield; ITIME, an INTEGER*4 subfield; IN, an INTEGER*2 subfield and VALUE, a REAL*8 subfield, as well as a variable length subfield, LTYPE, which consists of IN bytes of data. Furthermore, assume that JADD contains the address of the previous element in the list, that the maximum value IN can have is 50 and that LTYPE is defined as a LOGICAL*1 variable dimensioned by 50. One way to insert the element into the list is by the following set of FORTRAN statements.

```
.
CALL INSELP(1,JADD,IN+18,0,4,IDATE,IFLAG)
IF(IFLAG .NE.  0) GO TO 500
KADD=IUADLP(JADD)
CALL IMVCLP(KADD,4,ILALP(ITIME),0,4)
CALL IMVCLP(KADD,8,ILALP(IN),0,2)
CALL IMVCLP(KADD,10,ILALP(VALUE),0,8)
CALL IMVCLP(KADD,18,ILALP(LTYPE),0,IN)
```

```
      .
```

A much more efficient way to insert the element into the list is by the following set of FORTRAN statements.

```
      .
      COMMON /ELEM/ VALUE,IDATE,ITIME,IN,LTYPE
      CALL INSELP(1,JADD,IN+18,0,IN+18,VALUE,IFLAG)
      IF(IFLAG .NE.  0) GO TO 500
      .
```

Two things should be noted. First of all, the variable VALUE must be placed in the COMMON before the other variables to be sure that word boundary errors can't occur. This is a consideration which the user should think about as the user data fields are designed. Another point is that all TWLVE elements must be a multiple of four; but, the routines INSELP and INSLLP automatically round the user data field lengths up to the next multiple of four, if necessary. For example, in the above statements, If IN is equal to eight then the user data field length will be rounded up to 20 and the last two bytes in the field will be undefined.

V. SYSTEM CONSIDERATIONS

Operating System

TWLVE is being used on IBM/360 computers which are operating under OS in a MVT environment. It makes use of the system macros GETMAIN, FREEMAIN and ABEND. GETMAIN and FREEMAIN are part of the OS memory management facilities and

ABEND handles abnormal terminations. It is assumed that the user is somewhat familiar with IBM/360 terminology and JCL.

Region Size

The total amount of memory necessary for the TWLVE routines is about 12k bytes. The user must also consider the memory which will be needed because of the calls to INITLP and INITPS which cause GETMAIN macro instructions to be issued. The sample problem (described in Appendix E) requires 92K bytes of memory.

Execution Time Estimate

The sample problem (described in Appendix E) used 11.22 seconds of CPU time on the 360/75 to process 3176 input records (405 of the input records were added to the lists which were built).

Control Cards

If the output from the trace print (logical unit ITRACE) or the output from the error message print and the PDS dumps (logical unit IOUT) is not to the standard output unit (see section called "Modifying COMMON" in this chapter for more information about ITRACE and ICUT) then one or more DD cards will have to be supplied. The cards will be of the following form:

```
//GO.FTnnF001 DD SYSOUT=A,
// DCB=(RECFM=FBA,LRECL=133,BLKSIZE=1330)
```

where nn is the appropriate logical unit number.

System Errors

The user should take special care when using TWLVE because it is easy to attempt to store into memory which is not intended to be used for building lists or push down stores. Most of the time these "wild stores" will result in a system error message being generated. The message generated will depend, to a great extent, on what memory locations are "written over".

The user should also be aware of what happens if a call is made to INITLP or to INITPS with a request for more memory than is still available to the job step (determined by the REGION.GO field on the EXEC card). What happens is that the job terminates with a system completion code 804 or 80A.

VI. SUMMARY

TWLVE is a list processing system which offers the following advantages.

1) Elements are variable length.
2) All TWLVE subroutines are FORTRAN callable.
3) Elements use two-way pointers for efficient insertion and deletion.
4) A trace feature is available to be used as a debugging aid.
5) Return codes are issued by many of the TWLVE routines for greater user flexibility and

control.

6) Independent subsets of TWLVE are provided for logical operations and push down store (stack) operations.

7) TWLVE is constructed in a modular fashion for easy modification and/or additions.

The list processing techniques used by TWLVE versus the possible aternatives include the following.

1) TWLVE uses variable length elements rather than fixed length elements.

2) For returning memory to the LAS, TWLVE uses a reference count technique rather than "garbage collection".

3) For obtaining memory from the LAS, TWLVE uses "first fit" rather than "best fit".

4) For combining LAS fragments, TWLVE waits until a request for memory can't be satisfied rather than combining LAS elements as soon as memory is returned to the LAS.

A user-oriented summary of the TWLVE calling sequences follows. The arguments which must be defined in the user's calling routine are underlined. For a complete description of these calls, as well as the calls not summarized here, Chapter III should be consulted.

CALL INITLP(NB) to initialize the list of available space.

CALL RLSELP to release space allocated for lists.

CALL DEFNLP(NAME,IMARK,IADD,IFLAG) to define a list.

CALL INSELP(P,IADD,L1,D,L2,DATA,IFLAG) to insert an element into a list.

CALL INSLLP(P,IADD,NAME,IFLAG) to insert a LISTR without a user data field into a list.

CALL INSLLP(P,IADD,NAME,IFLAG,L1,D,L2,DATA) to insert a LISTR with a user data field into a list.

CALL DLTELP(IADD) to delete an element from a list.

CALL DLTLLP(NAME,IFLAG) to delete a list.

CALL TRVLLP(NAME,P,IADD,IFLAG) to traverse a list structure.

CALL TRVELP(P,IADD,IFLAG) to traverse a list.

IADD=IANMLP(NAME) to fetch the address of a LCB.

IPNT=IPNTLP(IADD,IP) to fetch the contents of a pointer.

ID=IIDLP(IADD) to fetch the ID of an element.

LENGTH=ILENLP(IADD) to fetch the length of an element's user data field.

IUADD=IUADLP(IADD) to fetch the address of an element's user data field.

J=INCLP(IADD,ID,JADD,JD,N) to perfcrm the logical AND operation.

J=IOCLP(IADD,ID,JADD,JD,N) to perfcrm the inclusive OR operation.

J=IXCLP(IADD,ID,JADD,JD,N) to perfcrm the exclusive OR

operation.

J=ITMLP(IADD,ID,M) to test under mask.

J=ICLCLP(IADD,ID,JADD,JD,N) to perform a logical compare.

CALL IMVCLP(IADD,ID,JADD,JD,K) to move data.

J=ILALP(IVAL) to load the address of a variable.

CALL DEFNPS(NAME1,IFLAG) to define a push down store.

CALL DELTPS(NAME1,IFLAG) to delete a push down store.

CALL PUSHPS(NAME1,IADD,IFLAG) to push an element into a push down store.

CALL POPPS(NAME1,IADD,IFLAG) to pop an element from the top of a push down store.

CALL RLSEPS to release memory allocated for push down stores.

CALL DUMPPS(ICNT,NAME1) to examine push down stores.

CHAPTER III

ROUTINE DESCRIPTIONS

This chapter describes the TWLVE routines. There are sections which describe how to maintain the list of available space, how to define and build two-way lists, how to traverse lists and fetch control information, how to perform logical operations, and how to use push down stores. Each section is divided into subsections each of which describes a specific function (i.e. initializing the list of available space, traversing a list, etc.). The first paragraph (or first few sentences, if there is only one paragraph) of any routine description contains the calling sequence. This is the only part that is required reading for the TWLVE user. The remaining details should be useful to the TWLVE implementor. The reader should also note that the logical operations routine and the push down store routine are both independent subsets of TWLVE.

I. MAINTAINING THE LIST OF AVAILABLE SPACE

The list of available space (LAS) is created and maintained using the IBM/360 assembly language routine named SPACELP. The routine contains entry points that are FORTRAN callable and that perform the following functions:

1) initialize the list of available space,

2) release the space allocated for lists,

3) obtain a block of memory from the LAS,

4) connect the LAS fragments,

5) return list space to the LAS,

6) return element space to the LAS, and

7) interrogate the LAS.

All of the calling sequences, required arguments and program logic are described below. The arguments must all be stored, in the calling routine, on a word boundary (i.e. INTEGER*4). Flow charts for the routine are given in Appendix C, the coding is contained in Appendix D and the LAS elements are described in Appendix B. It is expected that the information given in the Appendixes will be needed to fully understand the program logic. The following internal variable names are also important for understanding the program logic.

1) LAS - Contains the address of the first element in the list of available space.

2) LAST - Contains the address of the last element in the list of available space.

3) NPNT - Contains the address of the first LCB.

4) BLKSIZ - Contains the size of the memory block which is used for storing the lists.

5) BLKADD - Contains the address of the first byte of the memory block which is used for storing the lists.

6) NPELEMTS - Contains the default number of PDS elements which are initialized when INITLP is called (1000).

7) NPDS - Contains the default number of PDS name elements which are initialized when INITLP is called (50).

8) FLAGS - Used to indicate initialization.

Initializing the List of Available Space for Lists

The list of available space (LAS) is initialized by making the following call:

```
CALL INITLP(NB) .
```

NB is the length, in bytes, of the block of core which will be made available to the control program for building lists. It must be defined in the calling routine. When INITLP is entered, FLAGS is checked to see if the list of available space is already initialized. If so, control is returned to the calling program. If the LAS is not initialized, NB is stored in BLKSIZ and a GETMAIN macro is issued with a request for NB bytes of memory. The address of the allocated block of memory is stored in BLKADD, stored in LAS and stored in LAST; then the ID, LENGTH, and P1 fields of the element pointed to by LAS are set. ID is set equal to X'00', LENGTH is set equal to NB and P1 is set equal to $$$$. Next, a call is made to INITPS to initialize the push down store lists of available space (see section "Initializing Memory for Push Down Stores" in this chapter).

NPELEMTS PDS elements and NPDS stack elements are initialized. Finally, FLAGS is set to X'80' to indicate that the list of available space for the lists is initialized and then control is returned to the calling routine. A flow chart for INITLP is given in Appendix C, chart S1, and the coding is contained in Appendix D as part of the control section named SPACELP.

Releasing the Space Allocated for Lists

The space allocated to the control program for building lists can be released by making the following call:

```
CALL RLSELP .
```

When RLSELP is entered, FLAGS is checked to see if the space used for building lists is initialized. If not, the job is terminated with error LPS21 (see Appendix A for a description of the errors). If list space is initialized then a FREEMAIN macro is issued to release BLKSIZ bytes of memory beginning at the address specified in BLKADD. Next, a call is made to RLSEPS to release the space allocated for the push down stores. Finally, FLAGS is set to indicate that list space is not initialized. A flow chart for RLSELP is given in Appendix C, chart S2, and the coding is contained in Appendix D as part of the control section named SPACELP.

Obtaining a Block of Memory From the LAS

A block of memory can be obtained from the list of

available space (LAS) by making the following call:

CALL OBTNLP(L,IADD,IFLAG) .

L is the length, in bytes, of the block. It must be defined in the calling routine and must be an even multiple of four. IADD is returned with the address of a block of L contiguous bytes of memory and IFLAG is returned set to zero, after a successful operation. If L contiguous bytes of memory can't be obtained then IFLAG is returned set to one and the contents of IADD are unpredictable.

When OBTNLP is entered, the address of the first element in the list of available space (stored in the variable called LAS) is loaded into PNT1 and the address of LAS is loaded into PNT0. If PNT1 contains a null pointer ($$$$) then IFLAG is set to one and control is returned to the calling routine. If PNT1 does not contain a null pointer, the ID of the element pointed to by PNT1 is examined. If the ID indicates that the element pointed to is a LISTR, a branch is made to the section of coding which handles the decision of deleting, or not deleting, a list (the decision logic is described below). Next, PNT2 is loaded with P1 of the element pointed to by PNT1 and L1 is loaded with the length of the element pointed to by PNT1. L1 is compared with L giving three possible results. If L1 is less than L, PNT1 is updated to point to the next LAS element, PNT0 is updated to contain the address of the pointer which has just been loaded into PNT1 and processing

continues (at the point where PNT1 is checked to see if it contains a null pointer). If L1 is equal to L or if L1 is greater than L then a LAS element has been found which satisfies the request of the calling routine.

When the length of a LAS element, L1, is equal to the required length, L, the element is ready to be deleted from the LAS. Recall that PNT1 contains the pointer to the element and PNT0 contains the address of the pointer. PNT3 is loaded with P1 of the element pointed to by PNT1 and is then stored at the address contained in PNT0. If PNT1 is equal to LAST then LAST is set equal to PNT3. Finally, IADD is set equal to PNT1, IFLAG is set to zero and control is returned to the calling routine.

If L1 is greater than L then the element is not deleted from the LAS, but its length and beginning address are both modified. PNT3 is computed by adding L to PNT1 and L2 is computed by subtracting L from L1. The ID of the element pointed to by PNT3 is set equal to X'00', the LENGTH of the element pointed to by PNT3 is set equal to L2 and P1 of the element pointed to by PNT3 is set equal to PNT2 (see above). Next, PNT3 is stored at the address contained in PNT0, and if PNT1 is equal to LAST then LAST is set equal to PNT3. Finally, IADD is set equal to PNT1, IFLAG is set to zero and control is returned to the calling routine.

When a LISTR (element with an ID of X'40' or X'41') is encountered in the list of available space, a section of the

program is entered to see if the list pointed to by the LISTR should be deleted. Upon entry, the ID of the LISTR is changed to X'00' so that a LISTR can cause a branch only once. The ID of the element pointed to by P3 of the LISTR is tested to see if the element is a LCB. If not (or if P3 contains a null pointer), control is returned to the caller; otherwise, the list control block (LCB) is examined. If the LCB's MARK is equal to X'80' then control is returned to the caller. If MARK is not equal to X'80', the LCB's COUNT is decreased by one. If the resulting COUNT is less than or equal to zero, the list pointed to by the LISTR is ready to be deleted; otherwise, control is returned to the caller.

To delete a list, the elements of the list are chained to the end of the LAS. The program logic is the following. LAST is tested to see if it contains a null pointer and if it doesn't, P1 of the element pointed to by LAST is set equal to the address of the LCB of the list which is being deleted. P1 of the element that is pointed to by P2 of the LCB is set to be a null pointer (the new end of the LAS). The ID of the LCB is changed to X'00' and LAST is set equal to P2 of the LCB. If the list of available space was empty (LAS equal to $$$$) then LAS is set equal to the address of the deleted LCB. At this point, a call is made to DELTPS to delete the push down store associated with the deleted list. Finally, the LCB is deleted from the LCB chain. If the deleted LCB is the first in the chain NPNT is set equal to

P5 of the deleted LCB; otherwise, P5 of the previous LCB is set equal to P5 of the deleted LCB. If the deleted LCB is not the last LCB in the chain, P4 of the next LCB is set equal to P4 of the deleted LCB.

A flow chart for OBTNLP is given in Appendix C, chart S3. A flow chart for the section of coding which handles the decision of deleting, or not deleting, a list and a flow chart for list deletion are given in Appendix C, charts H1 and H2. The coding for all of the above is contained in Appendix D as part of the control section named SPACELP.

Connecting the LAS Fragments

Fragments in the list of available space (LAS) may be connected by making the following call:

CALL CONFLP(ICNT) .

ICNT is returned with the number of connections which have been made. When ICNT is returned set tc zero no connections were possible and the LAS remains unchanged.

When CONFLP is entered, ICNT is zeroed and PNT1 is initialized with the address of the first element in the list of available space. PNT1 is checked to see if it contains a null pointer and if it does control is returned to the calling routine. If PNT1 does not contain a null pointer, the ID of the element pointed to by PNT1 is checked to see if the element is a LISTR. If sc, a branch is made to the section of coding which handles the decision of deleting, or not deleting, a list (see section called

"Obtaining a Block of Memory From the LAS" in this chapter). ADDS is computed as PNT1 plus the length of the element pointed to by PNT1. PNT2 is initialized to be equal to the address, minus 4, of the variable called LAS and PNT3 is initialized to the contents of LAS. If PNT3 contains a null pointer, PNT1 is chained to point to the next LAS element and the above process (starting at the point where PNT1 is checked for a null pointer) is repeated. If PNT2 does not contain a null pointer, the ID of the element pointed to by PNT2 is checked to see if the element is a LISTH and, if so, a branch is made to the section of coding which handles the decision of deleting, or not deleting, a list. Next, PNT3 is compared with ADDS. If PNT3 is equal to ADDS a connection can be made. If PNT3 is not equal to ADDS then PNT2 is set equal to PNT3, PNT3 is chained to point to the next LAS element and the above process (starting at the point where PNT3 is checked for a null pointer) is repeated.

When a connection is made, ICNT is incremented by one. The length of the element pointed to by PNT1 is increased by the length of the element pointed to by PNT3. ADDS is also increased by the length of the element pointed to by PNT3. Next, P1 of the element pointed to by PNT2 is set equal to P1 of the element pointed to by PNT3. Finally, if LAST is equal to PNT3, LAST is set equal to PNT2. The above process (starting at the point where PNT2 is initialized) is repeated until all LAS elements have been looked at.

A flow chart for CONFLP is given in Appendix C, chart S4. A flow chart for the section of coding which handles the decision of deleting, or not deleting, a list and a flow chart for list deletion are given in Appendix C, charts H1 and H2. The coding for all of the above is contained in Appendix D as part of the control section named SPACELP.

Returning List Space to the LAS

The space used by a list can be returned to the list of available space (LAS) by making the following call:

```
CALL RTNLLP(IADD) .
```

IADD is defined in the calling routine to point to the LCB for a list which is to be deleted. PNT4 is set equal to IADD and then the ID of the element pointed to by PNT4 is checked to see if the element is a LCB. If not, control is returned to the calling routine. If so, a branch is made to the section of coding which handles the deletion of a list (see section called "Obtaining a Block of Memory From the LAS" in this chapter). A flow chart for RTNLLP is given in Appendix C, chart S5. A flow chart for the section of coding which handles list deletion is given in Appendix C, chart H2. The coding for both is contained in Appendix D as part of the control section named SPACELP.

Returning Element Space to the LAS

Space used by an element may be returned to the list of available space (LAS) by making the following call:

CALL RTNELP(IADD) .

IADD must be defined in the calling program and should not point to a LCB. When RTNELP is entered, PNT1 is set equal to IADD. If the ID of the element pointed to by PNT1 indicates that the element is a LCB, control is returned to the calling routine; otherwise, processing is continued. If LAST does not contain a null pointer then P1 of the element pointed to by LAST is set equal to PNT1 (the new last element in the LAS). LAST is set equal to PNT1. Next, if LAS contains a null pointer, LAS is set equal to PNT1. Finally, P1 of the element pointed to by PNT1 is set to be a null pointer and control is returned to the calling program. A flow chart for RTNELP is given in Appendix C, chart S6, and the coding is contained in Appendix D as part of the control section named SPACELP.

Interrogating the LAS

The list of available space (LAS) may be interrogated by making the following call:

CALL PLASLP(IADD,ILEN,IFLAG) .

IADD must be defined in the calling routine. If IADD is equal to zero, on entry, then the address of the first LAS element is returned in IADD and IFLAG is set to zero, unless the LAS is empty. If the LAS is empty then IFLAG is returned set to two. If IADD is not equal to zero, on entry (it is assumed, in this case, that IADD points to a LAS element), then IADD is returned set equal to P1 of the

element pointed to, ILEN is set equal to the first four bytes (ID and LENGTH) of the LAS element pointed to, and IFLAG is set to zero, unless P1 of the element pointed to is a null pointer. In this case IFLAG is set equal to one. A flow chart for PLASLP is given in Appendix C, chart S7, and the coding is contained in Appendix D as part of the control section named SPACELP.

II. DEFINING AND BUILDING TWO-WAY LISTS

Lists are built using the IBM/360 assembly language routine named DLINKLP. The routine contains entry points that are FORTRAN callable and that perform the following functions:

1) define a list,
2) insert an element into a list,
3) insert a LISTR (an element which points to a sublist) into a list,
4) delete an element from a list, and
5) delete a list.

The calling sequences, required arguments and program logic are described below. All of the arguments, unless stated otherwise, must be stored, in the calling routine, on a word boundary (i.e. INTEGER*4). Flow charts for the routine are given in Appendix D and the coding is contained in Appendix C. List elements, LISTRs and list control blocks (LCB) are described in Appendix B. To fully understand the program

logic, the information contained in Appendix B and in Appendix C should be examined before reading the program logic or as the logic is read. The internal variable NPNT contains the address of the first LCB.

Defining a List

A list may be defined by making the following call:

```
CALL DEFNLP(NAME,IMARK,IADD,IFLAG) .
```

NAME and IMARK must be defined in the calling routine. The defined list is identified by NAME, which can consist of any four bytes of data except the null pointer ($$$$). IMARK is used to set the MARK field in the list control block (LCB). If IMARK is not zero, MARK is set equal to X'80' so that the list can only be deleted by a call to DLTLLP (see section called "Deleting a List" in this chapter). If IMARK is zero then MARK is set equal to X'00' so that the list can be deleted if the count field of the LCB reaches zero. IADD is returned with the address of the defined list's LCB and IFLAG is returned set to zero if the list is defined successfully. If the LAS does not contain enough space to define the LCB, IFLAG is returned set to one. IFLAG is returned set to two if the push down store (PDS) name list of available space is empty causing the associated PDS, called NAME, to not be defined. If a list called name is defined on entry, then IADD is returned with the address of the list's LCB and IFLAG is returned set to zero.

When DEFNLP is entered the list control blocks are examined to see if a list called NAME is already defined. If not, a LCB for the list is defined and placed in the correct position in the LCB chain (the LCBs are organized in ascending order). The program logic follows. On entry, PNT2 is set equal to the null pointer and PNT1 is loaded with the address of the first LCB. If PNT1 is null then the list is not defined and the LCB will be defined, if LAS space is available. If PNT1 is not null, NM1 is set equal to the NAME field of the LCB pointed to by PNT1 and NAME is compared to NM1. If NAME is equal to NM1, the list is already defined so IADD is set equal to PNT1, IFLAG is set to zero, and control is returned to the calling routine. If NAME is less than NM1 then the list is not defined and the LCB will be defined, if LAS space is available. If NAME is greater than NM1 then PNT2 is set equal to PNT1, PNT1 is chained to point to the next LCB and the above process is repeated, beginning at the point where PNT1 is tested for a null pointer.

When defining a LCB, a branch is first made to the section of coding which gets space from the LAS (GETSPACE), see below. A LCB requires 28 bytes (see Appendix B for a complete description of a LCB). JFLAG is tested to see if the required space was obtained. If not, IFLAG is set to one and control is returned to the calling program. If the required space was obtained, processing continues. If PNT2

is null, NPNT is set equal to the address of the LCB; otherwise, P5 of the previous LCB (pointed to by PNT2) is set equal to the address of the LCB (contained in PNT3). If PNT1 is not null, P4 of the next LCB (pcinted to by PNT1) is set equal to PNT3. The LENGTH field of the new LCB is set equal to 28, the ID is set equal to X'80', P1 and P2 are both set equal to PNT3, COUNT is set equal to zero and the NAME field is set equal to NAME. Next, the MARK field, of the new LCB, is set according to IMARK. DEFNPS is called with NAME and a value for KFLAG is returned (see the section called "Defining a Push Down Store" in this chapter). If KFLAG is equal to two (not enough space to define a PDS called NAME) then IFLAG is set to two and control is returned to the calling routine. If KFLAG is not equal to two then IFLAG is set to zero, IADD is set equal to PNT3 and control is returned to the calling routine.

The GETSPACE section is entered with the required space, L. OBTNLP is called with L and values for KADD and KFLAG are returned (see section called "Obtaining a Block of Memory From the LAS" in this chapter). If KFLAG is equal to zero, the required space has been obtained so JFLAG is set to X'00', PNT3 is set equal to KADD and control is returned to the caller. If KFLAG is not equal tc zero then CONFLP is called and a value for ICNT is returned (see the section called "Connecting the LAS Fragments" in this chapter). If ICNT is equal to zero, there were no connections made so

JFLAG is set to X'80' and control is returned to the caller. If ICNT is not equal to zero, OBTNLP is called again. If KFLAG is equal to zero, the required space has been obtained so JFLAG is set to X'00', PNT3 is set equal to KADD and control is returned to the caller. If KFLAG is not equal to zero, the required space could not be obtained so JFLAG is set equal to X'80' and control is returned to the caller.

A flow chart for DEFNLP is given in Appendix C, chart D1. A flow chart for GETSPACE is given in Appendix C, chart G1. The coding for both is contained in Appendix D as part of the control section named DLINKLP.

Inserting an Element into a List

An element can be inserted into a list by making the following call:

```
CALL INSELP(P,IADD,L1,D,L2,DATA,IFLAG) .
```

P, IADD, L1, D, and DATA must be defined in the calling routine. An element is inserted after (P equal to one) or before (P equal to two) the element whose address is contained in IADD. L1 is the length, in bytes, of the element's user data field, D is the displacement, in bytes, into the user data field where DATA is to be moved. L2 is the length, in bytes, of DATA. DATA can be any type of variable, i.e. LOGICAL*1, INTEGER*4, etc. IADD is returned with the address of the inserted element and IFLAG is returned set to zero if the element is inserted successfully. If there is not enough space left in the LAS

for the element then IFLAG is returned set to one and IADD is unpredictable.

When INSELP is entered, PNT1 is loaded with IADD. If L1 is not a multiple of four, it is increased until it is an even multiple of four. LEN is set equal to L1 plus 12 then the section of coding which gets space is branched to with LEN as an argument (see the section called "Defining a List" in this chapter). If JFLAG is equal to X'80' then the required space can't be obtained from the LAS so IFLAG is set to one and control is returned to the calling routine. If JFLAG is not equal to X'80', processing continues with PNT3 containing the address of the element to be inserted. LENGTH of the inserted element is set equal to LEN and ID is set equal to X'00'. If P is equal to one, PNT2 is set equal to P1 of the element pointed to by PNT1, P1 of the element pointed to by PNT1 is set equal to PNT3, P1 of the inserted element is set equal to PNT2 and P2 of the inserted element is set equal to PNT1. If P is not equal to one, PNT2 is set equal to P2 of the element pointed to by PNT1, P2 of the element pointed to by PNT1 is set equal to PNT3, P1 of the element pointed to by PNT2 is set equal to PNT3, P2 of the inserted element is set equal to PNT2 and P1 of the inserted element is set equal to PNT1. FROM is set equal to the address of DATA, TO is computed as PNT3 plus D plus 12, LDATA is set equal to L2 and the section which moves data (MOVEDATA) is branched to, see below. Finally, IADD is set

equal to PNT3, IFLAG is set to zero and control is returned to the caller.

MOVEDATA is entered with FROM, TO and LDATA. ADD1 is set equal to TO, ADD2 is set equal to FROM and L is set equal to LDATA. If L is greater than 256 then 256 bytes of data are moved from ADD2 to ADD1, 256 is subtracted from L and is added to both ADD1 and ADD2, and then the above process is continued. If L is less than 256 then L bytes of data are moved from ADD2 to ADD1 and control is returned to the caller.

A flow chart for INSELP is given in Appendix C, chart D2. A flow chart for MOVEDATA is given in Appendix C, chart M1. The coding for both is contained in Appendix D as part of the control section named DLINKLP.

Inserting a LISTR into a List

A LISTR is an element which contains a pointer, P3, that points to a list control block (LCB). A LISTR may or may not contain a user data field (see Appendix B for a complete description of a LISTR). A LISTR with no user data field may be inserted into a list by making the following call:

```
CALL INSLLP(P,IADD,NAME,IFLAG) .
```

P, IADD, and NAME must be defined by the caller. A LISTR is inserted after (P equal to one) or before (P equal to two) the element whose address is in IADD. If NAME is null ($$$$) then P3 of the LISTR is set equal to null; otherwise,

P3 of the LISTR is set equal to the address of the LCB for the list called NAME. If the insertion is successful, IADD is returned with the address of the inserted LISTR and IFLAG is returned set to zero. IFLAG is returned set to one if a list called name is not defined and is returned set to two if there isn't enough space in the LAS for the LISTR.

A LISTR with an user data field may be inserted into a list by making the following call:

CALL INSLLP(P,IADD,NAME,IFLAG,L1,D,L2,DATA) .

P, IADD, NAME and IFLAG are the same as above. L1, D, L2 and DATA must be defined in the calling routine. L1 is the length, in bytes, of the LISTR's user data field. D is the displacement, in bytes, into the user data field where DATA is to be moved. L2 is the length, in bytes, of DATA. DATA can be any type of variable, i.e. LOGICAL*1, INTEGER*4, etc. The program logic follows.

When INSLLP is entered, NAME is checked to see if it contains a null pointer ($$$$). If so, PNT1 is set equal to null; otherwise, the LCBs are examined to see if a list called NAME is defined. If not, IFLAG is set equal to one and control is returned to the calling routine. If the list called NAME is defined then PNT1 is loaded with the address of the list's LCB.

LEN is set equal to 16 and LFLAG is set equal to X'00'. If IFLAG is the last argument, GETSPACE is branched to with a request for LEN bytes. If IFLAG is not the last argument

(LISTR will have an user data field) then LFLAG is set to X'80'. If L1 is not a multiple of four, it is increased until it is an even multiple of four. Next, L1 is added to LEN and GETSPACE is branched to, with a request for LEN bytes.

After returning from GETSPACE (see section called "Defining a List" in this chapter), JFLAG is tested to see if it is equal to X'80'. If so, there is not enough space in the LAS for the LISTR; therefore, IFLAG is set equal to two and control is returned to the calling routine. If JFLAG is not equal to X'80' then processing continues with PNT3 containing the address of the inserted LISTR. The COUNT field of the LCB pointed to by PNT1 is increased by one. LENGTH of the inserted LISTR is set equal to LEN, ID of the inserted LISTR is set equal to X'40' and P3 of the inserted LISTR is set equal to PNT1 (the address of the LCB for the list called NAME). PNT4 is set equal to IADD. If P is equal to one then PNT2 is set equal to P1 of the element pointed to by PNT4, P1 of the element pointed to by PNT4 is set equal to PNT3, P2 of the element pointed to by PNT2 is set equal to PNT3, P1 of the inserted LISTR is set equal to PNT2 and P2 of the inserted LISTR is set equal to PNT4. If P is not equal to one then PNT2 is set equal to P2 of the element pointed to by PNT4, P2 of the element pointed to by PNT4 is set equal to PNT3, P1 of the element pointed to by PNT2 is set equal to PNT3, P2 of the inserted LISTR is set

equal to PNT2 and P1 of the inserted LISTR is set equal to PNT4. Next, IADD is set equal to PNT3.

If LFLAG is equal to X'80' then MOVEDATA is branched to with FROM equal to the address of DATA, TO equal to PNT3 plus D plus 16 and with LDATA equal to L2. Finally, IFLAG is set to zero and control is returned to the calling routine. MOVEDATA is described in the section called "Inserting an Element into a List" in this chapter. A flow chart for INSLLP is given in Appendix C, chart D3. The coding for INSLLP is contained in Appendix D as part of the control section named DLINKLP.

Deleting an Element From a List

An element may be deleted from a list by making the following call:

```
CALL DLTELP(IADD) .
```

IADD must be defined in the calling routine as the address of an element. When DLTELP is entered, PNT1 is loaded with IADD. PNT2 is set equal to P1 of the element pcinted to by PNT1 PNT3 is set equal to P2 of the element pointed to by PNT1. P2 of the element pointed to by PNT2 is set equal to PNT3. Next, P1 of the element pointed to by PNT3 is set equal to PNT2. Finally, RTNELP is called with IADD as an argument, to return the space occupied by the element to the list of available space (LAS). After ccntrol is returned from RTNELP, the calling routine is returned to. RTNELP is described in the section called "Returning Element Space to

the LAS" in this chapter. A flow chart for DLTELP is given in Appendix C, chart D4. The coding for DLTELP is contained in Appendix D as part of the control section named DLINKLP.

Deleting a List

A list, as well as the associated push down store (PDS), can be deleted by making the following call:

CALL DLTLLP(NAME,IFLAG) .

NAME must be defined in the calling routine as the name of a list. IFLAG is returned set to zero if the list is deleted successfully, returned set to one if a list called NAME is not defined. When DLTLLP is entered PNT4 is loaded with the address of the first list control block (LCB). If PNT4 is null ($$$$), IFLAG is set to one and control is returned to the calling routine. If PNT4 is not null then NM1 is set equal to the NAME field of the LCB pointed to by PNT4. Next, NAME is compared with NM1. If NAME is less than NMM1 (recall that the LCBs are organized with the list names in ascending order), IFLAG is set to one and control is returned to the calling routine. If NAME is greater than NM1, PNT4 is chained to point to the next LCB and the above process is repeated. If NAME is equal to NM1 then KADD is set equal to PNT4 and RTNLLP is called, with KADD as an argument, to return the space occuppied by the list to the list of available space (LAS). Note that RTNLLP makes the call which deletes the associated PDS. Finally, IFLAG is set to zero and control is returned to the calling routine.

RTNLLP is described in the section called "Returning List Space to the LAS" in this chapter. A flow chart for DLTLLP is given in Appendix C, chart D5. The coding for DLTLLP is contained in Appendix D as part of the control section named DLINKLP.

III. TRAVERSING LISTS AND FETCHING CONTROL INFORMATION

The IBM/360 assembly language routine named TRAVSLP is used for traversing lists, traversing list structures, and also for obtaining control information. The routine contains entry points that are FORTRAN callable and that perform the following functions:

1) traverse a list structure,
2) traverse a list,
3) fetch the address of a list control block (LCB),
4) fetch the contents of a pointer,
5) fetch the ID of an element,
6) fetch the length of an element's user data field, and
7) fetch the address of an element's user data field.

The calling sequences, required arguments and program logic are described below. The arguments must be stored, in the calling routine, on a word boundary (i.e. INTEGER*4). A flow chart for list structure traversal is given in Appendix

C. The program logic of the other entries should be clear without the aid of flow charts. The coding is contained in Appendix D and the list elements are described in Appendix B.

Traversing a List Structure

A list structure may be traversed by making the following call:

CALL TRVLLP(NAME,P,IADD,IFLAG) .

NAME, P, and IADD must be defined in the calling routine. NAME tells which list is being traversed. IADD points to an element in the list (or to an element in a sublist of the list). P equal to one indicates that P1 is used for traversal (forwards). P equal to two indicates that P2 is used for traversal (backwards). The associated push down store (PDS), also called NAME, is used when a list structure is being traversed. IADD is returned with the address of the next element in the list structure and IFLAG is returned set to zero if the operation is completed normally. If a PDS called NAME is not defined, IFLAG is returned set equal to one. If the PDS called NAME is empty when the top element of the associated PDS is attempted to be popped, then IFLAG is returned set to two. If there is an attempt to push an element into the associated PDDS and the list of available space for the PDS elements is empty then IFLAG is returned set to three.

When TRVLLP is entered, JADD is set equal to IADD. The

ID of the element pointed to by JADD is tested to see if the element is a LISTR. If so, the section of coding which puts information into the associated PDS is branched to (see below). If not, JADD is chained (forward if P is equal to one or backward if P is equal to two) to point to the next element. If the ID of the element pointed to by the updated JADD indicates that the element is a LCB then the section of coding which gets information from the associated PDS is branched to (see below); otherwise, the ID is examined to see if the element pointed to is a LISTR with no user data field (ID equal to X'41'). If so, the section of coding which puts information into the PDS is branched to. If the element pointed to by the updated JADD is neither a LCB or a LISTR with no user data field then IADD is set equal to JADD, IFLAG is set to zero and control is returned to the calling routine.

When the section of coding which puts information into the PDS is entered, JADD contains the address of a LISTR. If P3 of the LISTR is null then control returns to the place where JADD is chained; otherwise, KADD is set equal to JADD and PUSHPS is called with NAME, KADD and KFLAG (see the section called "Pushing an Element into a Push Down Store" in this chapter). After control is returned from PUSHPS, KFLAG is tested. If KFLAG is equal to zero, JADD is set equal to the address of the LCB which is pointed to by P3 of JADD and control returns to the section where JADD is

chained. If KFLAG is equal to one then IFLAG is set to one and control is returned to the calling routine. If KFLAG is not equal to zero or one then IFLAG is set to three and control is returned to the calling routine.

When the section of coding which gets information from a PDS is entered, POPPS is called with NAME, KADD and KFLAG (see the section called "Popping an Element From a Push Down Store" in this chapter). After control is returned from POPPS, KFLAG is tested. If KFLAG is equal to zero then JADD is set equal to KADD and control is returned to the section of coding which chains JADD. If KFLAG is equal to one, IFLAG is set to one and control is returned to the calling routine. If KFLAG is not equal to zero or to one, IADD is set equal to JADD and IFLAG is set equal to two (this usually indicates that the LCB of the list called NAME has been encountered, i.e. the entire list structure has been traversed). Next, control is returned to the calling routine.

A flow chart for TRVLLP is given in Appendix C, chart T1. The coding for TRVLLP is contained in Appendix D as part of the control section named TRAVSLP.

Traversing a List

A list may be traversed by making the following call:

```
CALL TRVELP(P,IADD,IFLAG) .
```

P and IADD must be defined in the calling routine. IADD, on entry, contains the address of a list element. P indicates

whether IADD will be updated with the previous element address (P equal to two) or with the next element address (P equal to one). IADD is returned with the updated element address. IFLAG is returned set to zero if the element pointed to by the updated IADD is not a LISTR or a LCB. IFLAG is returned set to one if the element pointed to by the updated IADD is a LISTR. Finally, IFLAG is returned set to two if the element pointed to by the updated IADD is a LCB. The coding for TRVELP is contained in Appendix D as part of the control section named TRAVSLP.

Fetching the Address of a LCB

The address of a LCB can be fetched by using the function subprogram IANMLP as follows.

IADD=IANMLP(NAME) .

NAME must be defined in the control program. If NAME is equal to zero, on entry, then IADD will contain the address of the first LCB when control is returned from the function, unless there are no list control blocks. In this case, IADD will be set to zero upon return. If NAME is not equal to zero on entry, then IADD will contain the address of the LCB for the list called NAME when control is returned to the calling routine, unless there is not a list called NAME defined. In this case IADD will be set to zero upon return from the function. The coding for IANMLP is contained in Appendix D as part of the control section named TRAVSLP.

Fetching the Contents of a Pointer

The contents of a pointer can be fetched by using the function subprogram IPNTLP as follows.

```
IPNT=IPNTLP(IADD,IP) .
```

IADD and IP must be defined in the control routine. IPNT is set equal to the contents of pointer IP of the element pointed to by IADD, when the function returns control, if IP is valid for the element pointed to. IPNT is set to zero if IP is not valid. Valid IP pointers are: one and two for all elements; one, two and three if the element pointed to is a LISTR; and, one, two, four and five if the element pointed to is a LCB. The coding for IPNTLP is contained in Appendix D as part of the control section named TRAVSLP.

Fetching the ID of an Element

The ID of an element can be fetched by using the function subprogram IIDLP as follows.

```
ID=IIDLP(IADD) .
```

IADD must be defined in the control routine. ID is returned with zero if the element pointed to by IADD has an ID equal to X'00', four if the element pointed to is a LISTR with no user data field (ID equal to X'40'), five if the element pointed to is a LISTR with a user data field (ID equal to X'41') and eight if the element pointed to is a list control block (ID equal to X'80'). The coding for IIDLP is contained in Appendix D as part of the control section named TRAVSLP.

Fetching the Length of an Element's User Data Field

The length of an element's user data field can be fetched by using the function subprogram ILENLP as follows.

```
LENGTH=ILENLP(IADD) .
```

IADD must be defined in the control routine. When control is returned from the function, LENGTH is set equal to the user data field length for the element pointed to by IADD. The coding for ILENLP is contained in Appendix D as part of the control section named TRAVSLP.

Fetching the Address of an Element's User Data Field

The address of an element's user data field may be fetched by using the function subprogram IUADLP as follows.

```
JADD=IUADLP(IADD) .
```

IADD must be defined in the control routine. When control is returned from the function, JADD is set equal to the address of the user data field in the element pointed to by IADD, if there is one. If there is no user data field then JADD is set to zero, upon return from the function. The coding for IUADLP is contained in Appendix D as part of the control section named TRAVSLP.

IV. LOGICAL OPERATIONS

The IBM/360 assembly language routine called LOPERLP is used to perform logical operations. The routine contains entry points that are FORTRAN callable and that perform the following functions:

1) AND operation,

2) inclusive OR operation,

3) exclusive OR operation,

4) testing under mask,

5) logical comparing,

6) moving data, and

7) loading the address of a variable.

The calling sequences, required arguments and program logic are described below. The arguments, unless stated otherwise, must be stored, in the calling routine, on a word boundary (i.e. INTEGER*4). The routine gives the FORTRAN programmer the ability to perform operations which are normally only available at the machine language level. The coding for the routine is contained in Appendix D.

AND Operation

The logical AND operation can be performed by using the function subprogram INCLP as follows.

J=INCLP(IADD,ID,JADD,JD,N) .

IADD, ID, JADD, JD and N must be defined in the control routine. The logical product (AND) of the bits at location one and location two are stored at location one. Location one is computed as the address contained in IADD plus the displacement contained in ID. Location two is computed as the address contained in JADD plus the displacement contained in JD. N is the length, in bytes (one to 256), of the strings. J, on return from the function, is set to zero

if the result is zero, and is set to one if the result is not zero. The coding for INCLP is contained in Appendix D as part of the control section named LOPERLP.

Inclusive OR Operation

The inclusive OR operation can be performed by using the function subprogram IOCLP as follows.

```
J=IOCLP(IADD,ID,JADD,JD,N) .
```

IADD, ID, JADD, JD and N must be defined in the control routine. The program logic is the same as the logic described for the AND operation (see section called "AND Operation" in this chapter) except that the inclusive OR operation is performed. The coding for IOCLP is contained in Appendix D as part of the control section named LOPERLP.

Exclusive OR Operation

The exclusive OR operation can be performed by using the function subprogram IXCLP as follows.

```
J=IXCLP(IADD,ID,JADD,JD,N) .
```

IADD, ID, JADD, JD and N must be defined in the control routine. The program logic is the same as the logic described for the AND operation (see section called "AND operation" in this chapter) except that the exclusive OR operation is performed. The coding for IXCLP is contained in Appendix D as part of the control section named LOPERLP.

Testing Under Mask

The test under mask operation can be performed by using

the function subprogram ITMLP as follows.

```
J=ITMLP(IADD,ID,M) .
```

IADD, ID and M must be defined in the control routine. The address contained in IADD is added to the displacement contained in ID to form location one. M is a byte (LOGICAL*1) mask which is used to select bits that are tested at location one. A bit, at location one, is tested if the corresponding bit position in the mask is a one. After control is returned, J is set as follows. J is set to zero if the selected bits are all zero or if the mask is zero. J is set to one if the selected bits are mixed zero and one. J is set to three if the selected bits are all one. The coding for ITMLP is contained in Appendix D as part of the control section named LOPERLP.

Logical Comparing

A logical comparison of strings (character string or bit string) may be performed by using the function subprogram ICLCLP as follows.

```
J=ICLCLP(IADD,ID,JADD,JD,N) .
```

IADD, ID, JADD, JD and N must be defined in the control routine. The string at location one (STRING1) is compared to the string at location two (STRING2). location one is computed as the address contained in IADD plus the displacement contained in ID. Location two is computed as the address contained in JADD plus the displacement contained in JD. N is the length, in bytes (one to 256), of

the strings. When control is returned from the function, J is set as one of the following. J is set to zero if STRING1 is equal to STRING2. J is set to one if STRING1 is less than STRING2. Finally, J is set to two if STRING1 is greater than STRING2. The coding for ICLCLP is contained in Appendix D as part of the control section named LOPERLP.

Moving Data

Data can be moved by making the following call:

```
CALL IMVCLP(IADD,ID,JADD,JD,K) .
```

IADD, ID, JADD, JD and K must be defined in the calling routine. K bytes of data are moved from location two to location one. Location one is computed as the address contained in IADD plus the displacement contained in ID. Location two is computed as the address contained in JADD plus the displacement contained in JD. Note that K is not limited to 256. The coding for IMVCLP is contained in Appendix D as part of the control section named LOPERLP.

Loading the Address of a Variable

The address of a variable may be loaded by using the function subprogram ILALP as follows.

```
J=ILALP(IVAL) .
```

IVAL is a variable defined in the control routine. When control is returned from the function, J is set equal to the address of the variable, IVAL. The coding for ILALP is contained in Appendix D as part of LOPERLP.

V. PUSH DOWN STORES

Push down stores (PDS) are created and maintained using the IBM/360 assembly language routine named PUSHST. The routine contains entry points that are FORTRAN callable and that perform the following functions:

1) initialize memory for PDSs,
2) define a push down store,
3) delete a push down store,
4) push an element into a PDS,
5) pop an element from a PDS,
6) release memory allocated for PDSs, and
7) examine push down stores.

The calling sequences, required arguments and program logic are described below. The arguments must all be stored, in the calling routine, on a word boundary (i.e. INTEGER*4). Flow charts for the routine are given in Appendix C, the coding is contained in Appendix D and the PDS elements are described in Appendix B. It is expected that the information contained in Appendix B and Appendix C will be needed to fully understand the program logic. The following internal variables are used in the routine.

1) NPNT - Points to the first PDS name element.
2) PLAS - Contains the address of the first element in the PDS element list of available space (PLAS).
3) PLAST - Points to the last element in the PLAS.

4) NLAS - Contains the address of the first element in the PDS name element LAS (NLAS).

5) NLAST - Points to the last element in the NLAS.

6) PLENGH - Contains the size of the memory block which is used for storing PDS elements.

7) PBLKAD - Contains the address of the first byte of the memory block which is used for storing the PDS elements.

8) NLENGH - Contains the size of the memory block which is used for storing PDS name elements.

9) NBLKAD - Contains the address of the first byte of the memory block which is used for storing PDS name elements.

10) FLAGS - Used to indicate initialization.

Initializing Memory for Push Down Stores

Memory for storing push down stores (PDS) can be initialized by making the following call:

CALL INITPS(N,NPS) .

N and NPS must be defined in the calling routine. N tells how many elements are to be initialized in the list of available space for the PDS elements (PLAS). Each PDS element requires eight bytes of memory. NPS tells how many elements are to be initialized in the list of available space for the PDS name elements (NLAS). Each PDS name element requires 12 bytes of memory.

When INITPS is entered, FLAGS is checked to see if

memory for push down stores is already initialized. If so, control is returned to the calling routine, otherwise processing continues as follows. PLENGH is computed as N multiplied by eight and then a GETMAIN macro is issued with a request for PLENGH bytes of core. The address of the block of memory is stored in PBLKAD. Next, PLAS is set equal to PBLKAD and PLAST is computed as PBLKAD plus PLENGH minus eight. The PDS elements in the PLAS are chained together by setting P1 of each element, except the last, equal to the address of the next PDS element. P1 of the last PDS element is set equal to the null pointer ($$$$). The above process is repeated for the NLAS after NLENGH is computed as NPS multiplied by 12. Finally, NPNT is set equal to $$$$, FLAGS is set to X'80' and control is returned to the calling routine. A flow chart for INITPS is given in Appendix C, chart P1, and the coding is contained in Appendix D as part of the control section named PUSHST.

Defining a Push Down Store

A push down store (PDS or stack) may be defined by making the following call:

```
CALL DEFNPS(NAME1,IFLAG) .
```

NAME1 must be defined in the calling routine. The defined PDS is identified by NAME1 which can consist of any four bytes of data except the null pointer ($$$$). IFLAG is returned set equal to zero if the PDS is defined successfully. It is returned set to one if the PDS called

NAME1 is already defined and is returned set to two if the NLAS is empty.

When DEFNPS is entered, FLAGS is checked to see if memory for the PDSs has been initialized. If not, the job terminates with error PDS31. If memory is initialized then the PDS name elements are looked at to see if a PDS called NAME1 is already defined. First, PNT1 is set equal to the address of NPNT and PNT is set equal to the contents of NPNT. PNT is tested to see if it contains a null pointer. If so, all PDS name elements have been looked at without finding one whose NAME field is equal to NAME1; therefore, a PDS name element is ready to be defined. If PNT does not contain a null pointer, NAME1 is compared with NAME, of the element pointed to by PNT, giving three possible outcomes. If NAME1 is equal to NAME, the PDS is already defined so IFLAG is set to one and control is returned to the caller. If NAME1 is less than NAME, a PDS name element is ready to be defined (this is because the PDS name elements are chained with the NAMEs in ascending order). If NAME1 is greater than NAME then PNT1 is computed as PNT plus four, PNT is chained to point to the next PDS name element and the above process is repeated.

Before a PDS name element can be inserted, NLAS must be checked to see if it is empty (NLAS equal to $$$$). If so, IFLAG is set to two and control is returned to the calling routine. If not, PNT2 (the pointer to the address of the

new PDS name element) is set equal to NLAS and NLAS is updated to be equal to P1 of the element pointed to by the previous NLAS. PNT2 is stored at the address contained in PNT1. The NAME field of the new PDS name element is set equal to NAME1, P1 is set equal to PNT and P2 is set to $$$$. Finally, IFLAG is set to zero and control is returned to the caller. A flow chart for DEFNPS is given in Appendix C, chart P2, and the coding is contained in Appendix D as part of the control section named PUSHST.

Deleting a Push Down Store

A push down store (PDS) may be deleted by making the following call:

```
CALL DELTPS(NAME1,IFLAG) .
```

NAME1 must be defined in the calling routine. NAME1 identifies the PDS which is to be deleted. IFLAG is returned set to zero after the PDS is deleted successfully. It is returned set to one if the PDS called NAME1 is undefined.

When DELTPS is entered, FLAGS is checked to see if memory for the PDS has been initialized. If not, the job terminates with error PDS61. If memory is initialized, a PDS name element which has it's NAME field equal to NAME1 is searched for (see section called "Defining a Push Down Store" in this chapter). If a PDS called NAME1 is not defined, IFLAG is set to one and control is returned to the calling routine. If a PDS called NAME1 is found, PNT is

loaded with the address of the PDS name element and PNT1 is loaded with the address of the pointer which points to the PDS name element. Next, P1 of the element pointed to by PNT is stored at the address contained in PNT1. P1 of the element pointed to by NLAST is set equal to PNT and NLAST is set equal to PNT. After this, P1 of the element pointed to by the updated NLAST is set equal to $$$$. Finally, the elements of the deleted PDS are chained to the end of the PDS element list of available space (PLAS), IFLAG is set to zero and control is returned to the calling routine. A flow chart for DELTPS is given in Appendix C, chart P3, and the coding is contained in Appendix D as part of the control section named PUSHST.

Pushing an Element Into a Push Down Store

An element can be pushed to the top of a push down store (PDS) by making the following call:

CALL PUSHPS(NAME1,IADD,IFLAG) .

NAME1 and IADD must be defined in the calling routine. NAME1 specifies which PDS the element will be put into. IADD is placed in the user data field of the inserted element. IFLAG is returned set to zero if the element is pushed into the PDS successfully. IFLAG is returned set to one if there is no PDS called NAME1 defined and is returned set to two if the list of available space for the PDS elements (PLAS) is empty.

The program logic is the following. PNT is loaded with

NPNT. If PNT is equal to $$$$ then IFLAG is set to one and control is returned to the calling routine (this indicates that the PDS called NAME1 is not defined). If PNT is not equal to $$$$ then NAME1 is compared to the NAME field of the element pointed to by PNT. If NAME1 is less than NAME then IFLAG is set to one and control is returned to the calling routine. If NAME1 is greater than NAME then PNT is chained to point to the next PDS name element and the above logic is repeated. If NAME1 is equal to NAME, of the element pointed to by PNT, then a PDS name element for a PDS called NAME1 has been found and processing continues.

If the PDS element list of available space is empty (PLAS equal to $$$$) then IFLAG is set to two (indicating that there isn't enough memory space to insert an element into a PDS) and control is returned to the calling routine. If PLAS is not equal to $$$$ then PNT2 is set equal to PLAS and PLAS is updated to contain P1 of the element pointed to by PNT2. If the updated PLAS is equal to $$$$ then PLAST is loaded with the address of PLAS. Next, P1 of the element pointed to by PNT2 is set equal to P2 of the element pointed to by PNT (the previous top of the PDS). P2 of the PDS name element pointed to by PNT is set equal to PNT2. Finally, the user data field of the element pointed to by PNT2 is set equal to IADD, IFLAG is set to zero and control is returned to the calling routine. A flow chart for PUSHPS is given in Appendix C, chart P4, and the coding is contained in

Appendix D as part of the control section named PUSHST.

Popping an Element From a Push Down Store

The top element of a push down store (PDS) can be popped by making the following call:

CALL POPPS(NAME1,IADD,IFLAG) .

NAME1 must be defined in the calling routine. It specifies which PDS will have its top element popped. IADD is returned set equal to the contents of the user data field of the popped element. IFLAG is returned set to zero, if the element is popped successfully. If a PDS called NAME1 is not defined then IFLAG is returned set to one. If the PDS called NAME1 is empty, IFLAG is returned set to two.

The program logic is the following. First, a PDS called NAME1 is searched for (see section called "Pushing an Element Into a Push Down Store" in this chapter). If a PDS called NAME1 isn't found then IFLAG is set to one and control is returned to the calling routine. If a PDS called NAME1 is found then PNT is loaded with the address of the PDS name element whose NAME field is equal to NAME1. Next, PNT2 is loaded with P2 of the element pointed to by PNT. If PNT2 contains a null pointer ($$$$) then IFLAG is set to two (to indicate that the PDS called NAME1 is empty) and control is returned to the calling routine. If PNT2 is not null then P2 of the PDS name element pointed to by PNT is set equal to P1 of the element pointed to by PNT2. P1 of the element pointed to by PLAST is set equal to PNT2 (the new

last element in the PDS element list of available space). Next, PLAST is updated to point to PNT2 and P1 of the updated PLAST is set equal to $$$$. Finally, IADD is set equal to the contents of the user data field of the element pointed to by PNT2, IFLAG is set to zero and control is returned to the calling routine. A flow chart for POPPS is given in Appendix C, chart P5, and the coding is contained in Appendix D as part of the control section named PUSHST

Releasing Memory Allocated for Push Down Stores

Memory used for storing push down store (PDS) elements can be released by making the following call:

CALL RLSEPS .

When RLSEPS is entered, FLAGS is tested to see if memory space is initialized. If not, the job is terminated with error PDS21; otherwise, processing continues. PLENG and PBLKAD are loaded then a FREEMAIN macro is issued to release PLENG bytes of memory, beginning at the address specified in PBLKAD. Next, NLENG and NBLKAD are loaded then a FREEMAIN macro is issued to release NLENGH bytes of memory, beginning at the address specified in NBLKAD. Finally, FLAGS is zeroed and control is returned to the calling routine. A flow chart for RLSEPS is given in Appendix C, chart P6, and the coding is contained in Appendix D as part of the control section named SPACELP.

Examining Push Down Stores

Push down stores (PDSs) may be dumped by making the following call:

CALL DUMPPS(ICNT,NAME1) .

ICNT and NAME1 (if ICNT is equal to two) must be defined in the calling routine. ICNT is used in the following manner. If ICNT is equal to zero then all PDS names and the content of the PDSs are dumped. If ICNT is equal to one then only the PDS names are dumped. If ICNT is equal to two then the PDS called NAME1 is dumped. DUMPPS makes use of the routines PEEPS and NEEPS, which are described below. All information is dumped using a Z format specification (this gives hexadecimal representations). If IPNT, in the labelled COMMON COM$$$, is not equal to zero then only the counts are printed. All output is printed on logical unit IOUT which is defined in the COMMON COM$$$. See the sections called "PROGRAMMING CONSIDERATIONS" and "SYSTEM CONSIDERATIONS" , in Chapter II, for discussions dealing with the COMMON block COM$$$.

PEEPS is used in the following manner.

CALL PEEPS(NAME1,IADD1,IADD2,IFLAG) .

NAME1 and IADD1 must be defined in the calling routine. If IADD1 is zero, on entry, then IADD1 is loaded with P2 of the PDS name element which has a NAME field equal to NAME1 and IFLAG is returned set to zero, unless IADD1 has been loaded with a null pointer ($$$$). In this case (the PDS called

NAME1 is empty), IADD1 is returned set equal to two. IFLAG is returned set equal to one if a PDS called NAME1 is not defined. If IADD1 is not equal to zero, on entry, then IADD1 is returned set equal to P1 of the element pointed to by IADD1, on entry, and IADD2 is returned set equal to the contents of the user data field of the element pointed to by IADD1, on entry. IFLAG is returned set to zero unless IADD1 is loaded with a null pointer. In this case, IFLAG is returned set equal to three.

NEEPS is used in the following manner.

CALL NEEPS(NAME1,IADD,IFLAG) .

IADD must be defined in the calling routine. If IADD is not equal to zero, on entry, then NAME1 is loaded with the name of the PDS name element pointed to by IADD. IADD is then updated to point to the next PDS name element. IFLAG is returned set to zero, if the operation is completed successfully or set to one, if IADD is loaded with a null pointer. If IADD is equal to zero, on entry, then IADD is returned set equal to the address of the first PDS name element. IFLAG is returned set to zero if the operation is completed successfully, or set equal to two if there are no stacks defined.

The coding for DUMPPS is contained in Appendix D. A flow chart for PEEPS is given in Appendix C, chart P7, and the coding is contained in Appendix D as part of the control section named PUSHST. A flow chart for NEEPS is given in

Appendix C, chart P8, and the coding is contained in Appendix D as part of the control section named PUSHST.

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Knuth, D. E., Fundamental Algorithms, Addison-Wesly Publishing Company, Menlo Park, California, 1968.

2. Maurer, W. D., Programming: An Introduction to Computer Languages and Techniques, Holden-Day, Inc., San-Francisco, California, 1969.

3. Maisel, H., Introduction to Electronic Digital Computers, McGraw-Hill Book Company, New York, 1969.

4. Weizenbaum, J., "Symmetric List Processor", Comm. ACM 6,9 (Sept. 1963).

5. Bobrow, D. G. and Raphael, B., "A Comparison of List Processing Computer Languages", Comm. ACM 7,4 (April, 1964).

6. Newell, A., Information Processing Language-V Manual, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1961.

7. IBM System/360 Principles of Operation, GA22-6821, International Business Machine Corporation, 1970.

8. IBM System/360 Operating System Supervisor Services and Macro Instructions, GC28-6646, International Business Machine Corporation, 1972.

9. IBM System/360 Operating System: Job Control Language Reference, GC28-6704, International Business Machine Corporation, 1972.

APPENDIXES

APPENDIX A

ERROR MESSAGES

ERROR	DESCRIPTION
LPS21	On a call to RLSELP, there was an attempt to release space used for building lists when no space had been initialized.
PDS21	On a call to RLSEPS, there was an attempt to release space used for building push down stores, but no space was initialized.
PDS31	On a call to DEFNPS, it was found that space for defining push down stores had not been initialized.
PDS61	On a call to DELTPS, it was found that space for defining push down stores had not been initialized.

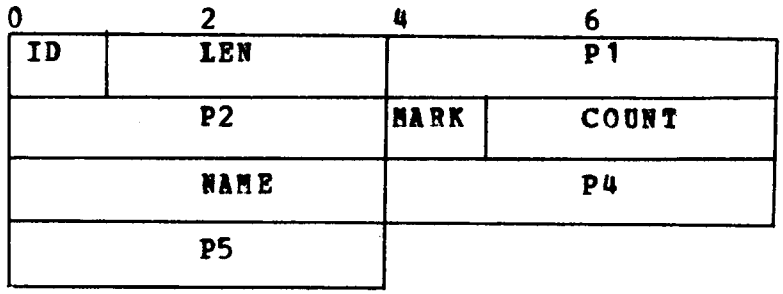
APPENDIX B

ELEMENT DESCRIPTIONS

I. LIST ELEMENTS

List Control Block

Displacement (Bytes)



Field Description

ID - Identifies the type of element; X'80' for a LCB.

LEN - Length of the element; 28 for a LCB.

P1 - Points to the first element in the list.

P2 - Points to the last element in the list.

MARK - Controls automatic deletion of list; X'80' for no automatic deletion.

COUNT - Number of times the list is used as a sublist.

NAME - Name of the list (any characters except $$$$).

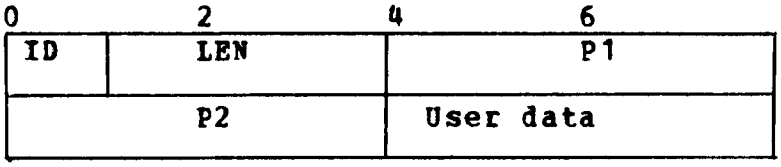
P4 - Points to the previous LCB.

P5 - Points to the next LCB.

Note: LCBs are stored in ascending collating sequence. For each LCB there is an associated PDS which has the same name.

List Element (Simple)

Displacement (Bytes)



Field Description

ID - Identifies the type of element; X'00' for a simple element

LEN - Length of the element; 12 plus the length of the user data field.

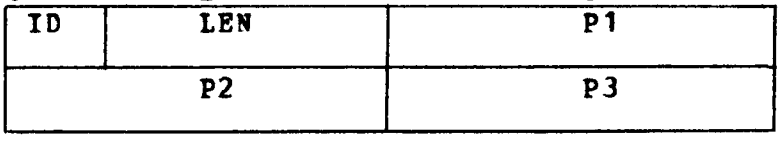
P1 - Points to the next element in the list.

P2 - Points to the previous element in the list.

User data - Any type of data; length must be an integer multiple of four.

LISTR with no User Data Field

Displacement (Bytes)



Field Description

ID - Identifies the type of element; X'40' for a LISTR with no user data field.

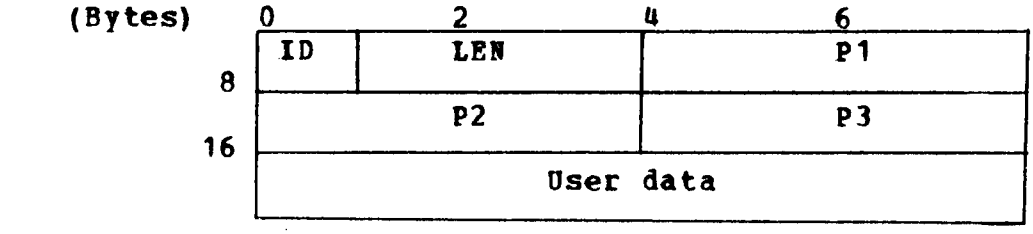
LEN - Length of the element; 16.

P1 - Points to the next element in the list.

P2 - Points to the previous element in the list.

P3 - Points to the LCB for a sublist.

LISTR with a User Data Field



Field	Description

ID - Identifies the type of element; X'41'.

LEN - Length of the element; 16 plus the length of the user data field.

P1 - Points to the next element in the list.

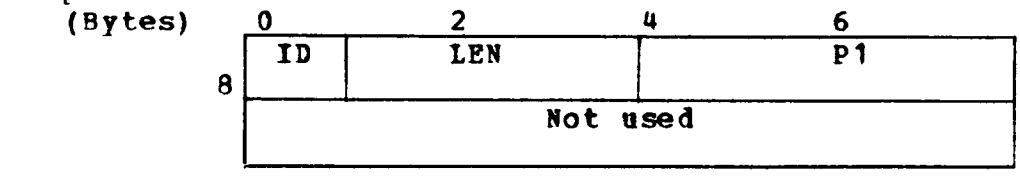
P2 - Points to the previous element in the list.

P3 - Points to the LCB for a sublist.

User data - Any type of data; length must be an integer multiple of four.

II. LAS ELEMENTS

Simple LAS element



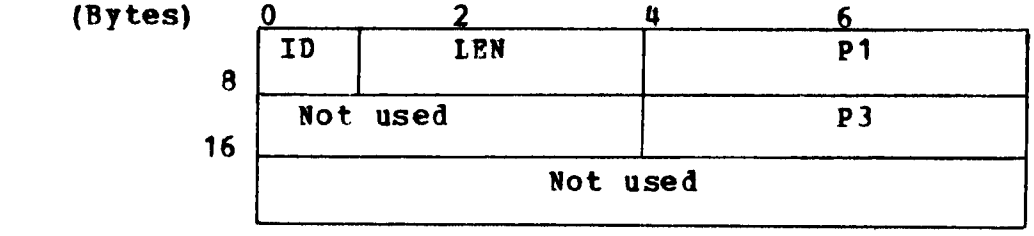
Field	Description

ID - Identifies the type of element; X'00' for a simple LAS element.

LEN - Length of the element.

P1 - Points to the next element in the LAS.

LAS LISTR



Field	Description

ID - Identifies the type of element; X'4-' for a LAS LISTR.

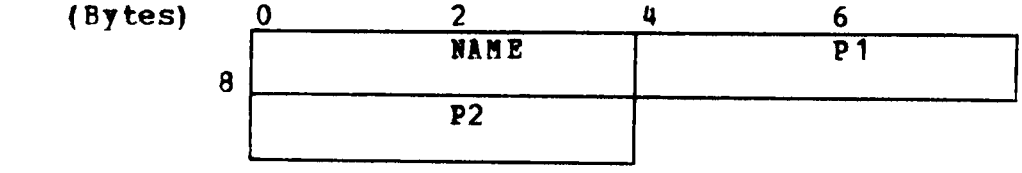
LEN - Length of the element.

P1 - Points to the next element in the LAS.

P3 - Points to the LCB for a list.

III. PDS ELEMENTS

PDS Name Element



Field	Description

NAME - Name of the PDS (any characters except $$$$).

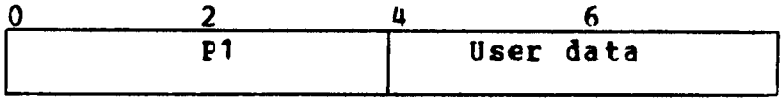
P1 - Points to the next PDS name element.

P2 - Points to the top element in the PDS called NAME.

Note: The PDS names are stored in ascending collating sequence.

PDS Element

Displacement (Bytes)



Field	Description
P1	Points to the next element in the PDS.
User data	A word (four bytes) of user data.

APPENDIX C

FLOW CHARTS

I. ABBREVIATIONS AND NOTATIONS

1) $$$$ - null pointer
2) E.P.T.B. - element pointed to by
3) LAS - list of available space
4) LCB - list control block
5) NLAS - list of available space for PDS name elements
6) PDS - push down store
7) PLAS - list of available space for PDS elements

II. CHARTS

DLINKLP

1) D1: DEFNLP
2) D2: INSELP
3) D3: INSLLP
4) D4: DLTELP
5) D5: DLTLLP

PUSHST

1) P1: INITPS
2) P2: DEFNPS
3) P3: DELTPS
4) P4: PUSHPS

5) P5: POPPS

6) P6: RLSEPS

7) P7: PEEPS

8) P8: NEEPS

SPACELP

1) S1: INITLP

2) S2: RLSELP

3) S3: OBTNLP

4) S4: CONFLP

5) S5: RTNLLP

6) S6: RTNELP

7) S7: PLASLP

TRAVSLP

1) T1: TRVLLP

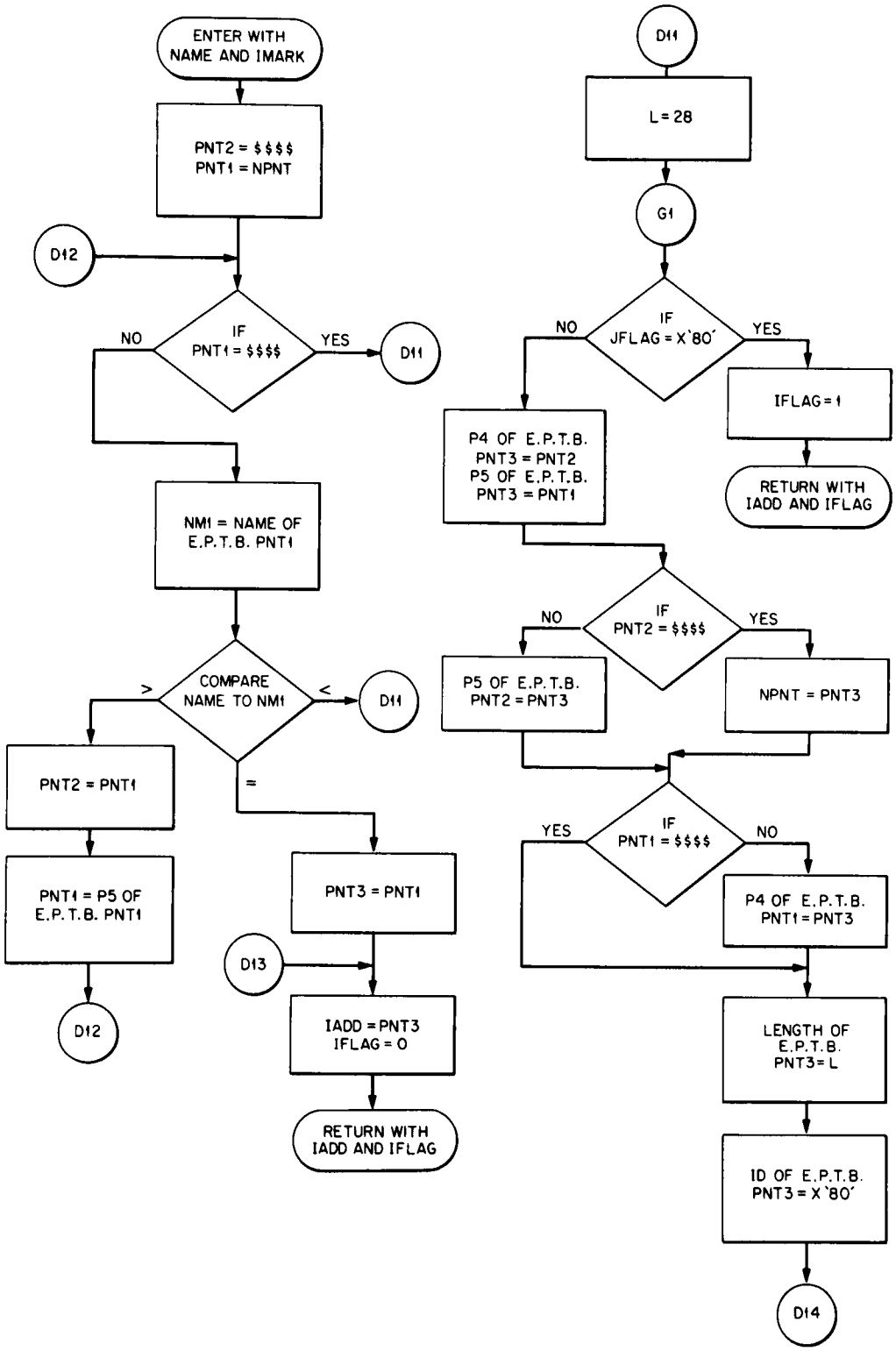


Chart D1: DEFNLP

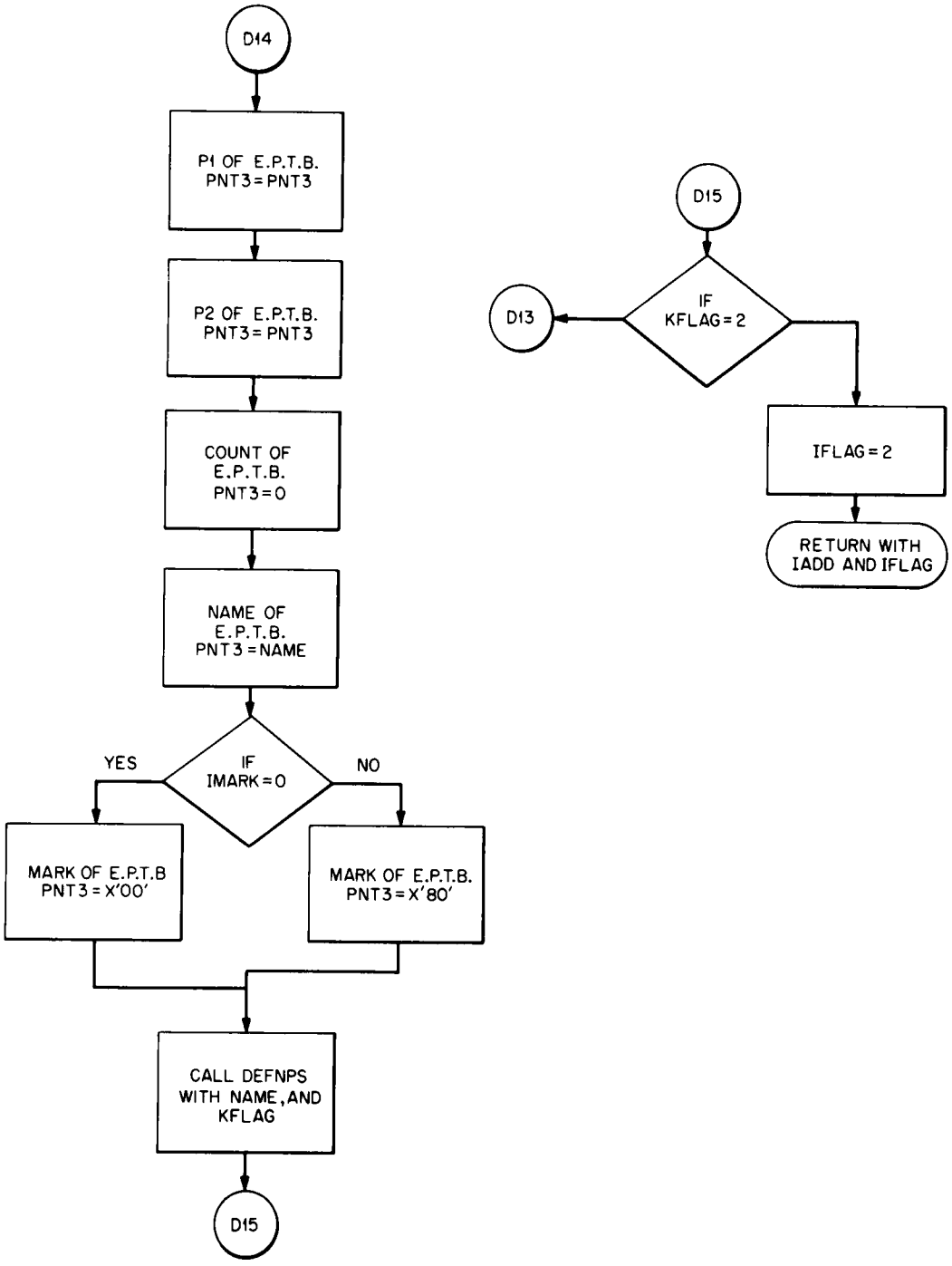


Chart D1: Continued

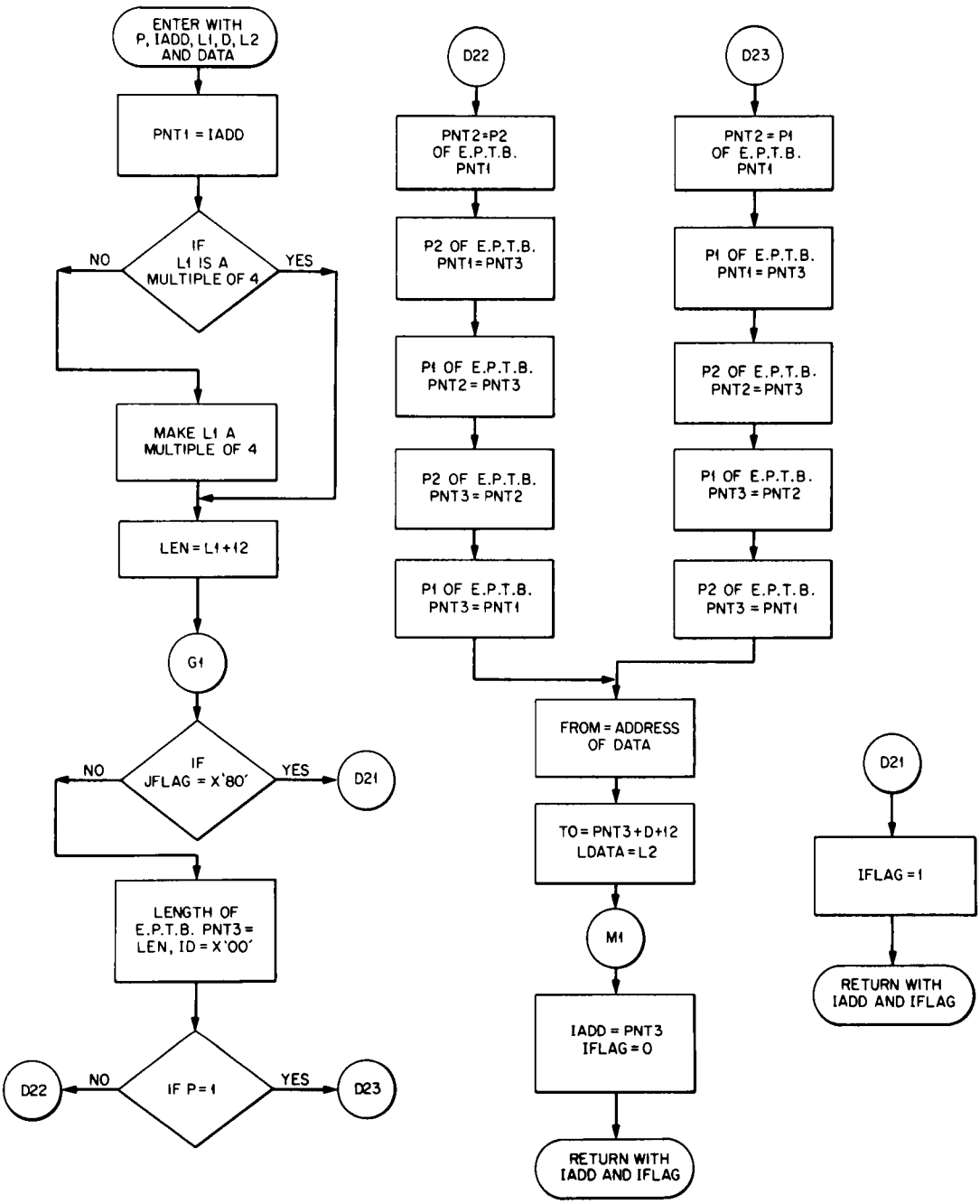


Chart D2: INSELP

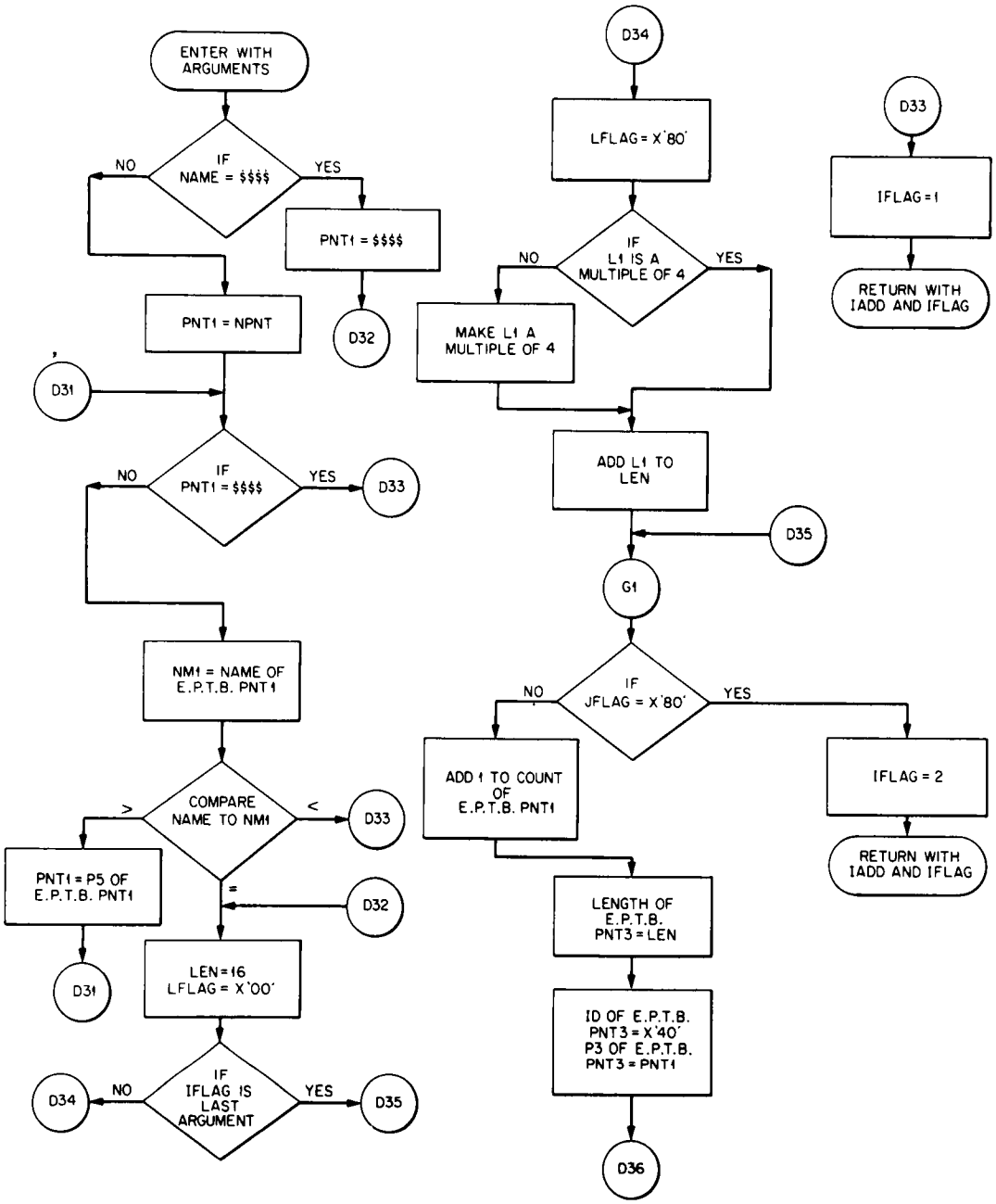


Chart D3: INSLLP

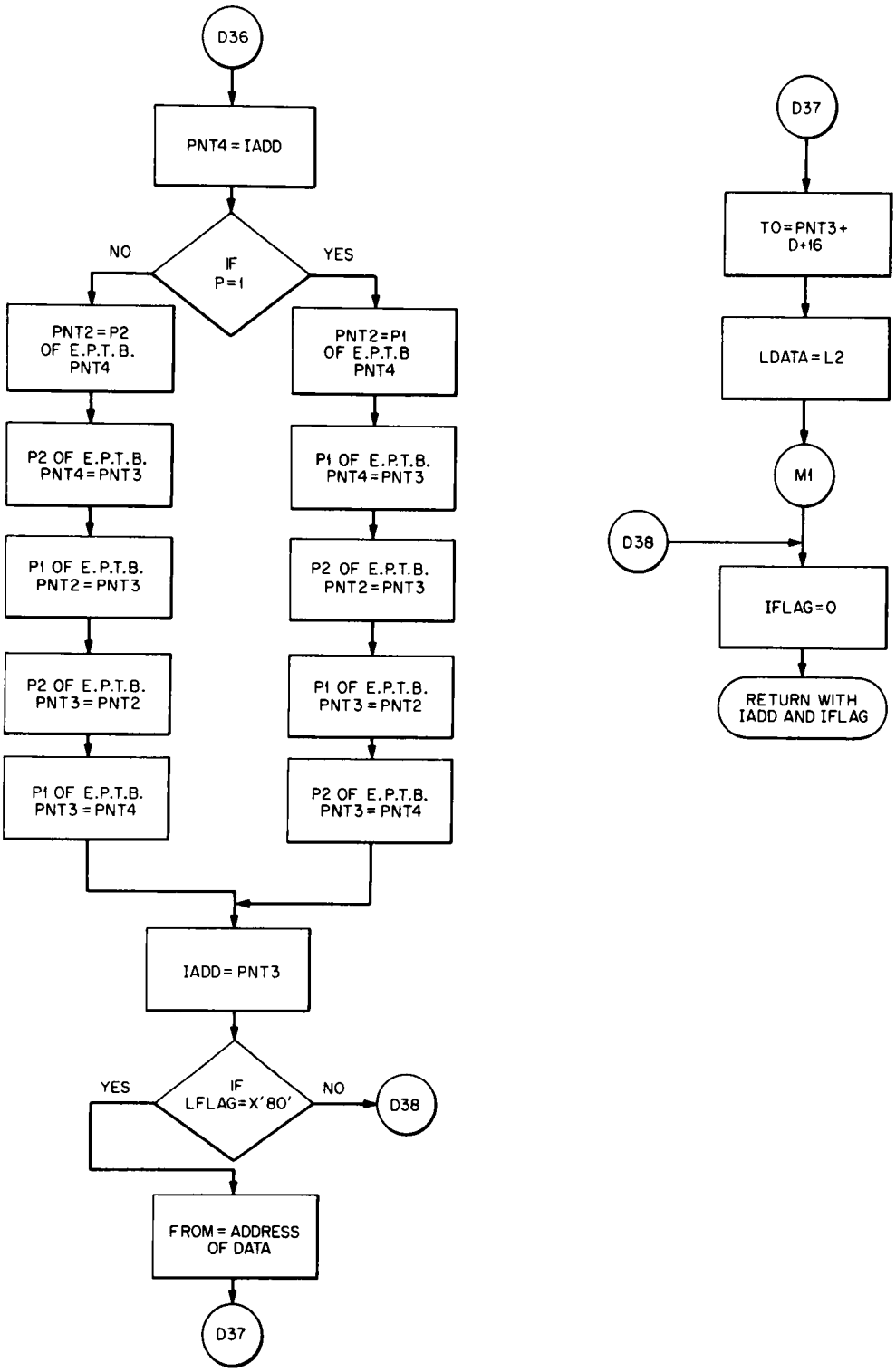


Chart D3: Continued

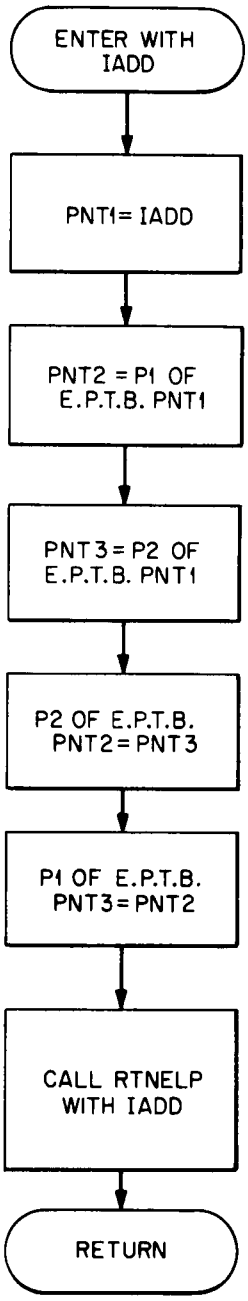


Chart D4: DLTELP

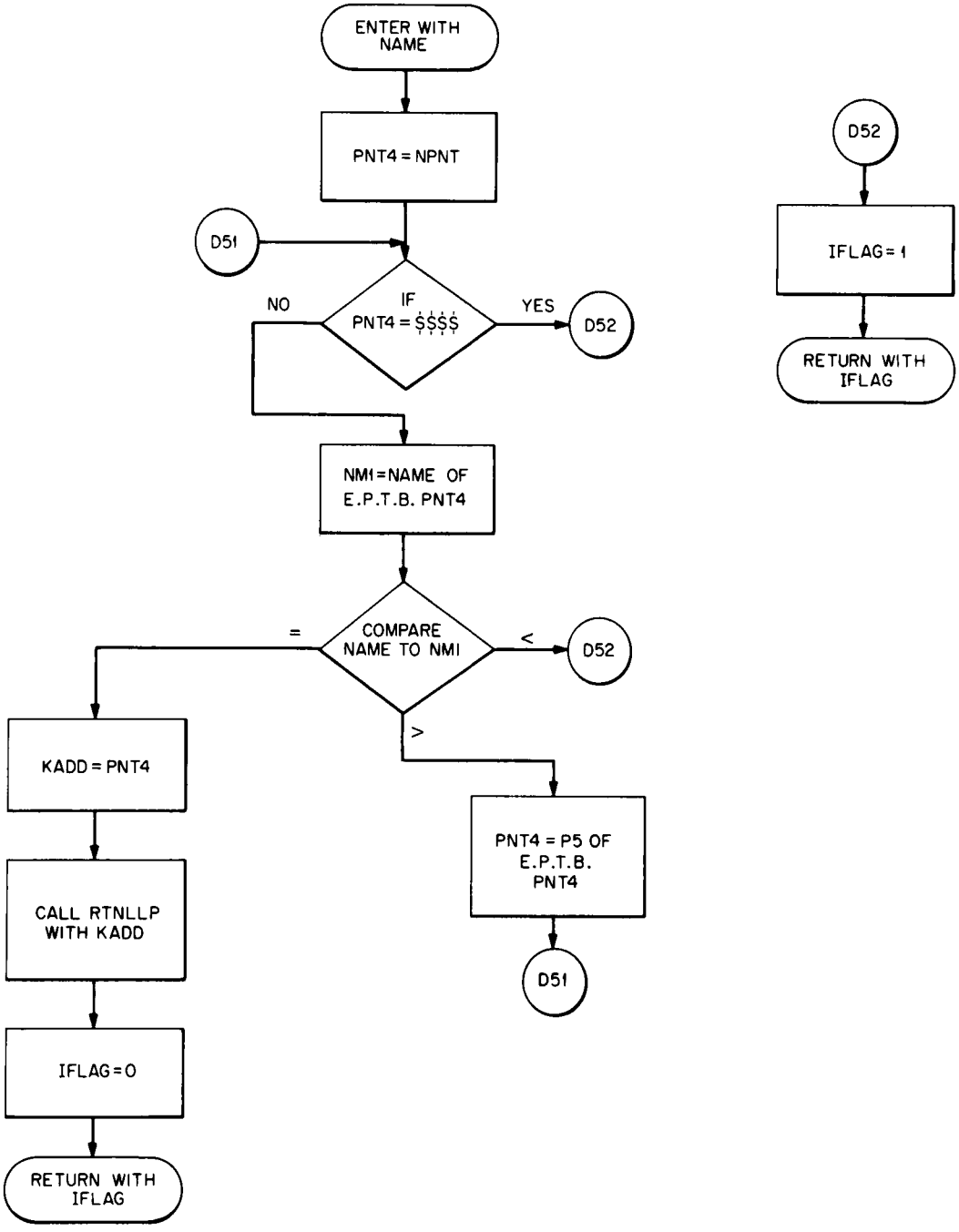


Chart D5: DLTLLP

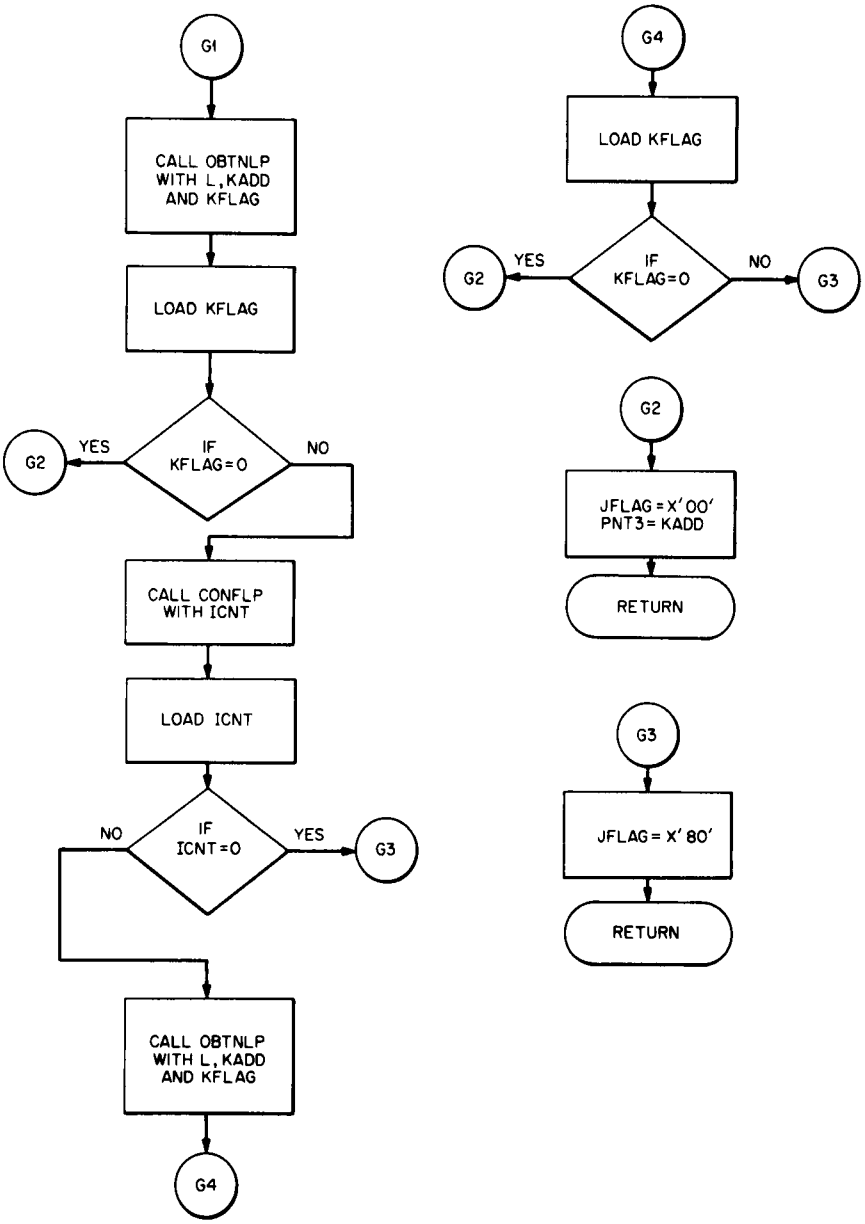


Chart G1: GETSPACE

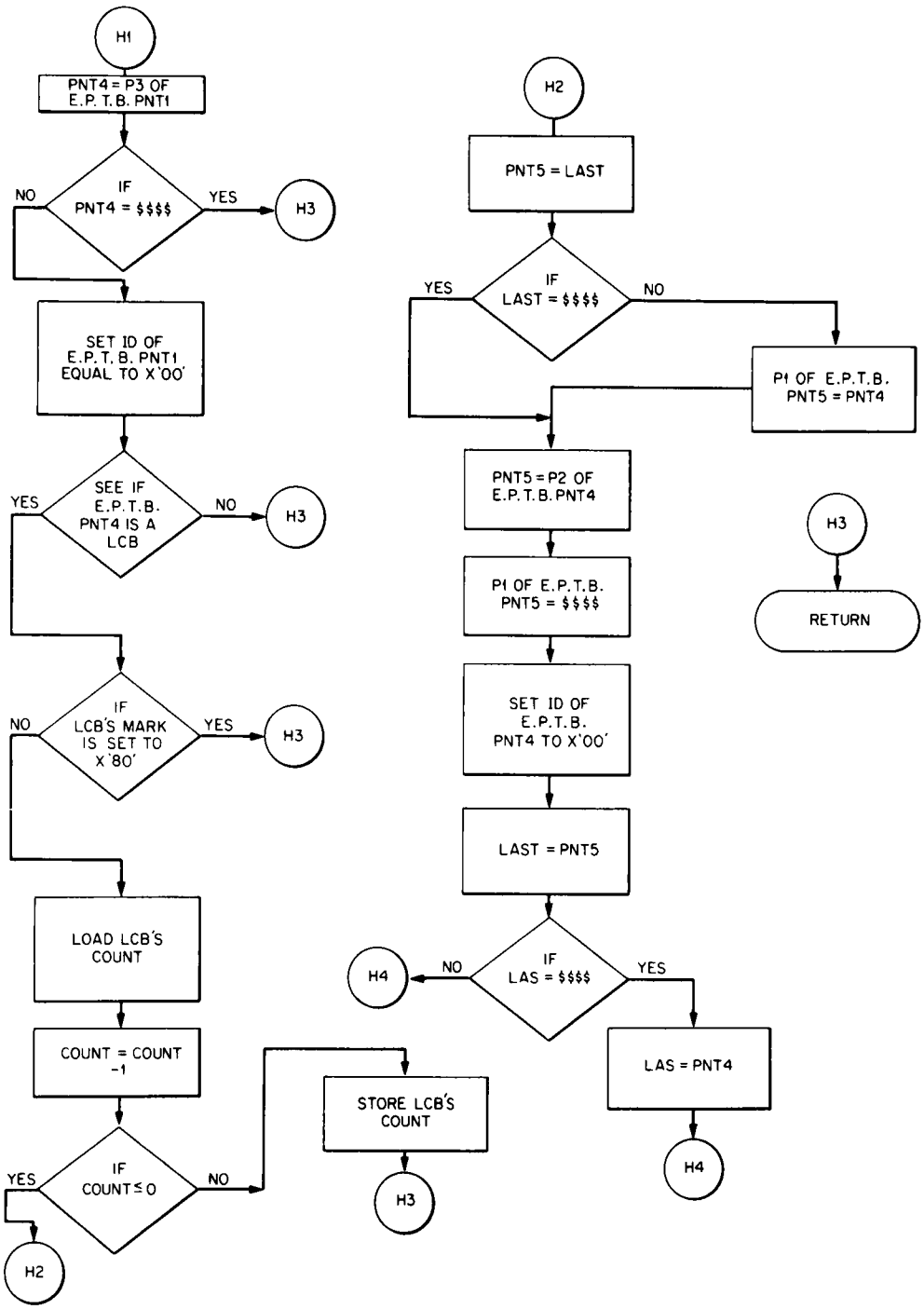


Chart H1: HANDLE

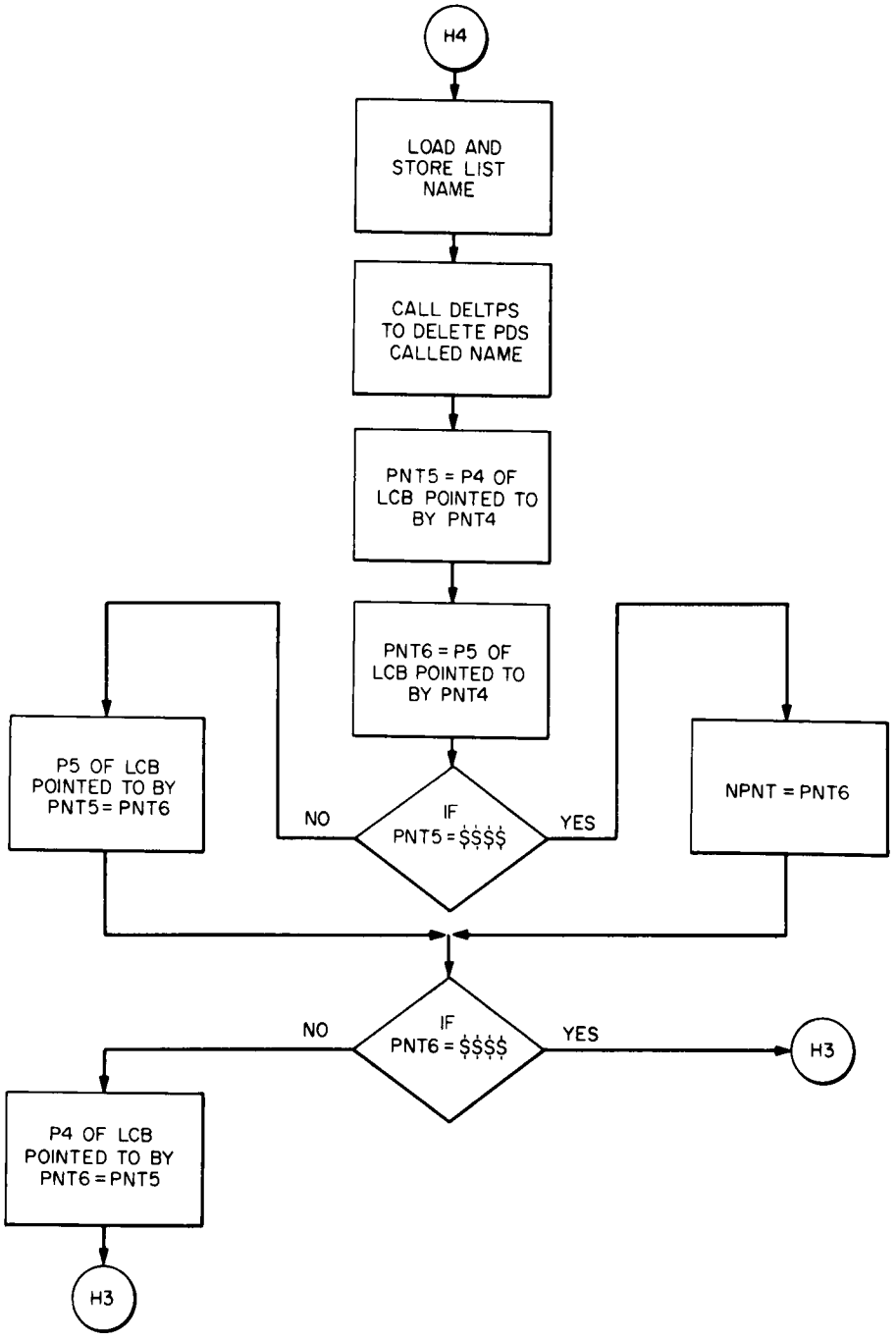


Chart H1: Continued

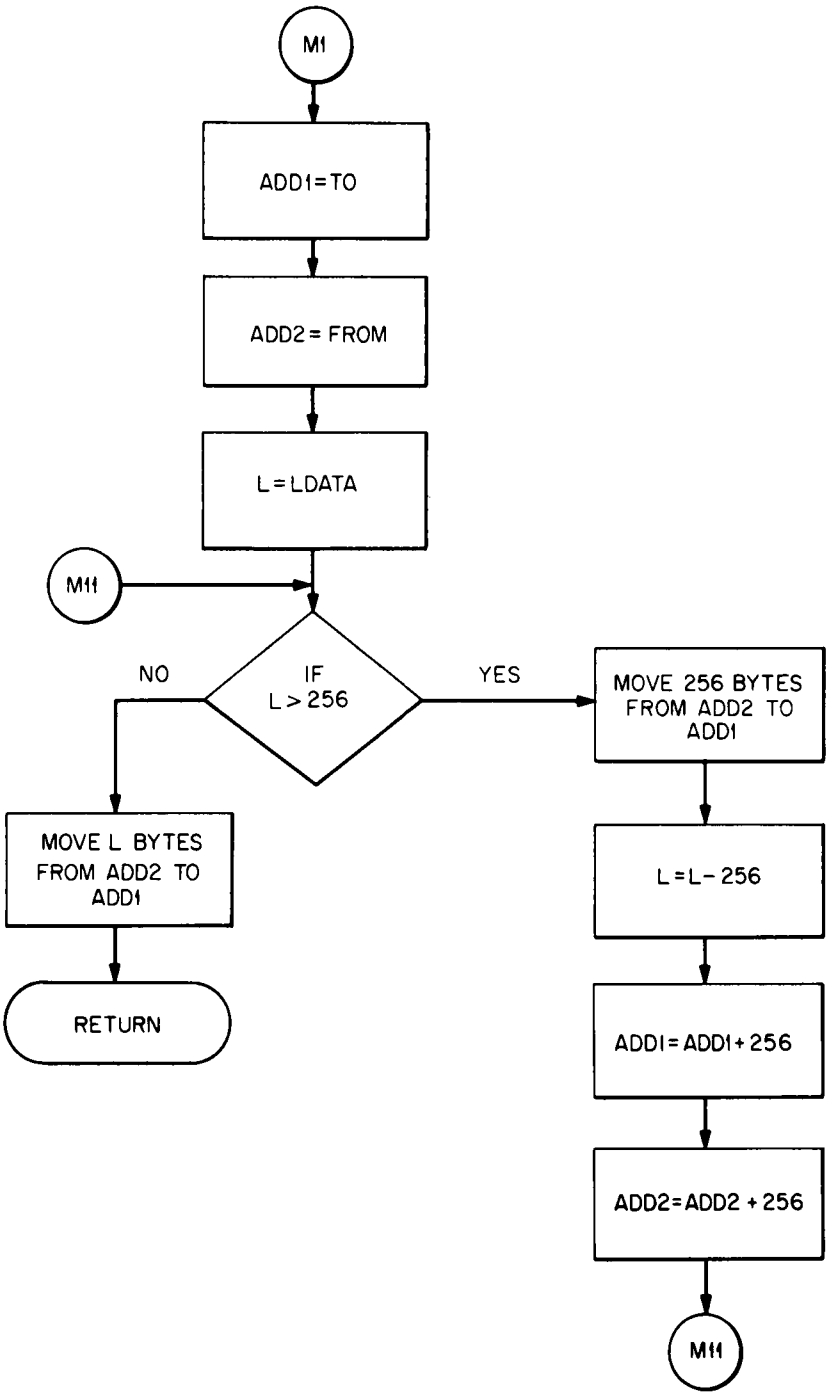


Chart M1: MOVEDATA

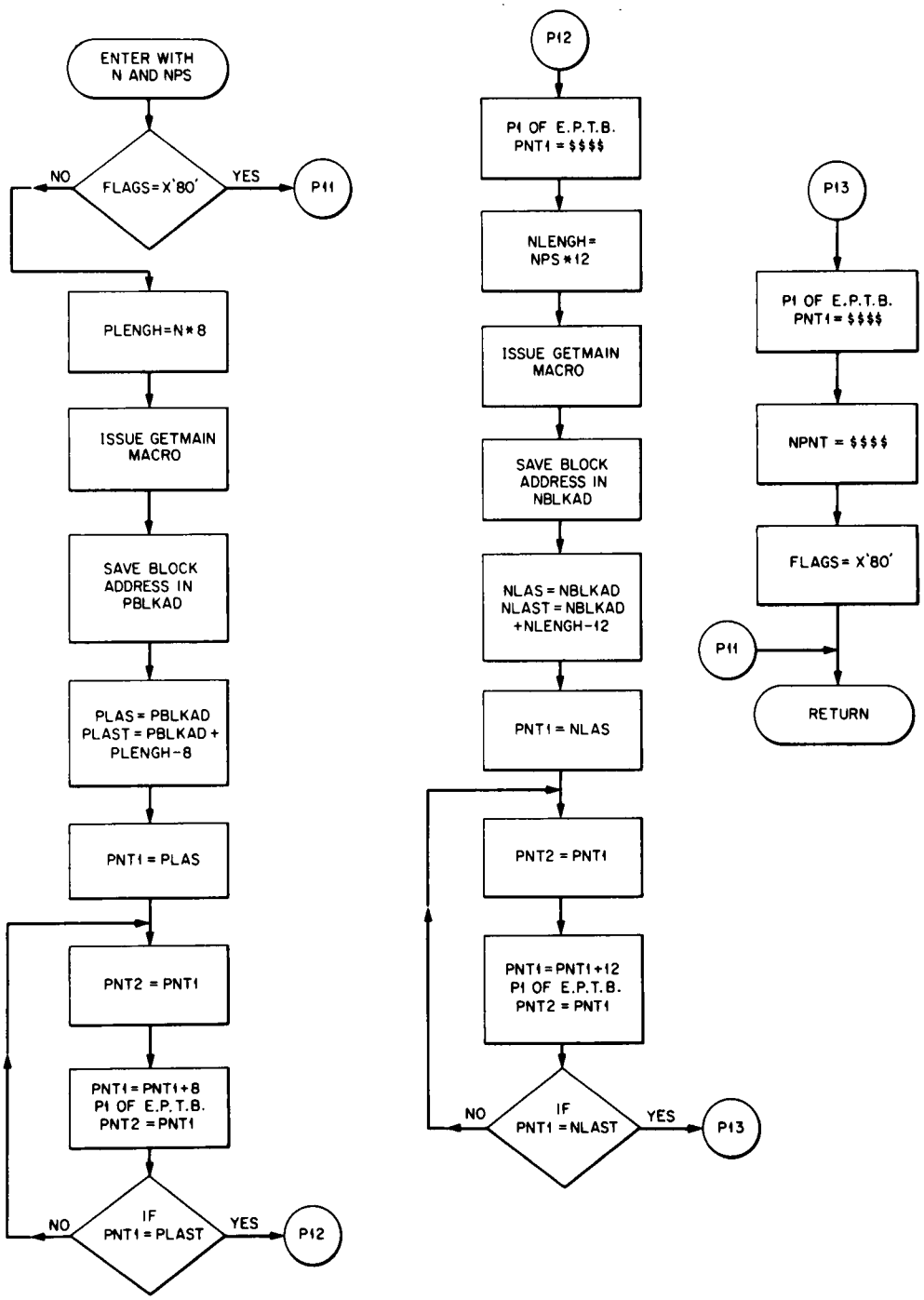


Chart P1: INITPS

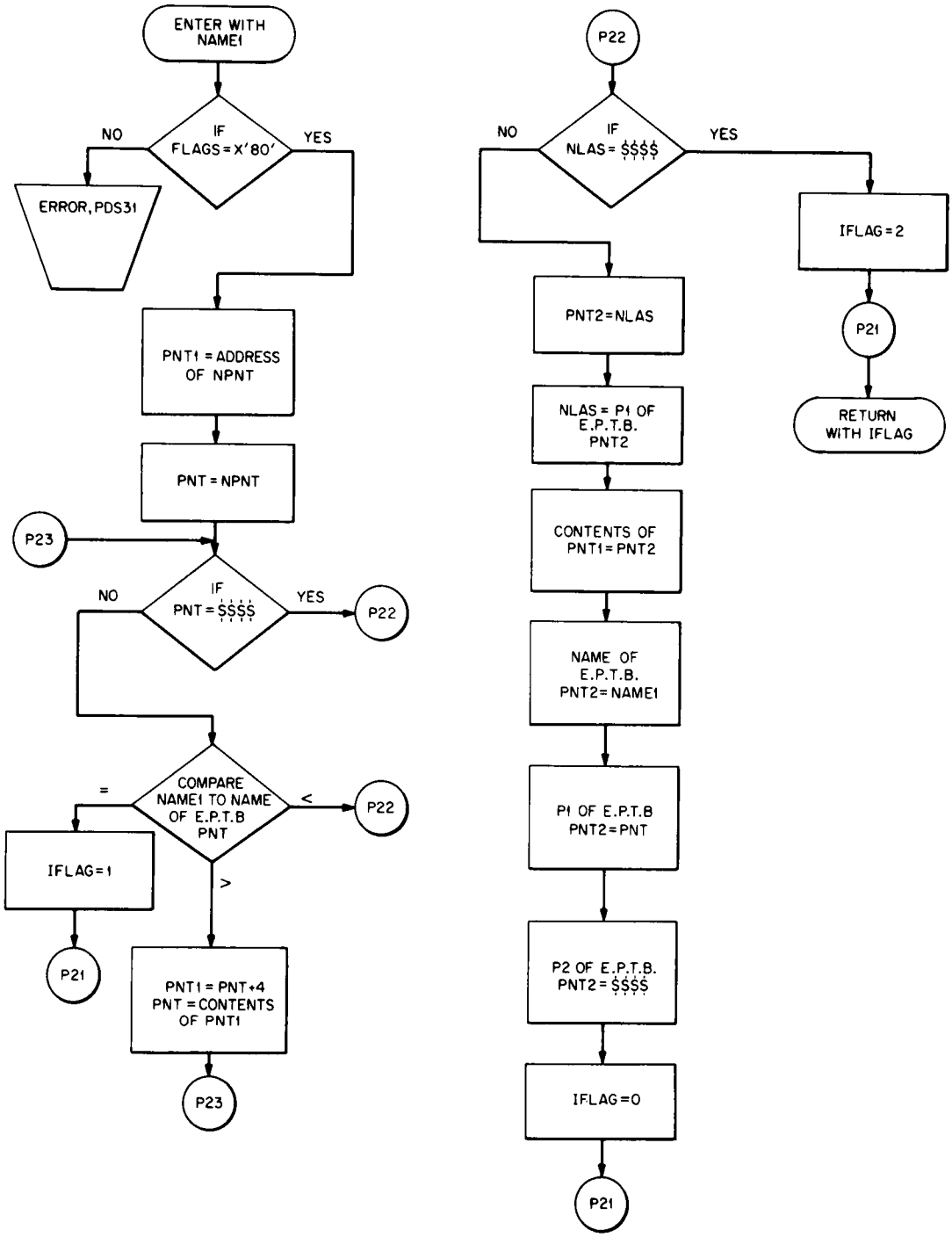


Chart P2: DEFNPS

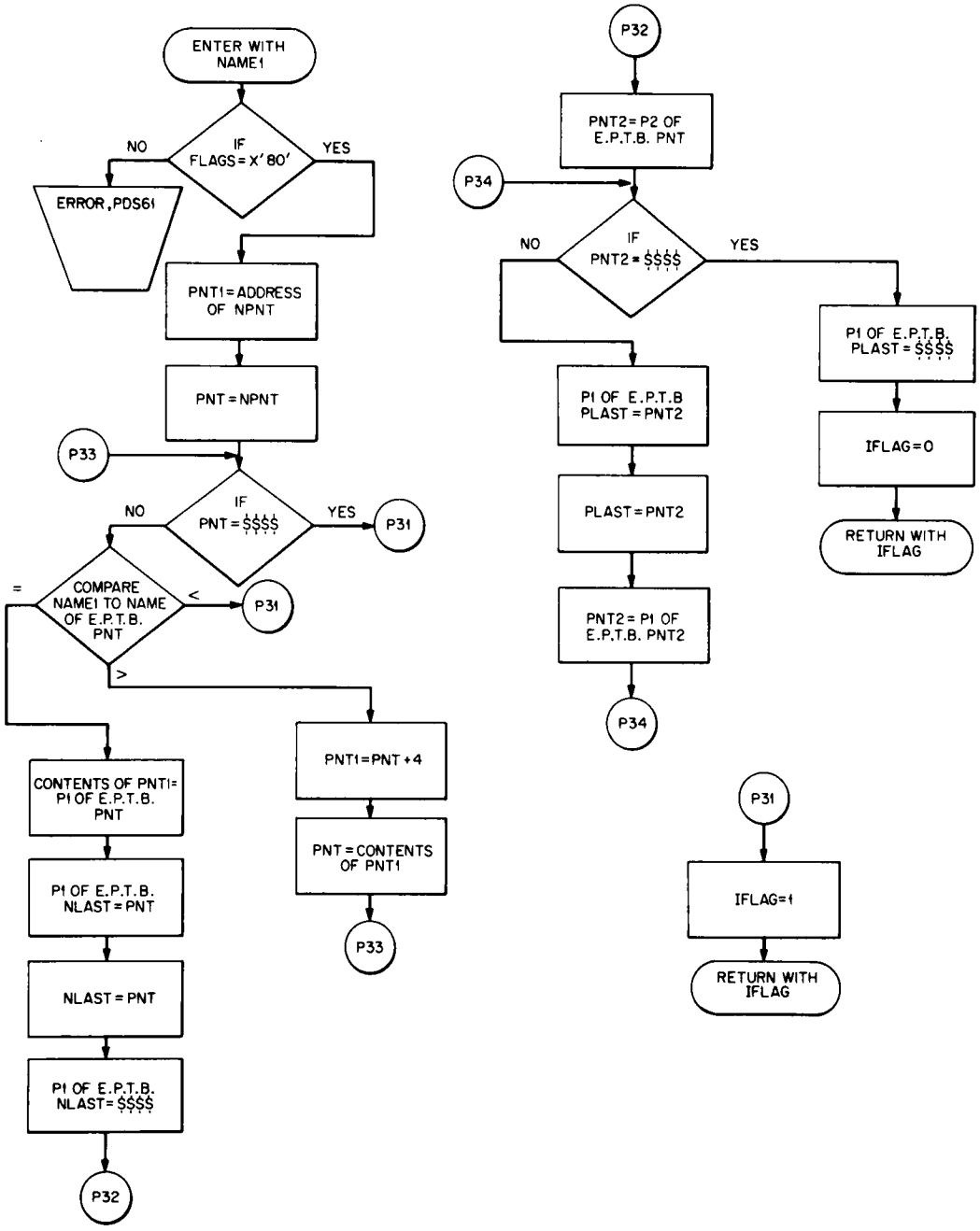


Chart P3: DELTPS

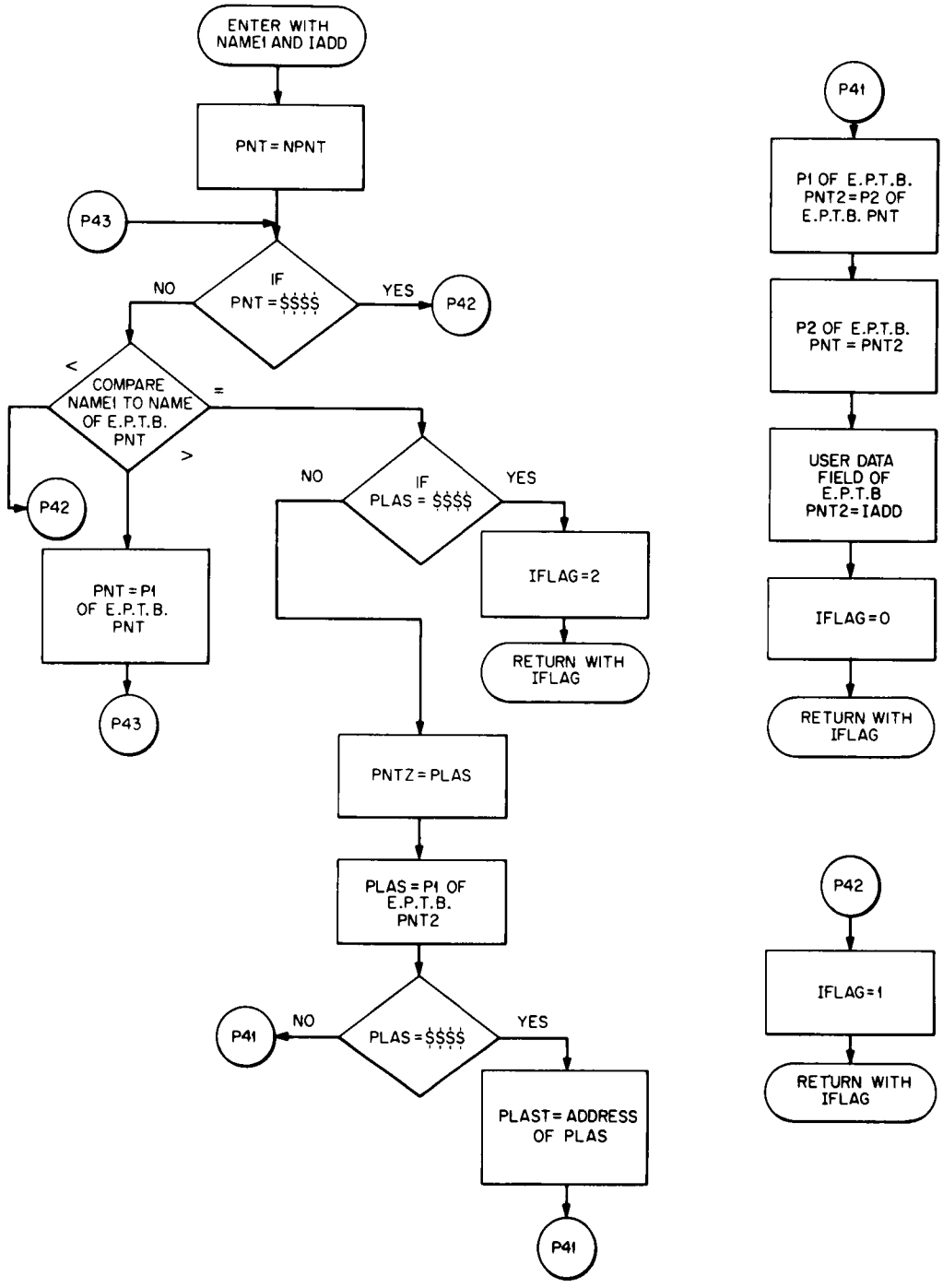


Chart P4: PUSHPS

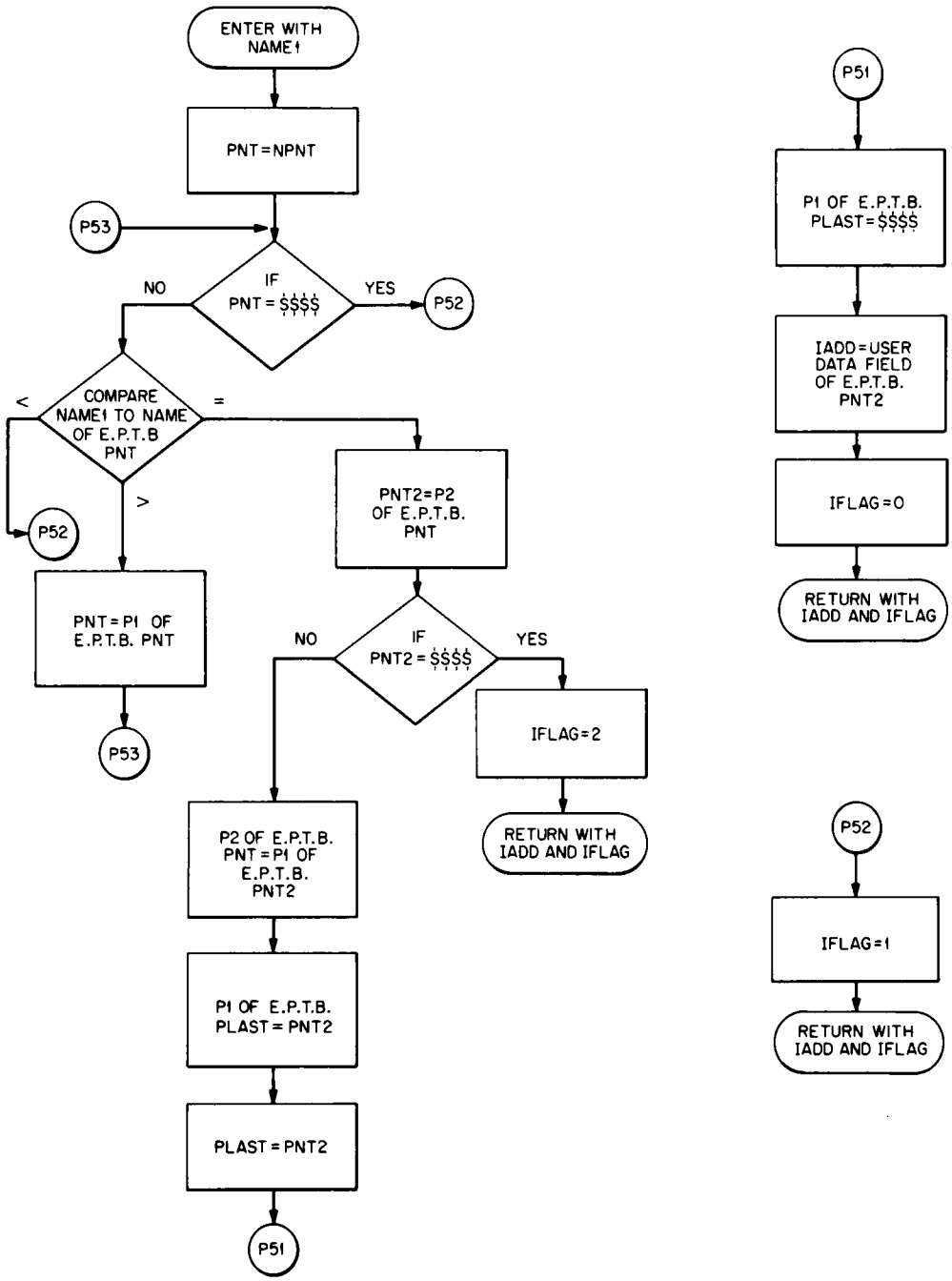


Chart P5: POPPS

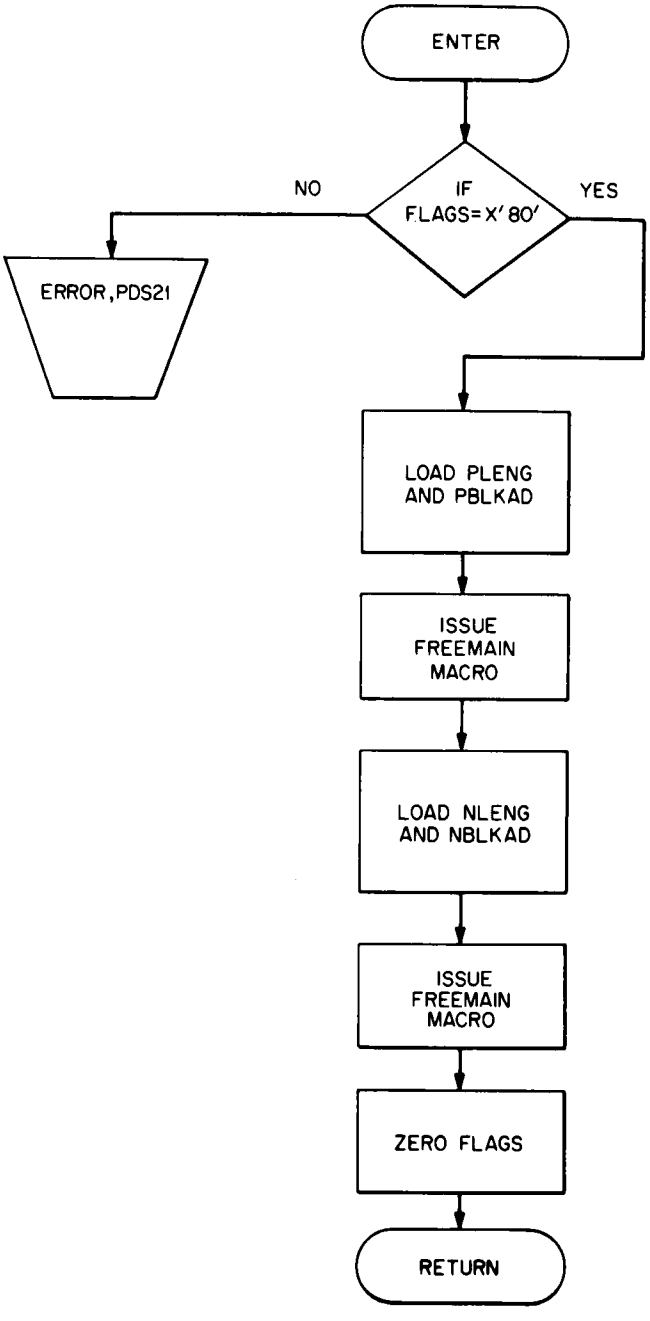


Chart P6: RLSEPS

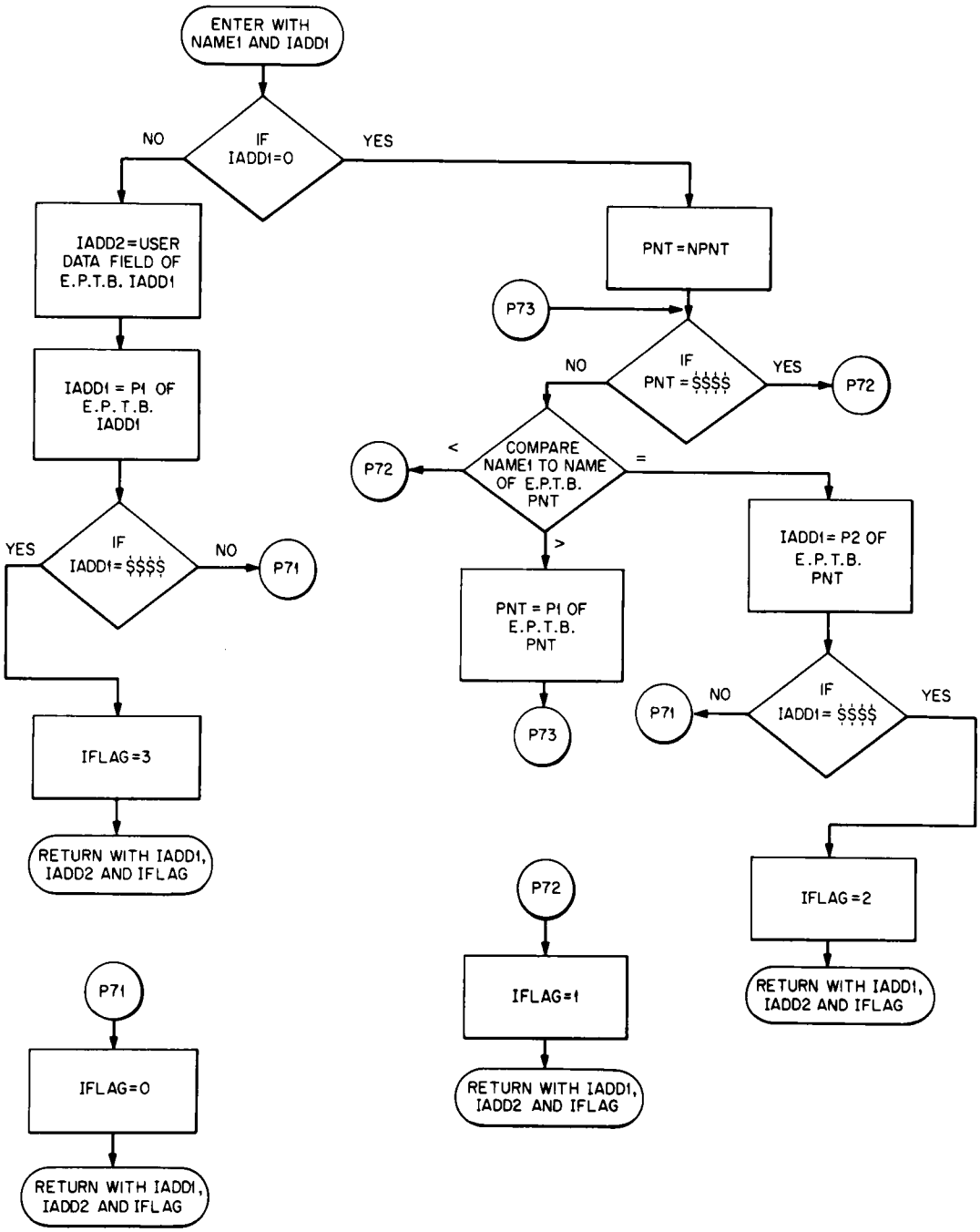


Chart P7: PEEPS

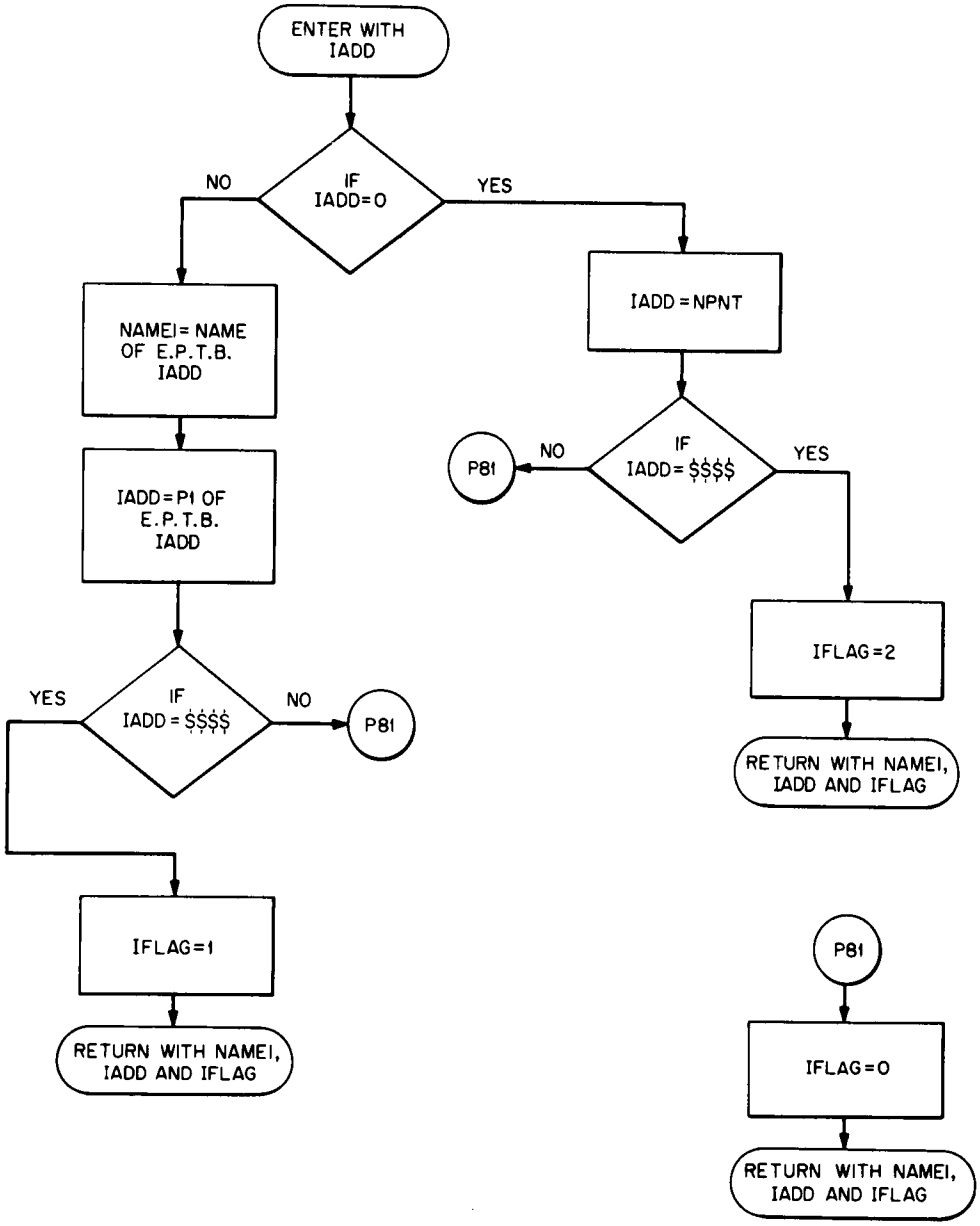


Chart P8: NEEPS

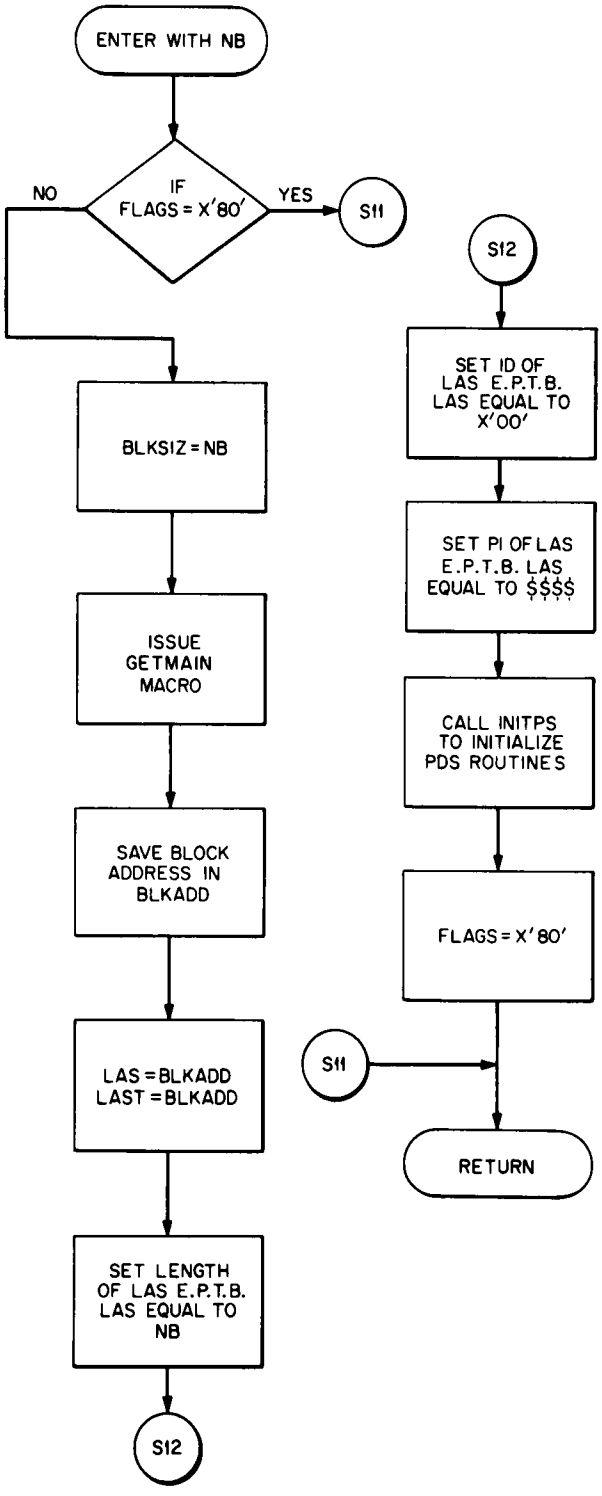


Chart S1: INITLP

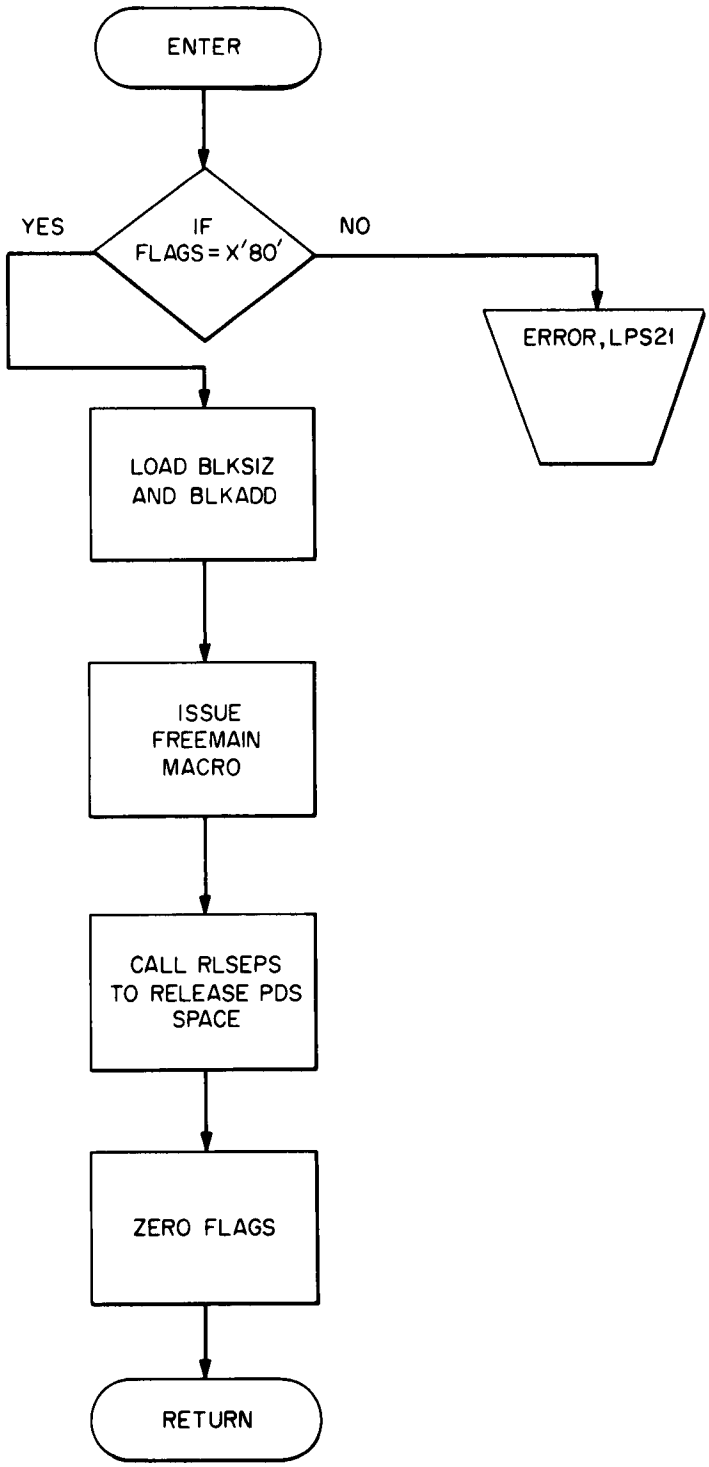


Chart S2: RLSELP

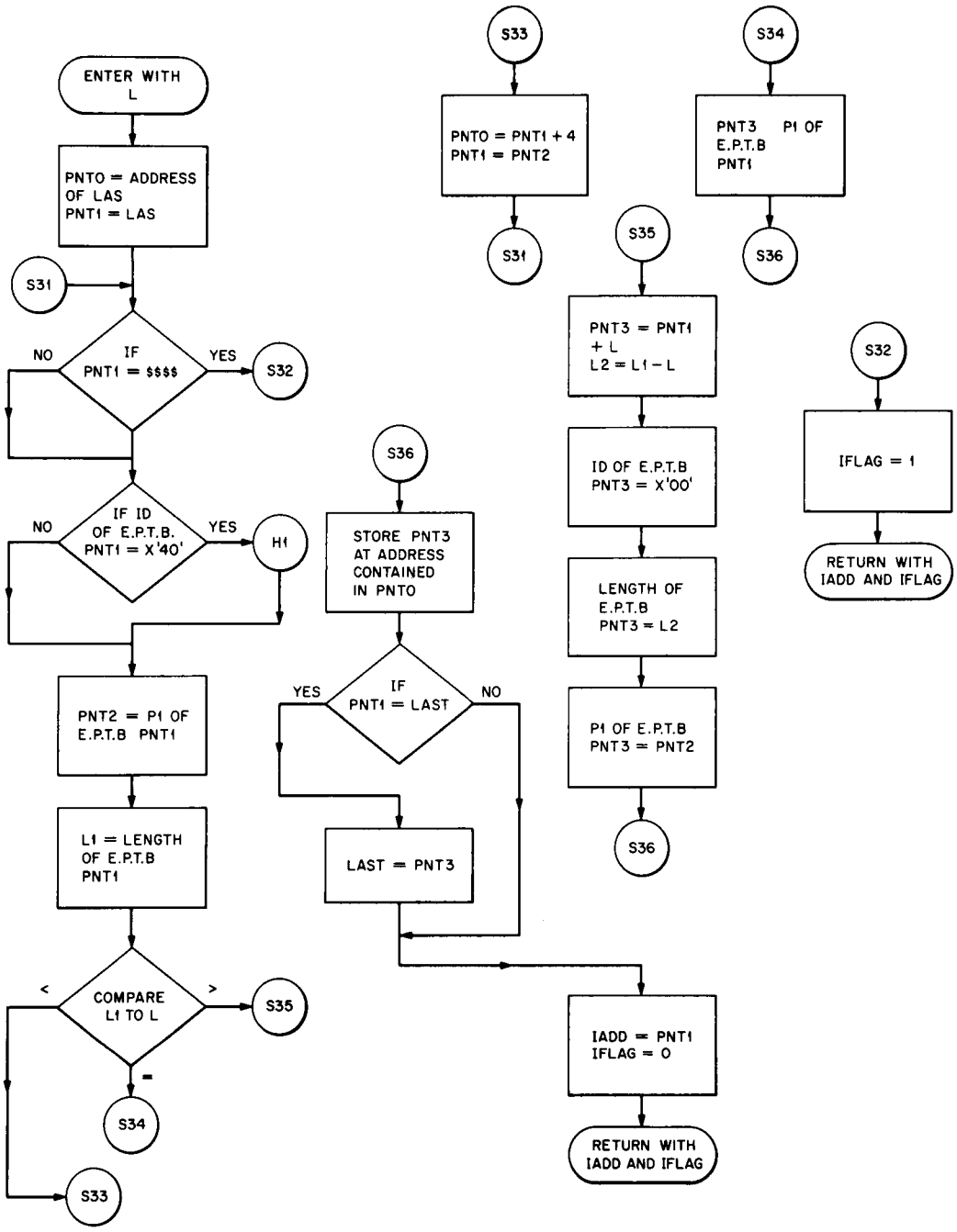


Chart S3: OBTNLP

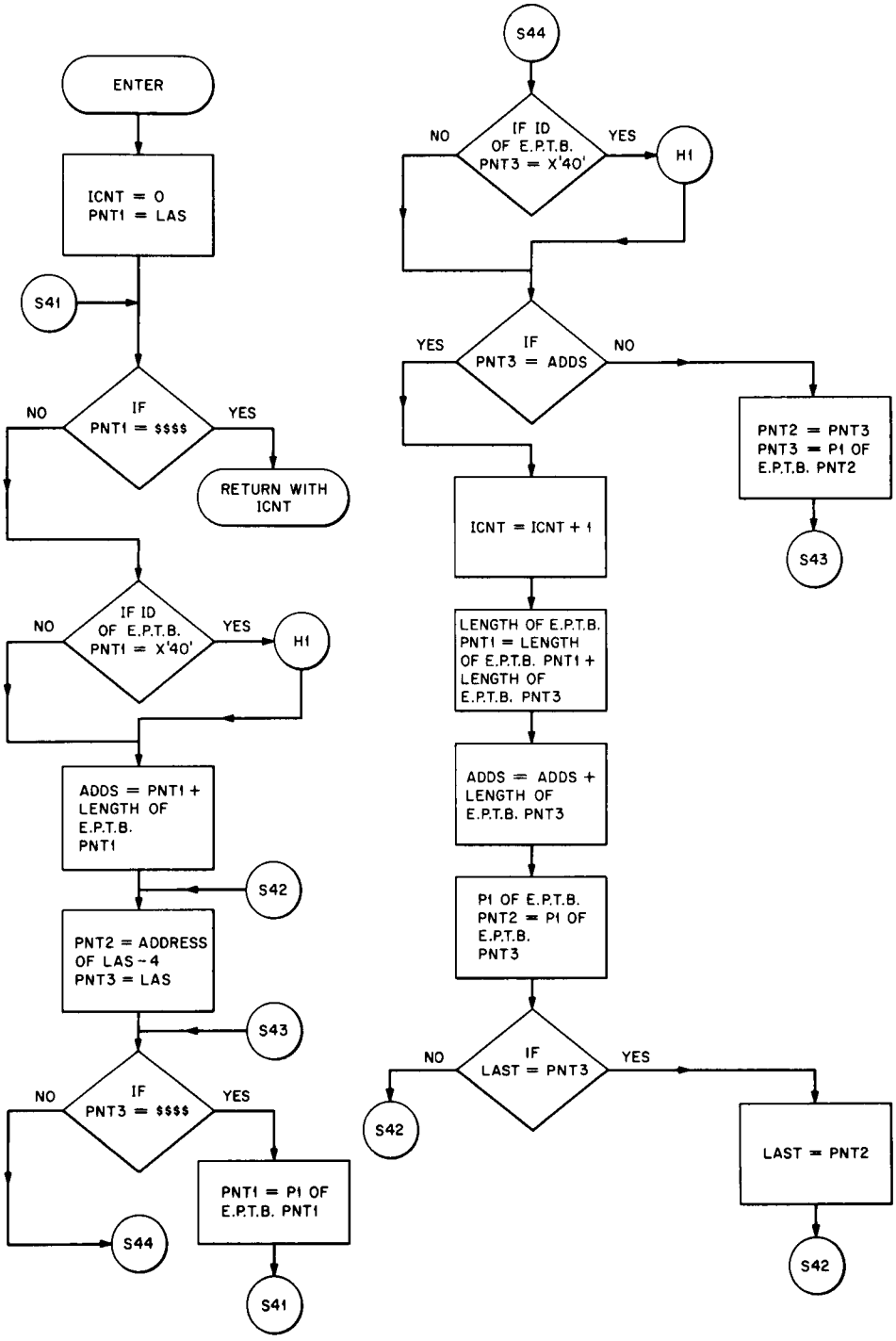


Chart S4: CONFLP

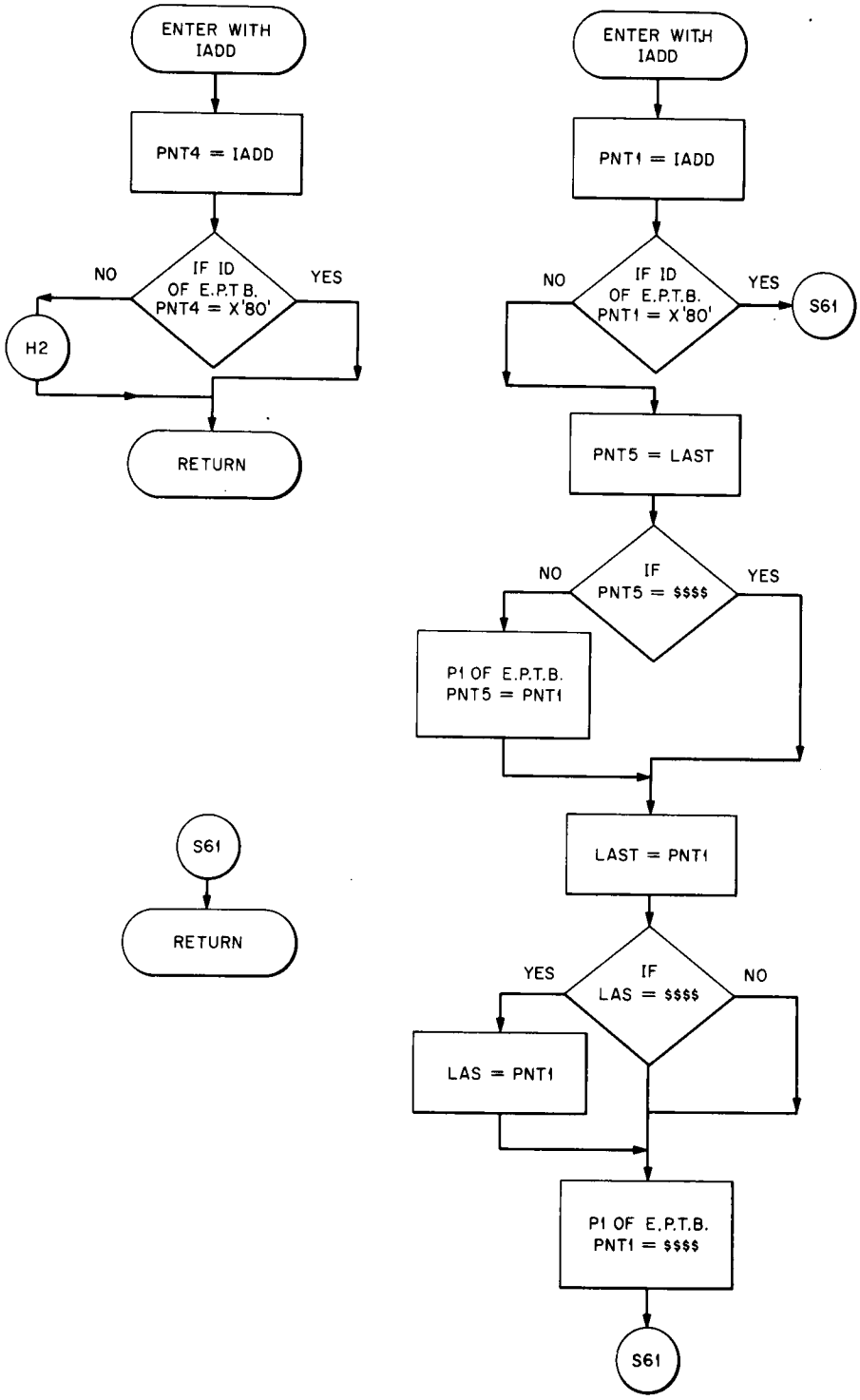


Chart S5: RTNLLP and S6: RTNELP

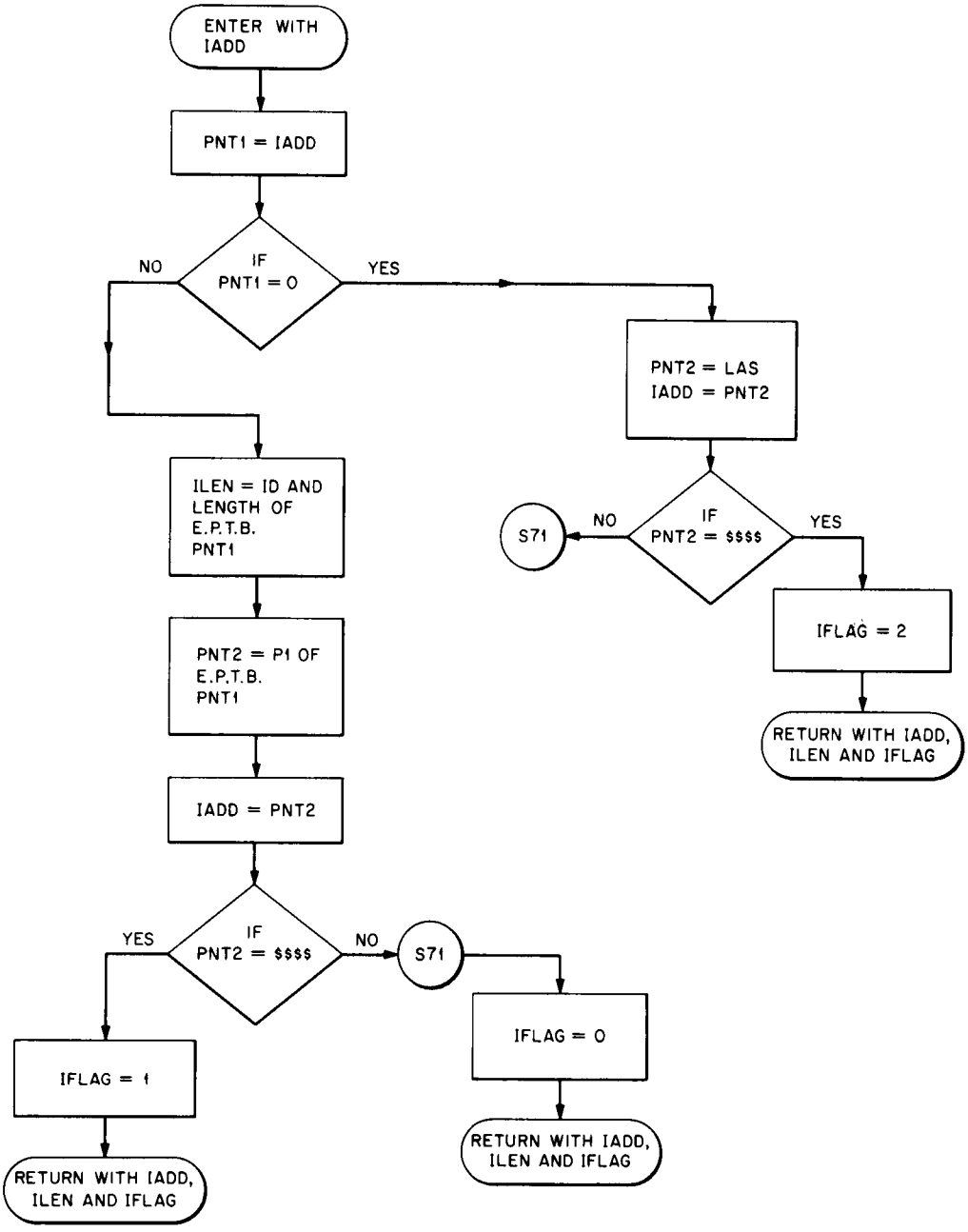


Chart S7: PLASLP

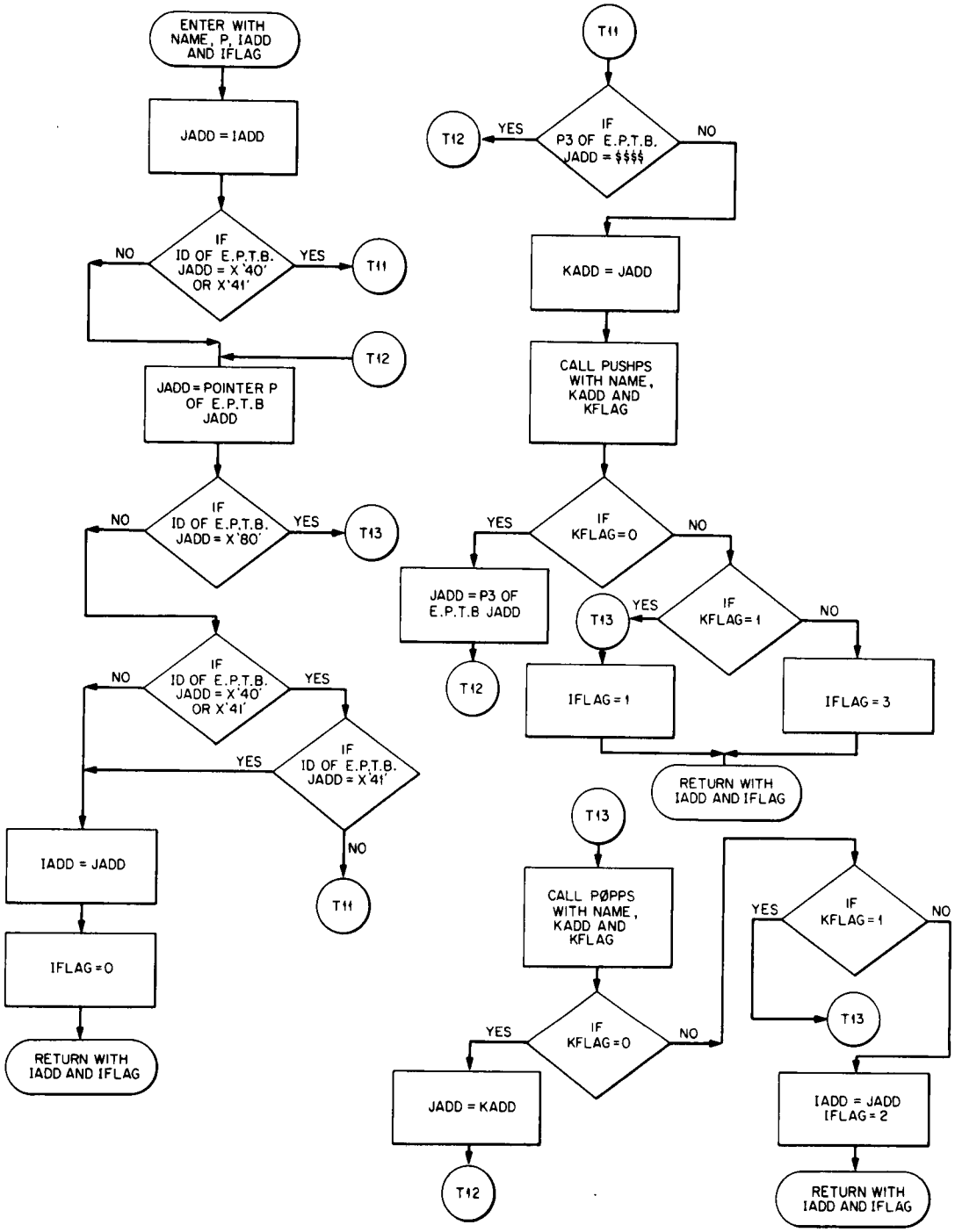


Chart T1: TRVLLP

APPENDIX D

PROGRAM LISTINGS

I. ASSEMBLY LANGUAGE ROUTINES

1) DLINKLP

2) LOPERLP

3) PUSHST

4) SPACELP

5) TRAVSLP

II. FORTRAN ROUTINES

1) BLOCKDATA - COMMON COM$$$

2) DUMPPS

3) MESS$$

4) MESSP$

5) MODCOM

```
DLNK     TITLE 'DLINKLP --- CONTROLS THE BUILDING OF A TWO-WAY LIST'    DLNK0010
DLINKLP  CSECT                                                          DLNK0020
*                                                                       DLNK0030
*        ROUTINE WHICH IS USED FOR DEFINING A LIST CONTROL BLOCK,       DLNK0040
*        INSERTING ELEMENTS INTO A LIST, INSERTING A SUB-LIST INTO A    DLNK0050
*        LIST, DELETING AN ELEMENT FROM A LIST, AND DELETING A LIST.    DLNK0060
*        ENTRY POINTS AND ARGUMENTS ARE ...                             DLNK0070
*                                                                       DLNK0080
*        1.  CALL DEFNLP(NAME,IMARK,IADD,IFLAG)                         DLNK0090
*              DEFINES A LIST CALLED NAME AND CREATES A LCB FOR THE     DLNK0100
*              LIST.  ALSO MAKES CALL TO DEFNPS TO DEFINE A PDS CALLED  DLNK0110
*              NAME.  IMARK = 0 FOR MARK = X'00', 1 FOR MARK = X'80'.   DLNK0120
*              IADD IS RETURNED WITH THE ADDRESS OF THE LCB FOR THE     DLNK0130
*              LIST CALLED NAME.  IFLAG IS SET TO 0 AFTER SUCCESSFUL    DLNK0140
*              OPERATION, SET TO 1 FOR NOT ENOUGH SPACE IN LIST         DLNK0150
*              ELEMENT LAS, SET TO 2 FOR PDS NAME LAS EMPTY.  IF A      DLNK0160
*              LIST CALLED NAME IS DEFINED ON ENTRY THEN IADD IS        DLNK0170
*              RETURNED WITH THE ADDRESS OF THE LCB.                    DLNK0180
*                                                                       DLNK0190
*        2.  CALL INSELP(P,IADD,L1,D,L2,DATA,IFLAG)                     DLNK0200
*              INSERTS AN ELEMENT AFTER (P=1) OR BEFORE (P=2) THE       DLNK0210
*              ELEMENT WHOSE ADDRESS IS IN IADD (ON ENTRY).  IADD IS    DLNK0220
*              RETURNED WITH THE ADDRESS OF THE INSERTED ELEMENT.  L1   DLNK0230
*              IS THE LENGTH OF THE ELEMENT'S USER DATA FIELD.  D IS    DLNK0240
*              THE DISPLACEMENT INTO THE USER DATA FIELD WHERE DATA,    DLNK0250
*              OF LENGTH L2 BYTES, IS TO BE PLACED.  IFLAG IS SET TO    DLNK0260
*              0 AFTER SUCCESSFUL INSERTION, SET TO 1 IF THERE IS NOT   DLNK0270
*              ENOUGH SPACE IN THE LIST ELEMENT LAS FOR THE ELEMENT.    DLNK0280
*                                                                       DLNK0290
*        3.1 CALL INSLLP(P,IADD,NAME,IFLAG)                             DLNK0300
*              INSERTS A LISTR (ELEMENT WHICH POINTS TO A LCB) AFTER    DLNK0310
*              (P=1) OR BEFORE (P=2) THE ELEMENT WHOSE ADDRESS IS IN    DLNK0320
*              IADD ON ENTRY.  IADD RETURNS WITH THE ADDRESS OF THE     DLNK0330
*              INSERTED LISTR.  THE LCB POINTED TO BY THE LISTR IS      DLNK0340
*              THE LCB FOR THE LIST CALLED NAME.  IF NAME IS NULL THEN  DLNK0350
*              P3 OF THE LISTR IS SET EQUAL TO NULL.  IFLAG IS SET TO 0 DLNK0360
*              AFTER SUCCESSFUL INSERTION, SET TO 1 IF A LIST CALLED    DLNK0370
*              NAME IS NOT DEFINED, AND SET TO 2 IF THERE IS NOT        DLNK0380
*              ENOUGH SPACE IN THE LIST ELEMENT LAS FOR THE LISTR.      DLNK0390
*                                                                       DLNK0400
*        3.2 CALL INSLLP(P,IADD,NAME,IFLAG,L1,D,L2,DATA)                DLNK0410
*              PERFORMS THE SAME FUNCTION AS THE CALL TO INSLLP         DLNK0420
*              DESCRIBED ABOVE (3.1) EXCEPT THAT A USER DATA FIELD      DLNK0430
*              IS INCLUDED IN THE LISTR.  THE PARAMETERS L1, D, L2,     DLNK0440
*              AND DATA ARE USED THE SAME AS IN THE CALL TO             DLNK0450
*              INSELP (2.).                                             DLNK0460
*                                                                       DLNK0470
*        4.  CALL DLTELP(IADD)                                          DLNK0480
*              DELETES ELEMENT WHOSE ADDRESS IS IN IADD ON ENTRY.       DLNK0490
*                                                                       DLNK0500
*        5.  CALL DLTLLP(NAME,IFLAG)                                    DLNK0510
*              DELETES THE LIST AND PDS CALLED NAME.  IFLAG IS SET      DLNK0520
*              TO 0 AFTER SUCCESSFUL DELETION, AND 1 IF LIST CALLED     DLNK0530
*              NAME IS NOT DEFINED.                                     DLNK0540
*                                                                       DLNK0550
*        6.  CALL TONDLP                                                DLNK0560
*              SETS TRACE FLAG TO INDICATE THAT CALLS TO ENTRY          DLNK0570
*              POINTS IN DLINKLP ARE RECORDED.                          DLNK0580
*                                                                       DLNK0590
*        7.  CALL TOFDLP                                                DLNK0600
*              SETS TRACE FLAG TO INDICATE THAT CALLS TO ENTRY          DLNK0610
*              POINTS IN DLINKLP ARE NOT RECORDED.                      DLNK0620
*                                                                       DLNK0630
*        A LCB  HAS THE FOLLOWING FORMAT.                               DLNK0640
```

```
*               ID = X'80' (BYTE 1),                                    DLNK0650
*               LCB ELEMENT LENGTH (BYTES 2-4),                         DLNK0660
*               P1 - POINTS TO THE FIRST LIST ELEMENT (BYTES 5-8),      DLNK0670
*               P2 - POINTS TO THE LAST LIST ELEMENT (BYTES 9-12),      DLNK0680
*               MARK - USED TO INDICATE (MARK = X'80') THAT A LIST      DLNK0690
*                      CAN ONLY BE RETURNED TO THE LAS BY USE OF A      DLNK0700
*                      DELETE COMMAND (BYTE 12).                        DLNK0710
*               COUNT - INDICATES THE NUMBER OF TIMES THE LIST IS USED  DLNK0720
*                       AS A SUBLIST (BYTES 13-16),                     DLNK0730
*               NAME - LIST NAME (BYTES 17-20),                         DLNK0740
*               P4 - POINTS TO THE PREVIOUS LCB, P4 IS NULL IF LCB IS   DLNK0750
*                    THE FIRST LCB (BYTES 21-24),                       DLNK0760
*               P5 - POINTS TO THE NEXT LCB, P5 IS NULL IF LCB IS       DLNK0770
*                    THE LAST LCB (BYTES 25-28).                        DLNK0780
*          A LIST ELEMENT HAS THE FOLLOWING FORMAT.                     DLNK0790
*               ID = X'00' (BYTE 1),                                    DLNK0800
*               LIST ELEMENT LENGTH (BYTES 2-4),                        DLNK0810
*               P1 - POINTS TO THE NEXT LIST ELEMENT, P1 POINTS TO THE  DLNK0820
*                    LCB IF IT IS THE LAST ELEMENT IN THE LIST          DLNK0830
*                    (BYTES 5-8),                                       DLNK0840
*               P2 - POINTS TO THE PREVIOUS LIST ELEMENT, P2 POINTS TO  DLNK0850
*                    THE LCB IF IT IS THE FIRST ELEMENT IN THE LIST     DLNK0860
*                    (BYTES 9-12),                                      DLNK0870
*               USER DATA - VARIABLE LENGTH DATA FIELD (BYTES 13-N,     DLNK0880
*                           WHERE N = LIST ELEMENT LENGTH).             DLNK0890
*          A LISTR HAS THE FOLLOWING FORMAT.                            DLNK0900
*               ID = X'40' NO USER DATA, X'41' USER DATA (BYTE 1),      DLNK0910
*               LISTR ELEMENT LENGTH (BYTES 2-4),                       DLNK0920
*               P1 - SAME AS FOR LIST ELEMENT (BYTES 5-8),              DLNK0930
*               P2 - SAME AS FOR LIST ELEMENT (BYTES 9-12),             DLNK0940
*               P3 - POINTS TO A LCB (BYTES 13-16), CAN BE NULL,        DLNK0950
*               USER DATA - OPTIONAL (BYTES 17-N, WHERE N = LIST        DLNK0960
*                           ELEMENT LENGTH), IF INCLUDED.               DLNK0970
*                                                                       DLNK0980
          ENTRY DEFNLP                                                  DLNK0990
          ENTRY INSELP                                                  DLNK1000
          ENTRY INSLLP                                                  DLNK1010
          ENTRY DLTELP                                                  DLNK1020
          ENTRY DLTLLP                                                  DLNK1030
          ENTRY TONDLP                                                  DLNK1040
          ENTRY TOFDLP                                                  DLNK1050
*                                                                       DLNK1060
GRL       EQU   15                  LINKING REGISTER.                   DLNK1070
GRP       EQU   14                  RETURN REGISTER.                    DLNK1080
GRB       EQU   2                   BASE REGISTER.                      DLNK1090
GRS       EQU   13                  SAVE REGISTER.                      DLNK1100
GR1       EQU   1                   ARGUMENT POINTER REGISTER.          DLNK1110
GR0       EQU   0                                                       DLNK1120
GR3       EQU   3                                                       DLNK1130
GR4       EQU   4                   WORK REGISTERS.                     DLNK1140
GR5       EQU   5                                                       DLNK1150
GR6       EQU   6                                                       DLNK1160
GR7       EQU   7                                                       DLNK1170
GR8       EQU   8                                                       DLNK1180
GR9       EQU   9                                                       DLNK1190
GRD       EQU   10                                                      DLNK1200
*                                                                       DLNK1210
          USING *,GRL                                                   DLNK1220
DEFNLP    B     CS11                                                    DLNK1230
          DC    XL1'07'                                                 DLNK1240
          DC    CL7'DEFNLP '                                            DLNK1250
CS11      STM   GRR,GRD,12(GRS)     SAVE CONTENTS OF GENERAL REGISTERS. DLNK1260
          L     GRB,BASE                                                DLNK1270
          USING DLINKLP,GRB                                             DLNK1280
          DROP  GRL                                                     DLNK1290
```

```
         LR     GR3,GRS           SWITCH SAVE AREA POINTERS.          DLNK1300
         LA     GRS,SAREA                                             DLNK1310
         ST     GR3,4(0,GRS)      SAVE PREVIOUS SAVE AREA POINTER.    DLNK1320
         ST     GRS,8(0,GR3)      SAVE PRESENT SAVE AREA POINTER.     DLNK1330
         ST     GR1,SAVE1         SAVE ADDRESS OF ARG. LIST.          DLNK1340
*                                 TRACE AT END SO ADD. OF LCB KNOWN.  DLNK1350
         L      GR3,0(0,GR1)                                          DLNK1360
         L      GR0,0(0,GR3)      LOAD NAME.                          DLNK1370
         ST     GR0,NAME          STORE NAME.                         DLNK1380
         L      GR3,NULL          LOAD NULL POINTER, PNT2.            DLNK1390
         L      GR9,12(0,GR1)     LOAD ADDRESS OF IFLAG.              DLNK1400
         L      GR4,8(0,GR1)                                          DLNK1410
         ST     GR4,AIADD         SAVE ADD. OF IADD.                  DLNK1420
         L      GR4,4(0,GR1)                                          DLNK1430
         ST     GR4,AIMARK        SAVE ADD. OF IMARK.                 DLNK1440
         L      GRD,ALP$$                                             DLNK1450
         USING  DALP$$,GRD                                            DLNK1460
         L      GR4,NPNT          INITIALIZE PNT1.                    DLNK1470
CS12     CL     GR4,NULL          SEE IF PNT1 IS NULL.                DLNK1480
         BE     CS16              YES, BRANCH.                        DLNK1490
         L      GR5,16(0,GR4)     LOAD NM1.                           DLNK1500
         CLR    GR0,GR5           COMPARE NAME TO NM1.                DLNK1510
         BL     CS16              NAME .LT. NM1, BRANCH.              DLNK1520
         BE     CS13              NAME = NM1, BRANCH.                 DLNK1530
         LR     GR3,GR4           PNT2 = PNT1.                        DLNK1540
         L      GR4,24(0,GR4)     UPDATE PNT1.                        DLNK1550
         B      CS12              CONTINUE.                           DLNK1560
CS13     LR     GR7,GR4           PNT3 = PNT1.                        DLNK1570
CS14     L      GR8,AIADD                                             DLNK1580
         ST     GR7,0(0,GR8)      IADD = PNT3.                        DLNK1590
         TM     TRACE,X'80'       TEST FOR TRACE.                     DLNK1600
         BZ     TR1               BRANCH IF NO TRACE.                 DLNK1610
         L      GR1,SAVE1         LOAD ADD. OF ARGUMENT LIST.         DLNK1620
         L      GRL,=V(DEFNL$)                                        DLNK1630
         BALR   GRR,GRL           TRACE.                              DLNK1640
TR1      SR     GR8,GR8           SET IFLAG TO INDICATE SUCCESS.      DLNK1650
         ST     GR8,0(0,GR9)                                          DLNK1660
RETRN1   L      GRS,SAREA+4                                           DLNK1670
         LM     GRR,GRD,12(GRS)   RESTORE GENERAL REGISTERS.          DLNK1680
         MVI    12(GRS),X'FF'                                         DLNK1690
         BR     GRR               RETURN TO CALLING ROUTINE.          DLNK1700
*                                                                     DLNK1710
CS16     LA     GR6,28(0,0)       LOAD LCB LENGTH.                    DLNK1720
         BAL    GR8,GETSPACE      GET LCB SPACE.  ADD. OF LCB IN GR7. DLNK1730
         TM     JFLAG,X'80'       SEE IF NEEDED SPACE OBTAINED.       DLNK1740
         BZ     CS17              YES, CONTINUE.                      DLNK1750
         LA     GR8,1(0,0)        SET IFLAG TO INDICATE THAT THERE IS DLNK1760
         ST     GR8,0(0,GR9)      NOT ENOUGH SPACE IN THE ELEMENT LAS DLNK1770
         B      RETRN1            FOR A LCB.                          DLNK1780
CS17     ST     GR3,20(0,GR7)     P4 OF LCB = PNT2.                   DLNK1790
         ST     GR4,24(0,GR7)     P5 OF LCB = PNT1.                   DLNK1800
         CL     GR3,NULL          SEE IF PNT2 IS NULL.                DLNK1810
         BE     CS18              YES, BRANCH.                        DLNK1820
         ST     GR7,24(0,GR3)     P5 OF PREVIOUS LCB = ADD. OF LCB.   DLNK1830
         B      CS185                                                 DLNK1840
CS18     ST     GR7,NPNT          NPNT = ADD. OF LCB.                 DLNK1850
CS185    CL     GR4,NULL          SEE IF PNT1 IS NULL.                DLNK1860
         BE     CS19              YES, BRANCH.                        DLNK1870
         ST     GR7,20(0,GR4)     P4 OF NEXT LCB = ADD. OF LCB.       DLNK1880
CS19     ST     GR6,0(0,GR7)      SET LCB LENGTH.                     DLNK1890
         MVI    0(GR7),X'80'      SET LCB ID.                         DLNK1900
         ST     GR7,4(0,GR7)      P1 OF LCB = ADD. OF LCB.            DLNK1910
         ST     GR7,8(0,GR7)      P2 OF LCB = ADD. OF LCB.            DLNK1920
         SR     GR8,GR8                                               DLNK1930
         ST     GR8,12(0,GR7)     ZERO LCB COUNT.                     DLNK1940
```

```
         ST    GR0,16(0,GR7)       SET LCB NAME.                          DLNK1950
         L     GR8,AIMARK                                                 DLNK1960
         L     GR8,0(0,GR8)        LOAD IMARK.                            DLNK1970
         LTR   GR8,GR8             TEST FOR 0.                            DLNK1980
         BZ    CS192               YES, BRANCH.                           DLNK1990
         MVI   12(GR7),X'80'       MARK = X'80'.                          DLNK2000
         B     CS195                                                      DLNK2010
CS192    MVI   12(GR7),X'00'       MARK = X'00'.                          DLNK2020
CS195    LA    GR1,LDEFNPS                                                DLNK2030
         L     GRL,ADEFNPS                                                DLNK2040
         BALR  GRR,GRL             DEFINE PDS CALLED NAME.                DLNK2050
         CLI   KFLAG+3,X'02'       SEE IF KFLAG = 2.                      DLNK2060
         BNE   CS14                NO, BRANCH.                            DLNK2070
         LA    GR8,2(0,0)          SET IFLAG TO INDICATE THAT PDS NAME    DLNK2080
         ST    GR8,0(0,GR9)        LAS IS EMPTY.                          DLNK2090
         B     RETRN1                                                     DLNK2100
*                                                                         DLNK2110
*        THIS IS THE SECTION OF CODING WHICH OBTAINS SPACE FOR A LIST     DLNK2120
*        ELEMENT OR A LCB.  IF NO AVAILABLE SPACE IS FOUND ON THE         DLNK2130
*        FIRST CALL TO OBTNLP THEN CONFLP IS CALLED TO CONNECT THE        DLNK2140
*        FRAGMENTS.  AFTER THIS (IF THERE WERE CONNECTIONS) OBTNLP        DLNK2150
*        IS CALLED AGAIN.  ON ENTRY TO THIS SECTION GR6 CONTAINS THE      DLNK2160
*        REQUIRED ELEMENT OR LCB LENGTH AND GR8 CONTAINS THE RETURN       DLNK2170
*        ADDRESS.  ON EXIT GR7 CONTAINS THE ADDRESS OF THE ELEMENT OR     DLNK2180
*        LCB.                                                             DLNK2190
*                                                                         DLNK2200
GETSPACE STM   GR1,GR3,SAVE1TO3                                          DLNK2210
         ST    GR6,L               STORE REQUIRED LENGTH.                 DLNK2220
         LA    GR1,LOBTNLP                                                DLNK2230
         L     GRL,AOBTNLP                                                DLNK2240
         BALR  GRR,GRL             BRANCH TO OBTNLP.                      DLNK2250
         CLI   KFLAG+3,X'00'       TEST FOR KFLAG = 0.                    DLNK2260
         BE    G2                  YES, BRANCH.                           DLNK2270
         LA    GR1,LCONFLP                                                DLNK2280
         L     GRL,ACONFLP                                                DLNK2290
         BALR  GRR,GRL             BRANCH TO CONFLP.                      DLNK2300
         L     GR3,ICNT            LOAD ICNT.                             DLNK2310
         LTR   GR3,GR3             CHECK FOR ICNT = 0.                    DLNK2320
         BE    G1                  YES, BRANCH.                           DLNK2330
         LA    GR1,LOBTNLP                                                DLNK2340
         L     GRL,AOBTNLP                                                DLNK2350
         BALR  GRR,GRL             BRANCH TO OBTNLP.                      DLNK2360
         CLI   KFLAG+3,X'00'       TEST FOR KFLAG = 0.                    DLNK2370
         BE    G2                                                         DLNK2380
G1       MVI   JFLAG,X'80'         SET JFLAG TO INDICATE THAT SPACE       DLNK2390
         B     G3                  COULD NOT BE OBTAINED.                 DLNK2400
G2       MVI   JFLAG,X'00'         SET JFLAG TO INDICATE THAT SPACE       DLNK2410
         L     GR7,KADD            WAS OBTAINED SUCCESSFULLY.             DLNK2420
G3       LM    GR1,GR3,SAVE1TO3                                           DLNK2430
         BR    GR8                                                        DLNK2440
*                                                                         DLNK2450
         DROP  GRB                                                        DLNK2460
         DROP  GRD                                                        DLNK2470
         USING *,GRL                                                      DLNK2480
INSELP   B     CS21                                                       DLNK2490
         DC    XL1'07'                                                    DLNK2500
         DC    CL7'INSELP '                                               DLNK2510
CS21     STM   GRR,GR8,12(GRS)     SAVE CONTENTS OF GENERAL REGISTERS.    DLNK2520
         L     GRB,BASE                                                   DLNK2530
         USING DLINKLP,GRB                                                DLNK2540
         DROP  GRL                                                        DLNK2550
         LR    GR3,GRS             SWITCH SAVE AREA POINTERS.             DLNK2560
         LA    GRS,SAREA                                                  DLNK2570
         ST    GR3,4(0,GRS)        SAVE PREVIOUS SAVE AREA POINTER.       DLNK2580
         ST    GRS,8(0,GR3)        SAVE PRESENT SAVE AREA POINTER.        DLNK2590
```

```
         TM    TRACE,X'80'          TEST FOR TRACE.                    DLNK2600
         BZ    TR2                  BRANCH IF NO TRACE.                DLNK2610
         L     GRL,=V(INSEL$)                                          DLNK2620
         BALR  GRR,GRL              TRACE.                             DLNK2630
TR2      L     GR3,4(0,GR1)                                            DLNK2640
         L     GR3,0(0,GR3)         PNT1 = IADD.                       DLNK2650
         L     GR6,8(0,GR1)                                            DLNK2660
         L     GR6,0(0,GR6)         LOAD L1.                           DLNK2670
         STC   GR6,LTEST            STORE RIGHT MOST CHARACTER.        DLNK2680
         TM    LTEST,X'03'          SEE IF L1 IS A PRODUCT OF 4.       DLNK2690
         BZ    CS215                YES, BRANCH.                       DLNK2700
         N     GR6,=X'00FFFFFC'     MAKE L1 A PRODUCT OF 4.            DLNK2710
         AL    GR6,=X'00000004'                                        DLNK2720
CS215    LA    GR6,12(0,GR6)        LEN = L1 + 12.                     DLNK2730
         BAL   GR8,GETSPACE         GET ELEMENT SPACE, RETURN ADD. IN  DLNK2740
         TM    JFLAG,X'80'          SEE IF JFLAG IS SET.               DLNK2750
         BO    CS24                 YES, BRANCH.                       DLNK2760
         ST    GR6,0(0,GR7)         GR7, PNT3.  STORE ELEMENT LENGTH,  DLNK2770
         L     GR5,0(0,GR1)         (ID = X'00').                      DLNK2780
         CLI   3(GR5),X'01'         SEE IF P = 1.                      DLNK2790
         BE    CS22                 YES, BRANCH.                       DLNK2800
         L     GR6,8(0,GR3)         LOAD ADDRESS OF PREVIOUS ELEMENT.  DLNK2810
         ST    GR7,8(0,GR3)         UPDATE P2 OF NEXT ELEMENT.         DLNK2820
         ST    GR7,4(0,GR6)         UPDATE P1 OF PREVICUS ELEMENT.     DLNK2830
         ST    GR6,8(0,GR7)         STORE INTO P2 OF INSERTED ELEMENT. DLNK2840
         ST    GR3,4(0,GR7)         STORE INTO P1 OF INSERTED ELEMENT. DLNK2850
         B     CS23                                                    DLNK2860
CS22     L     GR6,4(0,GR3)         LOAD ADDRESS OF NEXT ELEMENT.      DLNK2870
         ST    GR7,4(0,GR3)         UPDATE P1 OF PREVICUS ELEMENT.     DLNK2880
         ST    GR7,8(0,GR6)         UPDATE P2 OF NEXT ELEMENT.         DLNK2890
         ST    GR6,4(0,GR7)         STORE INTO P1 OF INSERTED ELEMENT. DLNK2900
         ST    GR3,8(0,GR7)         STORE INTO P2 OF INSERTED ELEMENT. DLNK2910
CS23     L     GR3,20(0,GR1)        FROM = ADD. OF DATA.               DLNK2920
         L     GR4,12(0,GR1)                                           DLNK2930
         L     GR4,0(0,GR4)         LOAD D.                            DLNK2940
         ALR   GR4,GR7              TO = ADDRESS OF INSERTED ELEMENT + DLNK2950
         LA    GR4,12(0,GR4)        D + 12.                            DLNK2960
         L     GR5,16(0,GR1)                                           DLNK2970
         L     GR5,0(0,GR5)         LDATA = L2.                        DLNK2980
         BAL   GR6,MOVEDATA         BRANCH TO MOVE DATA.               DLNK2990
         L     GR3,4(0,GR1)         LOAD ADDRESS OF IADD.              DLNK3000
         ST    GR7,0(0,GR3)         UPDATE IADD.                       DLNK3010
         SR    GR0,GR0              SET IFLAG TO INDICATE THAT AN      DLNK3020
         L     GR6,24(0,GR1)        ELEMENT HAS BEEN INSERTED          DLNK3030
         ST    GR0,0(0,GR6)         SUCCESSFULLY.                      DLNK3040
RETRN2   L     GR5,SAREA+4                                             DLNK3050
         LM    GRR,GR8,12(GR5)      RESTORE GENERAL REGISTERS.         DLNK3060
         MVI   12(GR5),X'FF'                                           DLNK3070
         BR    GRR                                                     DLNK3080
CS24     LA    GR0,1(0,0)           SET IFLAG TO INDICATE THAT THE LAS DLNK3090
         L     GR6,24(0,GR1)        FOR LIST ELEMENTS DOES NOT CONTAIN DLNK3100
         ST    GR0,0(0,GR6)         ENOUGH SPACE FOR THE ELEMENT.      DLNK3110
         B     RETRN2                                                  DLNK3120
*                                                                      DLNK3130
*        THIS IS THE SECTION OF CODING WHICH MOVES DATA.  ON ENTRY     DLNK3140
*        TO THIS SECTION GR3 CONTAINS FROM (ADDRESS WHERE DATA TO BE   DLNK3150
*        MOVED IS LOCATED), GR4 CONTAINS TO (ADDRESS WHERE DATA IS TO  DLNK3160
*        BE MOVED), GR5 CONTAINS LDATA (NUMBER OF BYTES OF DATA TO BE  DLNK3170
*        MOVED), AND GR6 CONTAINS THE RETURN ADDRESS.                  DLNK3180
*                                                                      DLNK3190
MOVEDATA STM   GR3,GR7,SAVE3TO7                                       DLNK3200
         LA    GR6,255(0,0)                                            DLNK3210
         LA    GR7,1(0,GR6)                                            DLNK3220
MLOOP    CR    GR5,GR7              TEST REMAINING DATA LENGTH.        DLNK3230
         BE    M1                   EQUAL TO 256 BYTES, BRANCH.        DLNK3240
```

```
         BL    M1                  LESS THAN 256 BYTES, BRANCH.        DLNK3250
         EX    GR6,MOVED           MOVE 256 BYTES.                     DLNK3260
         SLR   GR5,GR7             LDATA = LDATA - 256.                DLNK3270
         ALR   GR4,GR7             TO = TO + 256.                      DLNK3280
         ALR   GR3,GR7             FROM = FROM + 256.                  DLNK3290
         B     MLOOP                                                   DLNK3300
MOVED    MVC   0(0,GR4),0(GR3)                                         DLNK3310
M1       BCTR  GR5,0                                                   DLNK3320
         EX    GR5,MOVED           MOVE REMAINING DATA.                DLNK3330
         LM    GR3,GR7,SAVE3TO7                                        DLNK3340
         BR    GR6                 RETURN.                             DLNK3350
*                                                                      DLNK3360
         DROP  GR8                                                     DLNK3370
         USING *,GRL                                                   DLNK3380
INSLLP   B     CS31                                                    DLNK3390
         DC    XL1'07'                                                 DLNK3400
         DC    CL7'INSLLP '                                            DLNK3410
CS31     STM   GRR,GR8,12(GRS)     SAVE CONTENTS OF GENERAL REGISTERS. DLNK3420
         L     GR8,BASE                                                DLNK3430
         USING DLINKLP,GR8                                             DLNK3440
         DROP  GRL                                                     DLNK3450
         LR    GR3,GRS             SWITCH SAVE AREA POINTERS.          DLNK3460
         LA    GRS,SAREA                                               DLNK3470
         ST    GR3,4(0,GRS)        SAVE PREVIOUS SAVE AREA POINTER.    DLNK3480
         ST    GRS,8(0,GR3)        SAVE PRESENT SAVE AREA POINTER.     DLNK3490
         TM    TRACE,X'80'         TEST FOR TRACE.                     DLNK3500
         BZ    TR3                 BRANCH IF NO TRACE.                 DLNK3510
         L     GRL,=V(INSLL$)                                          DLNK3520
         BALR  GRR,GRL             TRACE.                              DLNK3530
TR3      L     GR3,8(0,GR1)                                            DLNK3540
         L     GR3,0(0,GR3)        LOAD NAME.                          DLNK3550
         L     GR8,12(0,GR1)       LOAD ADDRESS OF IFLAG.              DLNK3560
         ST    GR8,AIFLAG          STORE ADDRESS OF IFLAG.             DLNK3570
         CL    GR3,NULL            SEE IF NAME = NULL.                 DLNK3580
         BNE   CS312               NO, BRANCH.                         DLNK3590
         L     GR5,NULL            PNT1 = NULL.                        DLNK3600
         B     CS32                                                    DLNK3610
CS312    L     GR6,ALP$$                                               DLNK3620
         USING DALP$$,GR6                                              DLNK3630
         L     GR5,NPNT            PNT1 = NPNT. (ADD. OF FIRST LCB).   DLNK3640
         DROP  GR6                                                     DLNK3650
CS315    CL    GR5,NULL            SEE IF PNT1 IS NULL.                DLNK3660
         BE    CS35                YES, BRANCH.                        DLNK3670
         L     GR0,16(0,GR5)       NM1 = NAME OF LCB POINTED TO BY PNT1 DLNK3680
         CLR   GR3,GR0             COMPARE NAME TO NM1.                DLNK3690
         BL    CS35                NAME .GT. NM1, BRANCH.              DLNK3700
         BE    CS32                NAME = NM1, BRANCH.                 DLNK3710
         L     GR5,24(0,GR5)       UPDATE PNT1 SO THAT IT POINTS TO    DLNK3720
         B     CS315               THE NEXT LCB.                       DLNK3730
CS32     LA    GR6,16(0,0)         LOAD LISTR LENGTH.                  DLNK3740
         MVI   LFLAG,X'00'         INITIALIZE LFLAG.                   DLNK3750
         TM    AIFLAG,X'80'        SEE IF IFLAG IS THE LAST ARGUMENT.  DLNK3760
         BO    CS325               YES, NO LISTR USER DATA FIELD.      DLNK3770
         MVI   LFLAG,X'80'         INDICATE THAT LISTR HAS USER DATA.  DLNK3780
         L     GR8,16(0,GR1)                                           DLNK3790
         L     GR8,0(0,GR8)        LOAD L1.                            DLNK3800
         STC   GR8,LTEST           STORE RIGHT MOST CHARACTER.         DLNK3810
         TM    LTEST,X'03'         SEE IF L1 IS A PRODUCT OF 4.        DLNK3820
         BZ    CS323               YES, BRANCH.                        DLNK3830
         N     GR8,=X'00FFFFFC'    MAKE L1 A PRODUCT OF 4.             DLNK3840
         AL    GR8,=X'00000004'                                        DLNK3850
CS323    ALR   GR6,GR8             ADD USER DATA FIELD LENGTH TO 16.   DLNK3860
CS325    BAL   GR8,GETSPACE        GET ADD. OF LCB, PNT3.              DLNK3870
         TM    JFLAG,X'80'         SEE IF JFLAG IS SET.                DLNK3880
         BO    CS36                YES, BRANCH.                        DLNK3890
```

```
         L      GR8,12(0,GR5)       LOAD COUNT OF LCB CALLED NAME.      DLNK3900
         LA     GR8,1(0,GR8)        ADD 1 TO COUNT.                     DLNK3910
         ST     GR8,12(0,GR5)       STORE COUNT.                        DLNK3920
         ST     GR6,0(0,GR7)        STORE INSERTED LISTR LENGTH.        DLNK3930
         MVI    0(GR7),X'40'        ID OF LISTR = X'40'.                DLNK3940
         ST     GR5,12(0,GR7)       P3 OF LISTR = PNT1.                 DLNK3950
         L      GR3,4(0,GR1)                                            DLNK3960
         L      GR3,0(0,GR3)        PNT4 = IADD.                        DLNK3970
         L      GR4,0(0,GR1)                                            DLNK3980
         CLI    3(GR4),X'01'        SEE IF P = 1.                       DLNK3990
         BE     CS33                YES, BRANCH.                        DLNK4000
         L      GR5,8(0,GR3)        LOAD ADDRESS OF PREVIOUS ELEMENT.   DLNK4010
         ST     GR7,8(0,GR3)        UPDATE P2 OF NEXT ELEMENT.          DLNK4020
         ST     GR7,4(0,GR5)        UPDATE P1 OF PREVIOUS ELEMENT.      DLNK4030
         ST     GR5,8(0,GR7)        STORE INTO P2 OF INSERTED LISTR.    DLNK4040
         ST     GR3,4(0,GR7)        STORE INTO P1 OF INSERTED LISTR.    DLNK4050
         B      CS34                                                    DLNK4060
CS33     L      GR5,4(0,GR3)        LOAD ADDRESS OF NEXT ELEMENT.       DLNK4070
         ST     GR7,4(0,GR3)        UPDATE P1 OF PREVIOUS ELEMENT.      DLNK4080
         ST     GR7,8(0,GR5)        UPDATE P2 OF NEXT ELEMENT.          DLNK4090
         ST     GR5,4(0,GR7)        STORE INTO P1 OF INSERTED LISTR.    DLNK4100
         ST     GR3,8(0,GR7)        STORE INTO P2 OF INSERTED LISTR.    DLNK4110
CS34     L      GR8,4(0,GR1)                                            DLNK4120
         ST     GR7,0(0,GR8)        IADD = ADD. OF INSERTED LISTR.      DLNK4130
         TM     LFLAG,X'80'         TEST FOR LISTR USER DATA FIELD.     DLNK4140
         BZ     CS345               NONE, BRANCH.                       DLNK4150
         OI     0(GR7),X'01'        INDICATE LISTR HAS USER DATA FIELD. DLNK4160
         L      GR3,28(0,GR1)       FROM = ADDRESS OF DATA.             DLNK4170
         L      GR4,20(0,GR1)                                           DLNK4180
         L      GR4,0(0,GR4)        LOAD D.                             DLNK4190
         ALR    GR4,GR7             TO = ADDRESS OF INSERTED LISTR +    DLNK4200
         LA     GR4,16(0,GR4)       D + 16.                             DLNK4210
         L      GR5,24(0,GR1)                                           DLNK4220
         L      GR5,0(0,GR5)        LDATA = L2.                         DLNK4230
         BAL    GR6,MOVEDATA        BRANCH TO MOVE DATA.                DLNK4240
CS345    SR     GR0,GR0             SET IFLAG TO INDICATE THAT LISTR    DLNK4250
         L      GR8,AIFLAG          HAS BEEN INSERTED SUCCESSFULLY.     DLNK4260
         ST     GR0,0(0,GR8)                                            DLNK4270
         B      RETRN3                                                  DLNK4280
CS35     LA     GR0,1(0,0)          SET IFLAG TO INDICATE THAT A LCB    DLNK4290
         L      GR8,AIFLAG          FOR THE LIST CALLED NAME COULD NOT  DLNK4300
         ST     GR0,0(0,GR8)        BE FOUND.                           DLNK4310
         B      RETRN3                                                  DLNK4320
CS36     LA     GR0,2(0,0)          SET IFLAG TO INDICATE THAT THERE IS DLNK4330
         L      GR8,AIFLAG          NOT ENOUGH SPACE IN THE LIST ELEMENT DLNK4340
         ST     GR0,0(0,GR8)        LAS FOR A LISTR.                    DLNK4350
RETRN3   L      GR5,SAREA+4                                             DLNK4360
         LM     GR8,GR8,12(GR5)     RESTORE GENERAL REGISTERS.          DLNK4370
         MVI    12(GR5),X'FF'                                           DLNK4380
         BR     GR8                                                     DLNK4390
*                                                                       DLNK4400
         DROP   GR8                                                     DLNK4410
         USING  *,GRL                                                   DLNK4420
DLTELP   B      CS41                                                    DLNK4430
         DC     XL1'07'                                                 DLNK4440
         DC     CL7'DLTELP'                                             DLNK4450
CS41     STM    GR8,GR4,12(GR5)     SAVE CONTENTS OF GENERAL REGISTERS. DLNK4460
         L      GR8,BASE                                                DLNK4470
         USING  DLINKLP,GR8                                             DLNK4480
         DROP   GRL                                                     DLNK4490
         LR     GR3,GR5             SWITCH SAVE AREA POINTERS.          DLNK4500
         LA     GR5,SAREA                                               DLNK4510
         ST     GR3,4(0,GR5)        SAVE PREVIOUS SAVE AREA POINTER.    DLNK4520
         ST     GR5,8(0,GR3)        SAVE PRESENT SAVE AREA POINTER.     DLNK4530
         TM     TRACE,X'80'         TEST FOR TRACE                      DLNK4540
```

```
         BZ    TR4                BRANCH IF NO TRACE.                  DLNK4550
         L     GRL,=V(DLTEL$)                                          DLNK4560
         BALR  GRR,GRL            TRACE.                               DLNK4570
TR4      L     GR1,0(0,GR1)       LOAD THE ADDRESS OF THE ELEMENT      DLNK4580
         L     GR1,0(0,GR1)       WHICH IS TO BE DELETED.              DLNK4590
         L     GR3,8(0,GR1)       LOAD ADD. OF PREVIOUS ELEMENT.       DLNK4600
         L     GR4,4(0,GR1)       LOAD ADD. OF NEXT ELEMENT.           DLNK4610
         ST    GR3,8(0,GR4)       UPDATE P2 OF NEXT ELEMENT.           DLNK4620
         ST    GR4,4(0,GR3)       UPDATE P1 OF PREVIOUS ELEMENT.       DLNK4630
         ST    GR1,KADD                                                DLNK4640
         LA    GR1,LRTNELP        RETURN THE ELEMENT CT THE LAS FOR    DLNK4650
         L     GRL,ARTNELP        THE LIST ELEMENTS AND LCBS.          DLNK4660
         BALR  GRR,GRL                                                 DLNK4670
         L     GRS,SAREA+4                                             DLNK4680
         LM    GRR,GR4,12(GRS)    RESTORE GENERAL REGISTERS.           DLNK4690
         MVI   12(GRS),X'FF'                                           DLNK4700
         BR    GRR                RETURN TO CALLING RCUTINE.           DLNK4710
*                                                                      DLNK4720
         DROP  GRB                                                     DLNK4730
         USING *,GRL                                                   DLNK4740
DLTLLP   B     CS51                                                    DLNK4750
         DC    XL1'07'                                                 DLNK4760
         DC    CL7'DLTLLP '                                            DLNK4770
CS51     STM   GRR,GR5,12(GRS)    SAVE CONTENTS OF GENERAL REGISTERS.  DLNK4780
         L     GRB,BASE                                                DLNK4790
         USING DLINKLP,GRB                                             DLNK4800
         DROP  GRL                                                     DLNK4810
         LR    GR3,GRS            SWITCH SAVE AREA POINTERS.           DLNK4820
         LA    GRS,SAREA                                               DLNK4830
         ST    GR3,4(0,GRS)       SAVE PREVIOUS SAVE AREA POINTER.     DLNK4840
         ST    GR5,8(0,GR3)       SAVE PRESENT SAVE AREA POINTER.      DLNK4850
         TM    TRACE,X'80'        TEST FOR TRACE.                      DLNK4860
         BZ    TR5                BRANCH IF NO TRACE.                  DLNK4870
         L     GRL,=V(DLTLL$)                                          DLNK4880
         BALR  GRR,GRL            TRACE.                               DLNK4890
TR5      L     GR3,0(0,GR1)                                            DLNK4900
         L     GR3,0(0,GR3)       LOAD NAME.                           DLNK4910
         L     GR4,ALP$$                                               DLNK4920
         USING DALP$$,GR4                                              DLNK4930
         L     GR5,NPNT           PNT4 = NPNT (ADD. OF FIRST LCB).     DLNK4940
         DROP  GR4                                                     DLNK4950
         L     GR4,4(0,GR1)       LOAD ADDRESS OF IFLAG.               DLNK4960
SNLOOP5  CL    GR5,NULL           SEE IF PNT4 IS NULL.                 DLNK4970
         BE    CS53               YES, BRANCH.                         DLNK4980
         L     GR0,16(0,GR5)      NM1 = NAME OF LCB POINTED TO BY PNT4 DLNK4990
         CLR   GR3,GR0            COMPARE NAME TO NM1.                 DLNK5000
         BE    CS52               NAME = NM1, BRANCH.                  DLNK5010
         BL    CS53               NAME .LT. NM1, BRANCH.               DLNK5020
         L     GR5,24(0,GR5)      UPDATE PNT4 SO THAT IT POINTS TO     DLNK5030
         B     SNLOOP5            THE NEXT LCB.                        DLNK5040
CS52     ST    GR5,KADD                                                DLNK5050
         LA    GR1,LRTNLLP        RETURN THE LIST CALLED NAME TO THE   DLNK5060
         L     GRL,ARTNLLP        LAS FOR THE LIST ELEMENTS AND LCBS.  DLNK5070
         BALR  GRR,GRL                                                 DLNK5080
         SR    GR0,GR0            SET IFLAG TO INDICATE THAT LIST      DLNK5090
         ST    GR0,0(0,GR4)       CALLED NAME HAS BEEN DELETED.        DLNK5100
         B     RETRN5                                                  DLNK5110
CS53     LA    GR0,1(0,0)         SET IFLAG TO INDICATE THAT THE LIST  DLNK5120
         ST    GR0,0(0,GR4)       CALLED NAME COULD NCT BE FOUND.      DLNK5130
RETRN5   L     GRS,SAREA+4                                             DLNK5140
         LM    GRR,GR5,12(GRS)    RESTORE GENERAL REGISTERS.           DLNK5150
         MVI   12(GRS),X'FF'                                           DLNK5160
         BR    GRR                RETURN TO CALLING RCUTINE.           DLNK5170
*                                                                      DLNK5180
         DROP  GRB                                                     DLNK5190
```

```
         USING *,GRL                                                    DLNK5200
TONDLP   MVI   TRACE,X'80'       TURN TRACE FLAG ON.                    DLNK5210
         BR    GRR                                                      DLNK5220
*                                                                       DLNK5230
         USING *,GRL                                                    DLNK5240
TOFDLP   MVI   TRACE,X'00'       TURN TRACE FLAG OFF.                   DLNK5250
         BR    GRR                                                      DLNK5260
*                                                                       DLNK5270
         CNOP  0,4                                                      DLNK5280
SAREA    DS    18F                                                      DLNK5290
AIADD    DS    1F                                                       DLNK5300
AIMARK   DS    1F                                                       DLNK5310
AIFLAG   DS    1F                                                       DLNK5320
SAVE1    DS    1F                                                       DLNK5330
BASE     DC    A(DLINKLP)                                               DLNK5340
NULL     DC    CL4'$$$$'                                                DLNK5350
ADEFNPS  DC    V(DEFNPS)                                                DLNK5360
LDEFNPS  DC    A(NAME)                                                  DLNK5370
         DC    XL1'80'                                                  DLNK5380
         DC    AL3(KFLAG)                                               DLNK5390
ARTNELP  DC    V(RTNELP)                                                DLNK5400
LRTNELP  DC    XL1'80'                                                  DLNK5410
         DC    AL3(KADD)                                                DLNK5420
ARTNLLP  DC    V(RTNLLP)                                                DLNK5430
LRTNLLP  DC    XL1'80'                                                  DLNK5440
         DC    AL3(KADD)                                                DLNK5450
NAME     DS    1F                                                       DLNK5460
KFLAG    DS    1F                                                       DLNK5470
ALP$$    DC    V(LASLP$)                                                DLNK5480
SAVE1TO3 DS    3F                                                       DLNK5490
AOBTNLP  DC    V(OBTNLP)                                                DLNK5500
LOBTNLP  DC    A(L)                                                     DLNK5510
         DC    A(KADD)                                                  DLNK5520
         DC    XL1'80'                                                  DLNK5530
         DC    AL3(KFLAG)                                               DLNK5540
L        DS    1F                                                       DLNK5550
KADD     DS    1F                                                       DLNK5560
ACONFLP  DC    V(CONFLP)                                                DLNK5570
LCONFLP  DC    XL1'80'                                                  DLNK5580
         DC    AL3(ICNT)                                                DLNK5590
SAVE3TO7 DS    4F                                                       DLNK5600
ICNT     DS    1F                                                       DLNK5610
TRACE    DC    XL1'00'                                                  DLNK5620
JFLAG    DS    1X                                                       DLNK5630
LFLAG    DS    1X                                                       DLNK5640
LTEST    DS    1X                                                       DLNK5650
*                                                                       DLNK5660
DALP$$   DSECT                                                          DLNK5670
NPNT     DS    1F                                                       DLNK5680
         END                                                            DLNK5690
```

```
LOPR     TITLE 'LOPERLP --- LOGICAL OPERATIONS ROUTINE'                  LOPR0010
LOPERLP  CSECT                                                           LOPR0020
*                                                                        LOPR0030
*        ROUTINE WHICH IS USED TO PERFORM LOGICAL OPERATIONS ON STRINGS  LOPR0040
*        OF CHARACTERS AND/OR BITS.  ENTRY POINTS AND ARGUMENTS ARE ...  LOPR0050
*                                                                        LOPR0060
*        1.  J = INCLP(IADD,ID,JADD,JD,N)                                LOPR0070
*              THE LOGICAL PRODUCT (AND) OF THE BITS AT LOCATION 1       LOPR0080
*              AND LOCATION 2 IS STORED AT LOCATION 1.  LOCATION 1 IS    LOPR0090
*              EQUAL TO THE ADDRESS IN IADD PLUS THE CONTENTS OF ID.     LOPR0100
*              LOCATION 2 IS EQUAL TO THE ADDRESS IN JADD PLUS THE       LOPR0110
*              CONTENTS OF JD.  N IS THE LENGTH, 1 TO 256 BYTES, OF THE  LOPR0120
*              STRINGS.  THE VALUE RETURNED BY THE FUNCTION IS ...       LOPR0130
*                 0 IF RESULT IS 0,                                      LOPR0140
*                 1 IF RESULT NOT 0.                                     LOPR0150
*                                                                        LOPR0160
*        2.  J = IOCLP(IADD,ID,JADD,JD,N)                                LOPR0170
*              SAME AS ABOVE (1.) EXCEPT THAT THE INCLUSIVE OR           LOPR0180
*              OPERATION IS PERFORMED.                                   LOPR0190
*                                                                        LOPR0200
*        3.  J = IXCLP(IADD,ID,JADD,JD,N)                                LOPR0210
*              SAME AS ABOVE (1. AND 2.) EXCEPT THAT THE EXCLUSIVE OR    LOPR0220
*              OPERATION IS PERFORMED.                                   LOPR0230
*                                                                        LOPR0240
*        4.  J = ITMLP(IADD,ID,M)                                        LOPR0250
*              THE ONE BYTE MASK, M, IS USED TO SELECT BITS TO BE        LOPR0260
*              TESTED AT LOCATION 1 (SEE ABOVE).  A BIT AT LOCATION 1    LOPR0270
*              IS TESTED IF THE CORRESPONDING BIT POSITION IN THE        LOPR0280
*              MASK IS A ONE.  THE VALUE RETURNED BY THE FUNCTION IS ..  LOPR0290
*                 0 IF SELECTED BITS ALL 0, OR MASK IS ALL ZERO.         LOPR0300
*                 1 IF SELECTED BITS ARE MIXED ZERO AND ONE.             LOPR0310
*                 3 IF SELECTED BITS ARE ALL ONES.                       LOPR0320
*                                                                        LOPR0330
*        5.  J = ICLCLP(IADD,ID,JADD,JD,N)                               LOPR0340
*              THE STRING (S1) AT LOCATION 1 IS COMPARED TO THE          LOPR0350
*              STRING (S2) AT LOCATION 2.  LOCATION 1, LOCATION 2        LOPR0360
*              AND N ARE AS ABOVE (1. ETC.).  THE VALUE RETURNED IS ...  LOPR0370
*                 0 S1 = S2.                                             LOPR0380
*                 1 S1 LESS THAN S2.                                     LOPR0390
*                 2 S1 GREATER THAN S2.                                  LOPR0400
*                                                                        LOPR0410
*        6.  CALL IMVCLP(IADD,ID,JADD,JD,K)                              LOPR0420
*              K BYTES ARE MOVED FROM LOCATION 2 TO LOCATION 1.          LOPR0430
*              LOCATION 1 AND LOCATION 2 ARE AS ABOVE.                   LOPR0440
*                                                                        LOPR0450
*        7.  J = ILALP(IVAL)                                             LOPR0460
*              THE FUNCTION RETURNS THE ADDRESS OF THE VARIABLE WHICH    LOPR0470
*              IS PASSED TO THE SUBPROGRAM, IVAL.                        LOPR0480
*                                                                        LOPR0490
         ENTRY INCLP                                                     LOPR0500
         ENTRY IOCLP                                                     LOPR0510
         ENTRY IXCLP                                                     LOPR0520
         ENTRY ITMLP                                                     LOPR0530
         ENTRY ICLCLP                                                    LOPR0540
         ENTRY IMVCLP                                                    LOPR0550
         ENTRY ILALP                                                     LOPR0560
*                                                                        LOPR0570
GRL      EQU   15                 LINKING AND BASE REGISTER.             LOPR0580
GRR      EQU   14                 RETURN REGISTER.                       LOPR0590
GRS      EQU   13                 SAVE AREA REGISTER.                    LOPR0600
GR0      EQU   0                  RESULT REGISTER.                       LOPR0610
GR1      EQU   1                  ARGUMENT REGISTER.                     LOPR0620
GR2      EQU   2                                                         LOPR0630
GR3      EQU   3                  WORK REGISTER.                         LOPR0640
```

```
GR4      EQU   4                                                    LOPR0650
GR5      EQU   5                                                    LOPR0660
*                                                                   LOPR0670
         USING INCLP,GRL                                            LOPR0680
INCLP    B     CS11                                                 LOPR0690
         DC    XL1'07'                                              LOPR0700
         DC    CL7'INCLP  '                                         LOPR0710
CS11     STM   GRR,GR5,12(GRS)     SAVE CONTENTS OF GENERAL REGISTERS. LOPR0720
         SR    GR0,GR0             ZERO RESULT.                     LOPR0730
         LM    GR1,GR5,0(GR1)      LOAD ADDRESS OF ARGUMENTS.       LOPR0740
         L     GR1,0(0,GR1)        LOAD CONTENTS OF IADD.           LOPR0750
         A     GR1,0(0,GR2)        FIND LOCATION 1.                 LOPR0760
         L     GR3,0(0,GR3)        LOAD CONTENTS OF JADD.           LOPR0770
         A     GR3,0(0,GR4)        FIND LOCATION 2.                 LOPR0780
         L     GR5,0(0,GR5)        LOAD N.                          LOPR0790
         BCTR  GR5,0               SUBTRACT 1 FROM N.               LOPR0800
         EX    GR5,AND             EXECUTE AND OPERATION.           LOPR0810
         BC    8,CS12              IF RESULT IS ZERO, BRANCH.       LOPR0820
         LA    GR0,1(0,0)          RESULT NOT ZERO.                 LOPR0830
CS12     L     GRR,12(0,GRS)                                        LOPR0840
         LM    GR1,GR5,24(GRS)     RESTORE GENERAL REGISTERS.       LOPR0850
         MVI   12(GRS),X'FF'                                        LOPR0860
         BR    GRR                 RETURN.                          LOPR0870
AND      NC    0(0,GR1),0(GR3)                                      LOPR0880
         DROP  GRL                                                  LOPR0890
*                                                                   LOPR0900
         USING IOCLP,GRL                                            LOPR0910
IOCLP    B     CS21                                                 LOPR0920
         DC    XL1'07'                                              LOPR0930
         DC    CL7'IOCLP  '                                         LOPR0940
CS21     STM   GRR,GR5,12(GRS)     SAVE CONTENTS OF GENERAL REGISTERS. LOPR0950
         SR    GR0,GR0             ZERO RESULT.                     LOPR0960
         LM    GR1,GR5,0(GR1)      LOAD ADDRESS OF ARGUMENTS.       LOPR0970
         L     GR1,0(0,GR1)        LOAD CONTENTS OF IADD.           LOPR0980
         A     GR1,0(0,GR2)        FIND LOCATION 1.                 LOPR0990
         L     GR3,0(0,GR3)        LOAD CONTENTS OF JADD.           LOPR1000
         A     GR3,0(0,GR4)        FIND LOCATION 2.                 LOPR1010
         L     GR5,0(0,GR5)        LOAD N.                          LOPR1020
         BCTR  GR5,0               SUBTRACT 1 FROM N.               LOPR1030
         EX    GR5,IOR             EXECUTE INCLUSIVE OR OPERATION.  LOPR1040
         BC    8,CS22              IF RESULT IS ZERO, BRANCH.       LOPR1050
         LA    GR0,1(0,0)          RESULT NOT ZERO.                 LOPR1060
CS22     L     GRR,12(0,GRS)                                        LOPR1070
         LM    GR1,GR5,24(GRS)     RESTORE GENERAL REGISTERS.       LOPR1080
         MVI   12(GRS),X'FF'                                        LOPR1090
         BR    GRR                 RETURN.                          LOPR1100
IOR      OC    0(0,GR1),0(GR3)                                      LOPR1110
         DROP  GRL                                                  LOPR1120
*                                                                   LOPR1130
         USING IXCLP,GRL                                            LOPR1140
IXCLP    B     CS31                                                 LOPR1150
         DC    XL1'07'                                              LOPR1160
         DC    CL7'IXCLP  '                                         LOPR1170
CS31     STM   GRR,GR5,12(GRS)     SAVE CONTENTS OF GENERAL REGISTERS. LOPR1180
         SR    GR0,GR0                                              LOPR1190
         LM    GR1,GR5,0(GR1)      LOAD ADDRESS OF ARGUMENTS.       LOPR1200
         L     GR1,0(0,GR1)        LOAD CONTENTS OF IADD.           LOPR1210
         A     GR1,0(0,GR2)        FIND LOCATION 1.                 LOPR1220
         L     GR3,0(0,GR3)        LOAD CONTENTS OF JADD.           LOPR1230
         A     GR3,0(0,GR4)        FIND LOCATION 2.                 LOPR1240
         L     GR5,0(0,GR5)        LOAD N.                          LOPR1250
         BCTR  GR5,0               SUBTRACT 1 FROM N.               LOPR1260
         EX    GR5,XOR             EXECUTE EXCLUSIVE OR OPERATION.  LOPR1270
         BC    8,CS32              IF RESULT IS ZERO, BRANCH.       LOPR1280
         LA    GR0,1(0,0)          RESULT NOT ZERO.                 LOPR1290
```

```
CS32     L      GRR,12(0,GRS)                                    LOPR1300
         LM     GR1,GR5,24(GRS)     RESTORE GENERAL REGISTERS.   LOPR1310
         MVI    12(GRS),X'FF'                                    LOPR1320
         BR     GRR                 RETURN.                      LOPR1330
XOR      XC     0(0,GR1),0(GR3)                                  LOPR1340
         DROP   GRL                                              LOPR1350
*                                                                LOPR1360
         USING  ITMLP,GRL                                        LOPR1370
ITMLP    B      CS41                                             LOPR1380
         DC     XL1'07'                                          LOPR1390
         DC     CL7'ITMLP  '                                     LOPR1400
CS41     STM    GRR,GR3,12(GRS)     SAVE CONTENTS OF GENERAL REGISTERS. LOPR1410
         SR     GR0,GR0             ZERO RESULT.                 LOPR1420
         LM     GR1,GR3,0(GR1)      LOAD ADDRESS OF ARGUMENTS.   LOPR1430
         L      GR1,0(0,GR1)        LOAD CONTENTS OF IADD.       LOPR1440
         A      GR1,0(0,GR2)        FIND LOCATION 1.             LOPR1450
         IC     GR3,0(0,GR3)        INSERT MASK.                 LOPR1460
         EX     GR3,TESTM           EXECUTE TEST UNDER MASK INSTRUCTION. LOPR1470
         BC     8,CS43              IF ALL ZEROES, BRANCH.       LOPR1480
         BC     4,CS42              IF MIXED ZEROES AND ONES, BRANCH. LOPR1490
         LA     GR0,3(0,0)          RESULT ALL ONES.             LOPR1500
         B      CS43                                             LOPR1510
CS42     LA     GR0,1(0,0)          RESULT MIXED.                LOPR1520
CS43     L      GRR,12(0,GRS)                                    LOPR1530
         LM     GR1,GR3,24(GRS)     RESTORE GENERAL REGISTERS.   LOPR1540
         MVI    12(GRS),X'FF'                                    LOPR1550
         BR     GRR                 RETURN.                      LOPR1560
*                                                                LOPR1570
TESTM    TM     0(GR1),X'00'                                     LOPR1580
         DROP   GRL                                              LOPR1590
*                                                                LOPR1600
         USING  ICLCLP,GRL                                       LOPR1610
ICLCLP   B      CS51                                             LOPR1620
         DC     XL1'07'                                          LOPR1630
         DC     CL7'ICLCLP '                                     LOPR1640
CS51     STM    GRR,GR5,12(GRS)     SAVE CONTENTS OF GENERAL REGISTERS. LOPR1650
         SR     GR0,GR0             ZERO RESULT.                 LOPR1660
         LM     GR1,GR5,0(GR1)      LOAD ADDRESS OF ARGUMENTS.   LOPR1670
         L      GR1,0(0,GR1)        LOAD CONTENTS OF IADD.       LOPR1680
         A      GR1,0(0,GR2)        FIND LOCATION 1.             LOPR1690
         L      GR3,0(0,GR3)        LOAD CONTENTS OF JADD.       LOPR1700
         A      GR3,0(0,GR4)        FIND LOCATION 2.             LOPR1710
         L      GR5,0(0,GR5)        LOAD N.                      LOPR1720
         BCTR   GR5,0               SUBTRACT 1 FROM N.           LOPR1730
         EX     GR5,COMPARE         EXECUTE COMP. LOGICAL CHAR. INST. LOPR1740
         BC     8,CS53              S1 = S2, BRANCH.             LOPR1750
         BC     4,CS52              S1 .LT. S2, BRANCH.          LOPR1760
         LA     GR0,2(0,0)          S1 .GT. S2, RESULT = 2.      LOPR1770
         B      CS53                                             LOPR1780
CS52     LA     GR0,1(0,0)          S1 .LT. S2, RESULT = 1.      LOPR1790
CS53     L      GRR,12(0,GRS)                                    LOPR1800
         LM     GR1,GR5,24(GRS)     RESTORE GENERAL REGISTERS.   LOPR1810
         MVI    12(GRS),X'FF'                                    LOPR1820
         BR     GRR                 RETURN.                      LOPR1830
COMPARE  CLC    0(0,GR1),0(GR3)                                  LOPR1840
         DROP   GRL                                              LOPR1850
*                                                                LOPR1860
         USING  IMVCLP,GRL                                       LOPR1870
IMVCLP   B      CS61                                             LOPR1880
         DC     XL1'07'                                          LOPR1890
         DC     CL7'IMVCLP '                                     LOPR1900
CS61     STM    GRR,GR5,12(GRS)     SAVE CONTENTS OF GENERAL REGISTERS. LOPR1910
         LM     GR1,GR5,0(GR1)      LOAD ADDRESS OF ARGUMENTS.   LOPR1920
         L      GR1,0(0,GR1)        LOAD CONTENTS OF IADD.       LOPR1930
         A      GR1,0(0,GR2)        FIND LOCATION 1.             LOPR1940
```

```
         L      GR3,0(0,GR3)        LOAD CONTENTS OF JADD.                LCPR1950
         A      GR3,0(0,GR4)        FIND LOCATION 2.                      LOPR1960
         L      GR5,0(0,GR5)        LOAD K.                               LOPR1970
         LA     GR2,255(0,0)        LOAD CONSTANTS.                       LOPR1980
         LA     GR4,1(0,GR2)                                              LOPR1990
MLOOP    CR     GR5,GR4             COMP. REMAINING CHAR. LEN. WITH 256.  LOPR2000
         BNH    FMOVE               EQUAL OR LOW, BRANCH.                 LOPR2010
         EX     GR2,MOVE            EXECUTE MOVE INSTRUCTION.             LOPR2020
         ALR    GR1,GR4             LOCATION 1 = LOCATION 1 + 256.        LOPR2030
         ALR    GR3,GR4             LOCATION 2 = LOCATION 2 + 256.        LOPR2040
         SLR    GR5,GR4             K = K - 256.                          LOPR2050
         B      MLOOP                                                     LOPR2060
FMOVE    BCTR   GR5,0               K = K - 1.                            LOPR2070
         EX     GR5,MOVE            EXECUTE MOVE INSTRUCTION.             LOPR2080
         LM     GRR,GR5,12(GRS)     RESTORE GENERAL REGISTERS.            LOPR2090
         MVI    12(GRS),X'FF'                                             LOPR2100
         BR     GRR                 RETURN.                               LOPR2110
MOVE     MVC    0(0,GR1),0(GR3)                                           LOPR2120
         DROP   GRL                                                       LOPR2130
*                                                                         LOPR2140
         USING  ILALP,GRL                                                 LOPR2150
ILALP    B      CS71                                                      LOPR2160
         DC     XL1'07'                                                   LOPR2170
         DC     CL7'ILALP  '                                              LOPR2180
CS71     L      GR0,0(0,GR1)        LOAD ADDRESS OF ARGUMENT.             LOPR2190
         N      GR0,=X'00FFFFFF'                                          LOPR2200
         BR     GRR                 RETURN.                               LOPR2210
         DROP   GRL                                                       LOPR2220
*                                                                         LOPR2230
         END                                                              LOPR2240
```

```
PDS      TITLE 'PUSHST --- PUSH DOWN STORE ROUTINE'                    PDS 0010
PUSHST   CSECT                                                         PDS 0020
*                                                                      PDS 0030
*        ROUTINE WHICH IS USED FOR DEFINING PUSH DOWN STORES, PUSHING  PDS 0040
*        ELEMENTS (A POINTER PLUS ONE WORD OF USER DATA) INTO A PUSH   PDS 0050
*        DOWN STORE, AND FOR POPPING AN ELEMENT FROM THE TOP OF A PUSH PDS 0060
*        DOWN STORE.  THE ENTRY POINTS AND ARGUMENTS ARE ...           PDS 0070
*                                                                      PDS 0080
*        1.  CALL INITPS(N,NPS)                                        PDS 0090
*              INITIALIZES MEMORY FOR N ELEMENTS OF LENGTH 8 (BYTES).  PDS 0100
*              AND FOR NPS PUSH DOWN STORES (STACKS) OF LENGTH 12.     PDS 0110
*                                                                      PDS 0120
*        2.  CALL DEFNPS(NAME,IFLAG)                                   PDS 0130
*              DEFINES A PUSH DOWN STORE CALLED NAME (UP TO 4 ALPHA-   PDS 0140
*              NUMERIC CHARACTERS).  IFLAG IS SET EQUAL TO 0 WHEN      PDS 0150
*              DEFINITION IS ERROR FREE, SET TO 1 IF A PUSH DOWN STORE PDS 0160
*              CALLED NAME HAS PREVIOUSLY BEEN DEFINED, AND IS SET TO  PDS 0170
*              2 IF THE LIST OF AVAILABLE SPACE FOR THE NAMES (NLAS)   PDS 0180
*              IS EMPTY.                                               PDS 0190
*                                                                      PDS 0200
*        3.  CALL DELTPS(NAME,IFLAG)                                   PDS 0210
*              DELETES A PUSH DOWN STORE CALLED NAME.  IFLAG IS SET    PDS 0220
*              EQUAL TO 0 WHEN DELETION IS SUCCESSFUL, SET TO 1 IF     PDS 0230
*              PUSH DOWN STORE CALLED NAME IS UNDEFINED.               PDS 0240
*                                                                      PDS 0250
*        4.  CALL PUSHPS(NAME,IADD,IFLAG)                              PDS 0260
*              PUSHES THE USERS DATA IN IADD TO THE TOP OF THE         PDS 0270
*              STACK CALLED NAME.  IFLAG IS SET TO 0 ON NORMAL RETURN, PDS 0280
*              SET TO 1 IF THE PUSH DOWN STORE CALLED NAME IS UN-      PDS 0290
*              DEFINED, SET TO 2 IF THE LAS FOR THE ELEMENTS (ELAS) IS PDS 0300
*              EMPTY.  IADD MUST BE 4 BYTES IN LENGTH.                 PDS 0310
*                                                                      PDS 0320
*        5.  CALL POPPS(NAME,IADD,IFLAG)                               PDS 0330
*              POPS THE TOP ELEMENT OF THE STACK CALLED NAME TO THE    PDS 0340
*              VARIABLE IADD.  IFLAG IS SET TO 0 ON NORMAL COMPLETION, PDS 0350
*              SET TO 1 IF PUSH DOWN STORE CALLED NAME IS NOT FOUND,   PDS 0360
*              SET TO 2 IF STACK CALLED NAME IS EMPTY.                 PDS 0370
*                                                                      PDS 0380
*        6.  CALL RLSEPS                                               PDS 0390
*              RELEASES MEMORY WHICH WAS ALLOCATED BY THE CALL TO      PDS 0400
*              INITPS.                                                 PDS 0410
*                                                                      PDS 0420
*        7.  CALL PEEPS(NAME,IADD1,IADD2,IFLAG)                        PDS 0430
*              RETURNS THE CONTENTS OF THE ELEMENT POINTED TO BY IADD1 PDS 0440
*              IN IADD1 (POINTER) AND IADD2 (USER DATA) UNLESS IADD1   PDS 0450
*              IS ZERO ON ENTRY.  WHEN IADD1 IS ZERO ON ENTRY THEN THE PDS 0460
*              FIRST ELEMENT POINTED TO BY THE PDS NAME ELEMENT CALLED PDS 0470
*              NAME IS LOADED INTO IADD1.  IFLAG IS EQUAL TO 0 ON      PDS 0480
*              RETURN WHEN OPERATION IS COMPLETED SUCCESSFULLY, 1 IF   PDS 0490
*              PDS CALLED NAME IS NOT FOUND, 2 IF PDS CALLED NAME IS   PDS 0500
*              EMPTY, AND 3 IF IADD1 IS LOADED WITH THE NULL POINTER.  PDS 0510
*                                                                      PDS 0520
*        8.  CALL NEEPS(NAME,IADD,IFLAG)                               PDS 0530
*              RETURNS THE NAME OF THE PDS NAME ELEMENT POINTED TO BY  PDS 0540
*              IADD AND THEN UPDATES IADD SO THAT IT POINTS TO THE NEXT PDS 0550
*              PDS NAME ELEMENT UNLESS IADD IS EQUAL TO ZERO ON ENTRY. PDS 0560
*              IN THIS CASE IADD IS LOADED WITH THE ADDRESS OF THE     PDS 0570
*              FIRST PDS NAME ELEMENT.  IFLAG IS EQUAL TO 0 ON RETURN  PDS 0580
*              WHEN OPERATION IS COMPLETED SUCCESSFULLY, 1 IF IADD IS  PDS 0590
*              LOADED WITH A NULL POINTER, AND 2 IF THERE ARE NO       PDS 0600
*              STACKS.                                                 PDS 0610
*                                                                      PDS 0620
*        9.  CALL TRONPS                                               PDS 0630
*              SETS TRACE FLAG TO INDICATE THAT CALLS TO ENTRY         PDS 0640
```

```
*              POINTS IN PUSHST ARE RECORDED.                          PDS 0650
*                                                                      PDS 0660
*       10. CALL TOFFPS                                                PDS 0670
*              SETS TRACE FLAG TO INDICATE THAT CALLS TO ENTRY         PDS 0680
*              POINTS IN PUSHST ARE NOT RECORDED.                      PDS 0690
*                                                                      PDS 0700
*       NOTE *** CALLS TO PEEPS AND NEEPS ARE NOT RECORDED WHEN USING  PDS 0710
*              THE TRACE FEATURE BECAUSE THEY DO NOT MODIFY THE STACKS. PDS 0720
*                                                                      PDS 0730
*       THE LIST OF AVAILABLE SPACE FOR THE PDS ELEMENTS (PLAS) HAS    PDS 0740
*       THE FOLLOWING FORMAT.                                          PDS 0750
*              P1 - POINTS TO THE NEXT AVAILABLE ELEMENT (BYTES 1-4).  PDS 0760
*                   NULL IF NONE.                                      PDS 0770
*              NOT USED (BYTES 5-8).                                   PDS 0780
*       THE LIST OF AVAILABLE SPACE FOR THE PDS NAME ELEMENTS (NLAS)   PDS 0790
*       HAS THE FOLLOWING FORMAT.                                      PDS 0800
*              P1 - POINTS TO THE NEXT AVAILABLE ELEMENT (BYTES 1-4).  PDS 0810
*                   NULL IF NONE.                                      PDS 0820
*              NOT USED (BYTES 5-12).                                  PDS 0830
*       THE PDS NAME LIST HAS THE FOLLOWING FORMAT.                    PDS 0840
*              NAME - ALPHANUMERIC NAME OF A PDS (BYTES 1-4).          PDS 0850
*              P1 - POINTS TO THE NEXT PDS NAME ELEMENT (BYTES 5-8).   PDS 0860
*                   NULL IF NONE.                                      PDS 0870
*                   THE NAMES ARE IN ASCENDING COLLATING SEQUENCE.     PDS 0880
*              P2 - POINTS TO THE STACK CALLED NAME (BYTES 9-12).      PDS 0890
*                   NULL IF STACK IS EMPTY.                            PDS 0900
*       THE STACKS HAVE THE FOLLOWING FORMAT.                          PDS 0910
*              P1 - POINTS TO THE NEXT LOWER ELEMENT IN THE STACK      PDS 0920
*                   (BYTES 1-4).  NULL IF NONE.                        PDS 0930
*              USER DATA - UP TO 4 BYTES OF DATA (BYTES 5-8).          PDS 0940
*       NPNT - POINTS TO THE FIRST PDS NAME ELEMENT.                   PDS 0950
*       PLAS - POINTS TO THE FIRST ELEMENT IN THE PLAS.                PDS 0960
*       PLAST - POINTS TO THE LAST ELEMENT IN THE PLAS.                PDS 0970
*       NLAS - POINTS TO THE FIRST ELEMENT IN THE NLAS.                PDS 0980
*       NLAST - POINTS TO THE LAST ELEMENT IN THE NLAS.                PDS 0990
*                                                                      PDS 1000
       ENTRY INITPS                                                    PDS 1010
       ENTRY DEFNPS                                                    PDS 1020
       ENTRY DELTPS                                                    PDS 1030
       ENTRY PUSHPS                                                    PDS 1040
       ENTRY POPPS                                                     PDS 1050
       ENTRY RLSEPS                                                    PDS 1060
       ENTRY PEEPS                                                     PDS 1070
       ENTRY NEEPS                                                     PDS 1080
       ENTRY TRONPS                                                    PDS 1090
       ENTRY TOFFPS                                                    PDS 1100
*                                                                      PDS 1110
GRL    EQU   15                 LINKING REGISTER.                      PDS 1120
GRR    EQU   14                 RETURN REGISTER.                       PDS 1130
GRB    EQU   2                  BASE REGISTER.                         PDS 1140
GRS    EQU   13                 SAVE REGISTER.                         PDS 1150
GR1    EQU   1                  ARGUMENT POINTER REGISTER.             PDS 1160
GR0    EQU   0                                                         PDS 1170
GR3    EQU   3                                                         PDS 1180
GR4    EQU   4                  WORK REGISTERS.                        PDS 1190
GR5    EQU   5                                                         PDS 1200
GR6    EQU   6                                                         PDS 1210
GR7    EQU   7                                                         PDS 1220
*                                                                      PDS 1230
       USING *,GRL                                                     PDS 1240
INITPS B     CS11                                                      PDS 1250
       DC    XL1'07'                                                   PDS 1260
       DC    CL7'INITPS '                                              PDS 1270
CS11   STM   GRR,GR7,12(GRS)    SAVE CONTENTS OF GENERAL REGISTERS.    PDS 1280
       L     GRB,BASE                                                  PDS 1290
```

```
         USING PUSHST,GRB                                              PDS 1300
         DROP  GRL                                                     PDS 1310
         LR    GR3,GRS           SWITCH SAVE AREA POINTERS.            PDS 1320
         LA    GRS,SAREA                                               PDS 1330
         ST    GR3,4(GRS)        SAVE PREVIOUS SAVE AREA POINTER.      PDS 1340
         ST    GRS,8(GR3)        SAVE PRESENT SAVE AREA POINTER.       PDS 1350
         TM    TRACE,X'80'       TEST FOR TRACE.                       PDS 1360
         BZ    TR1               BRANCH IF NO TRACE.                   PDS 1370
         L     GRL,=V(INITP$)                                          PDS 1380
         BALR  GRR,GRL           TRACE.                                PDS 1390
TR1      TM    FLAGS,X'80'       TEST FOR PREVIOUS INITIALIZATION.     PDS 1400
         BO    RETRN             IF SO THEN RETURN.                    PDS 1410
         L     GR3,0(0,GR1)                                            PDS 1420
         L     GR3,0(GR3)        LOAD N.                               PDS 1430
         L     GR5,4(0,GR1)                                            PDS 1440
         L     GR5,0(0,GR5)      LOAD NPS.                             PDS 1450
         SLL   GR3,3(0)          COMPUTE PLENGH (LENGTH OF PDS BLK).   PDS 1460
         ST    GR3,PLENGH        STORE PLENGH.                         PDS 1470
         GETMAIN R,LV=(GR3)      ALLOCATE PDS BLOCK.                   PDS 1480
         ST    GR1,PBLKAD        STORE ADDRESS OF PDS BLOCK.           PDS 1490
         ST    GR1,PLAS          SAVE ADDRESS OF FIRST AVAILABLE       PDS 1500
         LR    GR7,GR1           SPACE FOR PDS ELEMENTS IN PLAS.       PDS 1510
         LA    GR6,8(0,0)        COMPUTE ADDRESS OF LAST AVAILABLE     PDS 1520
         ALR   GR7,GR3           SPACE FOR PDS ELEMENTS (PLAST).       PDS 1530
         SLR   GR7,GR6                                                 PDS 1540
         ST    GR7,PLAST         SAVE PLAST.                           PDS 1550
LOOP11   LR    GR4,GR1           CHAIN PDS LIST OF AVAILABLE SPACE -   PDS 1560
         BXH   GR1,GR6,ELOP11    PLAS.                                 PDS 1570
         ST    GR1,0(0,GR4)                                            PDS 1580
         B     LOOP11                                                  PDS 1590
ELOP11   L     GR1,NULL                                                PDS 1600
         ST    GR1,0(0,GR4)      END CHAIN OF PLAS.                    PDS 1610
         LA    GR6,12(0,0)                                             PDS 1620
         MR    GR4,GR6           COMPUTE NLENGH (LENGTH OF PDS NAME    PDS 1630
         ST    GR5,NLENGH        BLOCK) AND STORE.                     PDS 1640
         GETMAIN R,LV=(GR5)      ALLOCATE PDS NAME BLOCK.              PDS 1650
         ST    GR1,NBLKAD        STORE ADDRESS OF PDS NAME BLOCK.      PDS 1660
         ST    GR1,NLAS          SAVE ADDRESS OF FIRST AVAILABLE       PDS 1670
         LR    GR7,GR1           SPACE FOR PDS NAME ELEMENTS IN NLAS.  PDS 1680
         ALR   GR7,GR5           COMPUTE AND SAVE THE ADDRESS OF THE   PDS 1690
         SLR   GR7,GR6           LAST AVAILABLE SPACE FOR THE PDS      PDS 1700
         ST    GR7,NLAST         NAME ELEMENTS (NLAST).                PDS 1710
LOOP12   LR    GR4,GR1           CHAIN PDS NAME LIST OF AVAILABLE      PDS 1720
         BXH   GR1,GR6,ELOP12    SPACE - NLAS.                         PDS 1730
         ST    GR1,0(0,GR4)                                            PDS 1740
         B     LOOP12                                                  PDS 1750
ELOP12   L     GR1,NULL                                                PDS 1760
         ST    GR1,0(0,GR4)      END CHAIN OF NLAS.                    PDS 1770
         ST    GR1,NPNT          MAKE NPNT POINT TO NULL.              PDS 1780
         MVI   FLAGS,X'80'       SET INITIALIZATION FLAG.              PDS 1790
         B     RETRN             RETURN.                               PDS 1800
*                                                                      PDS 1810
         DROP  GRB                                                     PDS 1820
         USING *,GRL                                                   PDS 1830
RLSEPS   B     CS21                                                    PDS 1840
         DC    XL1'07'                                                 PDS 1850
         DC    CL7'RLSEPS '                                            PDS 1860
CS21     STM   GRR,GR3,12(GRS)   SAVE CONTENTS OF GENERAL REGISTERS.   PDS 1870
         L     GRB,BASE                                                PDS 1880
         USING PUSHST,GRB                                              PDS 1890
         DROP  GRL                                                     PDS 1900
         LR    GR3,GRS           SWITCH SAVE AREA POINTERS.            PDS 1910
         LA    GRS,SAREA                                               PDS 1920
         ST    GR3,4(0,GRS)      SAVE PREVIOUS SAVE AREA POINTER.      PDS 1930
         ST    GRS,8(0,GR3)      SAVE PRESENT SAVE AREA POINTER.       PDS 1940
```

```
         TM    TRACE,X'80'          TEST FOR TRACE.                    PDS 1950
         BZ    TR2                  BRANCH IF NO TRACE.                PDS 1960
         L     GRL,=V(RLSEP$)                                          PDS 1970
         BALR  GRR,GRL              TRACE.                             PDS 1980
TR2      TM    FLAGS,X'80'          TEST FOR PREVIOUS INITIALIZATION.  PDS 1990
         BO    CS22                 PREVIOUS INITIALIZATION, CONTINUE. PDS 2000
         LA    GR1,ERR21A           ERROR, NO PREVIOUS INITIALIZATION. PDS 2010
         L     GRL,AEOUT            PRINT ERROR MESSAGE.               PDS 2020
         BALR  GRR,GRL                                                 PDS 2030
         B     ABND                 TERMINATE JOB.                     PDS 2040
CS22     L     GR0,PLENGH           LOAD PDS BLOCK LENGTH.             PDS 2050
         L     GR1,PBLKAD           LOAD PDS BLOCK ADDRESS.            PDS 2060
         FREEMAIN R,LV=(GR0),A=(GR1) RELEASE BLOCK.                    PDS 2070
         L     GR0,NLENGH           LOAD PDS NAME BLOCK LENGTH.        PDS 2080
         L     GR1,NBLKAD           LOAD PDS NAME BLOCK ADDRESS.       PDS 2090
         FREEMAIN R,LV=(GR0),A=(GR1) RELEASE BLOCK.                    PDS 2100
         SR    GR1,GR1                                                 PDS 2110
         ST    GR1,FLAGS            ZERO FLAGS.                        PDS 2120
         L     GRS,SAREA+4                                             PDS 2130
         LM    GRR,GR3,12(GRS)      RESTORE GENERAL REGISTERS.         PDS 2140
         MVI   12(GRS),X'FF'                                           PDS 2150
         BR    GRR                                                     PDS 2160
*                                                                      PDS 2170
         DROP  GRB                                                     PDS 2180
         USING *,GRL                                                   PDS 2190
DEFNPS   B     CS31                                                    PDS 2200
         DC    XL1'07'                                                 PDS 2210
         DC    CL7'DEFNPS '                                            PDS 2220
CS31     STM   GRR,GR7,12(GRS)      SAVE CONTENTS OF GENERAL REGISTERS. PDS 2230
         L     GRB,BASE                                                PDS 2240
         USING PUSHST,GRB                                              PDS 2250
         DROP  GRL                                                     PDS 2260
         LR    GR3,GRS              SWITCH SAVE AREA POINTERS.         PDS 2270
         LA    GRS,SAREA                                               PDS 2280
         ST    GR3,4(0,GRS)         SAVE PREVIOUS SAVE AREA POINTER.   PDS 2290
         ST    GRS,8(0,GR3)         SAVE PRESENT SAVE AREA POINTER.    PDS 2300
         TM    TRACE,X'80'          TEST FOR TRACE.                    PDS 2310
         BZ    TR3                  BRANCH IF NO TRACE.                PDS 2320
         L     GRL,=V(DEFNP$)                                          PDS 2330
         BALR  GRR,GRL              TRACE.                             PDS 2340
TR3      TM    FLAGS,X'80'          TEST FOR PREVIOUS INITIALIZATION.  PDS 2350
         BO    CS35                 PREVIOUS INITIALIZATION, CONTINUE. PDS 2360
         LA    GR1,ERR31A           ERROR, NO PREVIOUS INITIALIZATION. PDS 2370
         L     GRL,AEOUT            PRINT ERROR MESSAGE.               PDS 2380
         BALR  GRR,GRL                                                 PDS 2390
         B     ABND                 TERMINATE JOB.                     PDS 2400
CS35     L     GR3,0(0,GR1)                                            PDS 2410
         L     GR3,0(0,GR3)         LOAD NAME.                         PDS 2420
         LA    GR6,NPNT             LOAD ADDRESS OF PDS NAME POINTER.  PDS 2430
         L     GR5,NPNT             LOAD PDS NAME POINTER.             PDS 2440
LOOP31   CL    GR5,NULL                                                PDS 2450
         BE    CS32                 NAME NOT PREVIOUSLY DEFINED.       PDS 2460
         CL    GR3,0(0,GR5)                                            PDS 2470
         BL    CS32                 NAME NOT PREVIOUSLY DEFINED.       PDS 2480
         BE    CS33                 NAME PREVIOUSLY DEFINED.           PDS 2490
         LR    GR6,GR5                                                 PDS 2500
         LA    GR6,4(0,GR6)         UPDATE ADDRESS OF PDS NAME POINTER.PDS 2510
         L     GR5,0(0,GR6)         UPDATE PDS NAME POINTER.           PDS 2520
         B     LOOP31                                                  PDS 2530
CS33     L     GR4,4(0,GR1)         SET IFLAG TO INDICATE THAT A PDS   PDS 2540
         LA    GR0,1(0,0)           CALLED NAME HAS PREVIOUSLY BEEN    PDS 2550
         ST    GR0,0(0,GR4)         DEFINED.                           PDS 2560
         B     RETRN                                                   PDS 2570
CS32     L     GR4,NLAS             PDS CALLED NAME NOT PREV. DEFINED. PDS 2580
         CL    GR4,NULL             SEE IF LAS FOR PDS NAMES IS EMPTY. PDS 2590
```

```
         BE    CS34              IF SO BRANCH.                          PDS 2600
         L     GR0,0(0,GR4)                                             PDS 2610
         ST    GR0,NLAS          UPDATE LAS FOR PDS NAMES.              PDS 2620
         ST    GR4,0(0,GR6)      ADD A PDS CALLED NAME TO THE LIST      PDS 2630
         ST    GR3,0(0,GR4)      AND CHAIN THE ELEMENTS.                PDS 2640
         ST    GR5,4(0,GR4)                                             PDS 2650
         L     GR0,NULL                                                 PDS 2660
         ST    GR0,8(0,GR4)      MAKE THE PDS CALLED NAME EMPTY.        PDS 2670
         L     GR4,4(0,GR1)      SET IFLAG TO INDICATE THAT THE PDS     PDS 2680
         SR    GR0,GR0           CALLED NAME HAS BEEN SUCCESSFULLY      PDS 2690
         ST    GR0,0(0,GR4)      DEFINED.                               PDS 2700
         B     RETRN                                                    PDS 2710
CS34     L     GR4,4(0,GR1)      SET IFLAG TO INDICATE THAT THE LAS     PDS 2720
         LA    GR0,2(0,0)        FOR THE PDS NAMES IS EMPTY, PDS        PDS 2730
         ST    GR0,0(0,GR4)      CALLED NAME COULD NOT BE DEFINED.      PDS 2740
         B     RETRN                                                    PDS 2750
*                                                                       PDS 2760
         DROP  GRB                                                      PDS 2770
         USING *,GRL                                                    PDS 2780
PUSHPS   B     CS41                                                     PDS 2790
         DC    XL1'07'                                                  PDS 2800
         DC    CL7'PUSHPS '                                             PDS 2810
CS41     STM   GRR,GR7,12(GRS)   SAVE CONTENTS OF GENERAL REGISTERS.    PDS 2820
         L     GRB,BASE                                                 PDS 2830
         USING PUSHST,GRB                                               PDS 2840
         DROP  GRL                                                      PDS 2850
         LR    GR3,GRS           SWITCH SAVE AREA POINTERS.             PDS 2860
         LA    GRS,SAREA                                                PDS 2870
         ST    GR3,4(0,GRS)      SAVE PREVIOUS SAVE AREA POINTER.       PDS 2880
         ST    GRS,8(0,GR3)      SAVE PRESENT SAVE AREA POINTER.        PDS 2890
         TM    TRACE,X'80'       TEST FOR TRACE.                        PDS 2900
         BZ    TR4               BRANCH IF NO TRACE.                    PDS 2910
         L     GRL,=V(PUSHP$)                                           PDS 2920
         BALR  GRR,GRL           TRACE.                                 PDS 2930
TR4      L     GR3,0(0,GR1)                                             PDS 2940
         L     GR3,0(0,GR3)      LOAD NAME.                             PDS 2950
         L     GR5,NPNT          LOAD PDS NAME POINTER.                 PDS 2960
LOOP41   CL    GR5,NULL                                                 PDS 2970
         BE    CS42              PDS CALLED NAME NOT FOUND IF EQUAL.    PDS 2980
         CL    GR3,0(0,GR5)                                             PDS 2990
         BL    CS42              PDS CALLED NAME NOT FOUND IF LOW.      PDS 3000
         BE    CS43              PDS CALLED NAME FOUND IF EQUAL.        PDS 3010
         L     GR5,4(0,GR5)      CONTINUE SEARCH FOR PDS CALLED NAME.   PDS 3020
         B     LOOP41                                                   PDS 3030
CS42     L     GR4,8(0,GR1)      SET IFLAG TO INDICATE THAT A PDS       PDS 3040
         LA    GR0,1(0,0)        CALLED NAME IS NOT DEFINED.            PDS 3050
         ST    GR0,0(0,GR4)                                             PDS 3060
         B     RETRN                                                    PDS 3070
CS43     L     GR6,PLAS                                                 PDS 3080
         CL    GR6,NULL          TEST TO SEE IF THE LAS FOR THE PDS     PDS 3090
         BE    CS44              ELEMENTS IS EMPTY, IF SO BRANCH.       PDS 3100
         L     GR7,0(0,GR6)                                             PDS 3110
         ST    GR7,PLAS          UPDATE LAS FOR PDS ELEMENTS.           PDS 3120
         CL    GR7,NULL                                                 PDS 3130
         BNE   CS435                                                    PDS 3140
         LA    GR7,PLAS                                                 PDS 3150
         ST    GR7,PLAST                                                PDS 3160
CS435    L     GR7,8(0,GR5)                                             PDS 3170
         ST    GR7,0(0,GR6)      CHAIN PDS ELEMENT.                     PDS 3180
         ST    GR6,8(0,GR5)      UPDATE PDS NAME POINTER.               PDS 3190
         L     GR4,4(0,GR1)                                             PDS 3200
         L     GR4,0(0,GR4)      LOAD IADD.                             PDS 3210
         ST    GR4,4(0,GR6)      STORE IADD IN PDS ELEMENT.             PDS 3220
         L     GR4,8(0,GR1)      SET IFLAG TO INDICATE SUCCESSFUL       PDS 3230
         SR    GR0,GR0           PUSH.                                  PDS 3240
```

```
         ST    GR0,0(0,GR4)                                          PDS 3250
         B     RETRN                                                 PDS 3260
CS44     L     GR4,8(0,GR1)        SET IFLAG TO INDICATE THAT THE LAS PDS 3270
         LA    GR0,2(0,0)          FOR THE PDS ELEMENTS IS EMPTY.    PDS 3280
         ST    GR0,0(0,GR4)                                          PDS 3290
         B     RETRN                                                 PDS 3300
*                                                                    PDS 3310
         DROP  GRB                                                   PDS 3320
         USING *,GRL                                                 PDS 3330
POPPS    B     CS51                                                  PDS 3340
         DC    XL1'07'                                               PDS 3350
         DC    CL7'POPPS  '                                          PDS 3360
CS51     STM   GRR,GR7,12(GRS)     SAVE CONTENTS OF GENERAL REGISTERS. PDS 3370
         L     GRB,BASE                                              PDS 3380
         USING PUSHST,GRB                                            PDS 3390
         DROP  GRL                                                   PDS 3400
         LR    GR3,GRS             SWITCH SAVE AREA POINTERS.        PDS 3410
         LA    GRS,SAREA                                             PDS 3420
         ST    GR3,4(0,GRS)        SAVE PREVIOUS SAVE AREA POINTER.  PDS 3430
         ST    GRS,8(0,GR3)        SAVE PRESENT SAVE AREA POINTER.   PDS 3440
         TM    TRACE,X'80'         TEST FOR TRACE.                   PDS 3450
         BZ    TR5                 BRANCH IF NO TRACE.               PDS 3460
         L     GRL,=V(POPP$)                                         PDS 3470
         BALR  GRR,GRL             TRACE.                            PDS 3480
TR5      L     GR3,0(0,GR1)                                          PDS 3490
         L     GR3,0(0,GR3)        LOAD NAME.                        PDS 3500
         L     GR5,NPNT            LOAD PDS NAME POINTER.            PDS 3510
LOOP51   CL    GR5,NULL                                              PDS 3520
         BE    CS42                PDS CALLED NAME NOT FOUND IF EQUAL. PDS 3530
         CL    GR3,0(0,GR5)                                          PDS 3540
         BL    CS42                PDS CALLED NAME NOT FOUND IF LOW. PDS 3550
         BE    CS52                PDS CALLED NAME FOUND IF EQUAL.   PDS 3560
         L     GR5,4(0,GR5)        CONTINUE SEARCH FOR PDS CALLED NAME. PDS 3570
         B     LOOP51                                                PDS 3580
CS52     L     GR6,8(0,GR5)        LOAD ADD. OF TOP ELEMENT OF PDS.  PDS 3590
         CL    GR6,NULL            TEST FOR EMPTY STACK.             PDS 3600
         BE    CS53                IF SO, BRANCH.                    PDS 3610
         L     GR7,0(0,GR6)        UPDATE PDS NAME ELEMENT.          PDS 3620
         ST    GR7,8(0,GR5)                                          PDS 3630
         L     GR7,PLAST                                             PDS 3640
         ST    GR6,0(0,GR7)        UPDATE LAS FOR PDS ELEMENTS.      PDS 3650
         ST    GR6,PLAST           (RETURN SPACE TO LAS).            PDS 3660
         L     GR7,NULL                                              PDS 3670
         ST    GR7,0(0,GR6)                                          PDS 3680
         L     GR7,4(0,GR6)        RETURN USER DATA IN VARIABLE IADD. PDS 3690
         L     GR4,4(0,GR1)        LOAD ADDRESS OF IADD.             PDS 3700
         ST    GR7,0(0,GR4)        STORE INTO IADD.                  PDS 3710
         L     GR4,8(0,GR1)        SET IFLAG TO INDICATE THAT TOP OF PDS 3720
         SR    GR0,GR0             STACK CALLED NAME HAS BEEN        PDS 3730
         ST    GR0,0(0,GR4)        SUCCESSFULLY POPPED TO IADD.      PDS 3740
         B     RETRN                                                 PDS 3750
CS53     L     GR4,8(0,GR1)        SET IFLAG TO INDICATE THAT STACK  PDS 3760
         LA    GR0,2(0,0)          CALLED NAME IS EMPTY.             PDS 3770
         ST    GR0,0(0,GR4)                                          PDS 3780
         B     RETRN                                                 PDS 3790
*                                                                    PDS 3800
         DROP  GRB                                                   PDS 3810
         USING *,GRL                                                 PDS 3820
DELTPS   B     CS61                                                  PDS 3830
         DC    XL1'07'                                               PDS 3840
         DC    CL7'DELTPS '                                          PDS 3850
CS61     STM   GRR,GR7,12(GRS)     SAVE CONTENTS OF GENERAL REGISTERS. PDS 3860
         L     GRB,BASE                                              PDS 3870
         USING PUSHST,GRB                                            PDS 3880
         DROP  GRL                                                   PDS 3890
```

```
         LR    GR3,GRS            SWITCH SAVE AREA POINTERS.          PDS 3900
         LA    GRS,SAREA                                              PDS 3910
         ST    GR3,4(0,GRS)       SAVE PREVIOUS SAVE AREA POINTER.    PDS 3920
         ST    GRS,8(0,GR3)       SAVE PRESENT SAVE AREA POINTER.     PDS 3930
         TM    TRACE,X'80'        TEST FOR TRACE.                     PDS 3940
         BZ    TR6                BRANCH IF NO TRACE.                 PDS 3950
         L     GRL,=V(DELTP$)                                         PDS 3960
         BALR  GRR,GRL            TRACE.                              PDS 3970
TR6      L     GR3,0(0,GR1)                                           PDS 3980
         L     GR3,0(0,GR3)       LOAD NAME.                          PDS 3990
         TM    FLAGS,X'80'        TEST FOR INITIALIZATION.            PDS 4000
         BO    CS625              IF SO CONTINUE.                     PDS 4010
         LA    GR1,ERR61A         ERROR, NO PREVIOUS INITIALIZATION.  PDS 4020
         L     GRL,AEOUT          PRINT ERROR MESSAGE.                PDS 4030
         BALR  GRR,GRL                                                PDS 4040
         B     ABND               TERMINATE JOB.                      PDS 4050
CS625    LA    GR6,NPNT           LOAD ADDRESS OF PDS NAME POINTER.   PDS 4060
         L     GR5,NPNT           LOAD PDS NAME POINTER.              PDS 4070
LOOP61   CL    GR5,NULL                                               PDS 4080
         BE    CS62               NAME NOT DEFINED, BRANCH.           PDS 4090
         CL    GR3,0(0,GR5)                                           PDS 4100
         BL    CS62               NAME NOT DEFINED, BRANCH.           PDS 4110
         BE    CS63               NAME DEFINED, BRANCH.               PDS 4120
         LR    GR6,GR5                                                PDS 4130
         LA    GR6,4(0,GR6)       UPDATE ADDRESS OF PDS NAME POINTER. PDS 4140
         L     GR5,0(0,GR6)       UPDATE PDS NAME POINTER.            PDS 4150
         B     LOOP61                                                 PDS 4160
CS62     L     GR4,4(0,GR1)       SET IFLAG TO INDICATE THAT PDS      PDS 4170
         LA    GR0,1(0,0)         CALLED NAME IS NOT DEFINED.         PDS 4180
         ST    GR0,0(0,GR4)                                           PDS 4190
         B     RETRN                                                  PDS 4200
CS63     L     GR7,4(0,GR5)       RETURN THE PDS NAME ELEMENT TO THE  PDS 4210
         ST    GR7,0(0,GR6)       LAS FOR NAME ELEMENTS, CHAIN        PDS 4220
         L     GR7,NLAST          REMAINING PDS NAME ELEMENTS.        PDS 4230
         ST    GR5,0(0,GR7)                                           PDS 4240
         ST    GR5,NLAST                                              PDS 4250
         L     GR7,NULL                                               PDS 4260
         ST    GR7,0(0,GR5)                                           PDS 4270
         L     GR5,8(0,GR5)       RETURN THE PDS ELEMENTS (FOR STACK  PDS 4280
LOOP62   CL    GR5,NULL           CALLED NAME) TO THE LAS FOR THE     PDS 4290
         BE    CS64               PDS ELEMENTS.                       PDS 4300
         LR    GR4,GR5                                                PDS 4310
         L     GR6,PLAST                                              PDS 4320
         ST    GR5,0(0,GR6)                                           PDS 4330
         ST    GR5,PLAST                                              PDS 4340
         L     GR5,0(0,GR4)                                           PDS 4350
         B     LOOP62                                                 PDS 4360
CS64     L     GR6,PLAST          PUT NULL POINTER IN LAST ELEMENT OF PDS 4370
         ST    GR7,0(0,GR5)       LAS FOR ELEMENTS, SET IFLAG TO      PDS 4380
         SR    GR7,GR7            INDICATE SUCCESSFUL DELETION.       PDS 4390
         L     GR4,4(0,GR1)                                           PDS 4400
         ST    GR7,0(0,GR4)                                           PDS 4410
         B     RETRN                                                  PDS 4420
*                                                                     PDS 4430
         DROP  GRB                                                    PDS 4440
         USING *,GRL                                                  PDS 4450
PEEPS    B     CS71                                                   PDS 4460
         DC    XL1'07'                                                PDS 4470
         DC    CL7'PEEPS  '                                           PDS 4480
CS71     STM   GRR,GR7,12(GRS)    SAVE CONTENTS OF GENERAL REGISTERS. PDS 4490
         L     GRB,BASE                                               PDS 4500
         USING PUSHST,GRB                                             PDS 4510
         DROP  GRL                                                    PDS 4520
         LR    GR3,GRS            SWITCH SAVE AREA POINTERS.          PDS 4530
         LA    GRS,SAREA                                              PDS 4540
```

```
         ST    GR3,4(0,GR5)                                          PDS 4550
         ST    GR5,8(0,GR3)                                          PDS 4560
         LM    GR3,GR6,0(GR1)      LOAD ADDRESSES OF ARGUMENTS.      PDS 4570
         L     GR7,0(0,GR4)        LOAD IADD1.                       PDS 4580
         LTR   GR7,GR7             SEE IF IADD1=0.                   PDS 4590
         BNE   CS74                IF NOT BRANCH.                    PDS 4600
         L     GR3,0(0,GR3)        LOAD NAME.                        PDS 4610
         L     GR7,NPNT            LOAD ADD. OF FIRST PDS NAME ELEMENT, PDS 4620
LOOP71   CL    GR7,NULL            TEST FOR NULL POINTER.            PDS 4630
         BE    CS73                BRANCH IF FOUND.                  PDS 4640
         CL    GR3,0(0,GR7)        COMPARE NAME TO PDS NAME ENTRY.   PDS 4650
         BL    CS73                LOW.                              PDS 4660
         BE    CS72                EQUAL.                            PDS 4670
         L     GR7,4(0,GR7)        HIGH, CONTINUE.                   PDS 4680
         B     LOOP71                                                PDS 4690
CS72     L     GR7,8(0,GR7)        LOAD ADD. OF TOP OF PDS CALLED NAME. PDS 4700
         ST    GR7,0(0,GR4)        STORE INTO IADD1.                 PDS 4710
         CL    GR7,NULL            SEE IF PDS CALLED NAME IS EMPTY.  PDS 4720
         BNE   CS75                NO.                               PDS 4730
         LA    GR0,2(0,0)          YES, SET IFLAG TO INDICATE THAT PDS PDS 4740
         ST    GR0,0(0,GR6)        CALLED NAME IS EMPTY.             PDS 4750
         B     RETRN                                                 PDS 4760
CS73     LA    GR0,1(0,0)          SET IFLAG TO INDICATE THAT PDS    PDS 4770
         ST    GR0,0(0,GR6)        CALLED NAME IS NOT DEFINED.       PDS 4780
         B     RETRN                                                 PDS 4790
CS74     L     GR0,4(0,GR7)        LOAD USER DATA POINTED TO BY IADD1. PDS 4800
         ST    GR0,0(0,GR5)        STORE IN IADD2.                   PDS 4810
         L     GR7,0(0,GR7)        UPDATE IADD1.                     PDS 4820
         ST    GR7,0(0,GR4)                                          PDS 4830
         CL    GR7,NULL            SEE IF IADD1 CONTAINS NULL POINTER. PDS 4840
         BNE   CS75                NO.                               PDS 4850
         LA    GR0,3(0,0)          YES, SET IFLAG TO INDICATE THAT   PDS 4860
         ST    GR0,0(0,GR6)        IADD1 CONTAINS A NULL POINTER.    PDS 4870
         B     RETRN                                                 PDS 4880
CS75     SR    GR0,GR0             SET IFLAG TO INDICATE SUCCESSFUL  PDS 4890
         ST    GR0,0(0,GR5)        COMPLETION.                       PDS 4900
         B     RETRN                                                 PDS 4910
*                                                                    PDS 4920
         DROP  GR8                                                   PDS 4930
         USING *,GRL                                                 PDS 4940
NEEPS    B     CS81                                                  PDS 4950
         DC    XL1'07'                                               PDS 4960
         DC    CL7'NEEPS  '                                          PDS 4970
CS81     STM   GR8,GR7,12(GR5)     SAVE CONTENTS OF GENERAL REGISTERS. PDS 4980
         L     GR8,BASE                                              PDS 4990
         USING PUSHST,GR8                                            PDS 5000
         DROP  GRL                                                   PDS 5010
         LR    GR3,GR5             SWITCH SAVE AREA POINTERS.        PDS 5020
         LA    GR5,SAREA                                             PDS 5030
         ST    GR3,4(0,GR5)                                          PDS 5040
         ST    GR5,8(0,GR3)                                          PDS 5050
         LM    GR3,GR5,0(GR1)      LOAD ADDRESSES OF ARGUMENTS.      PDS 5060
         L     GR6,0(0,GR4)        LOAD IADD.                        PDS 5070
         LTR   GR6,GR6             SEE IF IADD=0.                    PDS 5080
         BNE   CS82                NO.                               PDS 5090
         L     GR6,NPNT            LOAD ADDRESS OF FIRT PDS NAME     PDS 5100
         CL    GR6,NULL            ELEMENT, THEN TEST FOR EMPTY PDS  PDS 5110
         BNE   CS83                NAME LIST.  BRANCH NOT EMPTY.     PDS 5120
         LA    GR7,2(0,0)          SET IFLAG TO INDICATE THAT PDS NAME PDS 5130
         ST    GR7,0(0,GR5)        LIST IS EMPTY.                    PDS 5140
         B     RETRN                                                 PDS 5150
CS82     L     GR7,0(0,GR6)        LOAD PDS NAME POINTED TO BY IADD. PDS 5160
         ST    GR7,0(0,GR3)        STORE NAME.                       PDS 5170
         L     GR6,4(0,GR6)        UPDATE IADD.                      PDS 5180
         CL    GR6,NULL            SEE IF IADD CONTAINS NULL POINTER. PDS 5190
```

```
         BNE   CS83              NO.                              PDS 5200
         LA    GR7,1(0,0)        YES, SET IFLAG TO INDICATE THAT  PDS 5210
         ST    GR7,0(0,GR5)      IADD CONTAINS A NULL POINTER.    PDS 5220
         ST    GR6,0(0,GR4)      STORE IADD.                      PDS 5230
         B     RETRN                                              PDS 5240
CS83     ST    GR6,0(0,GR4)      STORE IADD.                      PDS 5250
         SR    GR7,GR7           SET IFLAG TO INDICATE SUCCESSFUL PDS 5260
         ST    GR7,0(0,GR5)      COMPLETION.                      PDS 5270
         B     RETRN                                              PDS 5280
*                                                                 PDS 5290
ABND     ABEND 10,DUMP                                           PDS 5300
*                                                                 PDS 5310
RETRN    L     GRS,SAREA+4                                       PDS 5320
         LM    GRR,GR7,12(GRS)   RESTORE GENERAL REGISTERS.       PDS 5330
         MVI   12(GRS),X'FF'                                     PDS 5340
         BR    GRR               RETURN TO CALLING ROUTINE.       PDS 5350
*                                                                 PDS 5360
         DROP  GRB                                               PDS 5370
         USING *,GRL                                             PDS 5380
TRONPS   MVI   TRACE,X'80'       TURN TRACE FLAG ON.              PDS 5390
         BR    GRR                                               PDS 5400
*                                                                 PDS 5410
         USING *,GRL                                             PDS 5420
TOFFPS   MVI   TRACE,X'00'       TURN TRACE FLAG OFF.             PDS 5430
         BR    GRR                                               PDS 5440
*                                                                 PDS 5450
         CNOP  0,4                                               PDS 5460
SAREA    DS    18F                                               PDS 5470
PLAS     DS    1F                                                PDS 5480
PLAST    DS    1F                                                PDS 5490
NPNT     DS    1F                                                PDS 5500
NLAS     DS    1F                                                PDS 5510
NLAST    DS    1F                                                PDS 5520
PLENGH   DS    1F                                                PDS 5530
PBLKAD   DS    1F                                                PDS 5540
NLENGH   DS    1F                                                PDS 5550
NBLKAD   DS    1F                                                PDS 5560
FLAGS    DC    F'0'                                              PDS 5570
BASE     DC    A(PUSHST)                                         PDS 5580
ARGADD   DS    1F                                                PDS 5590
AEOUT    DC    V(MESS$$)                                         PDS 5600
ERR21A   DC    XL1'80'                                           PDS 5610
         DC    AL3(ERR21)                                        PDS 5620
ERR31A   DC    XL1'80'                                           PDS 5630
         DC    AL3(ERR31)                                        PDS 5640
ERR61A   DC    XL1'80'                                           PDS 5650
         DC    AL3(ERR61)                                        PDS 5660
ERR21    DC    F'1'                                              PDS 5670
ERR31    DC    F'2'                                              PDS 5680
ERR61    DC    F'3'                                              PDS 5690
NULL     DC    CL4'$$$$'                                         PDS 5700
TRACE    DC    XL1'00'                                           PDS 5710
         END                                                     PDS 5720
```

```
SPCE     TITLE 'SPACELP --- MAINTAINS THE LIST OF AVAILABLE SPACE'      SPCE0010
SPACELP  CSECT                                                          SPCE0020
*                                                                       SPCE0030
*        ROUTINE WHICH IS USED FOR MAINTAINING THE LIST OF AVAILABLE    SPCE0040
*        SPACE (LAS) FOR THE LIST CONTROL BLOCKS (LCB) AND THE LIST     SPCE0050
*        ELEMENTS.  THE ENTRY POINTS AND ARGUMENTS ARE ...              SPCE0060
*                                                                       SPCE0070
*        1.  CALL INITLP(NB)                                            SPCE0080
*              INITIALIZES THE LAS SO THAT IT CONTAINS NB BYTES OF      SPCE0090
*              AVAILABLE SPACE.  ALSO MAKES A CALL TO INITPS TO         SPCE0100
*              INITIALIZE MEMORY FOR NPELEMTS PDS ELEMENTS AND NPDS     SPCE0110
*              PUSH DOWN STORES (STACKS).                               SPCE0120
*                                                                       SPCE0130
*        2.  CALL RLSELP                                                SPCE0140
*              RELEASES MEMORY WHICH WAS ALLOCATED BY THE CALL TO       SPCE0150
*              INITLP AND MAKES A CALL TO RLSEPS WHICH RELEASES         SPCE0160
*              MEMORY ALLOCATED BY THE CALL TO INITPS.                  SPCE0170
*                                                                       SPCE0180
*        3.  CALL OBTNLP(L,IADD,IFLAG)                                  SPCE0190
*              OBTAINS THE ADDRESS OF A BLOCK OF CONTIGUOUS BYTES OF    SPCE0200
*              LENGTH L AND RETURNS THIS ADDRESS IN IADD.  IFLAG IS     SPCE0210
*              SET TO 0 IF THE OPERATION IS SUCCESSFUL, SET TO 1 IF     SPCE0220
*              L CONTIGUOUS BYTES CAN'T BE FOUND IN THE LAS.            SPCE0230
*                                                                       SPCE0240
*        4.  CALL CONFLP(ICNT)                                          SPCE0250
*              CONNECTS THE LAS FRAGMENTS.  ICNT IS RETURNED WITH THE   SPCE0260
*              NUMBER OF CONNECTIONS MADE.                              SPCE0270
*                                                                       SPCE0280
*        5.  CALL RTNLLP(IADD)                                          SPCE0290
*              RETURNS THE SPACE OCCUPIED BY A LIST (POINTED TO BY      SPCE0300
*              IADD) TO THE LAS.  IF THE FIRST BIT OF THE LOCATION      SPCE0310
*              POINTED TO BY IADD IS NOT ONE THEN THE CALL IS           SPCE0320
*              IGNORED.                                                 SPCE0330
*                                                                       SPCE0340
*        6.  CALL RTNELP(IADD)                                          SPCE0350
*              RETURNS THE SPACE OCCUPIED BY AN ELEMENT (POINTED TO     SPCE0360
*              BY IADD) TO THE LAS.  IF THE FIRST BIT OF THE LOCATION   SPCE0370
*              POINTED TO BY IADD IS ONE THEN THE CALL IS IGNORED.      SPCE0380
*                                                                       SPCE0390
*        7.  CALL PLASLP(IADD,ILEN,IFLAG)                               SPCE0400
*              USED TO INTERROGATE THE LAS.  IF IADD = 0 ON ENTRY       SPCE0410
*              THEN THE ADDRESS OF THE FIRST LAS ELEMENT IS RETURNED    SPCE0420
*              IN IADD.  IF IADD IS NOT EQUAL TO 0 ON ENTRY THEN        SPCE0430
*              THE FIRST WORD (4 BYTES) OF THE LAS ELEMENT POINTED TO   SPCE0440
*              BY IADD IS RETURNED IN ILEN AND IADD IS UPDATED TO POINT SPCE0450
*              TO THE NEXT LAS ELEMENT.  IFLAG IS SET TO 0 WHEN IADD IS SPCE0460
*              UPDATED SUCCESSFULLY, 1 IF IADD IS LOADED WITH A NULL    SPCE0470
*              POINTER, AND 2 IF THE LAS IS EMPTY.  CALLS TO PLASLP     SPCE0480
*              ARE NOT TRACED.                                          SPCE0490
*                                                                       SPCE0500
*        8.  CALL TONSLP                                                SPCE0510
*              SETS TRACE FLAG TO INDICATE THAT CALLS TO ENTRY          SPCE0520
*              POINTS IN SPACELP ARE TO BE RECORDED.                    SPCE0530
*                                                                       SPCE0540
*        9.  CALL TOFSLP                                                SPCE0550
*              SETS TRACE FLAG TO INDICATE THAT CALLS TO ENTRY          SPCE0560
*              POINTS IN SPACELP ARE NOT TO BE RECORDED.                SPCE0570
*                                                                       SPCE0580
*        THE LAS ELEMENTS HAVE THE FOLLOWING FORMAT.                    SPCE0590
*              ID = X'00' FOR SIMPLE ELEMENT, X'4-' FOR LISTR (BYTE 1), SPCE0600
*              LENGTH - LENGTH OF THE LAS ELEMENT IN BYTES (BYTES 2-4), SPCE0610
*              P1 - POINTER TO NEXT LAS ELEMENT, P1 IS NULL IF THE      SPCE0620
*                   ELEMENT IS THE LAST LAS ELEMENT (BYTES 5-8),        SPCE0630
*              P3 - FOR LISTR ONLY, POINTS TO A LCB (BYTES 13-16).      SPCE0640
```

```
*        LAS - POINTS TO THE FIRST AVAILABLE LAS ELEMENT.                SPCE0650
*        LAST - POINTS TO THE LAST AVAILABLE LAS ELEMENT.                SPCE0660
*        NPNT - POINTS TO THE FIRST LCB.                                 SPCE0670
*                                                                        SPCE0680
         ENTRY INITLP                                                    SPCE0690
         ENTRY RLSELP                                                    SPCE0700
         ENTRY OBTNLP                                                    SPCE0710
         ENTRY CONFLP                                                    SPCE0720
         ENTRY RTNLLP                                                    SPCE0730
         ENTRY RTNELP                                                    SPCE0740
         ENTRY PLASLP                                                    SPCE0750
         ENTRY TONSLP                                                    SPCE0760
         ENTRY TOPSLP                                                    SPCE0770
         ENTRY LASLP$                                                    SPCE0780
*                                                                        SPCE0790
GRL      EQU   15                  LINKING REGISTER.                     SPCE0800
GRR      EQU   14                  RETURN REGISTER.                      SPCE0810
GRB      EQU   2                   BASE REGISTER.                        SPCE0820
GRS      EQU   13                  SAVE REGISTER.                        SPCE0830
GR1      EQU   1                   ARGUMENT POINTER REGISTER.            SPCE0840
GR0      EQU   0                                                         SPCE0850
GR3      EQU   3                                                         SPCE0860
GR4      EQU   4                   WORK REGISTERS.                       SPCE0870
GR5      EQU   5                                                         SPCE0880
GR6      EQU   6                                                         SPCE0890
GR7      EQU   7                                                         SPCE0900
*                                                                        SPCE0910
         USING *,GRL                                                     SPCE0920
INITLP   B     CS11                                                      SPCE0930
         DC    XL1'07'                                                   SPCE0940
         DC    CL7'INITLP '                                              SPCE0950
CS11     STM   GRR,GR4,12(GRS)     SAVE CONTENTS OF GENERAL REGISTERS.   SPCE0960
         L     GRB,BASE                                                  SPCE0970
         USING SPACELP,GRB                                               SPCE0980
         DROP  GRL                                                       SPCE0990
         LR    GR3,GRS             SWITCH SAVE AREA POINTERS.            SPCE1000
         LA    GRS,SAREA                                                 SPCE1010
         ST    GR3,4(0,GRS)        SAVE PREVIOUS SAVE AREA POINTER.      SPCE1020
         ST    GRS,8(0,GR3)        SAVE PRESENT SAVE AREA POINTER.       SPCE1030
         TM    TRACE,X'80'         TEST FOR TRACE.                       SPCE1040
         BZ    TR1                 BRANCH IF NO TRACE.                   SPCE1050
         L     GRL,=V(INITL$)                                            SPCE1060
         BALR  GRR,GRL             TRACE.                                SPCE1070
TR1      TM    FLAGS,X'80'         TEST FOR PREVIOUS INITIALIZATION.     SPCE1080
         BO    RETRN1              IF SO THEN RETURN.                    SPCE1090
         L     GR3,0(0,GR1)                                              SPCE1100
         L     GR3,0(0,GR3)        LOAD NB.                              SPCE1110
         ST    GR3,BLKSIZ          STORE NB IN BLKSIZ.                   SPCE1120
         GETMAIN R,LV=(GR3)        ALLOCATE BLOCK FOR LAS.               SPCE1130
         ST    GR1,BLKADD          STORE ADDRESS OF BLOCK.               SPCE1140
         ST    GR1,LAS             STORE LAS POINTER.                    SPCE1150
         ST    GR1,LAST            STORE LAST POINTER.                   SPCE1160
         ST    GR3,0(0,GR1)        STORE LENGTH PORTION OF LAS ELEMENT,  SPCE1170
         L     GR4,NULL            ALSO ID = X'00'.                      SPCE1180
         ST    GR4,4(0,GR1)        INDICATE THAT THERE ARE NO OTHER      SPCE1190
         ST    GR4,NPNT            LAS ELEMENTS AND NO LCB MEMBERS.      SPCE1200
         LA    GR1,IPLSARGL                                              SPCE1210
         L     GRL,AINITPS                                               SPCE1220
         BALR  GRR,GRL             INITIALIZE PDS SPACE.                 SPCE1230
         MVI   FLAGS,X'80'         SET INITIALIZATION FLAG.              SPCE1240
RETRN1   L     GRS,SAREA+4                                               SPCE1250
         LM    GRR,GR3,12(GRS)     RESTORE GENERAL REGISTERS.            SPCE1260
         MVI   12(GRS),X'FF'                                             SPCE1270
         BR    GRR                 RETURN TO CALLING ROUTINE.            SPCE1280
*                                                                        SPCE1290
```

```
         DROP  GRB                                                      SPCE1300
         USING *,GRL                                                    SPCE1310
RLSELP   B     CS21                                                     SPCE1320
         DC    XL1'07'                                                  SPCE1330
         DC    CL7'RLSELP '                                             SPCE1340
CS21     STM   GRR,GR3,12(GRS)      SAVE CONTENTS OF GENERAL REGISTERS. SPCE1350
         L     GRB,BASE                                                 SPCE1360
         USING SPACELP,GRB                                              SPCE1370
         DROP  GRL                                                      SPCE1380
         LR    GR3,GRS              SWITCH SAVE AREA POINTERS.          SPCE1390
         LA    GRS,SAREA                                                SPCE1400
         ST    GR3,4(0,GRS)         SAVE PREVIOUS SAVE AREA POINTER.    SPCE1410
         ST    GRS,8(0,GR3)         SAVE PRESENT SAVE AREA POINTER.     SPCE1420
         TM    TRACE,X'80'          TEST FOR TRACE.                     SPCE1430
         BZ    TR2                  BRANCH IF NO TRACE.                 SPCE1440
         L     GRL,=V(RLSEL$)                                           SPCE1450
         BALR  GRR,GRL              TRACE.                              SPCE1460
TR2      TM    FLAGS,X'80'          TEST FOR PREVIOUS INITIALIZATION.   SPCE1470
         BO    CS22                 YES, CONTINUE.                      SPCE1480
         LA    GR1,ERR21A           NO, PRINT ERROR MESSAGE.            SPCE1490
         L     GRL,AEOUT                                                SPCE1500
         BALR  GRR,GRL                                                  SPCE1510
         B     ABND                 TERMINATE JOB.                      SPCE1520
CS22     L     GR0,BLKSIZ           LOAD LAS BLOCK SIZE.                SPCE1530
         L     GR1,BLKADD           LOAD LAS BLOCK ADDRESS.             SPCE1540
         FREEMAIN R,LV=(GR0),A=(GR1) RELEASE LAS BLOCK.                 SPCE1550
         L     GRL,ARLSEPS                                              SPCE1560
         BALR  GRR,GRL              RELEASE PDS SPACE.                  SPCE1570
         SR    GR1,GR1                                                  SPCE1580
         ST    GR1,FLAGS            ZERO FLAGS.                         SPCE1590
         B     RETRN1                                                   SPCE1600
*                                                                       SPCE1610
         DROP  GRB                                                      SPCE1620
         USING *,GRL                                                    SPCE1630
OBTNLP   B     CS31                                                     SPCE1640
         DC    XL1'07'                                                  SPCE1650
         DC    CL7'OBTNLP '                                             SPCE1660
CS31     STM   GRR,GR7,12(GRS)      SAVE CONTENTS OF GENERAL REGISTERS. SPCE1670
         L     GRB,BASE                                                 SPCE1680
         USING SPACELP,GRB                                              SPCE1690
         DROP  GRL                                                      SPCE1700
         LR    GR3,GRS              SWITCH SAVE AREA POINTERS.          SPCE1710
         LA    GRS,SAREA                                                SPCE1720
         ST    GR3,4(0,GRS)         SAVE PREVIOUS SAVE AREA POINTER.    SPCE1730
         ST    GRS,8(0,GR3)         SAVE PRESENT SAVE AREA POINTER.     SPCE1740
         TM    TRACE,X'80'          TEST FOR TRACE.                     SPCE1750
         BZ    TR3                  BRANCH IF NO TRACE.                 SPCE1760
         L     GRL,=V(OBTNL$)                                           SPCE1770
         BALR  GRR,GRL              TRACE.                              SPCE1780
TR3      L     GR3,0(0,GR1)                                             SPCE1790
         L     GR0,0(0,GR3)         LOAD L.                             SPCE1800
         LA    GR3,LAS              LOAD ADDRESS OF LAS.                SPCE1810
         L     GR4,LAS              LOAD LAS (ADD. OF FIRST LAS ELEMENT) SPCE1820
CS315    CL    GR4,NULL             TEST FOR A NULL POINTER.            SPCE1830
         BNE   CS32                 NO, CONTINUE.                       SPCE1840
         LA    GR7,1(0,0)           SET IFLAG TO INDICATE THAT THE LAS  SPCE1850
         L     GR6,8(0,GR1)         DOES NOT CONTAIN AN ELEMENT WHICH   SPCE1860
         ST    GR7,0(0,GR6)         IS MADE UP OF AT LEAST L BYTES.     SPCE1870
         B     RETRN                                                    SPCE1880
CS32     TM    0(GR4),X'40'         CHECK ID OF LAS ELEMENT.            SPCE1890
         BZ    CS325                ID=X'00', CONTINUE.                 SPCE1900
         BAL   GR7,HLIST            LAS ELEMENT POINTS TO LIST, HANDLE. SPCE1910
CS325    L     GR5,4(0,GR4)         LOAD ADD. OF NEXT LAS ELEMENT.      SPCE1920
         L     GR6,0(0,GR4)         LOAD L1, LENGTH OF LAS ELEMENT.     SPCE1930
         CLR   GR0,GR6              COMPARE L TO L1.                    SPCE1940
```

```
         BL    CS34              L LESS THAN L1.                          SPCE1950
         BE    CS33              L EQUAL L1.                              SPCE1960
         LA    GR3,4(0,GR4)      L GREATER THAN L1, LOOK AT NEXT          SPCE1970
         LR    GR4,GR5           LAS ELEMENT.                             SPCE1980
         B     CS315                                                      SPCE1990
CS33     LR    GR7,GR5           LOAD ADDRESS OF NEXT LAS ELEMENT.        SPCE2000
         B     CS35                                                       SPCE2010
CS34     LR    GR7,GR4           COMPUTE NEW ADDRESS OF LAS ELEMENT,      SPCE2020
         ALR   GR7,GR0           (OLD ADDRESS + L).                       SPCE2030
         SLR   GR6,GR0           COMPUTE NEW LENGTH (L1 - L).             SPCE2040
         ST    GR6,0(0,GR7)      STORE NEW ELEMENT LENGTH (ID=X'00').     SPCE2050
         ST    GR5,4(0,GR7)      STORE ADD. OF NEXT LAS ELEMENT.          SPCE2060
CS35     ST    GR7,0(0,GR3)      UPDATE PREVIOUS LAS ELEMENT POINTER.     SPCE2070
         CL    GR4,LAST                                                   SPCE2080
         BNE   CS36                                                       SPCE2090
         ST    GR7,LAST          UPDATE LAST.                             SPCE2100
CS36     L     GR6,4(0,GR1)                                               SPCE2110
         ST    GR4,0(0,GR6)      STORE IADD.                              SPCE2120
         SR    GR7,GR7           SET IFLAG TO INDICATE THAT A LAS         SPCE2130
         L     GR6,8(0,GR1)      ELEMENT OF LENGTH L HAS BEEN             SPCE2140
         ST    GR7,0(0,GR6)      OBTAINED.                                SPCE2150
         B     RETRN                                                      SPCE2160
*                                                                         SPCE2170
*        THIS IS THE SECTION OF CODING WHICH HANDLES THE DECISION         SPCE2180
*        OF WHETHER OR NOT TO RETURN SPACE USED BY A LIST TO THE LAS.     SPCE2190
*        THIS SECTION IS ENTERED IF AN LAS ELEMENT WITH AN ID NOT         SPCE2200
*        EQUAL TO X'00' IS ENCOUNTERED.  THE ID OF THE ENCOUNTERED        SPCE2210
*        ELEMENT IS CHANGED TO X'00' ON ENTRY TO THIS SECTION SO THAT     SPCE2220
*        ANY LAS ELEMENT CAN CAUSE A BRANCH TO THIS SECTION ONLY ONCE.    SPCE2230
*        THE LIST CONTROL BLOCK IS EXAMINED TO SEE IF THE LIST SPACE      SPCE2240
*        IS TO BE RETURNED TO THE LAS.  A LCB HAS THE FOLLOWING FORMAT.   SPCE2250
*              ID = X'80' (BYTE 1),                                       SPCE2260
*              LCB ELEMENT LENGTH (BYTES 2-4),                            SPCE2270
*              P1 - POINTS TO THE FIRST LIST ELEMENT (BYTES 5-8),         SPCE2280
*              P2 - POINTS TO THE LAST LIST ELEMENT (BYTES 9-12),         SPCE2290
*              MARK - USED TO INDICATE (MARK = X'80') THAT A LIST         SPCE2300
*                     CAN ONLY BE RETURNED TO THE LAS BY USE OF           SPCE2310
*                     A DELETE COMMAND (BYTE 12),                         SPCE2320
*              COUNT - INDICATES THE NUMBER OF TIMES THE LIST IS USED     SPCE2330
*                      AS A SUBLIST (BYTES 13-16),                        SPCE2340
*              NAME - LIST NAME (BYTES 17-20),                            SPCE2350
*              P4 - POINTS TO THE PREVIOUS LCB, P4 IS NULL IF LCB IS      SPCE2360
*                   THE FIRST LCB (BYTES 21-24),                          SPCE2370
*              P5 - POINTS TO THE NEXT LCB, P5 IS NULL IF LCB IS          SPCE2380
*                   THE LAST LCB (BYTES 25-28).                           SPCE2390
*        ON ENTRY TO THIS SECTION GR4 CONTAINS THE ADDRESS OF THE LAS     SPCE2400
*        ELEMENT WHICH HAS AN ID NOT EQUAL TO X'00' AND GR7 CONTAINS      SPCE2410
*        THE RETURN ADDRESS.                                              SPCE2420
*                                                                         SPCE2430
HLIST    STM   GR1,GR7,SAVE7                                              SPCE2440
         L     GR3,12(0,GR4)     LOAD ADD. OF LCB POINTED TO.             SPCE2450
         MVI   0(GR4),X'00'      CHANGE ID OF ELEMENT WHICH POINTS        SPCE2460
         CL    GR3,NULL          TO LCB.  TEST FOR NULL POINTER.          SPCE2470
         BE    RETRNH            YES, BRANCH.                             SPCE2480
         TM    0(GR3),X'80'      SEE IF ELEMENT POINTED TO                SPCE2490
         BZ    RETRNH            IS REALLY AN LCB, IF NOT RETURN.         SPCE2500
         TM    12(GR3),X'80'     TEST LCB MARK.                           SPCE2510
         BO    RETRNH            RETURN IF MARK IS SET.                   SPCE2520
         L     GR5,12(0,GR3)     LOAD LCB COUNT.                          SPCE2530
         S     GR5,ONE           COUNT = COUNT - 1.                       SPCE2540
         BP    H4                BRANCH IF COUNT IS POSITIVE.             SPCE2550
H1       L     GR1,LAST          LOAD LAST.                               SPCE2560
         CL    GR1,NULL          SEE IF LAST = NULL.                      SPCE2570
         BE    H15               YES, BRANCH.                             SPCE2580
         ST    GR3,4(0,GR1)      UPDATE PREVIOUS LAST LAS ELEMENT BY      SPCE2590
```

```
H15      L      GR1,8(0,GR3)        MAKING P1 POINT TO THE LCB.          SPCE2600
         L      GR7,NULL            UPDATE NEW LAST LAS ELEMENT BY       SPCE2610
         ST     GR7,4(0,GR1)        MAKING P1 OF THE LAST ELEMENT OF THE SPCE2620
         MVI    0(GR3),X'00'        LIST NULL.  CHANGE ID OF LCB -X'00'. SPCE2630
         ST     GR1,LAST            UPDATE LAST.                         SPCE2640
         L      GR1,LAS                                                  SPCE2650
         CL     GR1,NULL            SEE IF LAS IS NULL.                  SPCE2660
         BNE    H17                 NO, BRANCH.                          SPCE2670
         ST     GR3,LAS             UPDATE LAS.                          SPCE2680
H17      L      GR1,16(0,GR3)       LOAD LIST NAME.                      SPCE2690
         ST     GR1,NAME            STORE NAME.                          SPCE2700
         LA     GR1,NMARGL                                               SPCE2710
         L      GRL,ADELTPS         DELETE PDS CALLED NAME               SPCE2720
         BALR   GRR,GRL                                                  SPCE2730
         L      GR1,20(0,GR3)       LOAD P4 OF THE LCB.                  SPCE2740
         L      GR6,24(0,GR3)       LOAD P5 OF THE LCB.                  SPCE2750
         CLR    GR1,GR7             SEE IF P4 IS NULL.                   SPCE2760
         BE     H2                  YES, BRANCH.                         SPCE2770
         ST     GR6,24(0,GR1)       UPDATE P5 OF LCB POINTED TO BY P4.   SPCE2780
         B      H3                                                       SPCE2790
H2       ST     GR6,NPNT            UPDATE NPNT.                         SPCE2800
H3       CLR    GR6,GR7             SEE IF P5 IS NULL.                   SPCE2810
         BE     RETRNH              YES, BRANCH.                         SPCE2820
         ST     GR1,20(0,GR6)       UPDATE P4 OF LCB POINTED TO BY P5.   SPCE2830
         B      RETRNH                                                   SPCE2840
H4       ST     GR5,12(0,GR3)       UPDATE LCB COUNT.                    SPCE2850
RETRNH   LM     GR1,GR7,SAVE7                                            SPCE2860
         BR     GR7                 RETURN.                              SPCE2870
*                                                                        SPCE2880
         DROP   GRB                                                      SPCE2890
         USING  *,GRL                                                    SPCE2900
CONFLP   B      CS41                                                     SPCE2910
         DC     XL1'07'                                                  SPCE2920
         DC     CL7'CONFLP '                                             SPCE2930
CS41     STM    GRR,GR7,12(GRS)     SAVE CONTENTS OF GENERAL REGISTERS.  SPCE2940
         L      GRB,BASE                                                 SPCE2950
         USING  SPACELP,GRB                                              SPCE2960
         DROP   GRL                                                      SPCE2970
         LR     GR3,GRS             SWITCH SAVE AREA POINTERS.           SPCE2980
         LA     GRS,SAREA                                                SPCE2990
         ST     GR3,4(0,GRS)        SAVE PREVIOUS SAVE AREA POINTER.     SPCE3000
         ST     GRS,8(0,GR3)        SAVE PRESENT SAVE AREA POINTER.      SPCE3010
         TM     TRACE,X'80'         TEST FOR TRACE.                      SPCE3020
         BZ     TR4                 BRANCH IF NO TRACE.                  SPCE3030
         L      GRL,=V(CONFL$)                                           SPCE3040
         BALR   GRR,GRL             TRACE.                               SPCE3050
TR4      SR     GR0,GR0             ZERO REGISTER.                       SPCE3060
         ST     GR0,ICNT            STORE 0 IN ICNT.                     SPCE3070
         L      GR4,LAS             LOAD ADD. OF FIRST LAS ELEMENT, PNT1 SPCE3080
CS415    CL     GR4,NULL            TEST FOR NULL POINTER.               SPCE3090
         BNE    CS42                                                     SPCE3100
         L      GR7,ICNT                                                 SPCE3110
         L      GR6,0(0,GR1)        RETURN THE VALUE OF ICNT.            SPCE3120
         ST     GR7,0(0,GR6)                                             SPCE3130
         B      RETRN                                                    SPCE3140
CS42     TM     0(GR4),X'40'        SEE IF ID = X'00'.                   SPCE3150
         BZ     CS43                YES, CONTINUE.                       SPCE3160
         BAL    GR7,HLIST           HANDLE LIST POINTED TO.              SPCE3170
CS43     LR     GR5,GR4             COMPUTE THE LAS ELEMENT ADD. TO BE   SPCE3180
         AL     GR5,0(0,GR4)        SEARCHED FOR, ADDS.                  SPCE3190
CS44     L      GR6,ALASM4          LOAD ADD OF LAS - 4, PNT2.           SPCE3200
         L      GR7,LAS             LOAD LAS, PNT3.                      SPCE3210
CS45     CL     GR7,NULL            SEE IF PNT3 IS NULL.                 SPCE3220
         BNE    CS46                NO, CONTINUE.                        SPCE3230
         L      GR4,4(0,GR4)        YES, UPDATE PNT1.                    SPCE3240
```

```
         B     CS415                                                  SPCE3250
CS46     TM    0(GR7),X'40'      TEST ID OF ELEM. POINTED TO BY PNT3. SPCE3260
         BZ    CS47              ID = X'00', CONTINUE.                SPCE3270
         ST    GR4,PNT1                                               SPCE3280
         LR    GR4,GR7           ID NOT X'00', HANDLE LIST POINTED    SPCE3290
         BAL   GR7,HLIST         TO.                                  SPCE3300
         LR    GR7,GR4                                                SPCE3310
         L     GR4,PNT1                                               SPCE3320
CS47     CLR   GR7,GR5           SEE IF PNT3 = ADDS.                  SPCE3330
         BE    CS48              YES, BRANCH.                         SPCE3340
         LR    GR6,GR7           NO, UPDATE PNT2.                     SPCE3350
         L     GR7,4(0,GR6)      UPDATE PNT3.                         SPCE3360
         B     CS45                                                   SPCE3370
CS48     L     GR0,ICNT          *** CONNECT TWO LAS ELEMENTS ***.    SPCE3380
         AL    GR0,ONE           ADD 1 TO ICNT.                       SPCE3390
         ST    GR0,ICNT                                               SPCE3400
         L     GR0,0(0,GR7)      UPDATE LENGTH OF LAS ELEMENT POINTED SPCE3410
         AL    GR0,0(0,GR4)      TO BY PNT1.  NEW LENGTH = OLD LENGTH SPCE3420
         ST    GR0,0(0,GR4)      + LEN. OF ELEM. POINTED TO BY PNT3.  SPCE3430
         AL    GR5,0(0,GR7)      UPDATE ADDS.                         SPCE3440
         L     GR0,4(0,GR7)                                           SPCE3450
         ST    GR0,4(0,GR6)      UPDATE PNT2.                         SPCE3460
         CL    GR7,LAST          SEE IF PNT3 = LAST.                  SPCE3470
         BNE   CS44              NO, BRANCH.                          SPCE3480
         ST    GR6,LAST          YES, UPDATE LAST, LAST = PNT2.       SPCE3490
         B     CS44                                                   SPCE3500
*                                                                     SPCE3510
         DROP  GRB                                                    SPCE3520
         USING *,GRL                                                  SPCE3530
RTNLLP   B     CS51                                                   SPCE3540
         DC    XL1'07'                                                SPCE3550
         DC    CL7'RTNLLP '                                           SPCE3560
CS51     STM   GRR,GR7,12(GRS)   SAVE CONTENTS OF GENERAL REGISTERS.  SPCE3570
         L     GRB,BASE                                               SPCE3580
         USING SPACELP,GRB                                            SPCE3590
         DROP  GRL                                                    SPCE3600
         LR    GR3,GRS           SWITCH SAVE AREA POINTERS.           SPCE3610
         LA    GRS,SAREA                                              SPCE3620
         ST    GR3,4(0,GRS)      SAVE PREVIOUS SAVE AREA POINTER.     SPCE3630
         ST    GRS,8(0,GR3)      SAVE PRESENT SAVE AREA POINTER.      SPCE3640
         TM    TRACE,X'80'       TEST FOR TRACE.                      SPCE3650
         BZ    TR5               BRANCH IF NO TRACE.                  SPCE3660
         L     GRL,=V(RTNLL$)                                         SPCE3670
         BALR  GRR,GRL           TRACE.                               SPCE3680
TR5      L     GR3,0(0,GR1)                                           SPCE3690
         L     GR3,0(0,GR3)      LOAD IADD.  MAKE SURE ELEMENT        SPCE3700
         TM    0(GR3),X'80'      POINTED TO BY IADD IS A LCB.         SPCE3710
         BZ    RETRN             IF NOT RETURN.                       SPCE3720
         LA    GR7,RETRN         IF SO RETURN LIST SPACE TO IAS.      SPCE3730
         STM   GR1,GR7,SAVE7                                          SPCE3740
         B     H1                                                     SPCE3750
*                                                                     SPCE3760
         DROP  GRB                                                    SPCE3770
         USING *,GRL                                                  SPCE3780
RTNELP   B     CS61                                                   SPCE3790
         DC    XL1'07'                                                SPCE3800
         DC    CL7'RTNELP '                                           SPCE3810
CS61     STM   GRR,GR3,12(GRS)   SAVE CONTENTS OF GENERAL REGISTERS.  SPCE3820
         L     GRB,BASE                                               SPCE3830
         USING SPACELP,GRB                                            SPCE3840
         DROP  GRL                                                    SPCE3850
         LR    GR3,GRS           SWITCH SAVE AREA POINTERS.           SPCE3860
         LA    GRS,SAREA                                              SPCE3870
         ST    GR3,4(0,GRS)      SAVE PREVIOUS SAVE AREA POINTER.     SPCE3880
         ST    GRS,8(0,GR3)      SAVE PRESENT SAVE AREA POINTER.      SPCE3890
```

```
         TM    TRACE,X'80'        TEST FOR TRACE.                          SPCE3900
         BZ    TR6                BRANCH IF NO TRACE.                      SPCE3910
         L     GRL,=V(RTNEL$)                                              SPCE3920
         BALR  GRR,GRL            TRACE.                                   SPCE3930
TR6      L     GR3,0(0,GR1)                                                SPCE3940
         L     GR3,0(0,GR3)       LOAD IADD.  MAKE SURE ELEMENT            SPCE3950
         TM    0(GR3),X'80'       POINTED TO BY IADD IS NOT A LCB.         SPCE3960
         BO    RETRN1             IF SO RETURN.                            SPCE3970
         L     GR1,LAST           IF NOT, RETURN ELEMENT SPACE TO LAS.     SPCE3980
         CL    GR1,NULL           SEE IF LAST = NULL.                      SPCE3990
         BE    CS62               YES, BRANCH.                             SPCE4000
         ST    GR3,4(0,GR1)       UPDATE P1 OF PREVICUS LAST ELEMENT.      SPCE4010
CS62     ST    GR3,LAST           UPDATE LAST.                             SPCE4020
         L     GR1,LAS            LOAD LAS.                                SPCE4030
         CL    GR1,NULL           SEE IF LAS IS NULL.                      SPCE4040
         BNE   CS63               NO, BRANCH.                              SPCE4050
         ST    GR3,LAS            UPDATE LAS.                              SPCE4060
CS63     L     GR1,NULL                                                    SPCE4070
         ST    GR1,4(0,GR3)       UPDATE P1 OF NEW LAST ELEMENT.           SPCE4080
         B     RETRN1                                                      SPCE4090
*                                                                          SPCE4100
         DROP  GRB                                                        SPCE4110
         USING *,GRL                                                      SPCE4120
PLASLP   B     CS71                                                        SPCE4130
         DC    XL1'07'                                                     SPCE4140
         DC    CL7'PLASLP '                                                SPCE4150
CS71     STM   GRR,GR7,12(GRS)    SAVE CONTENTS OF GENERAL REGISTERS.      SPCE4160
         L     GRB,BASE                                                    SPCE4170
         USING SPACELP,GRB                                                SPCE4180
         DROP  GRL                                                        SPCE4190
         LR    GR3,GRS            SWITCH SAVE AREA POINTERS.               SPCE4200
         LA    GRS,SAREA                                                   SPCE4210
         ST    GR3,4(0,GRS)       SAVE PREVIOUS SAVE AREA POINTER.         SPCE4220
         ST    GRS,8(0,GR3)       SAVE PRESENT SAVE AREA POINTER.          SPCE4230
         L     GR6,0(0,GR1)       LOAD ADDRESS OF IADD.                    SPCE4240
         L     GR3,0(0,GR6)       LOAD IADD.                               SPCE4250
         LTR   GR3,GR3            SEE IF IADD = 0.                         SPCE4260
         BE    CS72               YES, BRANCH.                             SPCE4270
         L     GR5,0(0,GR3)       IADD NOT 0, RETURN THE ID AND THE        SPCE4280
         L     GR7,4(0,GR1)       LENGTH OF THE ELEMENT POINTED TO         SPCE4290
         ST    GR5,0(0,GR7)       BY IADD IN ILEN.                         SPCE4300
         L     GR4,4(0,GR3)       UPDATE IADD.                             SPCE4310
         ST    GR4,0(0,GR6)                                                SPCE4320
         CL    GR4,NULL           SEE IF UPDATED IADD IS NULL.             SPCE4330
         BNE   CS73               NO, BRANCH.                              SPCE4340
         LA    GR0,1(0,0)         SET IFLAG TO INDICATE THAT THE           SPCE4350
         L     GR7,8(0,GR1)       UPDATED IADD CONTAINS A NULL             SPCE4360
         ST    GR0,0(0,GR7)       POINTER.                                 SPCE4370
         B     RETRN                                                       SPCE4380
CS72     L     GR4,LAS            LOAD LAS.                                SPCE4390
         ST    GR4,0(0,GR6)       STORE LAS INTO IADD.                     SPCE4400
         CL    GR4,NULL           SEE IF LAS IS EMPTY.                     SPCE4410
         BNE   CS73               NO, BRANCH.                              SPCE4420
         LA    GR0,2(0,0)         SET IFLAG TO INDICATE THAT               SPCE4430
         L     GR7,8(0,GR1)       THE LAS IS EMPTY.                        SPCE4440
         ST    GR0,0(0,GR7)                                                SPCE4450
         B     RETRN                                                       SPCE4460
CS73     SR    GR0,GR0            SET IFLAG TO INDICATE THAT IADD HAS      SPCE4470
         L     GR7,8(0,GR1)       BEEN UPDATED SUCCESSFULLY AND THAT       SPCE4480
         ST    GR0,0(0,GR7)       ILEN HAS BEEN STORED INTO (IF IADD       SPCE4490
         B     RETRN              WAS NOT EQUAL TO 0 ON ENTRY).            SPCE4500
*                                                                          SPCE4510
ABND     ABEND 11,DUMP                                                    SPCE4520
*                                                                          SPCE4530
RETRN    L     GRS,SAREA+4                                                 SPCE4540
```

```
         LM    GRR,GR7,12(GRS)     RESTORE GENERAL REGISTERS.      SPCE4550
         MVI   12(GRS),X'FF'                                       SPCE4560
         BR    GRR                 RETURN TO CALLING ROUTINE.      SPCE4570
*                                                                  SPCE4580
         DROP  GRB                                                 SPCE4590
         USING *,GRL                                               SPCE4600
TONSLP   MVI   TRACE,X'80'         TURN TRACE FLAG ON.             SPCE4610
         BR    GRR                                                 SPCE4620
*                                                                  SPCE4630
         USING *,GRL                                               SPCE4640
TOFSLP   MVI   TRACE,X'00'         TURN TRACE FLAG OFF.            SPCE4650
         BR    GRR                                                 SPCE4660
*                                                                  SPCE4670
         CNOP  0,4                                                 SPCE4680
LASLP$   DS    0F                                                  SPCE4690
NPNT     DS    1F                                                  SPCE4700
LAS      DS    1F                                                  SPCE4710
LAST     DS    1F                                                  SPCE4720
BLKSIZ   DS    1F                                                  SPCE4730
BLKADD   DS    1F                                                  SPCE4740
BASE     DC    A(SPACELP)                                          SPCE4750
SAREA    DS    18F                                                 SPCE4760
ONE      DC    F'1'                                                SPCE4770
FLAGS    DC    F'0'                                                SPCE4780
NULL     DC    CL4'$$$$'                                           SPCE4790
NPELEMTS DC    F'1000'             NUMBER OF PDS ELEMENTS.         SPCE4800
NPDS     DC    F'50'               NUMBER OF STACKS.               SPCE4810
IPDSARGL DC    A(NPELEMTS)                                         SPCE4820
         DC    XL1'80'                                             SPCE4830
         DC    AL3(NPDS)                                           SPCE4840
AINITPS  DC    V(INITPS)                                           SPCE4850
ARLSEPS  DC    V(RLSEPS)                                           SPCE4860
NMARGL   DC    A(NAME)                                             SPCE4870
         DC    XL1'80'                                             SPCE4880
         DC    AL3(DUMMY)                                          SPCE4890
ADELTPS  DC    V(DELTPS)                                           SPCE4900
NAME     DS    1F                                                  SPCE4910
DUMMY    DS    1F                                                  SPCE4920
SAVE7    DS    7F                                                  SPCE4930
AEOUT    DC    V(MESSP$)                                           SPCE4940
ERR21A   DC    XL1'80'                                             SPCE4950
         DC    AL3(ERR21)                                          SPCE4960
ERR21    DC    F'1'                                                SPCE4970
ICNT     DS    1F                                                  SPCE4980
PNT1     DS    1F                                                  SPCE4990
ALASM4   DC    A(LAS-4)                                            SPCE5000
TRACE    DC    XL1'00'                                             SPCE5010
         END                                                       SPCE5020
```

```
TRVS     TITLE 'TRAVSLP --- LIST TRAVERSAL ROUTINE'                    TRVS0010
TRAVSLP  CSECT                                                         TRVS0020
*                                                                      TRVS0030
*        ROUTINE WHICH IS USED FOR TRAVERSING LISTS, MOVING DATA TO    TRVS0040
*        OR FROM USER DATA FIELDS, AND EXTRACTING CONTROL INFORMATION  TRVS0050
*        FROM LIST ELEMENTS AND LCBS.  ENTRY POINTS AND ARGUMENTS      TRVS0060
*        ARE ...                                                       TRVS0070
*                                                                      TRVS0080
*        1.  CALL TRVLLP(NAME,P,IADD,IFLAG)                            TRVS0090
*              TRAVERSES A LIST STRUCTURE (I.E. IF A LISTR IS          TRVS0100
*              ENCOUNTERED THEN THE LIST POINTED TO IS TRAVERSED).     TRVS0110
*              NAME TELLS WHICH LIST IS BEING TRAVERSED.  P=1 FOR      TRVS0120
*              TRAVERSAL USING P1, P=2 FOR TRAVERSAL USING P2.  ON     TRVS0130
*              ENTRY IADD POINTS TO A LIST ELEMENT, ON EXIT IADD IS    TRVS0140
*              UPDATED TO POINT TO THE NEXT ELEMENT.  IFLAG IS SET TO  TRVS0150
*              0 FOR NORMAL RETURN, 1 IF PDS CALLED NAME NOT FOUND,    TRVS0160
*              2 IF PDS CALLED NAME IS EMPTY, AND 3 IF THE LAS FOR PDS TRVS0170
*              ELEMENTS IS EMPTY.                                      TRVS0180
*                                                                      TRVS0190
*        2.  CALL TRVELP(P,IADD,IFLAG)                                 TRVS0200
*              TRAVERSES A LIST (I.E. SUB-LISTS ARE NOT TRAVERSED).    TRVS0210
*              P=1 FOR TRAVERSAL USING P1, P=2 FOR TRAVERSAL USING P2. TRVS0220
*              ON ENTRY IADD POINTS TO A LIST ELEMENT, ON EXIT IADD IS TRVS0230
*              UPDATED TO POINT TO THE PREVIOUS ELEMENT (P=2) OR THE   TRVS0240
*              NEXT ELEMENT (P=1).  IFLAG IS SET ACCORDING TO THE ID   TRVS0250
*              OF THE ELEMENT POINTED TO BY IADD ON EXIT.              TRVS0260
*                   ID = X'00'  ---  IFLAG = 0,                        TRVS0270
*                   ID = X'4-'  ---  IFLAG = 1,                        TRVS0280
*                   ID = X'80'  ---  IFLAG = 2,                        TRVS0290
*                                                                      TRVS0300
*        3.  IADD = IANMLP(NAME)                                       TRVS0310
*              IF NAME EQUALS 0, THEN IADD IS RETURNED WITH THE        TRVS0320
*              ADDRESS OF THE FIRST LCB ELEMENT, 0 IF THERE ARE NO     TRVS0330
*              LCB ELEMENTS.  IF NAME NOT EQUAL TO 0, THEN IADD        TRVS0340
*              IS RETURNED WITH THE ADDRESS OF THE LCE FOR THE LIST    TRVS0350
*              CALLED NAME, 0 IF THERE IS NO LIST CALLED NAME.         TRVS0360
*                                                                      TRVS0370
*        4.  IPNT = IPNTLP(IADD,IP)                                    TRVS0380
*              IPNT IS RETURNED WITH THE CONTENTS OF POINTER IP OF THE TRVS0390
*              ELEMENT POINTED TO BY IADD IF IP IS VALID FOR THE       TRVS0400
*              ELEMENT POINTED TO.  IPNT IS RETURNED WITH 0 IF IP IS   TRVS0410
*              NOT VALID.  VALID IP VALUES ARE ...                     TRVS0420
*                   ELEMENT POINTED TO       VALID IP VALUES          TRVS0430
*                        ALL ELEMENTS             1,2                 TRVS0440
*                        LISTR                    1,2,3               TRVS0450
*                        LCB                      1,2,4,5             TRVS0460
*                                                                      TRVS0470
*        5.  ID = IIDLP(IADD)                                          TRVS0480
*              ID OF ELEMENT POINTED TO BY IADD IS RETURNED AS ...     TRVS0490
*                   ELEMENT ID               RETURNED ID VALUE        TRVS0500
*                        X'00'                    0                   TRVS0510
*                        X'40'                    4                   TRVS0520
*                        X'41'                    5                   TRVS0530
*                        X'80'                    8                   TRVS0540
*                                                                      TRVS0550
*        6.  LENGTH = ILENLP(IADD)                                     TRVS0560
*              LENGTH IS RETURNED WITH THE LENGTH OF THE USER DATA     TRVS0570
*              FIELD IN THE ELEMENT POINTED TO BY IADD.                TRVS0580
*                                                                      TRVS0590
*        7.  JADD = IUADLP(IADD)                                       TRVS0600
*              JADD IS RETURNED WITH THE ADDRESS OF THE USER DATA      TRVS0610
*              FIELD IN THE ELEMENT POINTED TO BY IADD, 0 IF THERE     TRVS0620
*              IS NO USER DATA FIELD IN THE ELEMENT.                   TRVS0630
*                                                                      TRVS0640
```

```
         ENTRY TRVLLP                                                   TRVS0650
         ENTRY TRVELP                                                   TRVS0660
         ENTRY IANMLP                                                   TRVS0670
         ENTRY IPNTLP                                                   TRVS0680
         ENTRY IIDLP                                                    TRVS0690
         ENTRY ILENLP                                                   TRVS0700
         ENTRY IUADLP                                                   TRVS0710
*                                                                       TRVS0720
GRL      EQU   15                 LINKING REGISTER.                     TRVS0730
GRR      EQU   14                 RETURN REGISTER.                      TRVS0740
GRB      EQU   2                  BASE REGISTER.                        TRVS0750
GRS      EQU   13                 SAVE REGISTER.                        TRVS0760
GR1      EQU   1                  ARGUMENT POINTER REGISTER.            TRVS0770
GR0      EQU   0                                                        TRVS0780
GR2      EQU   2                                                        TRVS0790
GR3      EQU   3                                                        TRVS0800
GR4      EQU   4                  WORK REGISTERS.                       TRVS0810
GR5      EQU   5                                                        TRVS0820
GR6      EQU   6                                                        TRVS0830
*                                                                       TRVS0840
         USING *,GRL                                                    TRVS0850
TRVLLP   B     CS11                                                     TRVS0860
         DC    XL1'07'                                                  TRVS0870
         DC    CL7'TRVLLP '                                             TRVS0880
CS11     STM   GRR,GR6,12(GRS)    SAVE CONTENTS OF GENERAL REGISTERS.   TRVS0890
         L     GRB,BASE                                                 TRVS0900
         USING TRAVSLP,GRB                                              TRVS0910
         DROP  GRL                                                      TRVS0920
         LR    GR3,GRS            SWITCH SAVE AREA POINTERS.            TRVS0930
         LA    GRS,SAREA                                                TRVS0940
         ST    GR3,4(0,GRS)       SAVE PREVIOUS SAVE AREA POINTER.      TRVS0950
         ST    GRS,8(0,GR3)       SAVE PRESENT SAVE AREA POINTER.       TRVS0960
         LM    GR3,GR6,0(GR1)                                           TRVS0970
         L     GR3,0(0,GR3)       LOAD NAME.                            TRVS0980
         ST    GR3,NAME           STORE NAME.                           TRVS0990
         LR    GR3,GR5            LOAD ADD. OF IADD.                    TRVS1000
         L     GR5,0(0,GR3)       JADD = IADD.                          TRVS1010
         L     GR4,0(0,GR4)       LOAD P.                               TRVS1020
         SLL   GR4,2(0)           PNT1 = P*4.                           TRVS1030
         TM    0(GR5),X'40'       SEE IF JADD POINTS TO A LISTR.        TRVS1040
         BO    CS15               YES, BRANCH.                          TRVS1050
CS115    L     GR5,0(GR4,GR5)     JADD = CONTENTS OF (JADD + PNT1).     TRVS1060
         TM    0(GR5),X'80'       SEE IF JADD POINTS TO A LCB.          TRVS1070
         BO    CS13               YES, BRANCH.                          TRVS1080
         TM    0(GR5),X'40'       SEE IF JADD POINTS TO A LISTR.        TRVS1090
         BZ    CS12               NO, BRANCH.                           TRVS1100
         TM    0(GR5),X'01'       DOES LISTR HAVE A USER DATA FIELD.    TRVS1110
         BZ    CS15               NO, BRANCH.                           TRVS1120
CS12     ST    GR5,0(0,GR3)       IADD = JADD.                          TRVS1130
         SR    GR0,GR0            SET IFLAG TO INDICATE NORMAL RETURN.  TRVS1140
         ST    GR0,0(0,GR6)                                             TRVS1150
         B     RETRN1                                                   TRVS1160
CS13     LA    GR1,LPANDP         LOAD ADD. OF ARG. LIST.               TRVS1170
         L     GRL,APOPPS                                               TRVS1180
         BALR  GRR,GRL            BRANCH TO POPPS.                      TRVS1190
         CLI   KFLAG+3,X'00'      SEE IF KFLAG = 0.                     TRVS1200
         BNE   CS14               NO, BRANCH.                           TRVS1210
         L     GR5,KADD           JADD = KADD.                          TRVS1220
         B     CS115                                                    TRVS1230
CS14     CLI   KFLAG+3,X'01'      SEE IF KFLAG = 1.                     TRVS1240
         BE    CS16               YES, BRANCH.                          TRVS1250
         L     GR5,0(0,GR3)       IADD = JADD.                          TRVS1260
         LA    GR0,2(0,0)         SET IFLAG TO INDICATE THAT EDS        TRVS1270
         ST    GR0,0(0,GR6)       CALLED NAME IS EMPTY.                 TRVS1280
         B     RETRN1                                                   TRVS1290
```

```
CS15     L      GR1,12(0,GR5)      LOAD P3.                                TRVS1300
         CL     GR1,NULL           SEE IF P3 = NULL.                       TRVS1310
         BE     CS115              YES, BRANCH.                            TRVS1320
         ST     GR5,KADD           KADD = JADD.                            TRVS1330
         LA     GR1,LPANDP         LOAD ADD. OF ARG. LIST.                 TRVS1340
         L      GRL,APUSHPS                                                TRVS1350
         BALR   GRR,GRL            BRANCH TO PUSHPS.                       TRVS1360
         CLI    KFLAG+3,X'00'      SEE IF KFLAG = 0.                       TRVS1370
         BNE    CS155              NO, BRANCH.                             TRVS1380
         L      GR5,12(0,GR5)      JADD = CONTENTS OF (JADD + 12).         TRVS1390
         B      CS115                                                      TRVS1400
CS155    CLI    KFLAG+3,X'01'      SEE IF KFLAG = 1.                       TRVS1410
         BE     CS16               YES, BRANCH.                            TRVS1420
         LA     GR0,3(0,0)         SET IFLAG TO INDICATE THAT THE LAS      TRVS1430
         ST     GR0,0(0,GR6)       FOR THE PDS ELEMENTS IS EMPTY.          TRVS1440
         B      RETRN1                                                     TRVS1450
CS16     LA     GR0,1(0,0)         SET IFLAG TO INDICATE THAT A PDS        TRVS1460
         ST     GR0,0(0,GR6)       CALLED NAME IS NOT DEFINED.             TRVS1470
RETRN1   L      GRS,SAREA+4                                                TRVS1480
         LM     GRR,GR6,12(GRS)    RESTORE GENERAL REGISTERS.              TRVS1490
         MVI    12(GRS),X'FF'                                              TRVS1500
         BR     GRR                RETURN                                  TRVS1510
*                                                                          TRVS1520
         DROP   GRB                                                        TRVS1530
         USING  *,GRL              SET UP BASE REGISTER.                   TRVS1540
TRVELP   B      CS21                                                       TRVS1550
         DC     XL1'07'                                                    TRVS1560
         DC     CL7'TRVELP '                                               TRVS1570
CS21     STM    GRR,GR3,12(GRS)    SAVE GENERAL REGISTERS.                 TRVS1580
         LM     GR1,GR3,0(GR1)                                             TRVS1590
         L      GR1,0(0,GR1)       LOAD P.                                 TRVS1600
         SLL    GR1,2(0)           PNT1 = P*4.                             TRVS1610
         L      GRR,0(0,GR2)       JADD = IADD.                            TRVS1620
         L      GRR,0(GR1,GRR)     JADD = CONTENTS OF (JADD + PNT1).       TRVS1630
         ST     GRR,0(0,GR2)       IADD = JADD                             TRVS1640
         SR     GR0,GR0                                                    TRVS1650
         IC     GR0,0(0,GRR)       INSERT ID OF ELEM. POINTED TO BY        TRVS1660
         SRL    GR0,6(0)           JADD THEN DIVIDE BY FOUR.               TRVS1670
         ST     GR0,0(0,GR3)       SET IFLAG = ID (POINTED TO BY JADD)     TRVS1680
         LM     GRR,GR3,12(GRS)    DIVIDED BY FOUR.                        TRVS1690
         MVI    12(GRS),X'FF'                                              TRVS1700
         BR     GRR                RETURN.                                 TRVS1710
*                                                                          TRVS1720
         DROP   GRL                                                        TRVS1730
         USING  *,GRL                                                      TRVS1740
IANMLP   B      CS31                                                       TRVS1750
         DC     XL1'07'                                                    TRVS1760
         DC     CL7'IANMLP '                                               TRVS1770
CS31     STM    GRR,GR1,12(GRS)    SAVE GENERAL REGISTERS.                 TRVS1780
         L      GR1,0(0,GR1)                                               TRVS1790
         L      GR1,0(0,GR1)       LOAD NAME.                              TRVS1800
         L      GRR,ANPNT                                                  TRVS1810
         USING  DNPNT,GRR                                                  TRVS1820
         L      GRR,NPNT           LOAD POINTER TO FIRST LCB ELEMENT.      TRVS1830
         DROP   GRR                                                        TRVS1840
         LTR    GR1,GR1            DOES NAME = 0.                          TRVS1850
         BE     CS34               YES, BRANCH.                            TRVS1860
CS32     CL     GRR,NULL           IS POINTER NULL.                        TRVS1870
         BE     CS35                                                       TRVS1880
         CL     GR1,16(0,GRR)      COMP. NAME TO LCB NAME.                 TRVS1890
         BL     CS35               NAME .LT. LCB NAME, BRANCH.             TRVS1900
         BE     CS33               NAME .EQ. LCB NAME, BRANCH.             TRVS1910
         L      GRR,24(0,GRR)      CHAIN POINTER TO NEXT LCB.              TRVS1920
         B      CS32                                                       TRVS1930
CS33     LR     GR0,GRR            IADD = POINTER.                         TRVS1940
```

```
         B     CS36                                                    TRVS1950
CS34     CL    GRR,NULL          IS THERE ANY LCB ELEMENTS.            TRVS1960
         BE    CS35              NO, BRANCH.                           TRVS1970
         LR    GR0,GRR           IADD = ADD. OF FIRST LCB ELEMENT.     TRVS1980
         B     CS36                                                    TRVS1990
CS35     SR    GR0,GR0           IADD = 0.                             TRVS2000
CS36     LM    GRR,GRL,12(GRS)   RESTORE GENERAL REGISTERS.            TRVS2010
         L     GR1,24(0,GRS)                                         TRVS2020
         MVI   12(GRS),X'FF'                                         TRVS2030
         BR    GRR               RETURN.                             TRVS2040
*                                                                     TRVS2050
         DROP  GRL                                                   TRVS2060
         USING *,GRL                                                 TRVS2070
IPNTLP   B     CS41                                                  TRVS2080
         DC    XL1'07'                                               TRVS2090
         DC    CL7'IPNTLP '                                          TRVS2100
CS41     STM   GRR,GR1,12(GRS)   SAVE GENERAL REGISTERS.             TRVS2110
         L     GRR,4(0,GR1)                                          TRVS2120
         L     GRR,0(0,GRR)      LOAD IP.                            TRVS2130
         L     GR1,0(0,GR1)                                          TRVS2140
         L     GR1,0(0,GR1)      LOAD IADD.                          TRVS2150
         C     GRR,ONE           SEE IF IP = 1.                      TRVS2160
         BE    CS42              YES, BRANCH.                        TRVS2170
         C     GRR,TWO           SEE IF IP = 2.                      TRVS2180
         BE    CS42              YES, BRANCH.                        TRVS2190
         C     GRR,THREE         SEE IF IP = 3.                      TRVS2200
         BE    CS43              YES, BRANCH.                        TRVS2210
         C     GRR,FOUR          SEE IF IP = 4.                      TRVS2220
         BE    CS44              YES, BRANCH.                        TRVS2230
         C     GRR,FIVE          SEE IF IP = 5.                      TRVS2240
         BE    CS44              YES, BRANCH.                        TRVS2250
         B     CS45                                                  TRVS2260
CS42     SR    GR0,GR0           IPNT = 0.                           TRVS2270
CS425    AR    GR0,GR1           ADD IADD TO IPNT.                   TRVS2280
         SLL   GRR,2(0)          MULTIPLY IP BY FOUR.                TRVS2290
         AR    GR0,GRR           ADD IP*4 TO IPNT.                   TRVS2300
         LR    GR1,GR0                                               TRVS2310
         L     GR0,0(0,GR1)      LOAD CONTENTS OF CORRECT POINTER.   TRVS2320
         B     CS46                                                  TRVS2330
CS43     TM    0(GR1),X'40'      TEST FOR LISTR.                     TRVS2340
         BO    CS42              YES, BRANCH.                        TRVS2350
         B     CS45                                                  TRVS2360
CS44     TM    0(GR1),X'80'      TEST FOR LCB.                       TRVS2370
         BZ    CS45                                                  TRVS2380
         LA    GR0,4(0,0)        IPNT = 4.                           TRVS2390
         B     CS425                                                 TRVS2400
CS45     SR    GR0,GR0           IPNT = 0.                           TRVS2410
CS46     LM    GRR,GRL,12(GRS)   RESTORE GENERAL REGISTERS.          TRVS2420
         L     GR1,24(0,GRS)                                         TRVS2430
         MVI   12(GRS),X'FF'                                         TRVS2440
         BR    GRR               RETURN.                             TRVS2450
*                                                                     TRVS2460
         DROP  GRL                                                   TRVS2470
         USING *,GRL                                                 TRVS2480
IIDLP    B     CS51                                                  TRVS2490
         DC    XL1'07'                                               TRVS2500
         DC    CL7'IIDLP  '                                          TRVS2510
CS51     STM   GRR,GR1,12(GRS)   SAVE GENERAL REGISTERS.             TRVS2520
         L     GRR,0(0,GR1)                                          TRVS2530
         L     GRR,0(0,GRR)      LOAD IADD.                          TRVS2540
         SR    GR0,GR0           ZERO REGISTERS.                     TRVS2550
         SR    GR1,GR1                                               TRVS2560
         IC    GR0,0(GRR)        INSERT ID OF ELEM. POINTED TO BY    TRVS2570
         SRDA  GR0,4(0)          ID.  SHIFT RIGHT MOST BITS OUT.     TRVS2580
         SRL   GR1,28(0)         LEFT JUSTIFY BITS.                  TRVS2590
```

```
         AR    GR0,GR1           ADD.                                  TRVS2600
         L     GRR,12(GRS)       RESTORE REGISTERS.                    TRVS2610
         L     GR1,24(GRS)                                             TRVS2620
         MVI   12(GRS),X'FF'                                           TRVS2630
         BR    GRR               RETURN.                               TRVS2640
*                                                                      TRVS2650
         DROP  GRL                                                     TRVS2660
         USING *,GRL                                                   TRVS2670
ILENLP   B     CS61                                                    TRVS2680
         DC    XL1'07'                                                 TRVS2690
         DC    CL7'ILENLP '                                            TRVS2700
CS61     STM   GRR,GR1,12(GRS)   SAVE GENERAL REGISTERS.               TRVS2710
         L     GR1,0(0,GR1)                                            TRVS2720
         L     GR1,0(0,GR1)      LOAD IADD.                            TRVS2730
         L     GR0,0(0,GR1)      LOAD FIRST WORD OF ELEMENT POINTED    TRVS2740
         N     GR0,LMASK         TO BY IADD.  MASK OFF ID.             TRVS2750
         TM    0(GR1),X'80'      TEST FOR LCB.                         TRVS2760
         BO    CS63              YES, BRANCH.                          TRVS2770
         TM    0(GR1),X'40'      TEST FOR LISTR.                       TRVS2780
         BO    CS62              YES, BRANCH.                          TRVS2790
         S     GR0,LELEM         SUBTRACT ELEM. CONTROL FROM LENGTH.   TRVS2800
         B     CS64                                                    TRVS2810
CS62     S     GR0,LLISTR        SUB. LISTR CONTROL FROM LENGTH.       TRVS2820
         B     CS64                                                    TRVS2830
CS63     SR    GR0,GR0           NO USER DATA.                         TRVS2840
CS64     L     GR1,24(GRS)       RESTORE REGISTER.                     TRVS2850
         MVI   12(GRS),X'FF'                                           TRVS2860
         BR    GRR               RETURN.                               TRVS2870
*                                                                      TRVS2880
         DROP  GRL                                                     TRVS2890
         USING *,GRL                                                   TRVS2900
IUADLP   B     CS71                                                    TRVS2910
         DC    XL1'07'                                                 TRVS2920
         DC    CL7'IUADLP '                                            TRVS2930
CS71     STM   GRR,GR1,12(GRS)   SAVE GENERAL REGISTERS.               TRVS2940
         L     GR1,0(0,GR1)                                            TRVS2950
         L     GR1,0(0,GR1)      LOAD IADD.                            TRVS2960
         CLI   0(GR1),X'40'      IS ID OF ELEM. POINTED TO BY IADD     TRVS2970
         BE    CS74              EQUAL TO X'40'.  YES, BRANCH.         TRVS2980
         CLI   0(GR1),X'80'      IS ID OF ELEM. POINTED TO BY IADD     TRVS2990
         BE    CS74              EQUAL TO X'80'.  YES, BRANCH.         TRVS3000
         LA    GR0,0(0,GR1)      LOAD IADD.                            TRVS3010
         CLI   0(GR1),X'41'      IS ID OF ELEM. POINTED TO BY IADD     TRVS3020
         BE    CS72              EQUAL TO X'41'.  YES, BRANCH.         TRVS3030
         A     GR0,LELEM         ADD ELEMENT CONTROL LENGTH.           TRVS3040
         B     CS75                                                    TRVS3050
CS72     A     GR0,LLISTR        ADD LISTR CONTROL LENGTH.             TRVS3060
         B     CS75                                                    TRVS3070
CS74     SR    GR0,GR0           JADD = 0.                             TRVS3080
CS75     L     GR1,24(0,GRS)     RESTORE GENERAL REGISTER.             TRVS3090
         MVI   12(GRS),X'FF'                                           TRVS3100
         BR    GRR               RETURN                                TRVS3110
*                                                                      TRVS3120
         CNOP  0,4                                                     TRVS3130
SAREA    DS    18F                                                     TRVS3140
BASE     DC    A(TRAVSLP)                                              TRVS3150
NAME     DS    1F                                                      TRVS3160
KADD     DS    1F                                                      TRVS3170
KFLAG    DS    1F                                                      TRVS3180
NULL     DC    CL4'$$$$'                                               TRVS3190
ONE      DC    F'1'                                                    TRVS3200
TWO      DC    F'2'                                                    TRVS3210
THREE    DC    F'3'                                                    TRVS3220
FOUR     DC    F'4'                                                    TRVS3230
FIVE     DC    F'5'                                                    TRVS3240
```

```
ANPNT    DC    V(LASLP$)                                        TRVS3250
LMASK    DC    XL4'00FFFFFF'                                    TRVS3260
LLISTR   DC    F'16'                                            TRVS3270
LELEM    DC    F'12'                                            TRVS3280
LPANDP   DC    A(NAME)                                          TRVS3290
         DC    A(KADD)                                          TRVS3300
         DC    XL1'80'                                          TRVS3310
         DC    AL3(KFLAG)                                       TRVS3320
APOPPS   DC    V(POPPS)                                         TRVS3330
APUSHPS  DC    V(PUSHPS)                                        TRVS3340
*                                                               TRVS3350
DNPNT    DSECT                                                  TRVS3360
NPNT     DS    1F                                               TRVS3370
         END                                                    TRVS3380
```

```
      BLOCK DATA                                                        0010
C                                                                       0020
      COMMON /COM$$$/ ITRACE,IOUT,IPRNT                                 0030
C                                                                       0040
      DATA ITRACE/40/, IOUT/51/, IPRNT/0/                               0050
      END                                                               0060
```

```
      SUBROUTINE DUMPPS(ICNT,NAME1)                                     0010
C                                                                       0020
C     USED TO DUMP PUSH DOWN STORE INFORMATION.  ICNT IS THE CONTROL    0030
C     VARIABLE.  ICNT=0  PRINT ALL PDS NAMES AND CONTENTS OF PDS'S.     0040
C                ICNT=1  PRINT ALL PDS NAMES.                           0050
C                ICNT=2  PRINT CONTENTS OF PDS CALLED NAME1.            0060
C     IF IPRNT NOT EQUAL 0 THEN ONLY COUNTS ARE PRINTED.                0070
C                                                                       0080
      COMMON /COM$$$/ ITRACE,IOUT,IPNT                                  0090
      DIMENSION ICONT(10)                                               0100
C                                                                       0110
      IF(ICNT .EQ. 1) GO TO 10                                          0120
      IF(ICNT .EQ. 2) GO TO 20                                          0130
      KNTN=0                                                            0140
      KNTT=0                                                            0150
      NADD=0                                                            0160
      CALL NEEPS(NAME,NADD,NFLAG)                                       0170
      IF(NFLAG .EQ. 2) GO TO 2                                          0180
    1 KNTN=KNTN+1                                                       0190
      CALL NEEPS(NAME,NADD,NFLAG)                                       0200
      IF(IPNT .EQ. 0) WRITE (IOUT,100) NAME                             0210
      IADD1=0                                                           0220
      CALL PEEPS(NAME,IADD1,IADD2,IFLAG)                                0230
      IF(IFLAG.EQ. 2) GO TO 9                                           0240
      KNTE=0                                                            0250
      KPNT=0                                                            0260
    3 KNTE=KNTE+1                                                       0270
      KNTT=KNTT+1                                                       0280
      CALL PEEPS(NAME,IADD1,IADD2,IFLAG)                                0290
      KPNT=KPNT+1                                                       0300
      ICONT(KPNT)=IADD2                                                 0310
      IF(KPNT .LT. 10) GO TO 4                                          0320
      IF(IPNT .EQ. 0) WRITE (IOUT,101) (ICONT(J),J=1,10)                0330
      KPNT=0                                                            0340
    4 IF(IFLAG .NE. 3) GO TO 3                                          0350
      IF(KPNT .EQ. 0) GO TO 5                                           0360
      IF(IPNT .EQ. 0) WRITE (IOUT,101) (ICONT(J),J=1,KPNT)              0370
    5 IF(IPNT .EQ. 0) WRITE (IOUT,102) KNTE                             0380
      GO TO 6                                                           0390
    9 IF(IPNT .EQ. 0) WRITE (IOUT,103)                                  0400
    6 IF(NFLAG .NE. 1) GO TO 1                                          0410
    2 WRITE (IOUT,104) KNTN,KNTT                                        0420
      RETURN                                                            0430
C                                                                       0440
   10 KNTN=0                                                            0450
      NADD=0                                                            0460
      KPNT=0                                                            0470
      CALL NEEPS(NAME,NADD,NFLAG)                                       0480
      IF(NFLAG .EQ. 2) GO TO 12                                         0490
      IF(IPNT .EQ. 0) WRITE (IOUT,105)                                  0500
   11 CALL NEEPS(NAME,NADD,NFLAG)                                       0510
      KNTN=KNTN+1                                                       0520
      KPNT=KPNT+1                                                       0530
      ICONT(KPNT)=NAME                                                  0540
      IF(KPNT .LT. 10) GO TO 14                                         0550
      IF(IPNT .EQ. 0) WRITE (IOUT,106) (ICONT(J),J=1,10)                0560
      KPNT=0                                                            0570
   14 IF(NFLAG .NE. 1) GO TO 11                                         0580
      IF(KPNT .EQ. 0) GO TO 12                                          0590
      IF(IPNT .EQ. 0) WRITE (IOUT,106) (ICONT(J),J=1,KPNT)              0600
   12 WRITE (IOUT,107) KNTN                                             0610
      RETURN                                                            0620
C                                                                       0630
   20 KNTE=0                                                            0640
```

```
      IADD1=0                                                          0650
      KPNT=0                                                           0660
      CALL PEEPS(NAME1,IADD1,IADD2,IFLAG)                              0670
      IF(IFLAG .EQ. 2) GO TO 29                                        0680
      IF(IFLAG .EQ. 1) GO TO 28                                        0690
      WRITE (IOUT,100) NAME1                                           0700
   21 CALL PEEPS(NAME1,IADD1,IADD2,IFLAG)                              0710
      KNTE=KNTE+1                                                      0720
      KPNT=KPNT+1                                                      0730
      ICONT(KPNT)=IADD2                                                0740
      IF(KPNT .LT. 10) GO TO 24                                        0750
      IF(IPNT .EQ. 0) WRITE (IOUT,101) (ICONT(J),J=1,10)               0760
      KPNT=0                                                           0770
   24 IF(IFLAG .NE. 3) GO TO 21                                        0780
      IF(KPNT .EQ. 0) GO TO 25                                         0790
      IF( IPNT .EQ. 0) WRITE (IOUT,101) (ICONT(J),J=1,KPNT)            0800
   25 WRITE (IOUT,102) KNTE                                            0810
      RETURN                                                           0820
   29 WRITE (IOUT,103)                                                 0830
      RETURN                                                           0840
   28 WRITE (IOUT,108)                                                 0850
      RETURN                                                           0860
  100 FORMAT('0PUSH DOWN STORE CALLED ',Z8,' (TOP TO BOTTOM) ')        0870
  101 FORMAT(6X,10Z10)                                                 0880
  102 FORMAT(1X,'NUMBER OF ELEMENTS =',I5)                             0890
  103 FORMAT(1X,'EMPTY')                                               0900
  104 FORMAT('0NUMBER OF PUSH DOWN STORES =',I5/' TOTAL NUMBER OF PUSH' 0910
     *,' DOWN STORE ELEMENTS =',I6)                                    0920
  105 FORMAT('0PUSH DOWN STORES')                                      0930
  106 FORMAT(1X,10Z12)                                                 0940
  107 FORMAT(' NUMBER OF PUSH DOWN STORES =',I5)                       0950
  108 FORMAT('0 NOT DEFINED')                                          0960
      END                                                              0970
```

```
      SUBROUTINE MESS$$(INUM)                                          0010
C                                                                      0020
C     I/O ROUTINE FOR PRINTING ERROR MESSAGES GENERATED BY THE PUSH-   0030
C     DOWN STORE ROUTINE (PUSHST) AND FOR RECORDING THE TRACE          0040
C     INFORMATION GENERATED BY CALLS TO THE ENTRY POINTS IN PUSHST.    0050
C                                                                      0060
      COMMON /COM$$$/ ITRACE,IOUT,IPNT                                 0070
      LOGICAL*1 MEST,MEST1,MEST2,MEST3                                 0080
      DIMENSION MESL(3),MESN(3),MEST(80,3),MEST1(80),MEST2(80),        0090
     * MEST3(80)                                                       0100
      EQUIVALENCE (MEST(1,1),MEST1(1)), (MEST(1,2),MEST2(1)),          0110
     * (MEST(1,3),MEST3(1))                                           0120
      DATA MESL/52,45,45/                                              0130
      DATA MESN/21,31,61/                                              0140
      DATA MEST1/'ATTEMPT TO RELEASE SPACE WHICH HAS NOT BEEN ALLOCATED'  0150
     1/                                                                0160
      DATA MEST2/'ON A CALL TO DEFNPS SPACE WAS NOT INITIALIZED'/      0170
      DATA MEST3/'ON A CALL TO DELTPS SPACE WAS NOT INITIALIZED'/      0180
C                                                                      0190
      L=MESL(INUM)                                                     0200
      WRITE (IOUT,100) MESN(INUM),(MEST(I,INUM),I=1,L)                 0210
  100 FORMAT('0*** ERROR  PDS',I2,5X,80A1)                             0220
      RETURN                                                           0230
C                                                                      0240
C     TRACE PRINT FOR CALL TO INITPS.                                  0250
C                                                                      0260
      ENTRY INITP$(N,NPS)                                              0270
      WRITE (ITRACE,101) N,NPS                                         0280
  101 FORMAT(' INITPS CALLED WITH N= ',Z8,' AND NPS= ',Z8)             0290
      RETURN                                                           0300
C                                                                      0310
C     TRACE PRINT FOR CALL TO DEFNPS.                                  0320
C                                                                      0330
      ENTRY DEFNP$(NAME)                                               0340
      WRITE (ITRACE,102) NAME                                          0350
  102 FORMAT(' DEFNPS CALLED WITH NAME= ',Z8)                          0360
      RETURN                                                           0370
C                                                                      0380
C     TRACE PRINT FOR CALL TO DELTPS.                                  0390
C                                                                      0400
      ENTRY DELTP$(NAME)                                               0410
      WRITE (ITRACE,103) NAME                                          0420
  103 FORMAT(' DELTPS CALLED WITH NAME= ',Z8)                          0430
      RETURN                                                           0440
C                                                                      0450
C     TRACE PRINT FOR CALL TO PUSHPS.                                  0460
C                                                                      0470
      ENTRY PUSHP$(NAME,IADD)                                          0480
      WRITE (ITRACE,104) NAME,IADD                                     0490
  104 FORMAT(' PUSHPS CALLED WITH NAME= ',Z8,' AND IADD= ',Z8)         0500
      RETURN                                                           0510
C                                                                      0520
C     TRACE PRINT FOR CALL TO POPPS.                                   0530
C                                                                      0540
      ENTRY POPP$(NAME)                                                0550
      WRITE (ITRACE,105) NAME                                          0560
  105 FORMAT(' POPPS CALLED WITH NAME= ',Z8)                           0570
      RETURN                                                           0580
C                                                                      0590
C     TRACE PRINT FOR CALL TO RLSEPS.                                  0600
C                                                                      0610
      ENTRY RLSEP$                                                     0620
      WRITE (ITRACE,106)                                               0630
  106 FORMAT(' RLSEPS CALLED')                                         0640
```

```
      RETURN                                                            0650
C                                                                       0660
      END                                                               0670
```

```
      SUBROUTINE MESSP$(INUM)                                           0010
C                                                                       0020
C     I/O ROUTINE FOR PRINTING ERROR MESSAGES GENERATED BY THE          0030
C     LAS SPACE MAINTAINING ROUTINE (SPACELP) AND FOR RECORDING THE     0040
C     TRACE INFORMATION GENERATED BY CALLS TO THE ENTRY POINTS          0050
C     IN SPACELP AND DLINKLP (THE TWO-WAY LIST BUILDING ROUTINE).       0060
C                                                                       0070
      LOGICAL*1 MEST,MEST1                                              0080
      COMMON /COM$$$/ ITRACE,IOUT,IPNT                                  0090
      DIMENSION MESL(1),MESN(1),MEST(80,1),MEST1(80)                    0100
      EQUIVALENCE (MEST(1,1),MEST1(1))                                  0110
      DATA MESL/59/                                                     0120
      DATA MESN/21/                                                     0130
      DATA MEST1/'ATTEMPTED TO RELEASE LAS SPACE WHICH HAS NOT BEEN ALLO 0140
     *CATED'/                                                           0150
C                                                                       0160
      L=MESL(INUM)                                                      0170
      WRITE (IOUT,100) MESN(INUM),(MEST(I,INUM),I=1,L)                  0180
  100 FORMAT('0*** ERROR LPS',I2,5X,80A1)                               0190
      RETURN                                                            0200
C                                                                       0210
C     TRACE PRINT FOR CALL TO INITLP.                                   0220
C                                                                       0230
      ENTRY INITL$(NB)                                                  0240
      WRITE (ITRACE,101) NB                                             0250
  101 FORMAT(' INITLP CALLED WITH NB= ',Z8)                             0260
      RETURN                                                            0270
C                                                                       0280
C     TRACE PRINT FOR CALL TO RLSELP.                                   0290
C                                                                       0300
      ENTRY RLSEL$                                                      0310
      WRITE (ITRACE,102)                                                0320
  102 FORMAT(' RLSELP CALLED')                                          0330
      RETURN                                                            0340
C                                                                       0350
C     TRACE PRINT FOR CALL TO OBTNLP.                                   0360
C                                                                       0370
      ENTRY OBTNL$(L)                                                   0380
      WRITE (ITRACE,103) L                                              0390
  103 FORMAT(' OBTNLP CALLED WITH I= ',Z8)                              0400
      RETURN                                                            0410
C                                                                       0420
C     TRACE PRINT FOR CALL TO CONFLP.                                   0430
C                                                                       0440
      ENTRY CONFL$                                                      0450
      WRITE (ITRACE,104)                                                0460
  104 FORMAT(' CONFLP CALLED')                                          0470
      RETURN                                                            0480
C                                                                       0490
C     TRACE PRINT FOR CALL TO RTNLLP.                                   0500
C                                                                       0510
      ENTRY RTNLL$(IADD)                                                0520
      WRITE (ITRACE,105) IADD                                           0530
  105 FORMAT(' RTNLLP CALLED WITH IADD= ',Z8)                           0540
      RETURN                                                            0550
C                                                                       0560
C     TRACE PRINT FOR CALL TO RTNELP.                                   0570
C                                                                       0580
      ENTRY RTNEL$(IADD)                                                0590
      WRITE (ITRACE,106)                                                0600
  106 FORMAT('RTNELP CALLED WITH IADD= ',Z8)                            0610
      RETURN                                                            0620
C                                                                       0630
C     TRACE PRINT FOR CALL TO DEFNLP.                                   0640
```

```
C                                                                       0650
      ENTRY DEFNL$(NAME,IMARK,IADD)                                     0660
      WRITE (ITRACE,301) NAME,IMARK,IADD                                0670
  301 FORMAT(' DEFNLP CALLED WITH NAME= ',Z8,', IMARK= ',Z8,            0680
     * ' AND IADD= ',Z8)                                               0690
      RETURN                                                            0700
C                                                                       0710
C     TRACE PRINT FOR CALL TO INSELP.                                   0720
C                                                                       0730
      ENTRY INSEL$(P,IADD,L1)                                           0740
      WRITE (ITRACE,302) P,IADD,L1                                      0750
  302 FORMAT(' INSELP CALLED WITH P= ',Z8,', IADD= ',Z8,' AND L1= ',Z8) 0760
      RETURN                                                            0770
C                                                                       0780
C     TRACE PRINT FOR CALL TO INSILP.                                   0790
C                                                                       0800
      ENTRY INSLL$(P,IADD,NAME)                                         0810
      WRITE (ITRACE,303) P,IADD,NAME                                    0820
  303 FORMAT(' INSLLP CALLED WITH P= ',Z8,', IADD= ',Z8,                0830
     * ' AND NAME= ',Z8)                                               0840
      RETURN                                                            0850
C                                                                       0860
C     TRACE PRINT FOR CALL TO DLTELP.                                   0870
C                                                                       0880
      ENTRY DLTEL$(IADD)                                                0890
      WRITE (ITRACE,304) IADD                                           0900
  304 FORMAT(' DLTELP CALLED WITH IADD= ',Z8)                           0910
      RETURN                                                            0920
C                                                                       0930
C     TRACE PRINT FOR CALL TO DLTILP.                                   0940
C                                                                       0950
      ENTRY DLTLL$(NAME)                                                0960
      WRITE (ITRACE,305) NAME                                           0970
  305 FORMAT(' DLTLLP CALLED WITH NAME= ',Z8)                           0980
      RETURN                                                            0990
      END                                                               1000
```

```
      SUBROUTINE MODCOM(I,IVALUE)                                       0010
C                                                                       0020
C     SUBROUTINE WHICH IS USED TO MODIFY THE CONTENTS OF THE LABELED    0030
C     COMMON BLOCK COM$$$.  IF I=0 THEN IPARM ARRAY IS PRINTED.         0040
C                                                                       0050
      COMMON /COM$$$/ ITRACE,IOUT,IPRNT                                 0060
      DIMENSION IPARM(1)                                                0070
      EQUIVALENCE (IPARM(1),ITRACE)                                     0080
      DATA NPARM/3/                                                     0090
C                                                                       0100
      IF(I .EQ. 0) GO TO 10                                             0110
      IF(I .GT. NPARM) RETURN                                           0120
      IPARM(I)=IVALUE                                                   0130
      RETURN                                                            0140
   10 WRITE (IOUT,100)                                                  0150
      WRITE (IOUT,101) (IPARM(J),J=1,NPARM)                             0160
      RETURN                                                            0170
  100 FORMAT(1H0,'CONTENTS OF IPARM ARRAY (COM$$$)')                    0180
  101 FORMAT(1X,10I10)                                                  0190
      END                                                               0200
```

APPENDIX E

TWLVE EXAMPLE

The sample FORTRAN program, PUP1, builds lists to print alphabetically sorted tables of users who have poor utilization factors. There is a list of user names called the master list and lists of user jobs called the detailed lists. The user data field of LISTERs in the master list contains only the user name (three alphanumeric characters, the first being alphabetic). The detail lists are named the same as the user name in the master list. Elements in the detail lists have user data fields which consist of the subfields: JOBNAM, CHARGE, MACHUS, REGINA, REGINU, DATE, KEY and CLASS. JOBNAM holds the name of a job (it can consist of up to eight alphanumeric characters, the first being alphabetic), CHARGE contains the user's charge number, MACHUS gives the total time (CPU time plus a factor for I/O requests etc.) for the job, REGINA holds the region allocated for the job, REGINU contains the maximum region used for the GO step of the job, DATE contains the date that the job was executed, KEY indicates which CPU the job was executed on and CLASS holds the class of the job (both as read in and as executed). The elements in the master list are sorted by user name and the elements in the detail lists are sorted by jobname then (on equal jobnames) by date. An

array named INDEX, contains the addresses of dummy elements in the master list. This array is used to speed up the process of examining the master list.

The program does the following things. It first reads in an accounting card which contains, among other things, a jobname, the amount of machine seconds used, the region allocated, the region used and the date. If one of the constraints is met (1: region allocated greater than or equal to MAXREG, or 2: the difference between the region allocated and the region used greater than MAXRD, the machine use greater than or equal to SMACHU and region used greater than or equal to MINREG) then the master list is examined to see if the user name is already in the list. If not the user name is added to the master list. Next, the jobname and other information about the job are added to the appropriate detail list (the one with its name the same as the user's name). After all accounting cards have been read, the detail lists are traversed, the poor utilization factor (PUF) for each job is computed and the information related to each job is printed out. The source listing of PUF1 and a sample page of output follow.

```
C      PROGRAM PUF1                                                      0010
C                                                                        0020
C      BUILDS LISTS TO PRINT ALPHABETICALLY SORTED TABLES OF USERS       0030
C      WHO HAVE POOR UTILIZATION FACTORS.                                0040
C                                                                        0050
       REAL*8 JOBNAM,DATER                                               0060
       REAL*4 MACHUS                                                     0070
       INTEGER*4 CHARGE,DATE,DATEF,DATEL,INDADD(26),DDISP,DATES,REGIND   0080
       INTEGER*2 REGINA,REGINU,KEY                                       0090
       LOGICAL*1 CLASS(2),BLANK,NAME(4),INDEX(26),INDEXE(4),SYMBOL,      0100
      * LIN1(4),LIN2(4),LIN3(4),ZERO                                     0110
       DATA MAXREG,MAXRD,NINDEX/1400, 80, 26/, LDLIST/28/, DDISP/20/     0120
       DATA INDEX/'ABCDEFGHIJKLMNOPQRSTUVWXYZ'/, SYMBOL/'$'/, IIN/50/    0130
       DATA FACTOR/7.5E5/, IREGIN/32000/, NPDS/200/, IDUM/20/, ICUT/51/  0140
       DATA LNAMES/'NAME'/, MINREG/  50/,   SMACHU/10.0/, ZERO/200/      0150
       DATA BLANK/' '/, MAXLIN /60/, CPUC/0.2/                           0160
       COMMON /DLISTE/ JOBNAM,CHARGE,MACHUS,REGINA,REGINU,DATE,          0170
      * KEY,CLASS                                                        0180
       EQUIVALENCE (IIN1,LIN1(1)),(IIN2,LIN2(1)),(IIN3,LIN3(1))          0190
C                                                                        0200
       DATEF=99999                                                       0210
       DATEL=0                                                           0220
       ILA1=ILALP(NAME(1))                                               0230
       ILA5=ILALP(DATES)                                                 0240
       CALL IDAY(DATER)                                                  0250
C***      PRINT CONTROL INFORMATION.                                     0260
       WRITE (IOUT,2000) DATER                                           0270
       WRITE (IOUT,2001) MAXREG                                          0280
       WRITE (IOUT,2002) MINREG                                          0290
       WRITE (IOUT,2003) MAXRD                                           0300
       WRITE (IOUT,2004) SMACHU                                          0310
       WRITE (IOUT,2006) FACTOR                                          0320
       WRITE (IOUT,2015) CPUC                                            0330
C***      INITIALIZE LIST SPACE AND BUILD INDEX ARRAY.                   0340
       CALL INITPS(IDUM,NPDS)                                            0350
       CALL INITLP(IREGIN)                                               0360
       CALL DEFNLP(LNAMES,1,LNADD,IFLAG)                                 0370
       IF(IFLAG .EQ. 0) GO TO 11                                         0380
       IERR=100+IFLAG                                                    0390
       GO TO 777                                                         0400
 11    NAME(2)=BLANK                                                     0410
       NAME(3)=BLANK                                                     0420
       NAME(4)=BLANK                                                     0430
       IADD=LNADD                                                        0440
       DO 14 I=1,NINDEX                                                  0450
       ILA2=ILALP(INDEX(I))                                              0460
       CALL IMVCLP(ILA1,0,ILA2,0,1)                                      0470
       CALL INSELP(1,IADD,4,0,4,NAME(1),IFLAG)                           0480
       IF(IFLAG .EQ. 0) GO TO 13                                         0490
       IERR=200+IFLAG                                                    0500
       GO TO 777                                                         0510
 13    INDADD(I)=IADD                                                    0520
 14    CONTINUE                                                          0530
C                                                                        0540
       NREAD=0                                                           0550
       NBADS=0                                                           0560
C***      READ ACCOUNTING CARD, SEE IF JOB HAD POOR UTILIZATION.         0570
C***      CONSTRAINTS ...                                                0580
C***      1 (REGION ALLOCATED - REGION USED) .GT. MAXRD AND MACHINE      0590
C***        USE .GE. SMACHU AND REGION USED .GE. MINREG.                 0600
C***      2 REGION ALLOCATED .GE. MAXREG.                                0610
       LIN1(1)=ZERO                                                      0620
       LIN2(1)=ZERO                                                      0630
       LIN3(1)=ZERO                                                      0640
```

```
 20   READ (IIN,1000,END=50) KEY,CHARGE,CLASS,(LIN1(I),I=2,4),          0650
     * (LIN2(I),I=2,4),(LIN3(I),I=2,4),JOBNAM,DATE                     0660
      IF(DATEF .GT. DATE) DATEF=DATE                                   0670
      IF(DATEL .LT. DATE) DATEL=DATE                                   0680
      MACHUS=FLOAT(IIN1)*0.01                                          0690
      REGINA=IIN2                                                      0700
      REGINU=IIN3                                                      0710
      NREAD=NREAD+1                                                    0720
      IF((REGINA-REGINU) .LE. MAXRD) GO TO 202                         0730
      IF(MACHUS .LT. SMACHU) GO TO 202                                 0740
      IF(REGINU .GE. MINREG) GO TO 205                                 0750
 202  IF(REGINA .GE. MAXREG) GO TO 205                                 0760
      GO TO 20                                                         0770
C***    PICK UP CORRECT ADDRESS TO BEGIN SEARCH OF MASTER LIST.        0780
C***      IF III .GT. 225 THEN S TO Z, SUBTRACT 207.                   0790
C***      IF III .GT. 208 THEN J TO R, SUBTRACT 198.                   0800
C***      IF III .GT. 192 THEN A TO I, SUBTRACT 192.                   0810
 205  ILA3=ILALP(JOBNAM)                                               0820
      III=0                                                            0830
      ILA4=ILALP(III)                                                  0840
      CALL IMVCLP(ILA4,3,ILA3,0,1)                                     0850
      IF(III .GT. 225) GO TO 21                                        0860
      IF(III .GT. 208) GO TO 22                                        0870
      IF(III .GT. 192) GO TO 23                                        0880
      IERR=250                                                         0890
      GO TO 777                                                        0900
 21   III=III-207                                                      0910
      GO TO 25                                                         0920
 22   III=III-198                                                      0930
      GO TO 25                                                         0940
 23   III=III-192                                                      0950
 25   IADD=INDADD(III)                                                 0960
      NAME(4)=SYMBOL                                                   0970
      CALL IMVCLP(ILA1,0,ILA3,0,3)                                     0980
C***    TRAVERSE LIST BEGINNING AT ADDRESS IN IADD.  IF NO MASTER      0990
C***    INDEX LISTR WITH NAME IN USER DATA FIELD IS FOUND THEN         1000
C***    INSERT LISTR.                                                  1010
 30   CALL TRVELP(1,IADD,IFLAG)                                        1020
      IF(IFLAG .NE. 1) GO TO 32                                        1030
      LUADD=IUADLP(IADD)                                               1040
      JTEST=ICLCLP(ILA1,0,LUADD,0,4)                                   1050
      IF(JTEST .EQ. 0) GO TO 36                                        1060
      IF(JTEST .EQ. 1) GO TO 32                                        1070
      GO TO 30                                                         1080
C***    DEFINE DETAIL SUB-LIST CALLED NAME AND INSERT LISTR.           1090
 32   CALL DEFNLP(NAME(1),0,KADD,IFLAG)                                1100
      IF(IFLAG .EQ. 0) GO TO 33                                        1110
      IERR=300+IFLAG                                                   1120
      GO TO 777                                                        1130
 33   CALL INSLLP(2,IADD,NAME(1),IFLAG,4,0,4,NAME(1))                  1140
      IF(IFLAG .EQ. 0) GO TO 40                                        1150
      IERR=350+IFLAG                                                   1160
      GO TO 777                                                        1170
C***                                                                   1180
 36   KADD=IPNTLP(IADD,3)                                              1190
C***    INSERT ELEMENT INTO THE DETAIL SUB-LIST.  ELEMENTS ARE         1200
C***    SORTED BY JOBNAME, THEN (ON EQUAL JOBNAMES), BY DATE.          1210
 40   CALL TRVELP(1,KADD,IFLAG)                                        1220
      IF(IFLAG .NE. 0) GO TO 45                                        1230
      LUADD=IUADLP(KADD)                                               1240
      JTEST=ICLCLP(ILA3,0,LUADD,0,8)                                   1250
      IF(JTEST .EQ. 0) GO TO 42                                        1260
      IF(JTEST .EQ. 1) GO TO 45                                        1270
      GO TO 40                                                         1280
 42   IUADD=IUADLP(KADD)                                               1290
```

```
      CALL IMVCLP(ILA5,0,IUADD,DDISP,4)                                 1300
      IF(DATE .LE. DATES) GO TO 45                                      1310
      GO TO 40                                                          1320
 45   CALL INSELP(2,KADD,LDLIST,0,LDLIST,JOBNAM,IFLAG)                  1330
      NBADS=NBADS+1                                                     1340
      IF(IFLAG .EQ. 0) GO TO 20                                         1350
      IERR=400+IFLAG                                                    1360
      GO TO 777                                                         1370
C***     TRAVERSE MASTER AND DETAIL LISTS AND PRINT SORTED              1380
C***     OUTPUT.                                                        1390
 50   IADD=LNADD                                                        1400
      WRITE (IOUT,2007) DATEF                                           1410
      WRITE (IOUT,2008) DATEL                                           1420
      WRITE (IOUT,2009) NREAD                                           1430
      WRITE (IOUT,2010) NBADS                                           1440
      PFLAG=FLOAT(NBADS)/FLOAT(NREAD)*100.0                             1450
      WRITE (IOUT,2011) PFLAG                                           1460
      WRITE (IOUT,2012)                                                 1470
      NLINES=0                                                          1480
 55   CALL TRVELP(1,IADD,IFLAG)                                         1490
 56   IF(IFLAG .EQ. 2) GO TO 80                                         1500
      IF(IFLAG  .EQ. 0) GO TO 55                                        1510
      KADD=IPNTLP(IADD,3)                                               1520
 60   CALL TRVELP(1,KADD,IFLAG)                                         1530
      IF(IFLAG .NE. 0) GO TO 65                                         1540
      LUADD=IUADLP(KADD)                                                1550
      CALL IMVCLP(ILA3,0,LUADD,0,LDLIST)                                1560
      IF(NLINES .LT. MAXLIN) GO TC 63                                   1570
      NLINES=0                                                          1580
      WRITE (IOUT,2012)                                                 1590
 63   REGIND=REGINA-REGINU                                              1600
      PUF=FLOAT(REGIND)*REGINA*MACHUS/FACTOR                            1610
      IF(KEY .EQ. 0)  PUF=PUF*CPUC                                      1620
      WRITE (IOUT,2005) JOBNAM,CHARGE,MACHUS,REGINA,REGINU,REGIND,      1630
     * DATE,KEY,CLASS,PUF                                              1640
      NLINES=NLINES+1                                                   1650
      GO TO 60                                                          1660
 65   JADD=IADD                                                         1670
      CALL TRVELP(1,IADD,IFLAG)                                         1680
C     CALL DLTELP(JADD)                                                 1690
      WRITE (IOUT,2013)                                                 1700
      NLINES=NLINES+1                                                   1710
      GO TO 56                                                          1720
C***                                                                    1730
C80   CALL RLSELP                                                       1740
 80   CONTINUE                                                          1750
      STOP 24                                                           1760
 777  WRITE (IOUT,2014) IERR                                            1770
      STOP 0                                                            1780
C                                                                       1790
 1000 FORMAT(8X,I1,I5,29X,2A1,3X,9A1,A8,7X,I5)                          1800
 2000 FORMAT(1H1,'PUF1',5X,'RAN ',A8)                                   1810
 2001 FORMAT('0MAXIMUM REGION USED',I6)                                 1820
 2002 FORMAT('0MINIMUM REGION USED',I6)                                 1830
 2003 FORMAT('0MAXIMUM REGION DIFFERENCE ALLOWED',I6)                   1840
 2004 FORMAT('0MINIMUM MACHINE USE',F8.2)                               1850
 2005 FORMAT(2X,A8,I7,F10.2,I7,2I8,I7,I3,2X,A1,1X,A1,F11.2)             1860
 2006 FORMAT('0FACTOR',1PE12.2)                                         1870
 2007 FORMAT('0FIRST DATE',I7)                                          1880
 2008 FORMAT('0LAST DATE',I7)                                           1890
 2009 FORMAT('0NUMBER OF CARDS READ IN',I7)                             1900
 2010 FORMAT('0NUMBER OF CARDS FLAGED',I7)                              1910
 2011 FORMAT('0PERCENT FLAGED',F7.2)                                    1920
 2012 FORMAT(1H1,18X,'MACHINE',3X,'REGION',2X,'REGION',2X,'REGION'/     1930
     * 2X,'JOBNAME',2X,'CHARGE',2X,'USE SEC.',2X,'ALLOC.',2X,'USED',3X,  1940
```

```
    * 'NOT USED',1X,'DATE',1X,'CPU',1X,'CLASS',5X,'PUF'/)                    1950
2013 FORMAT(1X)                                                              1960
2014 FORMAT(//1H0,'*** ERROR IN PUF1, ERROR NUMBER =',I5)                    1970
2015 FORMAT('0CPU CONSTANT',F8.3)                                            1980
     END                                                                     1990
```

JOBNAME	CHARGE	MACHINE USE SEC.	REGION ALLOC.	REGION USED	REGION NOT USED	DATE	CPU	CLASS	PUF
AIJ2F005	35005	67.83	1530	4	1526	73018	1	F L	211.16
AKPN	30096	22.76	300	186	114	73018	1	C C	1.04
HQSC	35629	21.41	400	304	96	73018	1	C C	1.10
ISB3	33469	41.19	300	184	116	73018	1	C C	1.91
JRGB	34746	28.38	540	404	136	73018	1	C C	2.78
NIPIF005	28946	155.68	1150	1040	110	73018	1	F F	26.26
VFRTC005	28684	91.15	500	344	156	73018	1	C C	9.48
WBJDC005	31735	299.28	540	424	116	73018	1	C C	25.00
XADR	33612	30.80	540	358	182	73018	1	C C	4.04

VITA

Eric Allen Ferguson was born in Chattanooga, Tennessee, on October 14, 1946. He graduated from Red Bank High School in June, 1964. In September, 1965, he entered Chattanooga State Technical Institute and in June, 1967, he recieved an Associate of Science degree in Computer Science. He has been employed by the Oak Ridge National Laboratory since June, 1967.

In September, 1967, he began study toward a Bachelor of Science degree in Business Administration at the University of Tennessee and in December, 1970, he recieved that degree. The following quarter he entered the Graduate School at the University of Tennessee. In August, 1973, he recieved a Master of Science degree in Computer Science. He is a member of Phi Kappa Phi.

He is married to the former Mildred Elouise Morgan of Chattanooga, Tennessee and they have two sons: Michael Allen and Mitchell Walter.