INFORMATION PROCESSING LANGUAGE V MANUAL
Section II. Programmers' Reference Manual

A. Newell
F. M. Tonge
E. A. Feigenbaum*
G. H. Mealy
Mathematics Division
The RAND Corporation

N. Saber**
B. F. Green, Jr.***
A. K. Wolf***

P-1918
31 March 1960

Correction Copy A.N.

*Consultant to the Mathematics Division of The RAND Corporation
**The University of Pittsburgh
***Lincoln Laboratory

Reproduced by

The RAND Corporation • Santa Monica • California

SUMMARY

This section of the manual sets out the complete rules for IPL-V. Section I* gives an introduction and simplified account of the language and should be read before trying to use this section. IPL-V is currently coded for several computers. Details of operation and the IPL-V interpretive systems for each machine are given in later sections of the IPL-V manual. The rules in Section II are common to all versions, and any program constructed according to these rules will be accepted on any of the object machines.

*The RAND Corporation paper, P-1897, " Information Processing Language V Manual Section I. The Elements of IPL Programming."

This section of the manual sets out the complete rules for IPL-V. Section I gives an introduction and simplified account of the language and should be read before trying to use this section. IPL-V is currently coded for several computers. Details of operation and the IPL-V interpretive systems for each machine are given in later sections of the IPL-V manual. The rules in Section II are common to all versions, and any program constructed according to these rules will be accepted on any of the object machines.

GENERAL DEFINITIONS

IPL LANGUAGE

IPL is a formal language in terms of which information can be stated and processes specified for processing the information. IPL allows two kinds of expressions: data list structures which contain the information to be processed; and routines which define information processes. A complete program consists of a set of data list structures and the set of routines that define the processing to be done.

IPL COMPUTER

No computer currently available can process the IPL language directly, but any general purpose digital computer can be made to interpret this language by writing a special program in the language of that computer. Such a program is called an IPL-V interpretive system. The interpretive system interprets IPL expressions as equivalent expressions in the language of the particular computer, and causes the computer to carry out IPL processes. When a computer is running with the IPL interpreter system, its main storage has two major sections, one containing the IPL interpretive system, and the rest, called the total available space, in

which IPL programs and data may be stored. The particular computer together with the interpretive system is known as the IPL computer. The total available space is the "storage" of the IPL computer.

The interpretive system consists of several parts:

(1) A loader, for loading IPL programs into the available space from cards or tape.

(2) A set of primitive processes, for manipulating IPL expressions.

(3) An interpreter, for executing the instructions in the IPL routines

(4) A monitor, for providing debugging information.

IPL SYMBOLS

IPL is a system for manipulating symbols. The IPL computer distinguishes three types of symbols--regional, internal, and local. It keeps track of the type of each symbol being used, and will behave differently in some cases, according to the type of symbol encountered.

To the programmer, a regional symbol is a letter or punctuation mark followed by a positive decimal integer no greater than 9999; e.g., A 1, *12, R3496. Regional symbols are the programmer's stock of symbols. An internal symbol is a positive decimal integer. Internal symbols are the computer's stock of symbols, and will generally not be used by programmers. Inside the computer--that is, except for input and output--internal and regional symbols are treated identically. Each symbol corresponds to a particular storage address. However, there are means to tell regional and internal symbols apart, if needed.

Local symbols are used to connect lists and list structures. Their identity is transitory--they are erased, generated, and changed at will by the IPL computer. To the programmer a local symbol is a 9 followed by a positive decimal integer no greater than 9999; e.g., 9-1, 9-34, 9-123.

The 9 takes the place of the letter in the regional symbols. The use of local symbols will be explained in the discussion of list structures.

All symbols are printed out in the same form as they are input: regionals are printed in the letter-numbers form; internals are printed as decimal integers; and locals are printed as integers prefixed by a 9.

STANDARD IPL WORDS

All IPL expressions, both data list structures and routines, are written in terms of an elementary unit, called the IPL word. Each word occupies a single cell of the total available space in the IPL computer. A standard word consists of four parts: P, Q, SYMB, and LINK. P and Q are called the prefixes of the word. Q is the designation prefix and P is the operation prefix (for routines) or the data type prefix (for data list structures). Each prefix is an octal digit--i.e., it may take on the values 0, 1, ..., 7. Its meaning depends on whether it occurs in routines or data. SYMB is an IPL symbol, and is called the symbol of the word. LINK is also an IPL symbol.

SPECIAL IPL WORDS: DATA TERMS

Different formats are necessary to represent integers, floating point numbers, alphabetic characters, etc. Words containing such information are called data terms, and have three parts: P, Q, DATA. P and Q are prefixes, and DATA contains the special datum. The Q prefix is always 1, indicating that the word is a special data term. The P prefix specifies the type of data. (Q=1 is also used in routines with a different meaning; program and data are kept separate by context.)

THE CODING FORM

To put IPL words into the IPL computer, they must first be coded and

punched into cards. The cards can then be read by the interpretive system. The cards are prepared from the standard coding form, one card per line, each card representing one IPL word (See Figure 1). For standard IPL words, the columns labelled NAME, P, Q, SYMB, and LINK are used. Type is 0 or blank, Sign (+-) is irrelevant (but see INITIAL LOADING), and all other columns are ignored by the IPL computer. (Certain columns are excluded from use.) P and Q may each contain any digit from 0 through 7. Blank is regarded as 0. For data lists, P and Q are always blank (or 0) unless the word is a data term. NAME, SYMB and LINK may contain any IPL symbol. If LINK is left blank, the IPL computer automatically fills in the address of the next cell, represented by the next line on the coding sheet. This is also true for SYMB. However, if the next line has a regional or internal symbol as NAME, the blank LINK or SYMB is taken as a termination symbol 0.

NAME, SYMB, and LINK each occupy five columns. The first (leftmost) column holds the region character--i.e., the letter for regions, or 9 for local symbols. The other four columns hold the four digit integer associated with the symbol. The integer may be located anywhere within the field in consecutive digits. For example, A1, A 1 A 1, and A0001, are all instances of A1. Likewise, 910, 9 10 and 9-10 are all instances of the local symbol 9-10, as long as the 9 occurs in the leftmost column. The exact rules for writing legitimate IPL symbols in NAME, SYMB, and LINK are the following:

-Regional and local symbols must have their initial character in the leftmost column of the field (columns 43, 51, and 57 respectively). Internal symbols may start anywhere in the field, except that if the initial digit is "9", that digit cannot be in the leftmost column.

-Except for the character in the leftmost column, all non-numeric characters and blanks are ignored.

-The numeric part of the symbol may occur anywhere in the field with any spacing. The field is scanned, and the digits are accumulated as they are found and composed into a number.

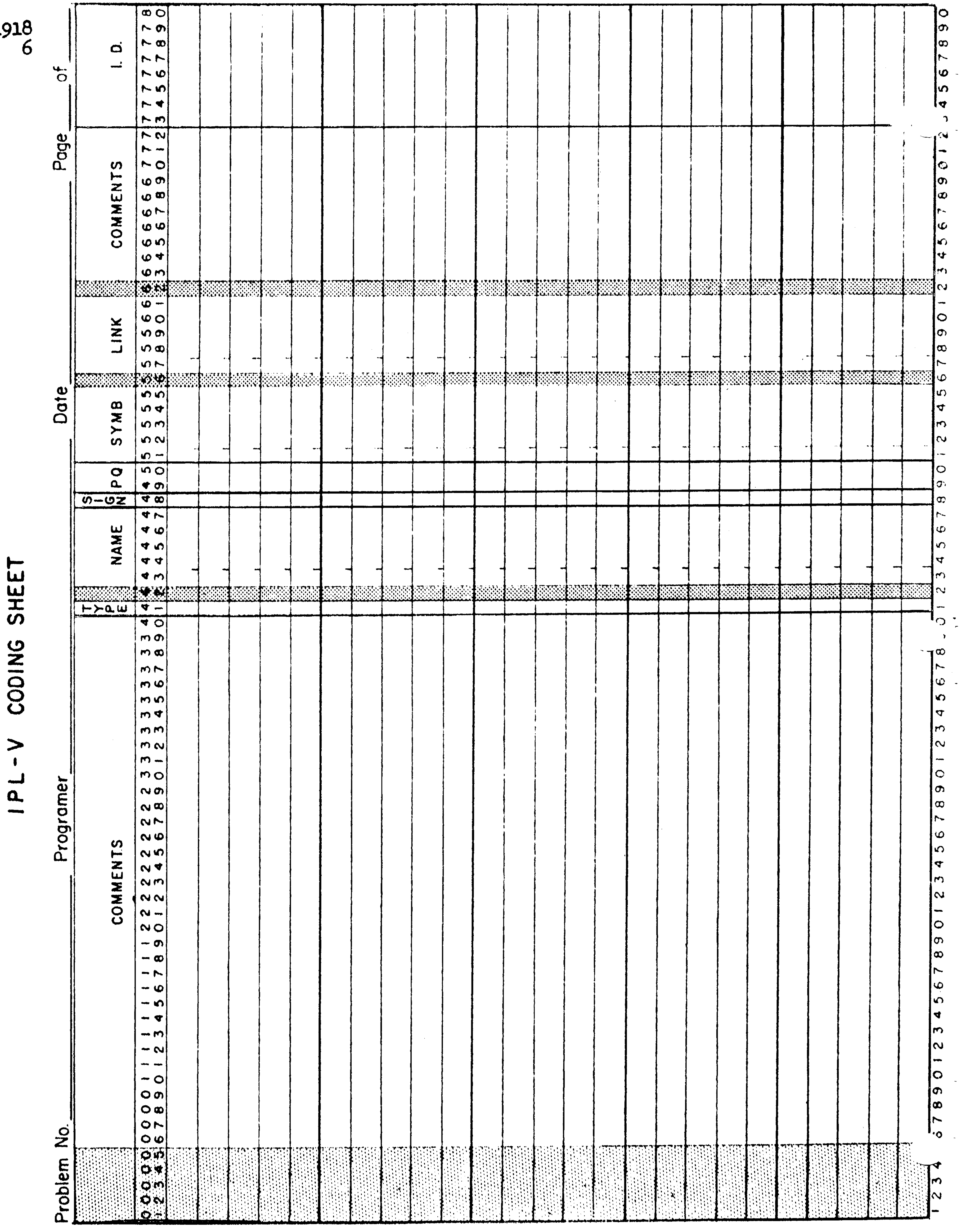
IPL-V CODING SHEET
Problem No.
Programer
Date
Page
of
COMMENTS
TYPE
NAME
SIGN
PQ
SYMB
LINK
COMMENTS
I. D.

Figure 1.

DATA TYPE CODE P

The format for data terms is shown in Figure 2. Data terms have been defined only for P from 0 to 3. The other four values, 4 through 7, are available for private use (see sections on the machine systems).

CELLS

Each IPL word resides in a cell in the IPL computer (that is, a register in the total available space). We say a cell contains the word, also that the cell contains a symbol; i.e., the SYMB part of the word. Alternatively, we refer to SYMB as the symbol in a cell. LINK is also a symbol, but this is referred to as the link in a cell.

AVAILABLE SPACE

Since each IPL word resides in a cell in the IPL computer, during a run the routines and data list structures require a certain amount of the total available space--that is, of the total set of cells. At any moment during a run there is a set of cells which is not part of any routine or data list structure. This set is called the available space at that moment. It is the stock of cells out of which new list structures can be constructed. The available space is continually depleted as new structures are built, but continually replenished as old structures are no longer needed and are erased--i.e., the cells composing them returned to available space. All the available space is on a list, named H2, and called the available space list. The mechanics for transferring cells to and from available space will be described later.

LIMITS ON THE NUMBER AND TYPES OF STRUCTURES

All data list structures and routines are built up from the available space, and any cell may be used for any purpose in such

IPL-V CODING SHEET

Problem No. _____ Programer _____ Date _____ Page ____ of ____

COMMENTS	TYPE	NAME	SIGN	PQ	SYMB	LINK	COMMENTS	I. D.
P = 0 DECIMAL INTEGER								
GENERAL FØRMAT -			±	01	dddd	dddd		
EXAMPLES: -12,192			-	1	1	2192		
15				1		15		
P = 1 FLØATING PØINT								
GENERAL FØRMAT -			±	11	ddddd	d ee		
(Exponent ee ranges from +50 to -50. Any character in column 59 is taken as "-".)								
EXAMPLES: $-.15\times10^{3}$			-	11	15	3		
$.732\times10^{-2}$				11	732	- 2		
P = 2 ALPHANUMERICAL								
GENERAL FØRMAT -				21	aaaaa			
EXAMPLE: IPL V				21	IPL 5			
P = 3 ØCTAL								
GENERAL FØRMAT -			±	31	ddddd	ddddd		
EXAMPLE: 7777777777				31	77777	77777		

Figure 2

constructions. Consequently, as long as cells are available construction can continue. No separate limits exist on how many data list structures, storage cells, symbols, and so on, can be used. The only limit is in the total amount of available space.

AUXILIARY STORAGE

The storage that holds the interpretive system and the available space is called the main storage. Access is also possible to secondary storages--fast auxiliary storage and slow auxiliary storage--when available on the object machine.

CELL NAMES

Access to a word requires access to the cell that holds the word, and this requires that the cell have a known IPL name. The name of a cell is the IPL symbol that represents the machine address of the cell. All cells in use have names, either regional, local, or internal. The cells in available space are not considered to have names since only when they are taken for a specific use is the name determined. On the coding sheet putting a symbol in the NAME field specifies that the word on that line will be in the cell named. In essence, cells are named by writing a symbol for NAME. The programmer need name only those cells he wishes to refer to explicitly; hence NAME is left blank in most instances.

HEADS, LIST CELLS, TERMINATION CELLS

Cells are used to construct the various structures in IPL. There are three kinds of cells: heads, which start structures; list cells, which form the bodies of structures; and termination cells, which mark the end of structures. (Data terms occur in heads.) We will need these distinctions in giving the conventions for each type of structure. A

termination cell contains the word 00 00000 00000, and the symbol that names it is called a termination symbol. The symbol 0 is a termination symbol, and is used by the programmer in preference to other termination symbols. Hence, it is referred to as the termination symbol. The need for other termination symbols arises from the delete processes (see DELETE). Any cell containing 0--i.e., SYMB = 0--is called empty. (0 is an internal symbol.)

STORAGE CELLS

A storage cell is one whose purpose is to hold symbols. A storage cell is created simply by giving a cell a regional name and putting the termination symbol, 0, for LINK. SYMB is then the symbol contained in the cell; it may be put in initially by writing in the symbol on the coding sheet, or the cell may be left empty and a symbol put in during processing.

Examples:

	NAME	PQ	SYMB	LINK
The empty storage cell, A1:	A1		0	0
The cell, A2, containing B3:	A2		B3	0

Any cell may function as a storage cell (assuming it is not being used in some other capacity).

PUSH DOWN LISTS FOR STORAGE CELLS

Associated with each storage cell is a system for storing symbols contained in the cell. This system is a data list, called a push down list. The storage cell is the head of the list, and the cells used in the storage system are list cells. The symbol currently in the storage cell may be put onto the push down list, so that the cell can be used for another purpose, and then recovered at a later time. The system is a "Last-In-First-Out" system (LIFO); that is, the symbols are recovered from

storage in the inverse order of their entry. The most recently preserved symbol is the first one recovered. The system is fully specified by the operation for putting symbols in storage, preserve or push down, and the operation for recovering symbols from storage, restore or pop up.

PRESERVE To preserve a storage cell is to put a copy of the symbol contained in the cell on the push down list associated with the cell. The operation leaves the symbol still in the cell.

RESTORE To restore a storage cell is to move into the cell the symbol most recently put on the associated push down list of that cell. The symbol occurrence in the cell just prior to restoring is lost, and the symbol moved from the push down list is no longer on the list.

Examples: Let the storage cell W3 contain the symbol S5:

NAME	PQ	SYMB	LINK
W3		S5	0

If W3 is preserved, then a copy of S5 goes into storage, while W3 continues to hold S5:

W3		S5
		S5	0

If another symbol, B1, is now put into W3, we have:

W3		B1
		S5	0

If W3 is preserved again, we have:

W3		B1
		B1
		S5	0

And if another symbol, G3, is put into W3, we have:

W3		G3
		B1
		S5	0

If W3 is restored, then:

W3		B1
		S5	0

And if W3 is restored again:

W3		S5	0

After two preserves followed by two restores, W3 is brought back to the original condition; and similarly for any number of preserves followed by the same number of restores.

Each cell, then, really consists of a stack of symbols. The one on top is accessible, and the others are in storage in the order in which they are put in the stack. There is no limit to the number of symbols that may be stored in a push down list; it is always possible to add another as long as some available space remains in the IPL system.

DATA LIST STRUCTURES

The data list structure is the IPL expression that contains the data to be processed. The total data for a program will be given as a set of data list structures. Each data list structure is made up of data lists, which in turn are made up of IPL words. (Routines are also list structures, but satisfy different conventions.)

DATA LISTS

A data list is a sequence of cells containing IPL words whose order is defined by the rule: the LINK part of the cell contains the name of the next cell in the list. The first cell in a list--the cell which does not have its name as the LINK of any cell of the list--is the head of the list. All other cells of the list are list cells. Cells containing data terms (cells with Q = 1) are also heads, and are treated as special cases of data lists. The following rules apply to all regular data lists:

- -Only names of list cells can occur as the LINK of a cell.
- -Only names of heads can occur as the SYMB of a cell.
- -The name of each list cell occurs once and only once as LINK (this is equivalent to making lists linear, without cycles).
- -The LINK of the last cell in a list is a termination symbol.

A list with 0 for the LINK of the head is called an empty list.

To create a data list write down a symbol in the NAME field of some line. This symbol is the name of the list, and the cell corresponding to it is the head of the list. (Thus the same symbol names both the list and the head cell.) Write down the IPL words of the list in successive lines of the coding sheet. These lines are the list cells, and they occur in the list in the order they appear on the coding sheet. No names are given to the list cells (NAME left blank) and the LINKs of all cells

but the last one are also left blank. The public termination symbol, 0, is written for LINK of the last cell.

Examples:	NAME	PQ	SYMB	LINK
The list with name L1, containing the symbols S1, S5, S12 and S7 in that order: (the first symbol occurs in the head here; conventions for heads will be given presently).	L1		S1	
			S5	
			S12	
			S7	0
The list with name 9-5, containing the symbols A5 and 9-3.	9-5		A5	
			9-3	0

The termination symbol, 0, is used, although any other termination symbol is perfectly legal. The latter would require an additional cell, and thus take extra space without any compensating gain.

NAMING LIST CELLS

The IPL computer will assign an internal name to any cell that is not explicitly named by the programmer. The programmer may give names to list cells by using local symbols. (Using regional symbols would start a new list, in effect.) The IPL computer interprets a blank SYMB or LINK in a cell as referring to the next cell, and the name of this next cell is filled in. This occurs properly either when the next cell has a blank NAME or a local symbol for NAME. If the next cell has a regional name, the blank SYMB or LINK is taken as the termination symbol, 0.

Example:	NAME	PQ	SYMB	LINK
The usual reason for naming data list cells is to break the sequential order on the coding sheet:	L1		0	9-1
	9-2		S2	9-3
	9-1		S1	9-2
	9-3		S3	0

DESCRIBABLE LISTS

It is possible to associate with a list a description list, similar in concept to a function table, which can contain information about the

list being described. The SYMB of the head is reserved for the name of the description list. A list with the head so reserved is called describable. If a list is describable, descriptive information can be added to it or requested about it, at any time during processing, by means of a set of processes, J10 - J15. Since the head of a describable list is reserved, the first symbol on the list is in the first cell after the head, the second symbol is in the second list cell after the head, and so on. Lists that use the head for any other purpose are called non-describable. If no information has been associated with a describable list, then there will exist no description list. However, the head is still reserved, and hence is empty. (The list in the previous example has no description list associated with it but has a reserved head.)

POLICY ON DESCRIBABLE LISTS

The basic processes (the J's) assume that data lists are describable whenever this is relevant to their operation. In the manual we will assume a list to be describable, unless explicitly stated otherwise.

ATTRIBUTES AND VALUES

The information that can be associated with a describable list is in the form of values to specified attributes. Suppose L1 is a describable list, and A1 is some attribute, say the number of symbols on a list. Then the value of A1 for L1 is some symbol, say N3. This can be expressed in mathematical notation as A1(L1) = N3. Any symbol at all may be used as an attribute, no matter what its other functions in the total program might be. The value of an attribute is always a single symbol. However, any symbol may be the value--for example, the name of a data term, the name of a list, or the name of a list structure--so that there is no

restriction at all on the kind of information that can effectively be the value of an attribute. Only a single value is possible for a given attribute, but it is always possible for the value of an attribute to be the name of a list of "values," thus achieving the effect of multivalued attributes. The usefulness of descriptions stems from the generality of what constitutes an attribute or a value. Any number of attribute values may be associated with a describable list.

DESCRIPTION LISTS

A description list is a list that contains alternately the symbols for attributes and their values. The attribute symbol occurs first, followed by its value for the list the description list is describing. Description lists are themselves describable, so that the first attribute symbol occurs in the first list cell, its value in the second list cell, the next attribute symbol in the third, and so on. The same symbol cannot occur more than once as an attribute on the description list.

CREATING DESCRIPTION LISTS

Processes exist to create, modify, interrogate and erase description lists during processing (see J10 to J15). Such lists can also be created on the coding sheet prior to loading. A local name is written for SYMB of the head of the list to be described. The description list is defined in the same manner as any other list: its name is written for NAME on some line (the same symbol as occurred in the head of the main list); the head of the description list is made blank since the description list is describable; then follow the attributes and values in sequence on the coding sheet; the final value has a termination symbol for LINK. (No other

list structures may intervene on the coding sheet between the describable list and the description. See DOMAIN OF DEFINITION OF LOCAL SYMBOLS.)

Examples:

	NAME	PQ	SYMB	LINK
The describable list, L1, with no descriptions:	L1		0	
			S1	
			S2	
			S3	0
L2 described by the attributes A1 and A2 with values V1 and V2 respectively:	L2		9-0	
			S1	
			S2	
			S3	0
	9-0		0	
			A1	
			V1	
			A2	
			V2	0

DATA LIST STRUCTURES

A list structure is a set of lists connected together by the fact that the names of some of the lists occur on other lists in the set. A data list structure is characterized by the following conditions:

- All the component lists are data lists (hence linear--that is, not re-entrant).
- There is one list, called the main list, that has a regional name (internal, if created by the IPL computer).
- All lists, except the main list, have local names, and are called sublists.
- All local names that occur in the list structure--that is, as SYMB of some cell--name lists that belong to the list structure.
- No cell belongs to more than one list (no merging of lists).
- The name of each component list except the main list occurs at least once on some list of the list structure; it may occur many times.
- The main list is always describable; the sublists may either be describable or non-describable.

A data list structure is thus a fairly simple form of list structure--many complicating ways of linking lists together having been excluded. It is not the simplest, which would be a tree, since it is possible for the name of a sublist to appear in several places in the structure. Data terms are included in the definition, as are storage cells since they are also data lists. The name of a list structure is the name of its main list. (Thus this symbol does triple duty as the name of a list structure, list and cell.) Not all symbols occurring in a list structure refer to other lists in the structure: if they are regional or internal symbols their referents cannot belong to the same list structure. Thus there can be complicated cross references between a set of data list structures.

DOMAIN OF DEFINITION OF LOCAL SYMBOLS

The domain of definition of a local symbol is a list structure. Within a single list structure a local symbol can be the name of only one data list--that for which it occurs as NAME. All occurrences of a local symbol within a list structure are understood to refer to this data list. However, there is no connection between the local symbols in one list structure and those in another (which is why they are called local). Thus the symbol 9-1 will stand for many things in a total program. Contrariwise, a regional symbol, like A1, or an internal symbol, like 1622, always stands for the same object throughout the total program. On the coding sheet the occurrence of a regional or internal symbol for NAME marks the start of a list structure. All local symbols that occur after this line belong to this list structure, until another regional or internal NAME occurs.

LEVELS

It is often convenient to refer to the lists of a data list structure as having levels. The main list has the highest level, and a sublist is one level below its superlist--i.e., the list on which its name occurs. (It is possible for the name of a list to occur on several lists at different levels.) If numbers need to be assigned to levels, the main list is assigned level 1 and increasing positive integers are used for successively lower levels.

Examples:

A single list can be a data list structure:

A single data term can be a data list structure:

A list of lists can be a data list structure. (The spaces between lists are for clarity in the manual; no such spaces need occur on the coding sheet):

A list of numbers can be a list structure. In the example, two of the numbers belong to the structure and the other, N3, does not:

A list can have multiple occurrences of sublists, as well as mutual references and self references:

NAME	PQ	SYMB	LINK
L1		0	
		S1	
		S2	
		S3	0
B5	21	BILL	
L2		0	
		9-1	
		9-2	
		9-3	0
9-1		0	
		S1	
		S2	
		S3	0
9-2		0	
		S3	
		S1	
		S2	0
9-3		0	
		S2	
		S3	
		S1	0
L3		0	
		9-3	
		N3	
		9-1	0
9-1	1		15
9-3	- 1		19
L4		0	
		9-1	
		9-1	0
9-1		0	
		9-2	0
9-2		0	
		9-1	
		9-2	0

If the name of the main list, which is internal or regional, appears in the list structure it is treated like any other regional or internal symbol. The example, L5, is a simple list.

NAME	PQ	SYMB	LINK
L5		0	
		L5	
		L5	
		L5	0

The algebraic expression, $(X1+X2)\cdot(X3-X4)$ can be written as a list structure where the sublist arrangement indicates the parenthetical structure. We have used +, -, and · instead of admissible IPL symbols to make the correspondence clear.

X0		0	
		9-1	
		·	
		9-2	0
9-1		X1	
		+	
		X2	0
9-2		X3	
		-	
		X4	0

OTHER LIST STRUCTURES

Other kinds of list structures besides data list structures are possible and useful--e.g., circular lists, in which the "last" cell links to the "first" cell. The programmer is free to invent and use any such structures he desires, but he is then responsible for being aware of their special nature. Almost any kind of structure can be loaded in the computer (see INITIAL LOADING). We have defined the class of data list structures, in order to provide useful processes which take into account their particular conventions--e.g., copy and erase an entire data list structure.

ROUTINES AND PROGRAMS

The IPL expressions used to specify information processes are generally similar to their data counterparts, but differ in detail. Corresponding to the word of data is the instruction, to the data list is the program list, and to the data list structure is the routine.

PRIMITIVE PROCESSES

A primitive process is one that can be directly performed by the computer without further IPL interpretation; i.e., one that is coded directly in machine language. IPL symbols can name primitives. Most of the basic processes (the J's) are primitives, and it is possible to add primitives to the language (see the sections on machine systems for details).

INSTRUCTIONS

The IPL word that specifies an information process is called an instruction. It always has the standard form: PQ SYMB LINK. The process to be done is designated by PQ SYMB, while the LINK, as usual, designates the next cell in a list. The P and Q codes are entirely different from the data P and Q codes. They denote operations to be carried out rather than types of symbols and data. (The information that SYMB is regional, internal, or local is lost in an instruction, but is not needed for interpretation.) The definitions of P and Q, given presently, completely define the process designated by an instruction.

PROGRAM LISTS

A program list is a sequence of cells containing instructions, whose order is defined by the following rule: the LINK of a cell is the name of the next cell in the list. The first cell in a list is the head; all others are list cells. The head contains an instruction, so no program list is describable. In interpretation, the program list gives a sequence of instructions to be carried out in the order of the list. Almost anything is possible with program lists: they may be reentrant, or merge. They may have regional symbols as LINKs, and names of list cells as SYMB. These

various possibilities are governed by the interpretation of the P and Q code.

ROUTINES AND PROGRAMS

A routine is a list structure characterized by the following conditions:

-Some of the lists are program lists.

-There is one program list, called the main list, that has a regional name.

-All lists, except the main list, have local names and are called sublists (and initiate local subroutines).

-All local names that occur in the list structure as SYMB of some cell name lists that belong to the list structure.

-The name of each sublist occurs at least once on some list of the list structure; it may occur many times.

-The main list is not describable (since it is a program list).

Local symbols follow the same rules for the domain of definition given in connection with data list structures. It is also possible to talk about the levels in a routine in the same manner as with data list structures. Each routine specifies a process. A routine is executed when this specified process is carried out by the IPL computer. This implies that the subroutines out of which the process is composed are also executed (as required). A program is the set of routines that specifies a process in terms of primitive processes. The routine first executed is at the highest level. The routines of the program are all routines required in the execution of this top routine, taking into account that routines require other routines for their execution.

DATA IN ROUTINES

Normally, routines consist purely of program lists. However, it is sometimes convenient to include various kinds of data along with the

routine, such as constants, storage cells, and so on. Since data list structures are handled differently from program lists on input (P and Q are treated differently), it is necessary to indicate which cells are to be interpreted as data. A + or - in the Sign column is used for this, and every cell to be interpreted as data must be so marked. (The + or - contributes to the data only in the case of numeric data terms, as defined earlier; in all other cases it has no effect.)

SAFE CELL

A storage cell is called safe over a routine if that routine leaves the symbol in the cell (and the push down list) the same as it was prior to the execution of the routine, except as modification is explicitly required by the definition of the routine. If there is no guarantee that the contents of the storage cell will remain unmolested, the cell is called unsafe over the routine. A routine can use a safe cell, as long as it returns the cell to the original condition. Safe cells are useful in IPL because the preserve and restore operations make it easy to use a storage cell and then return it to an earlier condition. From the point of view of the using routine, a safe cell is one it can put a symbol in, execute a subroutine, and expect to find the symbol still in the cell afterwards.

INPUTS AND OUTPUTS OF ROUTINES, HO

A routine can have a set of operands, called the input symbols. It can also produce a set of symbols as outputs. It may also modify existing data list structures, either those designated by input symbols, or those inplicit in the construction of the routine. The number of inputs or outputs is unlimited. They are always symbols, but these symbols can

name list structures (either data or routines), so that the types of inputs and outputs are completely general.

All inputs for a routine are placed in a special storage cell, H0, called the communication cell. If there are multiple inputs, they are placed in the push down list of H0 in a sequence determined by the definition of the routine. All outputs from a routine are also placed in the communication cell, H0. If there are multiple outputs, they are placed in the push down list of H0 in a sequence determined by the definition of the routine. In the IPL-V manual we will let (0), (1) ..., represent, respectively, the symbols in H0 and its push down list. They will serve as names for the inputs and outputs. The communication cell is safe over all routines. In connection with inputs this means that a routine must remove (before it terminates) all the input symbols from the communication push down list. The outputs, of course, are explicitly required to be in H0 at the end of processing. (Of course, routines can be defined with any input-output conventions the programmer desires. The above ones are used by the basic processes (the J's), and means are provided to make them easy to use generally.)

EXPLICIT STATEMENT OF INPUTS AND OUTPUTS

The safety of H0 implies that a routine must remove all its input symbols from H0. Its outputs, of course, are to be left in H0. In order to avoid confusion we adopt the policy of explicitly stating all inputs and outputs. For example, if a routine leaves one of its input symbols in H0, this is to be stated explicitly as one of the outputs.

TEST CELL, H5

The result of many processes involves a binary distinction-- a "yes"

or "no." For example, a process may be a "test" whose purpose is to make a binary choice, or it may produce an output where there is no guarantee that the output can be produced, so that a binary indication, "yes, the output was produced," or "no, the output was not produced," is needed as well as the output symbol in those cases where it can be produced. A special storage cell, H5, called the test cell, is used for this binary information. It can contain either of two special symbols, "+," which stands for yes, or "-," which stands for no. The + and - are symbols used only in the manual. In the computer, J4 is the symbol for + and J3 for -. These are, respectively, the names of the basic processes that set H5 + or -. The test cell is safe over the basic processes (the J's); that is, if a J-process does not set H5 as part of its definition, then H5 will be the same after performance of the process as it was before. (This means that conditional transfers may be delayed after the decision has been made and recorded in H5, as long as only J's which do not set H5 are performed.)

THE DESIGNATION OPERATION, Q, AND THE DESIGNATED SYMBOL, S

In instructions the Q prefix specifies an operation, called the designation operation, whose operand is SYMB. The result of performing the designation operation on SYMB is a new symbol, S, called the designated symbol of the instruction. We give below all eight values of Q. The first five Q's Q = 0, 1,..., 4 are normally the only ones that appear on the coding sheet.

Q = 0 S = the symbol in the instruction itself--i.e., SYMB.

Q = 1 S = the symbol in the cell named in the instruction--i.e., in SYMB.

Q = 2 S = the symbol in the cell whose name is in the cell named in the instruction--i.e., in the cell named in SYMB.

Q = 3 Trace this program list (otherwise equivalent to Q = 0).

Q = 4 Continue tracing (otherwise equivalent to Q = 0).

Q = 5 SYMB is the address of a primitive--i.e., of a machine language subroutine.

Q = 6 Routine is in fast auxiliary storage.

Q = 7 Routine is in slow auxiliary storage.

Examples:

Given the memory situation:

NAME	PQ	SYMB	LINK
B1		C1	0
C1		D1	0

For the three instructions below we get the following designated symbol:

S = B1	0 B1
S = C1	1 B1
S = D1	2 B1

THE OPERATION CODE, P

The P prefix specifies an operation, called simply the operation of the instruction, whose operand is the designated symbol, S. The result is an action related to the set up, execution, and clean up of routines. The eight operations are:

P = 0 EXECUTE S. S is assumed to name a routine or a primitive; it is executed--i.e., the process it specifies is carried out--before the next instruction is performed.

P = 1 INPUT S. HO is preserved; then a copy of S is put in HO.

P = 2 OUTPUT TO S. A copy of (0) is put in cell S; then HO is restored.

P = 3 RESTORE S. The symbol most recently stored in the push down list of S is moved into S; the current symbol in S is lost.

P = 4 PRESERVE S. A copy of the symbol in S is stored in the push down list of S; the symbol still remains in S.

P = 5 REPLACE (0) BY S. A copy of S is put in HO; the current (0) is lost.

P = 6 COPY (0) IN S. A copy of (0) is put in S; the current symbol in S is lost, and (0) is unaffected.

P = 7 BRANCH TO S IF H5 -. The symbol in H5 is always either + or -. If H5 is +, then LINK names the cell containing the next instruction to be performed. (This is the normal sequence.) If H5 is -, then S names the cell containing the next instruction to be performed.

Thus, P = 0 is used to execute subroutines; P = 1, 2, 5, and 6, are used to transfer symbols to and from the communication cell, HO; P = 3 and 4 are used in connection with safe cells; and P = 7 is a centralized transfer of control.

Examples:

At the right we give small segments of program lists--i.e., sequences of instructions. Below we give a verbal statement of the action.

	NAME	PQ	SYMB	LINK
It takes two instructions to put the symbol in W0 into the cell W1. The first instruction, 11W0, inputs the symbol 1W0 to H0, and the second, 20W1, moves the symbol into cell W1.		11	W0	
		20	W1	
It is desired to execute a process, P15, which takes two inputs and produces one output. The inputs are to be 'L1' and the symbol in W0; and the output is to be in W1. 10L1 inputs 'L1' to H0, pushing the symbol in H0 down, so it is not destroyed. 11W0 inputs the symbol in W0 to H0, again pushing down. Then P15 is fired: it removes the two symbols just put in H0, and places its own output there. 20W1 takes this output from H0 and puts it in W1 (destroying the symbol in W1). H0 is left as it was at the beginning.		10	L1	
		11	W0	
			P15	
		20	W1	
It is desired to put (0) into Y5, but without destroying the symbol already there. Hence, 20Y5 is preceeded by 40Y5, which preserves Y5.		40	Y5	
		20	Y5	
It is desired to replace a symbol in the cell named in W1 by the symbol in the cell named in W0. 12W0 brings the symbol into H0, and 21W1 puts it in 1W1--i.e., in the cell named in W1. Notice that H0 is left just as it was before the two operations were performed.		12	W0	
		21	W1	
A process whose name is in Y2 is fired with input from W0. Assume it has one output. This is put into W1 by 60W1, which also leaves it in H0 so that J2 can test if it is equal to S5. The result of J2 is either a + or - in H5. 709-1 transfers control to the part of the program list starting at 9-1 if H5 is -. If H5 +, then control proceeds down the list.		11	W0	
		1	Y2	
		60	W1	
		10	S5	
			J2	
		70	9-1	
			....	
	9-1		....	
Process P30 is fired on an input from W0. W0 is restored by 30W0 to bring it back to its previous condition.		11	W0	
			P30	
		30	W0	

INTERPRETATION

The interpretation of a program consists of generating a sequence of primitives according to the lists in the program, and executing each primitive in turn. The part of the IPL-computer that carries this process out is called the interpreter. The process consists of a cycle of operations, which we define in two ways: first, as a series of rules, from the most generally applicable to the most special; second, as a step-by-step sequence of interpretive actions, similar to a flow diagram.

CURRENT INSTRUCTION ADDRESS CELL, H1

Execution of a routine in a program involves executing its subroutines. While executing a subroutine it is necessary to remember the current location in the higher routine, so that when the subroutine is finished, interpretation can proceed from the correct instruction in the higher routine. The hierarchy of in-process subroutines is necessarily unlimited, since a subroutine can be composed of other subroutines of unknown composition. A special storage cell, H1, called the current instruction address cell or CIA, is used to mark locations in the hierarchy of inprocess routines. The symbol in H1 is the address of the current instruction; the symbol one down in the push down list is the address of the instruction in the routine one level up; the next symbol down is the address of the instruction in the routine two levels up; and so on. (The programmer never uses H1; it is used solely by the interpreter.)

RULES OF INTERPRETATION

1. An instruction is interpreted by first applying Q to SYMB to get S and then applying P to S to get the action.

2. Generally, the instructions in a program list are interpreted in the order of the list. Control advances.

3. In case P = 7 the sequence may be broken (if H5-), but control remains at the same level and continues along the list from the cell with name S. Control branches.

4. A process designated in a program list is executed by remembering the address of its instruction in H1 (with a preserve), and then interpreting its program list--i.e., the list whose name is the designated symbol--starting with the instruction in the head. Control descends a level.

5. A primitive process designated in a program list is executed by transferring machine control to the machine language subroutine corresponding to the primitive process; no descent occurs.

6. Interpretation of a program list terminates with a LINK = 0, the end of the list; or with LINK = name of a routine, in which case this routine is executed as the last process of the program list. (Termination is also achieved by branching to a 0 or the name of a routine via P = 7.)

7. Upon termination of a program list, control ascends a level, and interpretation proceeds in the program list that contained the name of the program list just finished, from the point at which it was executed (H1 is restored). If H1 is empty, the computer halts.

8. If the routine of a designated process is in auxiliary storage, it is brought into main storage, and interpretation proceeds.

THE INTERPRETATION CYCLE

START: H1 contains the name of the cell holding the instruction to be interpreted.

INTERPRET Q
- Q = 0, 1, 2: apply Q to SYMB to yield S; go to INTERPRET P.
- Q = 3, 4: execute monitor action (see MONITOR SYSTEM); take S = SYMB; go to INTERPRET P.
- Q = 5: transfer machine control to SYMB (executing primitive); go to ASCEND.
- Q = 6, 7: bring routine in from auxiliary storage; put name of auxiliary region in H1, go to INTERPRET Q.

INTERPRET P
- P = 0: go to TEST FOR PRIMITIVE.
- P = 1, 2, 3, 4, 5, 6: perform the operation; go to ADVANCE.
- P = 7: go to BRANCH.

TEST FOR PRIMITIVE: Q of S
- Q = 5: transfer machine control to SYMB of S (executing primitive); go to ADVANCE.
- Q ≠ 5: go to DESCEND.

ADVANCE: interpret LINK
- LINK = 0: termination; go to ASCEND.
- LINK ≠ 0: LINK is the name of the cell containing the next instruction; put LINK in H1; go to INTERPRET Q.

ASCEND: restore H1 (returning to H1 the name of the cell holding the current instruction, one level up); restore auxiliary region if required; go to ADVANCE.

DESCEND: preserve H1: put S into H1 (H1 now contains the name of the cell holding the first instruction of the sub program list); go to INTERPRET Q.

BRANCH: interpret sign in H5
- H5 - : put S as LINK (control transfers to S); go to ADVANCE.
- H5 + : go to ADVANCE.

Figure 3 gives a schematic picture of the connections between the parts of the interpretive cycle.

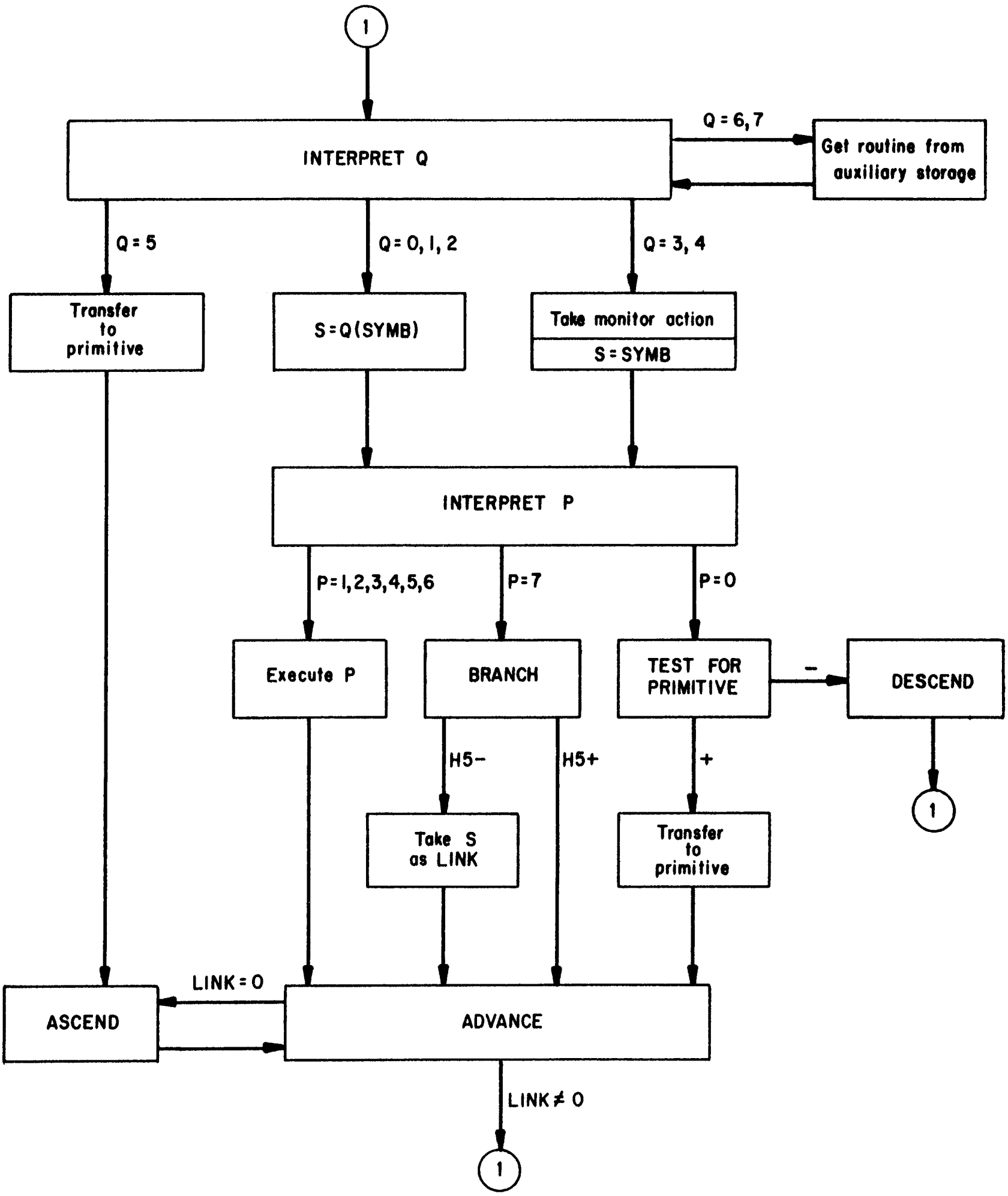
1
INTERPRET Q
Q = 6, 7
Get routine from auxiliary storage
Q = 5
Q = 0, 1, 2
Q = 3, 4
Transfer to primitive
S = Q(SYMB)
Take monitor action
S = SYMB
INTERPRET P
P = 1, 2, 3, 4, 5, 6
P = 7
P = 0
Execute P
BRANCH
TEST FOR PRIMITIVE
−
DESCEND
H5−
H5+
+
1
Take S as LINK
Transfer to primitive
LINK = 0
ASCEND
ADVANCE
LINK ≠ 0
1

FIG. 3

TALLY OF INTERPRETATION CYCLES, H3

The interpreter counts the number of cycles executed by tallying 1 into H3 every time an ADVANCE occurs. H3 is an integer data term. It is set to zero at the beginning of a run by the loader. It is available to the program during running--that is, it can be copied, reset to 0 at various points in the program, and so on. It provides a useful measure of the amount of processing done.

BASIC SYSTEM OF PROCESSES

The system of prefixes, P and Q, the interpreter, and the rules for constructing list structures, are essentially the grammar of IPL. In order to construct useful programs it is necessary to add a set of basic processes for manipulating symbols, lists, description lists, list structures, and special format words. The system provided here is general purpose, in that any process can be accomplished with it. It is focussed on list manipulation, however, with the consequence that arithmetical processes are inefficient in comparison with their machine code counterparts. The system consists of a set of storage cells with special functions (some of which have already been described), and a set of basic information processes. Some of the basic processes are primitives; some are elementary IPL routines included to complete the repertoire.

SYSTEM REGIONS (EXCLUDED FROM OTHER USE)

The regions Hdddd, Jdddd, and Wdddd are used by the system, and no new symbols in these regions may be defined by the programmer.

SYSTEM CELLS

The following cells have special functions. They are all storage cells and safe, except H3 and W11, which are integer data terms.

H0 Communication cell.

H1 Current instruction address cell (CIA); never used by programmer.

H2 Available space list; never used by programmer, except to count with J126.

H3 Tally of interpretation cycles executed; an integer data term.

H4 Current auxiliary routine cell; never used by programmer.

H5 Test cell; safe only over J's.

W0 to W9 Ten cells for common working storage (see WORKING STORAGE PROCESSES and GENERATOR PROCESSES).

W10 Random number control cell; holds the name of integer data term used to produce random number in J129 and J16.

W11 Remainder of integer division; an integer data term (see J113).

See MONITOR SYSTEM FOR W12 through W15, W23, W29.

W12 Monitor start cell; holds name of routine executed at start of trace (Q=3).

W13 Monitor end cell; holds name of routine executed at return to Q=3 point.

W14 Monitor terminate cell; holds name of routine executed at signalled termination.

W15 Monitor save cell; holds name of routine executed at signalled save for restart.

See INPUT-OUTPUT for W16 through W22, W24, W25.

W16 Input mode cell; holds name of integer determining input mode.

W17 Output mode cell; holds name of integer determining output mode.

W18 Read unit cell; holds name of integer determining unit used by J140.

W19 Write unit cell; holds name of integer determining unit used by J142.

W20 Print unit cell; holds name of integer determining unit for J150's.

W21 Print column cell; holds name of integer determining print column.

W22 Print spacing cell; holds name of integer determining line and page spacing.

W23 Post mortem cell; holds name of list determining information to be printed on post mortem dump.

W24 Print line cell; holds name of present print line.

W25 Entry column cell; holds name of integer determining entry position in print line.

See ERROR TRAP for W26 through W28.

W26 Error trap cell; holds name of list, in description list form, of trap symbols and associated processes.

W27 Trap address cell; holds CIA at the time of the trap.

W28 Trap symbol cell; holds symbol indicating cause of trap.

W29 Monitor point address cell; holds name of cell holding instruction with Q = 3.

GENERAL PROCESSES, J0 to J9

In this and following sections we give the definitions of the basic processes, accompanied by whatever general explanations are appropriate. Note that all outputs are explicitly named, and that only these outputs remain in H0 after completion of a routine. We include definitions of some terms with a circumscribed meaning.

TEST A test is a process whose only result is to set H5 + or -. Its definition is of the form: "TEST X" where X is any statement. If X is true, then H5 is set +; if X is false, then H5 is set -. Any number of inputs is permissible.

FIND A find is a process with a single symbol as output, but where it is uncertain whether the output can be produced (can be found). If the output is produced it is put in H0, and H5 is set +. If the output is not produced, there is no output in H0, and H5 is set -. Any number of inputs is permissible.

MOVE In normal computing one never destroys the information in the originating location when reading it into a new place; i.e., readouts are "non-destructive." In IPL, with the operation of restore, a "destructive" read becomes useful. Thus, move means to put in the newly designated place, but not to leave in the original place. If a symbol is being moved from a storage cell, then the cell is restored; if a list structure is being moved to auxiliary storage, then it is erased in main storage.

J0 NO OP. Proceed to the next instruction.

J1 EXECUTE (0). The process, (0), is removed from H0, H0 is restored (this positions the process's inputs correctly), and the process is executed (as if its name occurred in the instruction instead of 'J1').

J2 TEST IF (0) ≡ (1). (The identity test is on the SYMB part only; P and Q are ignored.)

J3 SET H5-. The symbol in H5 is replaced by the symbol J3.

J4 SET H5 +. The symbol in H5 is replaced by the symbol J4.

J5 REVERSE H5. If H5 +, it is set -; if H5 is -, it is set +.

J6 REVERSE (0) AND (1). Permutes the symbol in H0 with the first symbol down in the H0 push down list.

J7 HALT, PROCEED ON GO. The computer stops; if started again it interprets the next instruction in sequence.

J8 RESTORE H0. (Identical to 30H0, but can be executed as LINK.)

J9 ERASE CELL (0). The cell whose name is (0) is returned to the available space list, without regard to the contents of the cell.

DESCRIPTION PROCESSES, J10 to J16

As described earlier in the section on DATA LIST STRUCTURES, there are processes for manipulating descriptions and description lists. For all of them the name of the describable list is input, and not the name of the description list. The name of the description list is found in the head of the describable list, and, whenever created by these processes, is a local symbol. (This allows the description list to be erased automatically whenever the list is erased as a list structure--see J72.)

J10 FIND THE VALUE OF ATTRIBUTE (0) OF (1). If the symbol (0) is on the description list of list (1) as an attribute, then its value--i.e., the symbol following it--is output as (0) and H5 set +; if not found or if the description list doesn't exist, there is no output and H5 set -. (J10 is accomplished by a search and test of all attributes on the description list.)

J11 ASSIGN (1) AS THE VALUE OF ATTRIBUTE (0) OF (2). After J11 the symbol (1) is on the description list of list (2) as the value of attribute (0). If (0) was already on the description list, the old value has been removed, and (1) has taken its place; if the old value was local, it has been erased as a list structure (J72). If (0) is a new attribute, it is placed at the front of the description list. J11 will create the description list (with a local name), if it does not exist (head of (2) blank). There is no output in H0.

J12 ADD (1) AT FRONT OF VALUE LIST OF ATTRIBUTE (0) OF (2). The value of (0) is assumed to be the name of a list. The symbol, (1), is inserted on the front of this list (behind head, as in J64). If the attribute is not on the description list, it is put on and a list is created as its value (with a local name). As in J11, if the description list doesn't exist, it is created.

J13 ADD (1) AT END OF VALUE LIST OF ATTRIBUTE (0) OF (2). Identical to J12, except that (1) is inserted at the end of the list, rather than the front.

J14 ERASE ATTRIBUTE (0) OF (1). If the symbol (0) exists on the description list of list (1) as an attribute, both it and its value symbol are removed from the list. If either is local, it is erased as a list structure (J72). If (0) is not an attribute on the description list of (1), nothing is done. (In all cases the description list is left.)

J15 ERASE ALL ATTRIBUTES OF (0). The description list of list (0) is erased as a list structure (J72), and the head of (0) is put blank.

J16 FIND ATTRIBUTE RANDOMLY FROM DESCRIPTION LIST OF (0). All the attributes on the description list of list (0) that have positive numerical data terms as values (integer or floating point) are taken as a population from which a random selection is made with relative weights given by their values. Thus if there are attributes A_i with values $N_i > 0$ then:

$$\text{Probability of } A_j \text{ being selected} = \frac{N_j}{\sum_{\text{all } i} N_i}$$

The output (0) is the attribute symbol selected, and H5 is set +. If there are no positive numerical data terms on the description list, there is no output and H5 is set -. The random number used in J16 is generated as in J129, and is therefore controlled by W10.

GENERATOR HOUSEKEEPING PROCESSES, J17 to J19

GENERATORS

Repetitive operations can be handled in IPL by means of loops, utilizing the conditional branch, just as in normal programming. They can also be handled by means of generators. A generator is a process that produces a sequence of outputs and applies to each a specified process. The process that the generator applies is called the subprocess of the generator, and is an input. Thus, the generator is associated with the kind of sequence it produces, and will apply any process whatsoever to these outputs. The only thing a generator knows about the subprocess is the name of its routine, plus a convention allowing the subprocess to control whether or not the generator will continue to produce outputs of the sequence. This latter convention is necessary if generators are to be used conditionally--e.g., to search for a member of a sequence with certain properties.

What makes generators different from all the other processes considered so far, is that two contexts of information--that of the generator, and that of the subprocesses and superprocess--must coexist in the computer at the same time. Hence, the strict hierarchy of routines and subroutines is violated, and special pains have to be taken to see that information remains safe, and that each routine is always working in its appropriate context. To see this, define the context of a routine to be the set of symbols in the working storages that it is using. We will assume that any routine using n+1 symbols of information, stores these in W0 through Wn, rather than some arbitrary subset of W's. The routine that uses a generator, which we will call the superroutine, has a certain context.

The subprocess is in the same context as the superroutine. The generator is being used to provide a sequence of information to be processed in the routine using the generator, and the subprocess is simply that part of the superroutine that does the processing. In general it needs access to all the symbols in the context of the superroutine. It is given a name only to communicate to the generator what processing to do. The generator has an entirely different context in order to produce the sequence. The purpose of the generator is to separate the processing that goes into producing a sequence from the processing that is to be done to the sequence. There is an alternation between generator and subprocess which is both an alternation of control and an alternation of context: to produce an element of the sequence, the generator must be in control, and its context should occupy the W's; and to process the element the subprocess must be in control, and the context of the superroutine should occupy the W's. Thus, whenever the generator fires the subprocess, it is necessary to remove the context of the generator from the W's, thus revealing the prior context, which is that of the superroutine. At the termination of the subprocess the context of the generator must be returned to the W's (pushing down the W's, of course).

To handle the special housekeeping associated with generators, three routines are provided: J17 is used at the beginning of a generator to set up the housekeeping, J18 is used to fire the subprocess, and shuffles the contexts back and forth, and J19 is used at the end of a generator to clean up the housekeeping structures.

J17 GENERATOR SETUP. Has two inputs:
(0) = Wn, the highest W that will be used for working storage--
e.g., (0) = W6, if cells W0 through W6 will be used.
(1) = the name of the subprocess to be executed by generator.
J17 does three things (and has no output):
-Preserves the cells W0 through Wn;
-Stores Wn and the name of the subprocess in storage cells, and preserves a third cell for the output sign of H5 (these three storage cells are called the generator hideout);
-Obtains the trace mode of the superroutine (Q one down in H1); and records it in one of the hideout cells (see MONITOR SYSTEM).

J18 EXECUTE SUBPROCESS. Has no input. It does six things:
-Removes the symbols in W0 through Wn, returning the previous context of symbols to the top of the W's;
-Stacks these symbols in one of the hideout cells;
-Sets the trace mode of the subprocess to be that of the superroutine (see MONITOR SYSTEM);
-Executes the subprocess;
-Returns the symbols of the generator's context from the hideout to the W's, pushing the W's down;
-Records H5, the communication of the subprocess to the generator (see J19), in one of the hideout cells.

J19 GENERATOR CLEANUP. Has no input. Does three things:
-Restores W0 through Wn;
-Restores all the cells of the hideout;
-Places in H5 the recorded sign, which will be + if the generator went to completion (last subprocess communicated +), and - if the generator was stopped (last subprocess communicated -).

GENERATOR CONVENTIONS

We can now summarize the conventions for the use and construction of generators.

-In the superroutine the generator is executed like any other routine. Its inputs are placed in H0:
(0) is always the name of the subprocess;
(1), (2), ..., define the kind of sequence to be produced.

-Start the generator routine by doing J17: input (1), the subprocess, is already in place; do a 10Wn, where Wn is the highest working cell to be used, for input (0).

-Produce the first member of the sequence, and put it in H0 as input to the subprocess. The member may be given by any number of symbols, (0), (1), ...

-Fire the subprocess by executing J18. At the time of execution the generator's symbols cannot be stacked up more than one deep in the W's or J18 will fail to clear the context.

-The subprocess operates in the context of the superroutine, taking as input the symbols provided by the generator, above. Thus the symbols in the W's are the ones placed there by the superroutine, or by one of the earlier executions of the subprocess. Likewise, the subprocess can put symbols in the W's (or H0), which are then available to later executions of the subprocess, or to the superroutine after the termination of the generator.

-The subprocess sets H5 upon termination: + if the generator is to produce the next member of the sequence; - if the generator is to terminate.

-Within the generator, after executing J18, if H5 is + produce the next member of the sequence. If there are no more members, clean up and quit with J19, which will pop up the W's and set H5 for output. If H5 is -, then immediately clean up and quit with J19.

-There is no output from the generator to the superroutine except H5, which is + if the generator went to completion--i.e., produced all members of the sequence--and is - if the generator was terminated. J19 sets this output.

-There is no restriction on the nesting or cascading of generators: a generator may use other generators as subroutines; and a generator can be in the form of a subprocess operating on the output of another generator. (The subprocess of a generator is part of its context, so that J18 always fires the subprocess of the generator currently in context.)

-If the generator is in main storage, the subprocess to it may have either a regional or local name. If the generator is in auxiliary storage, the subprocess to it must have a regional name (see AUXILIARY STORAGE PROCESSES).

WORKING STORAGE PROCESSES, J20 to J59

Storage cells can be created at will by the programmer, and can be used either as permanent or temporary storage for any purpose the programmer desires. The only advantage in using the W's lies in the following forty processes for manipulating them, together with their built-in use in the generator processes.

J2n MOVE (0), (1), ..., (n) INTO W0, W1, ..., Wn RESPECTIVELY. Ten routines, J20 - J29 that provide block transfers out of H0 into working storage. The symbols currently in W0 to Wn are lost.

J3n RESTORE W0, W1, ..., Wn. (Ten routines, J30 - J39)

J4n PRESERVE W0, W1, ..., Wn. (Ten routines, J40 - J49)

J5n PRESERVE W0, W1, ..., Wn, THEN MOVE (0), (1), ..., (n) INTO W0, W1, ..., Wn, RESPECTIVELY. (Ten routines, J50 to J59, combining J4n and J2n.)

LIST PROCESSES, J60 to J104

PRESERVE AND RESTORE AS GENERAL LIST OPERATIONS

The preserve and restore operations were defined earlier for storage cells. We describe below the mechanics underlying them. It can be seen that these operations can apply to any list, given the name of a cell in the list: preserve will insert an additional cell with the same PQ SYMB as the given cell, and restore will replace the contents of the given symbol with the contents of the following cell, and remove the following cell from the list, thus performing a deletion.

To the right we are given, initially, the available space list, H2, and a cell, W0, with a list proceeding from its LINK:

NAME	PQ	SYMB	LINK
H2		0	1000
1000		0	1050
1050		0	1020
1020		0	....
W0		B2	500
500		C1	505
505		C2	....

If we preserve W0, then a word is obtained from available space and inserted in the list following W0, with a copy of SYMB of W0:

H2		0	1050
1050		0	1020
1020		0	

Notice that all words in the list except W0 remained unchanged, and that all the conditions for preserve are satisfied. Note also that the amount of processing is independent of how many items are on the list.

W0		B2	1000
1000		B2	500
500		C1	505
505		C2	....

If we now put into W0 a new SYMB, D1, we get (with no change in the H2 list):

W0		D1	1000
1000		B2	500
500		C1	505
505		C2	....

Restoring W0 reverses the operation, deleting the cell next after W0, putting it back on the available space, but putting its SYMB in W0:

H2		0	1000
1000		0	1050
1050		0	1020
1020		0	....
W0		B2	500
500		C1	505
505		C2	....

Restoring W0 again yields:
Notice that cells are returned on the front of the available space list, H2, so that the amount of processing required is independent of the size of available space.

H2		0	500
500		0	1000
1000		0	1050
1050		0	1020
1020		0	....
W0		C1	505
505		C2	....

LOCATE

A locate produces an output which is the name of the cell containing the desired symbol. Since there is no guarantee that the symbol is locatable, H5 is set + if it is, and - if it is not located. In the

negative case an output is still produced: in the locate processes in the basic system, J60, J61, J62, the output is the name of the last cell in the list.

INSERT

In an insert two symbols are designated, either directly as inputs or as the result of preliminary processing by the insert processes: a symbol in a list cell, and a symbol that is to be inserted in the list relative to the first symbol. A new cell from available space is put in the list to hold the new symbol, which is then located in the appropriate relationship to the symbol already in the list. There are no outputs in HO.

Consider the mechanics for two relationships: insert before and insert after. Suppose the symbol to be inserted is A1, the symbol in the list is B1, and its list cell is 1000:

NAME	PQ	SYMB	LINK
	900	....	1000
	1000	B1	910
	910	....	....

In both cases we start by preserving 1000:

	900	....	1000
	1000	B1	1010
	1010	B1	910
	910	....	....

For insert before, we put A1 in 1000:

	900	....	1000
	1000	A1	1010
	1010	B1	910
	910	....	....

For insert after, we put A1 in 1010:

	900	....	1000
	1000	B1	1010
	1010	A1	910
	910	....	....

Notice that the symbols bear the appropriate relationship of before and after, but not necessarily the cells. Given the name of a cell, there is no way to insert a cell in front of it, since the cell that links to it is unknown.

DELETE

In a delete a symbol in a list is designated, either directly as input or as the result of preliminary processing, and it is desired to remove this symbol from the list, reducing the number of list cells by one. H5 is set - for appropriate special cases; e.g., if the symbol designated for deletion does not exist. Otherwise it is set +.

Suppose the designated symbol is A1 and it is in list cell 1000:

NAME	PQ	SYMB	LINK
900		....	1000
1000		A1	910
910		B1	920
920		....	....

Then deletion is accomplished by restoring 1000:

900		....	1000
1000		B1	920
920		....	....

Notice that it is the cell after 1000 that is removed. It is not possible to remove a cell knowing only the name of the cell, since the name of the cell linking to it is unknown.

Suppose, however, that cell 1000 was the last cell in the list:

900		....	1000
1000		A1	0

Then, it is not possible to remove the next cell, which is 0, the termination symbol. Instead, 1000 is made into a private termination cell. This is the only way to make cell 900 the last cell in the list. H5 is set - to indicate that we have deleted the last symbol.

900		....	1000
1000	0	0	0

POLICY ON PRIVATE TERMINATION CELLS

Private termination cells are introduced to allow deletion of final symbols on lists. They occur in no other way. They can gradually accumulate during processing, using up space. Consequently, J60, the process which locates the next symbol on a list, automatically returns private termination cells to available space. substituting the termination symbol, 0. (J60 can do this, since when it detects a termination cell it still has available the name of the previous cell.) Any J's that use J60 as a subroutine will also have this feature (see sections on machine systems).

ERASE

To erase a structure of any kind is to return all the cells comprising it to available space. There is no output in HO.

COPY

To copy a structure of any kind is to produce a new set of cells from available space and link them together isomorphically to the given structure. All the cells of the new set will contain exactly the same symbols as their correspondents, except those that contain symbols used to link the structure together; e.g., local names in list structures. These contain the names of the copies of the corresponding lists. The name of the new structure is the output, (0).

LIST PROCESSES

J60 LOCATE NEXT SYMBOL AFTER CELL (0). (0) is the name of a cell. If a next cell exists (LINK of (0) not a termination symbol), then the output (0) is the name of the next cell, and H5 is set +. If LINK is a termination symbol, then the output (0) is the input (0), which is the name of the last cell on the list, and H5 is set -.

If the next cell is a private termination cell, J60 will work as specified above, but in addition the private termination cell will be returned to available space and the LINK of the input cell (0) will be changed to hold 0.

No test is made to see that (0) is not a data term, and J60 will attempt to interpret a data term as a standard IPL cell.

J61 LOCATE LAST SYMBOL ON LIST (0). (0) is assumed to be the name of a cell in a list (either a head or list cell; it makes no difference). The output (0) is the name of the last cell in the list, and H5 set +. If there is no cell after (0), then the output (0) is the input (0) and H5 is set -.

J62 LOCATE (0) ON LIST (1). A search of list with name (1) is made, testing each symbol against (0) (starting with cell after cell (1)). If (0) is found, the output (0) is the name of the cell containing it and H5 set +. Hence, J62 locates the first occurrence of (0) if there are several. If (0) is not found, the output (0) is the name of the last cell on the list, and H5 set -.

J63 INSERT (0) BEFORE SYMBOL IN (1). (1) is assumed to name a cell in a list. A new cell is inserted in the list behind (1). The symbol in (1) is moved into the new cell, and (0) is put into (1). The end result is that (0) occurs in the list before the symbol that was originally in cell (1). There is no output in H0.

J64 INSERT (0) AFTER SYMBOL IN (1). Identical with J63, except the symbol in (1) is left in (1), and (0) is put into the new cell, thus occurring after the symbol in (1). (If (1) is a private termination symbol, (0) is put in cell (1), which agrees with the definition of insert after.)

J65 INSERT (0) AT END OF LIST (1). Identical with J64, except that the location of the last cell is obtained first, prior to inserting.

J66 INSERT (0) AT END OF LIST (1), IF NOT ALREADY ON IT. Identical with J62 followed by J64, if (0) is not found. If (0) is found, J66 does nothing.

J67 REPLACE (1) BY (0) ON LIST (2) (FIRST OCCURRENCE ONLY). J62 followed by putting (0) in the cell occupied by (1). This only replaces the first occurrence of (1). If (1) doesn't occur on list (2), J67 does nothing.

J68 DELETE SYMBOL IN CELL (0). (0) names a cell in a list. The symbol in it is deleted by replacing it with the next symbol down the list (the next cell is removed from the list and returned to available space, so that the list is now one cell shorter). H5 is set + unless (0) is the last cell in the list or a termination cell. Then H5 is set -. Thus, H5 - means that, after J68, (0) is a termination cell.

J69 DELETE SYMBOL (0) FROM LIST (1) (FIRST OCCURRENCE ONLY). J62 is executed, followed by a delete if (0) is found. H5 is set + if (0) deleted, and set - if (0) not on list.

J70 DELETE LAST SYMBOL FROM LIST (0). A J61, followed by a delete if list (0) was not empty. H5 is set + if there was a last symbol, and set - if list (0) was empty.

J71 ERASE LIST (0). (0) is assumed to name a list. All cells of the list--both head and list cells--are returned to available space. (Nothing else is returned, not even the description list of (0) if it exists.) There is no output in H0. If (0) names a list cell, the cell linking to it will be linking to available space after J71, a dangerous but not always fatal situation.

J72 ERASE LIST STRUCTURE (0). (0) is assumed to name a list structure or a sublist structure. List (0) is erased, as are all lists with local names on list (0), and all lists with local names on them, and so on. Thus, description lists get erased, since they have local names. If the list is on auxiliary storage (Q of (0) = 6 or 7), then the list structure is erased from auxiliary, and the head, (0), is also erased. J72 works for lists in both main and auxiliary storage.

J73 COPY LIST (0). The output (0) names a new list, with the identical symbols in the cells as are in the corresponding cells of list (0), including the head. If (0) is the name of a list cell, rather than a head, the output (0) will be a copy of the remainder of the list from (0) on. (Nothing else is copied, not even the description list of (0), if it exists.) The name is local if the input (0) is local; otherwise it is internal.

J74 COPY LIST STRUCTURE (0). A new list structure is produced, the cells of which are in one to one correspondence with the cells of list structure (0). All the regional and internal symbols in the cells will be identical to the symbols in the corresponding cells of (0), as will the contents of data terms. There will be new local symbols, since these are the names of the sublists of the new structure. Description lists will be copied, if their names are local. If (0) is in auxiliary storage (Q of (0) = 6 or 7), the copy will be produced in main storage. In all cases, list structure (0) remains unaffected. The output (0) names the new list structure. It is local if the input (0) is local; it is internal otherwise.

J75 DIVIDE LIST AFTER LOCATION (0). (0) is assumed to be the name of a cell on a list. A termination symbol is put for LINK of (0), thus making (0) the last cell on the list. The output (0) names the remainder list: a new blank head followed by the string of list cells that occurred after cell (0).

J76 INSERT LIST (0) AFTER CELL (1), AND LOCATE LAST SYMBOL. List (0) is assumed to be describable. Its head is erased (if local, the symbol in the head is erased as a list structure). The string of list cells is inserted after cell (1): LINK of cell (1) is the name of the first list cell, and LINK of the last cell of the string is the name of the cell originally occurring after cell (1). The output (0) is the name of the last cell in the inserted string and H5 is set +. If list (0) has no list cells, then the output (0) is the input (1) and H5 is set -.

J77 TEST IF (0) IS ON LIST (1). Assumes (1) is the name of a cell on a list. A search is done of all cells after (1); H5 is set + if (0) is found, and set - if not.

J78 TEST IF LIST (0) IS NOT EMPTY. H5 is set - if LINK of (0) is a termination symbol, and set + if not.

J79 TEST IF CELL (0) IS NOT EMPTY. H5 is set - if SYMB of (0) is 0, and set + otherwise.

J8n FIND THE nth SYMBOL ON LIST (0), $0 \leq n \leq 9$. (Ten routines, J80-J89.) Set H5 + if the nth symbol exists, - if not. Assumes list (0) describable, so that J81 finds symbol in first list cell, etc. J80 finds symbol in head; and sets H5 - if (0) is a termination symbol.

J9n CREATE A LIST OF THE n SYMBOLS (n-1), (n-2), ..., (1), (0), $0 \leq n \leq 9$. The order is (n-1) first, (n-2) second, ..., (0) last. The output (0) is the name (internal) of the new list; it is describable. J90 creates an empty list (also used to create empty storage cells, and empty data terms).

J100 GENERATE SYMBOLS FROM LIST (1) FOR SUBPROCESS (0). The subprocess named (0) is performed successively with each of the symbols of list named (1) as input. The order is the order on the list, starting with the first list cell. H5 is safe over the generator: The sign of H5 left by the subprocess at one occurrence will exist at the next occurrence (it must be + to keep the generator going).

J101 GENERATE CELLS OF LIST STRUCTURE (1) FOR SUBPROCESS (0). The subprocess named (0) is performed successively with each of the names of the cells of list structure named (1) as input. The order (called print order) is as follows:

1. List (0) is generated first.
2. All cells of a list are generated in contiguous sequence, starting with the head.
3. After a list has been generated, the sublists of the list structure that occur on the list are generated in the order they occur on the list.
4. Lower level sequences of sublists occur after the higher level sequence is finished, and are not interpolated.
5. Each list is generated only once, at the first opportunity.

The name of the cell is output to the subprocess as (0). H5 is set + if the cell is the head of a list (so that J101 is starting to generate a new sublist). In this case J101 has already marked the sublist processed (J137), so that the head contains the processed mark and a blank symbol. The original contents of the head are one down in the list, and will occur as the next cell to be generated. In case the cell output to the subprocess is a list cell H5, is set -.

J101 has available the name of the next cell to be generated prior to executing the subprocess (which determines how manipulations of the list structure by the subprocess will affect generation).

J101 cleans up the processing marks that it puts in the list structure, returning the list structure to its original state (except as modified by the subprocess). Structures whose names have been put by the subprocess in the blank heads created by marking processed are not erased by the generator.

J101 will move in list structure (0) if it is on auxiliary.

J102 GENERATE CELLS OF TREE (1) FOR SUBPROCESS (0). The subprocess named (0) is performed successively with each of the names of the cells of the tree named (1) as input. A tree is a data list structure in which each sublist appears once and only once. The cells of each sublist are generated before going on with the superlist; the cell containing the name of the sublist occurs immediately after the sublist and all its sublists are generated. H5 is set + to the subprocess if input (0) is the head of a new sublist, and is set - otherwise. (Nothing is marked processed, since there is no need to keep track of multiple occurrences.) The name of the next cell to be generated is found before the cell is presented to the subprocess--i.e., it is possible to erase a tree with J102.

J102 will move in list structure (0) if it is on auxiliary.

AUXILIARY STORAGE PROCESSES, J105 to J109

There are two types of auxiliary storage--fast and slow--and two separate auxiliary storage systems--one for data list structures, and the other for routines.

AUXILIARY STORAGE FOR DATA LIST STRUCTURES

The system for data list structures is patterned after a file drawer. The file holds data list structures. A list structure can be filed in auxiliary storage (it is the programmer's decision whether in fast or slow storage). When filed, the structure is no longer in main storage, and all the space it used is made available (except the head--see below). The programmer must be aware of when he has filed a list structure in auxiliary, since most of the processes do not check for this. Thus, doing a J60, which locates the next symbol, on the name of filed list structure can only lead to chaos. The system determines where a list structure shall be filed, and records this information in the control word for the list

structure. This is kept in the head of the list structure--i.e., in the cell whose name is the name of the list structure. Thus, a list structure has the same name throughout a run, no matter how often it is shuffled between main and auxiliary storage: when it is in auxiliary, the cell of the name holds the control information to get the list structure back.

A filed list structure may be moved back into main storage, in which case it is no longer filed, and no trace of it remains in auxiliary. This can be done any time the name of the list structure is encountered, since the head holds the control information that locates it in auxiliary. It is also possible to copy or erase list structures in auxiliary using the regular list processes, J74 and J72. Thus, the repertoire of processes for handling auxiliary storage of data list structures consists of the following processes:

J72 ERASE LIST STRUCTURE (0). (See definition in LIST PROCESSES.)

J74 COPY LIST STRUCTURE (0). (See definition in LIST PROCESSES.)

J105 MOVE LIST STRUCTURE (0) IN FROM AUXILIARY. The control word in cell (0) determines the location of the list structure, including whether it is in fast (Q = 6) or slow (Q = 7) storage. The list structure is returned to main storage, using words from available space, and the head replaced by the head of the list structure, so that the list structure is identical to itself prior to filing (except that different list cells are used). H5 is set +. If the list structure (0) was already in main storage (Q ≠ 6 or 7), J105 does nothing and H5 is set -. The output (0) is the input (0).

J106 FILE LIST STRUCTURE (0) IN FAST AUXILIARY STORAGE. Creates a copy of list structure (0) in a unit of the fast storage (the system selects unit and the space within the unit). Erases the list structure in main storage, except for head. Creates control word (Q = 6) and places it in the head. There is no output. (If there is no space in the fast auxiliary, it is filed in the slow auxiliary.)

J107 FILE LIST STRUCTURE (0) IN SLOW AUXILIARY STORAGE. Identical to J106 except uses slow auxiliary storage (Q = 7). (If there is no space in the slow auxiliary, an error signal occurs.)

J108 TEST IF LIST STRUCTURE (0) IS ON AUXILIARY. Sets H5 + if (0) is on either fast or slow auxiliary, and H5 - in all other cases.

AUXILIARY STORAGE FOR ROUTINES

The system for routines is used by the interpreter to bring in routines for execution. It consists of an auxiliary block into which all routines stored in auxiliary (either fast or slow) are copied, and executed. All routines to be stored in auxiliary are assembled into this block during loading, so that no further assembly is needed to execute them once they have been brought in (see INTERPRETATION). Since all auxiliary routines use the same block, if an auxiliary routine uses an auxiliary routine, the copy of the higher one in main storage is destroyed when the lower one is called in. It is necessary to bring the higher auxiliary routine back into main storage again when the lower is finished. This leads to a "two call" system, in which every routine requires two reads from auxiliary storage: one to bring the routine in, and one to bring back its predecessor in the auxiliary block. It is necessary to use a storage cell, the current auxiliary routine cell, H4, to keep track of the routines in the auxiliary block, since the nesting of auxiliary routines is unlimited. The symbols stacked in H4 are names of the control words, so the routines can be called back. These considerations lead to the following restrictions:

- There is a fixed upper limit to the size of an auxiliary routine, given by the size of the block. However, this block can be set arbitrarily for each run (see type 3: BLOCK RESERVATION CARDS).

- No auxiliary routine shall modify itself during execution. If it did, the call back from auxiliary would not be the same as the initial--and now modified--copy read in from auxiliary. (There are other reasons for not allowing self-modification--e.g., recursions.)

-Subprocesses used with generators in auxiliary must be independent routines--i.e., have regional names--so that every time the generator executes the subprocess it can be brought in from auxiliary. If the subprocess were a sub-list-structure of the superroutine (with a local name), then when the generator was brought in from auxiliary it would destroy the copy of the superroutine--and with it, the subprocess--and chaos would result when the generator tried to execute the subprocess (see GENERATORS).

Routines cannot be created or manipulated during processing, so there are no routines for moving routines from main storage to auxiliary or vice versa.

CONTROL OF AUXILIARY UNITS

Both fast and slow auxiliaries are repacked when full or inefficient (see sections on machine systems). Thus the location of a list structure in auxiliary is variable. This means that copies of the control words have no validity, and hence should never be used by the programmer. It also means that the programmer has no control over which auxiliary units are used (see INITIAL LOADING).

ARITHMETIC PROCESSES, J110 to J129

All the input and output symbols in this section are the names of numeric data terms. Most operations admit only integers (P = 0, Q = 1) or floating point numbers (P = 1, Q = 1), but some admit any data term. In the arithmetic operations, if all factors are integers, then the result will be an integer. If either factor is floating point, the result will be a floating point number. Note that the prior nature of the cell holding the answer is immaterial. Thus, for example, J90 is used to create new result cells, even though it does not create data terms. None of the factors are affected by the operations, unless they are also named as the result. Any illegal operation--overflow, divide check, etc.--produces an error condition (see ERROR TRAP).

J110 (1) + (2) ⟶ (0). The number named (0) is set equal to the algebraic sum of the numbers named (1) and (2). The output (0) is the input (0); i.e., the result.

J111 (1) - (2) ⟶ (0). The number (0) is set equal to the algebraic difference between numbers (1) and (2). The output (0) is the input (0).

J112 (1) x (2) ⟶ (0). The number (0) is set equal to the product of the numbers (1) and (2). The output (0) is the input (0).

J113 (1) / (2) ⟶ (0). The number (0) is set equal to the quotient of the number (1) divided by the number (2). The output (0) is the input (0). If division is integer division, then the remainder is the data term, W11 (consequently, the remainder is unsafe over divisions).

J114 TEST IF (0) = (1). Tests identity, including prefixes, of any two data terms, named (0) and (1). Hence will always give H5 - if an integer is tested against a floating point.

J115 TEST IF (0) > (1).

J116 TEST IF (0) < (1).

J117 TEST IF (0) = 0.

J118 TEST IF (0) > 0.

J119 TEST IF (0) < 0.

J120 COPY (0). The output (0) names a new cell containing the identical contents to (0). The name is local if the input (0) is local; otherwise it is internal.

J121 SET (0) IDENTICAL TO (1). The contents of the cell named (1) is placed in the cell (0). The output (0) is the input (0).

J122 TAKE ABSOLUTE VALUE OF (0). The number (0) is modified by setting its sign +. It is left as the output (0).

J123 TAKE NEGATIVE OF (0). The number (0) is modified by changing its sign--i.e., by multiplication by -1. It is left as the output (0). (Zero is signed; J123 takes zero into minus zero.)

J124 CLEAR (0). The number (0) is set to be 0. If the cell is not a data term, it is made an integer 0. If a number, its type, integer, or floating point, is unaffected. It is left as the output (0).

J125 TALLY 1 IN (0). An integer 1 is added to the number (0). The type of the result is the same as the type of (0). It is left as the output (0).

J126 COUNT LIST (0). The output (0) is an integer data term, whose value is the number of list cells in list (0) (i.e., it doesn't count the head). If (0) = H2, J126 will count the available space list. This is the only place where H2 can be used safely by the programmer.

J127 TEST IF DATA TYPE (0) = DATA TYPE (1). Tests if P of cell (0) is the same as the P of cell (1). (Assumes (0) and (1) are data terms, hence, uses P of data term representation, which is not the same as P of instructions--see sections on machine systems.)

J128 TRANSLATE (0) TO BE DATA TYPE OF (1). The output (0) is the input (0), translated according to the data type of data term (1). This translation is not defined for all data terms. It will float integers (P = 0 to P = 1) and fix floating point numbers (P = 1 to P = 0). It can be expanded to include other P's--see sections on machine systems.

J129 PRODUCE RANDOM NUMBER IN RANGE 0 TO (0). The output (0) is a new number chosen from the uniform distribution over the interval 0 up to number (0) (the endpoint (0) is excluded). It is an integer or floating point number according to (0). It is produced by first generating a random number in the interval 0 up to 1, and then multiplying this number by (0). The random fraction is generated by multiplying the number named in storage cell W10 by a fixed number and taking the low order digits. This new number is returned to W10 to become the factor in the next random number generated. Thus, starting W10 with a specified integer leads to a fixed sequence with random properties, which can be repeated. Different random sequences, such as are needed in statistical replication, are generated by starting W10 with different initial numbers.

Note that if the input is the integer n, the selection is from the n integers, 0, 1, ..., n-1, each with probability 1/n.

DATA PREFIX PROCESSES, J130 to J139

The reason for defining the data list structure as a unit of information is to allow processes that work for the list structure as a whole. We have processes like J72, erase a list structure; J74, copy a list structure; and J140, read a list structure into the computer. One erase process is sufficient to cover almost all possible types of data. It is desirable to be able to construct additional higher IPL routines that also work for list structures. To do this requires the ability to detect and manipulate the three kinds of symbols: regional, internal, and local. This is possible (for data only) since the Q prefix is used internally to encode the symbol type with each occurrence. Upon loading data list structures (see INPUT-OUTPUT), the following coding takes place:

Q = 0 SYMB is regional.
Q = 1 Word is data term.
Q = 2 SYMB is local.
Q = 3 Unassigned.
Q = 4 SYMB is internal.
Q = 5 Word is data term (same as Q = 1).
Q = 6 List structure is in fast auxiliary storage.
Q = 7 List structure is in slow auxiliary storage.

P = 0 For all standard IPL words, and as assigned for data terms.

The only values of Q and P that appear externally are those connected with data terms. We give the others here to make it clear what processes are being performed with the data prefix processes; details can be found in the sections on machine systems.

RECURSIONS

Besides the processes implied above, it is necessary to be able to work on all parts of the list structure--e.g., in an erase, every cell must be erased. The basic technique in processing list structures is

recursion. Since a list structure is recursively defined, the kind of operations that can be defined for a list structure involve defining what is to be done to each list of the structure and then recursing through the structure. That is, the total process has the form:

-Do what you have to to this list;
-Find all the local names on this list;
-Do the total process to each sub-list-structure
 defined by these local names.

Eventually, all the lists in the list structure get processed and the recursion will stop; the recursive character of the routine and the fact that all connections in the structure are marked by local names assures this. Since, however, the name of a list can occur in many places in a list structure, there must be some device for avoiding multiple processing of the same list if this is not desired (and it must not be allowed for list structures which allow the name of a list to appear on one of its sublists). For example, in erasing a list of lists which consists of three occurrences of the same sublist--e.g., L1: 9-1, 9-1, 9-1--the sublist, 9-1, must be erased only once, not just as a matter of efficiency, but because chaos will result if an erased list is erased.

MARKING A LIST PROCESSED

The solution provided in the basic system to keep track of multiple processing is a technique for marking a list "processed": J137 (taking the name of a list as input) preserves the list, makes the head blank (Q = 4, SYMB = 0), and marks it with P = 1. Since throughout the rest of the data, P = 0, it is possible to detect if the sublist has already been processed by testing whether P = 1 (J133). The mark can be removed and the list returned to its initial condition by a restore. The blank head can hold temporary information relevant to each sublist during a list

structure process. For example, a new temporary description list could be put in the head. It would not get mixed up with the normal description list, which is one down in the push down list. Of course, this temporary description list must be cleaned up at the end, say by J15.

It is possible to avoid some of the problems of keeping track of list structure by using J101, the generator of the cells of a data list structure. J101 uses the device of marking processed--every sublist is marked processed when first presented--but much of the mechanics is buried in J101, and need not be repeated by the subprocess that uses it.

J130 TEST IF (0) IS REGIONAL SYMBOL. Tests if Q = 0 in HO.

J131 TEST IF (0) NAMES DATA TERM. Tests if Q = 1 or 5 in the cell whose name is (0).

J132 TEST IF (0) IS LOCAL SYMBOL. Tests if Q = 2 in HO.

J133 TEST IF LIST (0) HAS BEEN PROCESSED. Tests if P = 1 (and Q ≠ 1 or 5) in the cell whose name is (0). It will only be 1 if list (0) has been preserved and P = 1 put in its head by J137. This means list (0) has been processed.

J134 TEST IF (0) IS INTERNAL SYMBOL. Tests if Q = 4 in HO.

J136 MAKE SYMBOL (0) LOCAL. The output (0) is the input (0) with Q = 2. Since all copies of this symbol carry along the Q value, if a symbol is made local when created, it will be local in all its occurrences.

J137 MARK LIST (0) PROCESSED. List (0) is preserved, its head made blank (Q = 4, SYMB = 0), and P set to be 1. Restoring (0) will return (0) to its initial state. This will work even with data terms. The output (0) is the input (0).

J138 MAKE SYMBOL (0) INTERNAL. The output (0) is the input (0) with Q = 4. Best considered as "unmake local symbol."

INPUT-OUTPUT CONVENTIONS

Input and output comprise several pieces: initial loading; translation from one representation to another; reading data list structures during running; writing data list structures created during running so they can be reloaded; printing; and monitoring the running program. All of these utilize common conventions about format and designation of units.

EXTERNAL TAPES (BCD TAPES)

It is possible to use tapes for input and output, rather than the on-line card readers, punches, and printers. Such tapes are called external tapes, to distinguish them from the tapes used for auxiliary storage. An external tape is functionally identical with a deck of cards outside the IPL computer. It consists of a sequence of independent list structures (and machine language code, if it is being used for initial load). It can be removed from a computer and put on again at a later time. External tapes are not generally compatible accross different types of machines (but see sections on machine systems for details). Tapes can be used as intermediate storage, since tapes written by the write processes can later be read back in by the read processes. An external tape can hold information in any of the representations defined below.

INPUT-OUTPUT UNIT CODE

The units used for input and output are named by small integers as follows:

0 The "normal" value for an installation. This will depend on the operating system being used at the installation and the kind of machine. It will include on-line card read and punch for some signal from the console.

1-10 External tapes. The connection between these names and physical units is again dependent on the machine and the installation.

The sections on the machine system should be consulted for more information.

INPUT-OUTPUT REPRESENTATION MODE

The information being input and output is in one of several modes, each of which has an integer code:

0 = IPL standard (one IPL word per card, as represented on the coding sheet).

1 = IPL compressed (about 7 IPL words per card).

2 = IPL binary (about 20 IPL words per card).

3, 4, 5, 6, 7 = Machine language for various object machines. See sections on machine systems for further details.

IPL COMPRESSED REPRESENTATION

See sections on machine systems for information.

IPL BINARY REPRESENTATION

(See sections on machine systems for further information.) The information is put on the card in column binary, although the notation used is as if it were row binary:--e.g., 9L is the 36 bit word in the left half of the 9 row of the card. The 9 row is special:

9LP = 6 (= 7 if wish to ignore checksum).
9LD = v + 500_8, where v = word count and is at most 22.
9LA = sequence number of card in deck.
9R = checksum = (9L) + (8L) + ... + (v^{th} information word).

All the v information words, starting with 8L and working back, are considered one long string of bits. The string is divided up into units by the following heading code and convention:

Heading code (bits)

0 = end of list.

10 = IPL word: followed by Q LINK P SYMB NAME.

11 = data term: followed by Q P DATA NAME.

P and Q each coded into 3 bits.

NAME, SYMB, LINK, each coded into 1 bit (=0) if blank; or into 6 bit region plus 15 bit relative number if not blank.

DATA is coded into 30 bits.

READ AND WRITE PROCESSES, J140 to J146

These are processes that allow the input and output of data list structures during running, under the control of the program. Only data list structures, not routines, can be input or output by these processes. The form of the data list structures is identical to that of initial loading, and may be in any of the three modes of representation: IPL standard, IPL compressed, or IPL binary (if possible for the object machine). A safe storage cell, W16 for reading and W17 for writing, determines the mode. The symbol in the cell is the name of the integer data term giving the code stated earlier. The list structures are handled independently, and not as blocks (as in initial loading), and no header cards are used. No translation, assembly listing, or direct input to auxiliary (all inputs being to main storage) is possible. The unit to be used must be selected, and safe storage cells, W18 for read and W19 for write, are used for this. The symbol in the cell names the integer data term giving the unit (see INPUT-OUTPUT UNIT CODE).

J140 READ LIST STRUCTURE TO (0). A list structure on cards (or external tape) in any of the admissible forms (IPL, compressed, binary) is read into main storage, its name input to (0), and H5 set +. Blank records are treated as end-of-list-structure marks. If the first record read by J140 is blank, it is ignored. If there is no list structure (card hopper empty or end of file) then there is no input and H5 is set -. Internal symbols are assumed to already exist in the IPL computer: internal symbol 1345 is assigned address 1345.

J141 READ A SYMBOL FROM CONSOLE. Inputs a symbol or data term from the console into H0. Sets H5 + if there is an input, and - if there is not. An input data term is put in a new cell and given an internal name.

The console conventions depend on the particular machine, and the sections of the manual on machine systems should be consulted for the exact definition of J141.

J142 WRITE LIST STRUCTURE (0). (0) is assumed to name a list structure. It is punched (or written on external tape) in any of the admissible forms (IPL, compressed, IPL binary). Regional symbols are converted back to external form, adddd; internal symbols are converted directly--address 1345 to symbol 1345; and local symbols are expressed as 9dddd, where the dddd are small integers that start with 0 for each list structure. The order of writing is that of J101, so that all the symbols of a list are written consecutively. Thus there is no need for local names for list cells--i.e., no link is needed except for 0, the termination symbol.

J143 REWIND TAPE (0). The external tape named by the data term (0) is rewound.

J144 SKIP TO NEXT TAPE FILE. The external tape named in W18 is positioned past the next end of file mark.

J145 WRITE END OF FILE. The end of file mark is written on the external tape named in W19.

J146 WRITE END OF BLOCK. A blank record (appropriate to mode 1W17) is written on the external tape named in W19. (See INITIAL LOADING for use of blank records.)

MONITOR SYSTEM, J147 to J149

Three kinds of facilities are available for monitoring the running program and controlling it. First, it is possible to take a "snap shot" of the program to see what it is doing. Second, it is possible to get

"post mortem" information after a program has stopped. Third, it is possible to trace the program, printing information on each instruction as it is executed. The sections of the manual on machine systems should be consulted on the conventions for using the console to accomplish the features described below.

MONITOR POINT, Q = 3

Any instruction with Q = 3 is called a monitor point in the program. As far as execution of the program is concerned it is treated as Q = 0. However, when it is encountered, the interpreter takes the following monitoring action:

-It turns the trace on, also marking that a monitor point has occurred.

-It pushes down the safe storage cell W29 and stores the current instruction address (the name of the cell holding the instruction with Q = 3) as 1W29.

-It checks the console for the following signals:

-Terminate the program for restart (see SAVING FOR RESTART): it executes the routine named in the safe storage cell, W15, and then continues with the program. Terminating this routine with J166 accomplishes the terminate for restart.

-Terminate the program: it executes the routine named in the safe storage cell, W14, and then continues with the program. Terminating this routine with J7 accomplishes the program terminate.

-External trace mode: no trace, selective trace, full trace.

-If neither of the terminates occur, it executes the routine named in the safe storage cell, W12, and then continues the program.

-When the program list in which Q = 3 occurred is finished--i.e., when the marked routine is finished--it executes the routine named in the safe storage cell, W13.

-It then pops up W29 and continues with the program.

It is normal to mark a routine by putting the monitor mark in the head.

SNAPSHOTS

The four cells, W12, W13, W14, and W15, hold four routines, called snapshot routines. As seen above, they will be executed under various conditions associated with the monitor points, Q = 3. There is no restriction to the routine that could be executed, although the normal use is to print out various lists to see how the program is progressing. The snapshot for W15 must end with J146 to make the program terminate for restart, and the snapshot for W14 must end with J7 to stop the program.

POST MORTEM DUMP

In the event the program stops from some internal error, it is possible to print out information about the terminating condition of the machine. The routine that does this is self contained, and is therefore normally unaffected by whatever error stopped the IPL program. It is executed manually from the console. W23 is used to select the information obtained. The dump routine varies with machine, and the sections of the manual on each machine system should be consulted for details.

TRACING

There are two trace modes, "on" and "off." In addition there are three externally imposed conditions: no trace, in which the trace mode is "off" no matter what is indicated internally; selective trace, in which the trace mode is as indicated internally; and full trace, in which the trace mode is "on" no matter what is indicated internally. If the trace mode is on, then for each instruction the following information is printed:

-Level number, counting down from the initial routine as level 1.

-CIA, the current instruction address (the symbol in H1).

-Test signal, the contents of H5 (+ or -) prior to execution.

-Instruction being executed, PQ SYMB LINK (the contents of CIA).

-S, the designated symbol.

-(0), the symbol in H0 prior to execution.

-The contents of cell (0), printed in appropriate form (data term or PQ SYMB LINK).

-H3, the number of interpretation cycles since H3 was last reset. (H3 will include one count for each line of trace that would have printed had full trace been on.)

The format is as follows:

←——Level CIA——→H5 P Q SYMB LINK S (0) CONTENTS H3

The level and CIA are indented according to the level, modulo the printing interval available. The symbols are translated back into IPL representation (this is not possible on all machines). The Q of (0) is printed, indicating whether the symbol is internal or local.

TRACE MARKS

The trace mode is carried by a mark in H1. This mark encodes whether the trace mode is on or off, and also whether a monitor point occurred. On selective trace, the interpreter consults this mark each cycle (after INTERPRET Q but before INTERPRET P) and if it reads on, prints the trace information. This mark is governed by the occurrence of Q = 3, and Q = 4, in the instructions of the program. Both of these are treated as Q = 0 in determining the designated symbol. The following rules describe their function:

- If a Q = 3 is encountered, set trace on.

- If the trace is on, it remains on as we advance along a program list (always at the same level)--i.e., the trace mark propagates down a list.

- When the program descends a level, the trace is always off, a priori --i.e., the trace mark does not propagate down levels.

- If a Q = 4 is encountered, the trace mark is set to equal the trace mark one level up--i.e., the trace is propagated down a level by Q = 4.

- In ascending Hl is restored and the trace mark of the higher level again becomes operative.

These rules mean the following: putting Q = 3 in the head of a program list will cause that list to be traced. Putting Q = 4 in the head of a program list will cause that list to be traced, if the program list calling upon it is tracing. Hence, putting Q = 4 in the heads of all local sublists of a routine makes the routine a tracing unit; all instructions of the routine will trace if Q = 3 in the head of the routine; the whole routine will trace conditionally if Q = 4 is put in the head; and none will trace if Q ≠ 3 or 4 in any instruction.

The Q's can be written in the routines at the time of coding by the programmer. Since Q = 3 and 4 are equivalent to Q = 0, they can often be put in without adding space to the system. If the head of a routine does not have Q = 0, then an additional instruction, say with SYMB = JO, is necessary. Since the routines that are traced are changed often, it is desirable to specify the Q's at the beginning of each run, without permanently marking the routines. This can be done by means of three IPL processes:

J147 MARK ROUTINE (0) TO TRACE. If Q = 0, 3, or 4, in cell (0), changes Q to be 3. If not, preserves (0), and places the instruction 03 J0 in cell (0).

J148 MARK ROUTINE (0) TO PROPAGATE TRACE. Identical to J147 except uses Q = 4.

J149 MARK ROUTINE (0) NOT TO TRACE. If Q = 3 or 4, in cell (0), puts Q = 0; unless SYMB is also J0, in which case J149 restores (0). If Q ≠ 3 or 4, does nothing.

PRINT PROCESSES, J150 to J161

Two classes of printing processes are provided, those for printing IPL units of data (symbols, lists, list structures, data terms) and those for composing and printing a line of information. Each of the printing processes is related to:

-the unit that will print, given by the integer data term named in the safe storage cell W20. (See INPUT-OUTPUT UNIT CODE.)

-the column in which the leftmost character of the format will print, given by the integer data term named in the safe storage cell W21. The columns run from 1 at the far left of the page to 120 at the right. The entire format of 37 spaces must fit onto the page, independent of whether particular fields are going to print or not. If the column number shifts the format too far to the right or left, the format will print at the rightmost or leftmost possible position.

-the line spacing that will occur between a line and the previous printing, given by the integer data term named in the safe storage cell W22. The spacing code is the following:

0 if spacing is suppressed--i.e., print on the same line;
1 if start printing on the next line;
2 if skip one line before starting to print;
3 if skip to next page, and start printing at the top.

Not all the object machines have the full flexibility, so the sections on machine systems should be consulted.

PRINTING IPL UNITS OF DATA

J150 PRINT LIST STRUCTURE (0). The contents of all the cells of the data list structure named (0) are printed. Regional symbols are translated to the form adddd; internals are printed as the decimal integer corresponding to the address; and local symbols are translated to the form 9dddd , where dddd are small integers starting with 0 for each list structure. All data terms are translated to their external form.

Each list of the list structure is printed in an uninterrupted vertical column, so that neither LINK nor the NAME of any list cell is ever printed. If the SYMB names a data term, then this data term is printed to the right on the same line. If the NAME is a local name (which can occur only in printing the head of a sublist), its corresponding address is printed to the left. The local name, 9dddd, bears no relation to this address. The full format is shown below. (Column 1 corresponds to the column specified by the integer data term named in W21.)

-column:	12345	67	89111 012	111 345	11112 67890	2222 1234	22 56	22233333333 78901234567
	addr. of NAME if local		NAME	PQ	SYMB		PQ	DATA if SYMB names data term

The lists of the list structure are printed in the order of J101.

J151 PRINT LIST (0). The contents of all the cells of the list named (0) are printed in an uninterrupted vertical column. The format is the same as that of J150, except that local symbols are not translated to form 9dddd; but instead their addresses are printed, and the Q = 2 identifies them as locals.

J152 PRINT SYMBOL (0). The symbol (0) is printed. The format is the same as J150, where (0) is placed at SYMB, and if it names a data term, this is printed to the right. Locals are handled as in J151.

J153 PRINT DATA TERM (0) WITHOUT NAME OR TYPE. (0) is assumed to name a data term (if not, nothing is printed and the designated spacing occurs.) The DATA part of the data term is printed in its location in the format of J150, but neither (0) nor the PQ of the data term is printed. This process, in connection with the suppression of spacing, allows alphanumeric characters to be placed along a line in any pattern.

LINE PRINTING

In addition to the output unit, left margin, and line spacing controls given previously, line printing is controlled by:

-the current print line, named by the symbol in the safe storage cell W24. Print lines are reserved during loading (see TYPE 3: BLOCK RESERVATION CARDS), when the symbol naming the line and the size of the line are specified. All print lines start with column 1; the specified line size determines the right margin of the line.

-the current column at which information will be entered in the current print line, given by the integer data term named in the safe storage cell W25. Information can be entered either left-justified--1W25 specifying the position of the first character of the field being entered--or right-justified--1W25 specifying the position of the last character of the field. After an entry, 1W25 is set to the next column following the last character of the field entered, and H5 is set +. If the entire field cannot be entered because it would exceed the line size, no information is entered, 1W25 is left unchanged, and H5 is set -.

Symbols are entered on the print line in compact form; that is, as A1, B10, etc. Data terms are entered according to the following rules:

Integers: Leading zeros are eliminated. Plus signs are not entered, but minus signs are. Examples: "00273" entered as "273" (3 columns); "-01050" entered as "-1050" (5 columns).

Floating Point: The entire number is entered, signed value followed by signed exponent. Only minus signs are entered. Examples: ".505135x10^{5}" entered as "505135 05" (9 columns); ".14x10^{-16}" entered as "140000 -16" (10 columns).

Alphanumeric: Trailing blanks--that is, blanks that follow some non-blank character and are not followed by some non-blank character--are eliminated. Example: "_A_F_" entered as "_A_F" (4 characters);"_____" entered as "_____" (5 characters.)

All Other: The entire value of the data term is entered as a ten digit octal integer. Example: "0000567234" entered as "0000567234".

J154 CLEAR PRINT LINE. Print line 1W24 is cleared and the current entry column, 1W25, is set equal to the left margin, 1W21.

J155 PRINT LINE. Line 1W24 is printed, according to spacing control 1W22. The print line is not cleared.

J156 ENTER SYMBOL (0) LEFT-JUSTIFIED. Symbol (0) is entered in the current print line with its leftmost character in print position 1W25, 1W25 is advanced to the next column after these in which (0) is entered, and H5 is set +. If (0) exceeds the remaining space in the print line, no characters are entered, 1W25 is not advanced, and H5 is set -.

J157 ENTER DATA TERM (0) LEFT-JUSTIFIED. Data term (0) is entered in the current print line with its leftmost character in print position 1W25, 1W25 is advanced, and H5 set +. If (0) exceeds the remaining space, no entry is made and H5 is set -.

J158 ENTER SYMBOL (0) RIGHT-JUSTIFIED. Symbol (0) is entered as in J156, except that 1W25 names the print position of the last character of the field. If entry is possible, 1W25 is advanced and H5 set +; if not, H5 is set -.

J159 ENTER DATA TERM (0) RIGHT-JUSTIFIED. Data term (0) is entered as in J157, except that 1W25 names the print position of the last character of the field. If entry is possible, 1W25 is advanced and H5 set +; if not, H5 is set -.

J160 TAB TO COLUMN (0). (0) is taken as the name of an integer data term. Current entry column, 1W25, is set equal to 1W21 + (0).

J161 INCREMENT COLUMN BY (0). (0) is taken as the name of an integer data term. Current entry column, 1W25, is set equal to 1W25 + (0).

In addition to lines composed using these primitives, complete headings and partial lines can be specified at loading (see TYPE 3: BLOCK RESERVATION CARDS).

INITIAL LOADING

To use IPL, the computer must first be turned into an IPL computer by loading the IPL interpretive system, either from cards or tape. Then the IPL computer must load the user's program into the total available space. This requires a deck of cards (or external tape) containing the IPL words, as well as some special cards to identify the program and to define the regional symbols that are used in the program. These special cards are called type cards, and they are identified by a non-zero digit in the TYPE column (column 41). The cards that have been described up till now

have all been type 0 cards (TYPE may be left blank on type 0 cards). The following additional types are recognized.

TYPE 1: COMMENT CARDS.

All columns (except 41) are available for anything the programmer wishes to write. Comment cards are listed on the assembly listing, but have no other effect on the loading process.

TYPE 2: REGION CARDS

All the regional symbols with the same initial letter constitute a region. Each region is translated into an interval of addresses in the computer. For example, the R region might correspond to addresses 1000 to 1018: then R0 would correspond to 1000, R1 to 1001, and R18 to 1018. The interval for each region must be specified at loading time by a type 2 card. One type 2 card is used for each region. The first symbol of the region--e.g., R or R0--is put in the NAME field, SYMB is left blank, and the number of cells in the interval is put in LINK. The IPL computer assigns the next block of contiguous cells available in the loading process to this region. Thus the origin is assigned arbitrarily. There is normally no need to know the origin, since all regional symbols are translated back into the letter-number form for output. However, for some purposes it may be desirable to specify the origin. This is done by placing the address of the origin in SYMB. The origin can also be specified in terms of another region, provided the other region is first defined. See the sections on machine systems for further details.

Examples:	TYPE	NAME	PQ	SYMB	LINK
Ten symbols for the M region M0 to M9:	2	M0			10
Starting the M region at address 1000:	2	M0		1000	10
Making M0 synonymous with B37	2	M0		B37	10

There are 36 possible regions: A B ... Z + - / = . , $ * (and). Three regions, H, J, and W, have already been permanently specified for the basic system. The regions are all part of the IPL computer. All the regional symbols that are not actually used during loading--i.e., do not occur as some NAME, SYMB, or LINK on the coding sheet--are made part of the available space for the IPL computer at the end of the loading, and thus lose their regional character. All regional symbols mentioned (in SYMB or LINK) but not defined (in NAME) are blank. If the exact limits of intervals are specified, then the intervals corresponding to different regions may overlap and need not be contiguous. If origins are assigned by the IPL computer, the regions are adjacent and disjoint.

TYPE 3: BLOCK RESERVATION CARDS:

It is necessary to reserve blocks of space for various purposes, and sometimes desirable to set a number of regional symbols to be blank without mentioning them, say for later input. Type 3 cards are used to accomplish this. They fit the same format as type 2 cards: SYMB indicates the base, if appropriate; and LINK indicates the size of the block (NAME is always blank). Q is used to indicate the purpose of the block, according to the following table:

Q = 0 Reserve regional symbols. If SYMB is A5 and LINK is 10, then A5 through A14 inclusive are set blank, and will not be put back on available space. The symbols reserved must have previously been covered by a type 2 card.

Q = 1 Reserve print line. SYMB is the regional symbol naming the line. LINK is the number of words to be set aside for the print line. (These words are taken from available storage, not from the region. See sections of the manual on machine systems for details of how SYMB refers to the line, and of how many characters are stored per word in a particular machine.) If P is not 0 or blank, the immediately following record is a Hollerith record to be loaded into the block starting with column 1 into the first character position and continuing to the end of the block.

Q = 2 Reserve primitive block. Space can be provided with known addresses to hold additional primitives. See the sections of the manual on the machine systems.

Q = 3 Reserve auxiliary routines buffer. This space is used by all the routines on auxiliary storage. Its size limits the maximum size of a routine on auxiliary. See AUXILIARY STORAGE.

Q = 4 Specify available space. If this card is absent from the loading deck, or if it is present with LINK blank, all the available space possible will be assigned, including all the unmentioned and unreserved regional symbols. If LINK is specified, only that much space will be provided, and all in one continuous block if possible. In any event, various scraps of space (unmentioned symbols, interstices between regions, etc.) will be put on the end of available space. This is often useful in debugging.

TYPE 4: LISTING CARDS

Type 4 cards represent printed output from computers which must output via cards and therefore require a way of distinguishing printed output (J150's) from punched output (J142). They are generated by the computer, and not by the programmer. If input, they are listed on the assembly listing, but have no other effect on loading.

TYPES 5, 6, AND 7: HEADER CARDS

Data or routines are loaded in a series of separate blocks, each of which is preceded by a header card that governs the loading process.

The input deck may be in several modes: IPL standard (one word per card); IPL compressed; IPL binary; or one of the machine codes. It may also come from one of several input units: tapes or the card reader. It is possible to specify an output during initial loading, which serves the purpose of translating from one form, such as IPL standard, to another, such as IPL binary, for subsequent use. An assembly listing is usually produced during loading, to indicate the machine location assigned to each IPL word in order to facilitate debugging. This may be suppressed, if desired.

The block may contain routines or data, and it is necessary to specify which, as the P and Q codes are treated differently. Also, the block may go into main storage (type 5), or to one of the auxiliary storages (type 6 for fast, type 7 for slow). In the latter case, it is necessary to specify the units available to the auxiliary system, although it is not necessary (or possible) to specify what units are used for what list structures.

Finally, a header card is used to specify that loading has finished, and to indicate where the program starts.

The codes for these various items of information are given in the following table:

TYPE: Type of storage to be used:

5 = main storage
6 = fast auxiliary
7 = slow auxiliary

NAME: Name of storage unit:

Blank for main storage (type 5)
See sections on machine systems for designation of auxiliary storage units on the various object machines (types 6 and 7).

P: Input mode:

0 = IPL standard (1 word per card)
1 = IPL compressed
2 = IPL binary
3, 4, 5, 6, 7 — Machine language for various object machines. See sections on machine systems for details.

Q: Type of input:

0 = Routines. Internal symbols are considered pure symbolics. Undefined internal symbols (internal symbols not in the internal symbol table) are assigned equivalents from available space.
1 = Data list structures. Internal symbols are considered pure symbolics. Undefined internal symbols are assigned equivalents from available space.
2 = Routines. Internal symbols are considered pure symbolics. The internal symbol table is reset (thus undefining all internal symbols) and undefined internal symbols are assigned equivalents from available space.
3 = Data list structures. Internal symbols are considered pure symbolics. The internal symbol table is to be reset and undefined internal symbols are to be assigned equivalents from available space.
4 = Routines. Internal symbols are considered machine addresses (and so no equivalent need be assigned).
5 = Data list structures. Internal symbols are considered machine addresses.

P or Q blank are interpreted as P or Q = 0.

SYMB: Input unit:

0 = "normal" for installation. May be left blank.
1-10 for external tapes

If SYMB contains a regional symbol, loading terminates and the program begins at the named routine.

LINK: Output mode: of form bbbcd

b = blank (columns 57-59)
c = 0 or blank if assembly listing desired.
1 or any other character, if assembly listing to be suppressed.
d = 0 or blank if no output desired
1 if output in IPL compressed.
2 if output in IPL binary
All other input-output modes are illegal.
The output unit is the one given in W19.
Each block of IPL compressed or IPL binary output ends with a blank record appropriate to that mode (see ALTERNATE INPUT UNITS).

TYPE 9: FIRST CARD

The very first card of a program to be loaded must be a type 9 card. Except for the type designation, a type 9 card is treated like a comment card (type 1) and may be used to identify the program. The use of type 9 cards allows several programs to be stacked on an external tape for batch execution.

ALTERNATE INPUT UNITS

As indicated on the type 5, 6, or 7 header card, it is possible to read a block of input from an input unit other than the primary one. The unit on which the first type 9 card is read is the controlling unit. If any header card read on that unit refers to any other input unit (in SYMB), the block that would follow that header card is read from the alternate unit. The header card on the controlling unit completely specifies the block--input mode, type of input, destination in storage, output mode and unit. Discrepancy between the header and the actual information on the alternate input unit causes a loading error. The block on the alternate unit is terminated by a blank record or by a header card, at which time the next record on the controlling unit is read. Any non-type 0 cards on the alternate unit, except header cards that terminate blocks, are treated as type 1 cards.

ASSEMBLY LISTING

As indicated on the type 5, 6, or 7 header card, it is possible to obtain an assembly listing of the program being loaded. This consists of a replica of the cards being input alongside the machine locations they correspond to with the assembled contents in decimal. The assembly listing of type 0 and 1 cards can be suppressed for any block by a signal in the LINK of the header card. Other type cards are printed under all conditions.

LOADING DECK

The IPL deck for initial loading consists of the following parts in order:

1. One type 9 card.
2. All type 2 cards with exact limits, if any, in any order.
3. All type 3 cards with exact limits, if any, in any order.
4. All type 2 cards giving only region size, if any, in any order.
5. All type 3 cards giving only block size, if any, in any order.

   Only regions and blocks defined by these cards (plus the H, J and W regions) exist for the IPL computer this run. The type 2 and 3 cards with exact limits must go first to insure that their cells will be available.

6. Blocks of data and routines, in any order.

   Each block is preceded by an appropriate type 5, 6, or 7 card. For IPL standard and IPL compressed cards, the end of the block is signalled by the next type 5, 6, or 7 card. For binary and machine modes, a special termination signal is required in the last card (see machine systems for details).

   The input unit that initiates loading--the one containing the type 9 card--becomes the controlling unit. If a type 5, 6, or 7 card indicates that the block is to come from another input unit (SYMB of the type card), then after the block is through loading, the next type card is picked up from the original controlling unit.

7. A final type 5 card with a regional symbol for SYMB to terminate loading and start the program.

   Any violation of this order will result in an on-line printed error message.

(It may be noted that the process of loading an IPL program is a one-pass symbolic assembly, hence the need to define symbols before loading the data and routines.)

In loading type 0 cards, the IPL computer assigns locations from available space to local symbols. A list of local symbol definitions is kept. The list is cleared whenever a regional or internal symbol is encountered in NAME (the start of a new list structure), and at the end of the loading process.

Internal symbols are likewise assigned locations from available space and thus redefined. A list of internal symbol definitions is kept. This list is cleared upon the appropriate signal from a header card, and at the end of the loading process. The programmer knows the correspondence of input symbols and their redefinitions only by means of the assembly listing. Any subsequent output of internal symbols will be in terms of their redefinitions.

Regional cells may be defined more than once in the loading sequence. The latest occurring definition is the effective one. (This is often useful in making corrections.)

INPROCESS LOADING

More routines and data can be loaded during interpretation of an IPL program. All options as to mode, unit, etc., available during initial loading are present during inprocess loading. No new regions or blocks can be specified during inprocess loading. (Not all object machines have full flexibility, so the sections on machine systems should be consulted.)

J165 LOAD ROUTINES AND DATA. More routines and data are read, with the input unit specified by 1W18 as the controlling unit, The load deck consists of header cards (type 5, 6, or 7) each followed by a block of routines or data (except when an alternate input unit is specified) and terminated by a type 5 card with a regional SYMB. The routine named as SYMB on the final type 5 card is taken as the next routine to be interpreted. If there are no routines or data, interpretation continues with the instruction following J165.

SAVE FOR RESTART

A primitive process is provided that allows a running program to be terminated at any point, read out on tape or cards, and restarted again by reading the tape or cards back into the machine. This process may be externally executed at a monitor point (see MONITOR SYSTEM) or may be put in the program at any point.

J166 SAVE ON UNIT (0) FOR RESTART. The entire contents of main storage is written onto a single external tape (or punched on cards, according to the unit named by data term (0).) Auxiliary storage is also saved in some form. Identification of the auxiliary units and external tapes being used by the IPL computer are printed out. Then the program stops. If the specified auxiliary units and external tapes are provided, and the tape (deck) is input under control of a one card loader (specified for each machine system), the program will commence at the instruction following J166. (See sections on machine systems for more details.)

J166 does not save external tapes. The programmer saving for restart must provide routines to record the position of external tapes before executing J166 and to reposition those tapes where continuing after restart.

An additional primitive is provided for use in repositioning tapes:

J167 SKIP LIST STRUCTURE. A single list structure on cards or external tape (as specified by 1W18) in any of the admissible forms--IPL, compressed, binary--(as specified by 1W16) is skipped over, and H5 set +. A blank record is treated as an end-of-list-structure mark. Immediately subsequent blank records are ignored. If there is no list structure (card hopper empty or end of file), then H5 is set -. J167 behaves as does J140, except that the structure is not entered into storage.

It is anticipated that "save for restart" will be used to provide a fast-loading version of checked-out routines, to which additional routines to be debugged can be added by "load more routines and data."

ERROR TRAP, J170

Many different error conditions can occur during processing by the IPL computer--for example, multiple definition of local symbols within a list structure during loading, specifying other than a data term as operand for an arithmetic process. These conditions cause a system error trap to occur. The action taken upon trapping depends on the routine currently associated with the particular error condition. (See sections on machine systems for the normal error conditions and associated trapping actions.) When an error condition occurs, the following steps take place:

- The safe storage cell W27 is preserved and the CIA at the time of the trap is stored as 1W27. This is the name of the instruction word designating the trapped process, except for primitives executed as links, when it is the name of the primitive.

- The safe storage cell W28 is preserved and the symbol associated with the trapping condition, the trap attribute, is stored as 1W28.

- The description list of W26 (that is, the list 1W26) is searched (as in J10) for the trap attribute. If the trap attribute exists as an attribute of W26, its value names the routine to be executed as the trapping action. That routine is executed. If no value is associated with the trap attribute, the routine associated with the attribute 'internal zero' (the symbol'0') is executed as the trapping action. If no value is associated with 'internal zero', no trapping action is taken.

  The trapping action is executed as a subprocess of the trapped process--that is, as though it were designated directly in the trapped process. Because H0, H5, and the W's are not disturbed by the error trap mechanism, the trapping action can repeat the trapped process under its own control if desired. If the trapping action is marked with Q=4, it will trace conditionally.

- When the trapping action terminates, W27 and W28 are restored and interpretation continues with the process following the trapped process.

The standard description list form of W26 allows any trapping action to be modified or disabled by assigning a different value to the trap attribute. Also, additional trap attributes and associated actions can be added. A primitive process is provided to take trapping action at any point in the program.

J170 TRAP ON (0). J170 preserves W27 and W28, stores the appropriate CIA in W27 and (0) in W28, searches the description list of W26 for the attribute (0), and executes as a subprocess of the process designating J170 the routine named by the associated value. If (0) is not an attribute of W26, the routine associated with 'internal zero' is executed. If 'internal zero' is not an attribute of W26, no trapping action is taken. J170 then restores W27 and W28 and terminates.

INDEX

SHORT LIST OF BASIC PROCESSES

* means sets H5

J0 No operation
J1 Execute (0) after restoring H0
*J2 TEST (0) = (1)
*J3 Set H5 -
*J4 Set H5 +
*J5 Reverse sense of H5
J6 Reverse (0) and (1)
J7 Halt, proceed on GO
J8 Restore H0
J9 ERASE cell (0)

*J10 FIND value of attribute (0) of (1)
J11 Assign (1) as value of attribute (0) of (2)
J12 Add (1) at front of value list of attribute (0) of (2)
J13 Add (1) at end of value list of attribute (0) of (2)
J14 ERASE attribute (0) of (1)
J15 ERASE all attributes of (0)
*J16 FIND attribute of (0) randomly

J17 GEN set up: context (0), suppr. (1)
*J18 Execute subprocess of GEN
*J19 GEN clean up

J2n MOVE (0)-(n) to W0-Wn
J3n Restore W0-Wn
J4n Preserve W0-Wn
J5n Preserve W0-Wn; MOVE (0)-(n) to W0-Wn

*J60 LOCATE next symbol after cell (0)
*J61 LOCATE last symbol on list (0)
*J62 LOCATE (0) on list (1)
J63 INSERT (0) before symbol in cell (1)
J64 INSERT (0) after symbol in cell (1)
J65 INSERT (0) at end of list (1)
J66 INSERT (0) at end if not on list (1)
J67 Replace (1) by (0) on list (2) (lst)
*J68 DELETE symbol in cell (0)
*J69 DELETE (0) from list (1) (lst)
*J70 DELETE last symbol from list (0)
J71 ERASE list (0)
J72 ERASE list structure (0)
J73 COPY list (0)
J74 COPY list structure (0)
J75 Divide list after location (0); name of remainder is output (0)
*J76 INSERT list (0) after (1), LOCATE last
*J77 TEST if (0) is on list (1)
*J78 TEST if list (0) is not empty
*J79 TEST if cell (0) is not empty
*J8n FIND the nth symbol on list (0)
J9n Create list of n symbols, (n-1) to (0)

*J100 GEN symbols on list (1) for (0)
*J101 GEN cells of list structure (1) for (0)
*J102 GEN cells of tree (1) for (0)
J103
J104

*J105 MOVE list structure (0) in from aux.
J106 File list structure (0) in fast aux.
J107 File list structure (0) in slow aux.
*J108 TEST if list structure (0) is on aux.
J109

J110 (1) + (2)—>(0), leave (0)
J111 (1) - (2)—>(0), leave (0)
J112 (1) x (2)—>(0), leave (0)
J113 (1) / (2)—>(0), leave (0)

*J114 TEST if (0) = (1)
*J115 TEST if (0) > (1)
*J116 TEST if (0) < (1)
*J117 TEST if (0) = 0
*J118 TEST if (0) > 0
*J119 TEST if (0) < 0
J120 COPY (0)
J121 Set (0) identical to (1), leave (0)
J122 Take absolute value of (0), leave (0)
J123 Take negative of (0), leave (0)
J124 Clear (0), leave (0)
J125 Tally 1 in (0), leave (0)
J126 Count list (0)
*J127 TEST if data type (0) = data type (1)
J128 Translate (0) to be data type of (1)
J129 Produce random number between 0 and (0)

*J130 TEST if (0) is regional symbol
*J131 TEST if (0) names data term
*J132 TEST if (0) is local symbol
*J133 TEST if list (0) has been processed
*J134 TEST if (0) is internal symbol
J135
J136 Make (0) local, leave (0)
J137 Mark list (0) processed, leave (0)
J138 Make (0) internal, leave (0)
J139

*J140 Read list structure to (0)
*J141 Read symbol from console to (0)
J142 Write list structure (0)
J143 Rewind tape (0)
J144 Skip to next tape file
J145 Write end of file
J146 Write end of block

J147 Mark routine (0) to trace
J148 Mark routine (0) to propagate trace
J149 Mark routine (0) to not trace

J150 Print list structure (0)
J151 Print list (0)
J152 Print symbol (0)
J153 Print data term (0) w/o name or type
J154 Clear print line
J155 Print line
*J156 Enter symbol (0) left-justified
*J157 Enter data term (0) left-justified
*J158 Enter symbol (0) right-justified
*J159 Enter data term (0) right-justified
J160 Tab to column (0)
J161 Increment column by (0)
J162
J163
J164

J165 Load routines and data
J166 Save on unit (0) for restart
*J167 Skip list structure
J168
J169

J170 Trap on (0)

IPL INSTRUCTION: PQ SYMB LINK

P is operation code
- P=0 Execute S
- P=1 Input S (after preserving H0)
- P=2 Output to S (then restore H0)
- P=3 Restore (pop up) S
- P=4 Preserve (push down) S
- P=5 Replace (0) by S
- P=6 Copy (0) in S
- P=7 Branch to S if H5 -

Q is designation code
- Q=0 S=SYMB
- Q=1 S=symbol in cell named SYMB
- Q=2 S=symbol in cell named in cell named SYMB
- Q=3 S=SYMB; start selective trace
- Q=4 S=SYMB; continue selective trace

SYMB is symbol operated on by Q
LINK is address of next instruction (0 for end of routine)

SYSTEM STORAGE CELLS

H0	Communication cell
H1	Current instruction address cell
H2	Available space list
H3	Tally of interpretation cycles
H4	Current auxiliary routine cell
H5	Test cell
W0-W9	Common working storage
W10	Random number control cell
W11	Integer division remainder
W12	Monitor start cell (Q=3)
W13	Monitor end cell (Q=3)
W14	Monitor terminate cell
W15	Monitor save for restart cell
W16	Input mode cell
W17	Output mode cell
W18	Read unit cell
W19	Write unit cell
W20	Print unit cell
W21	Print column cell
W22	Print spacing cell
W23	Post mortem dump cell
W24	Print line cell
W25	Print entry column cell
W26	Error trap cell
W27	Trap address cell
W28	Trap symbol cell
W29	Monitor point address cell

IPL DATA: PQ SYMB LINK

Q=0 Standard list cell
- P is irrelevant
- SYMB is symbol
- LINK is address of next list cell (0 for end of list)

Q=1 Data term

	± PQ	SYMB	LINK
Decimal integer	1	dddd	dddd
Floating point	11	ddddd	d ±ee
Alphanumerical	21	aaaaa	
Octal	31	ddddd	ddddd

TYPE CARDS

0 (blank) Routines and data
1 Comments
2 Region definition
- NAME=regional symbol
- SYMB=origin (if given)
- LINK=size

3 Block reservation
- Q=0 Reserve regional symbols
- Q=1 Reserve print line
- Q=2 Reserve primitive block
- Q=3 Reserve auxiliary buffer
- Q=4 Specify available space

4 Listing cards
5 Main storage header
6 Fast auxiliary storage header
7 Slow auxiliary storage header
- NAME=name of storage unit
- P =input mode
  - P=0 IPL standard
  - P=1 IPL compressed
  - P=2 IPL binary
- Q =type of input
  - Q=0 Routines; internals symbolic
  - Q=1 Data; internals symbolic
  - Q=2 Routines; internals symbolic; reset internal symbol table
  - Q=3 Data; internals symbolic; reset internal symbol table
  - Q=4 Routines; internals absolute
  - Q=5 Data; internals absolute
- SYMB=input unit
  - 0 = "normal" for installation
  - 1-10 = external tapes
- Regional SYMB names first routine (terminate loading)
- LINK=output mode (of form bbcd)
  - b=blank (columns 57-59)
  - c=non-zero if assembly listing
  - d=1 output in IPL compressed
  - d=2 output in IPL binary

9 First card

SHORT LIST OF BASIC PROCESSES

* means sets H5

J0 No operation
J1 Execute (0) after restoring H0
*J2 TEST (0) = (1)
*J3 Set H5 -
*J4 Set H5 +
*J5 Reverse sense of H5
J6 Reverse (0) and (1)
J7 Halt, proceed on G0
J8 Restore H0
J9 ERASE cell (0)

*J10 FIND value of attribute (0) of (1)
J11 Assign (1) as value of attribute (0) of (2)
J12 Add (1) at front of value list of attribute (0) of (2)
J13 Add (1) at end of value list of attribute (0) of (2)
J14 ERASE attribute (0) of (1)
J15 ERASE all attributes of (0)
*J16 FIND attribute of (0) randomly

J17 GEN set up: context (0), suppr. (1)
*J18 Execute subprocess of GEN
*J19 GEN clean up

J2n MOVE (0)-(n) to W0-Wn
J3n Restore W0-Wn
J4n Preserve W0-Wn
J5n Preserve W0-Wn; MOVE (0)-(n) to W0-Wn

*J60 LOCATE next symbol after cell (0)
*J61 LOCATE last symbol on list (0)
*J62 LOCATE (0) on list (1)
J63 INSERT (0) before symbol in cell (1)
J64 INSERT (0) after symbol in cell (1)
J65 INSERT (0) at end of list (1)
J66 INSERT (0) at end if not on list (1)
J67 Replace (1) by (0) on list (2) (lst)
*J68 DELETE symbol in cell (0)
*J69 DELETE (0) from list (1) (lst)
*J70 DELETE last symbol from list (0)
J71 ERASE list (0)
J72 ERASE list structure (0)
J73 COPY list (0)
J74 COPY list structure (0)
J75 Divide list after location (0); name of remainder is output (0)
*J76 INSERT list (0) after (1), LOCATE last
*J77 TEST if (0) is on list (1)
*J78 TEST if list (0) is not empty
*J79 TEST if cell (0) is not empty
*J8n FIND the nth symbol on list (0)
J9n Create list of n symbols, (n-1) to (0)

*J100 GEN symbols on list (1) for (0)
*J101 GEN cells of list structure (1) for (0)
*J102 GEN cells of tree (1) for (0)
J103
J104

*J105 MOVE list structure (0) in from aux.
J106 File list structure (0) in fast aux.
J107 File list structure (0) in slow aux.
*J108 TEST if list structure (0) is on aux.
J109

J110 (1) + (2)—>(0), leave (0)
J111 (1) - (2)—>(0), leave (0)
J112 (1) x (2)—>(0), leave (0)
J113 (1) / (2)—>(0), leave (0)

*J114 TEST if (0) = (1)
*J115 TEST if (0) > (1)
*J116 TEST if (0) < (1)
*J117 TEST if (0) = 0
*J118 TEST if (0) > 0
*J119 TEST if (0) < 0
J120 COPY (0)
J121 Set (0) identical to (1), leave (0)
J122 Take absolute value of (0), leave (0)
J123 Take negative of (0), leave (0)
J124 Clear (0), leave (0)
J125 Tally 1 in (0), leave (0)
J126 Count list (0)
*J127 TEST if data type (0) = data type (1)
J128 Translate (0) to be data type of (1)
J129 Produce random number between 0 and (0)

*J130 TEST if (0) is regional symbol
*J131 TEST if (0) names data term
*J132 TEST if (0) is local symbol
*J133 TEST if list (0) has been processed
*J134 TEST if (0) is internal symbol
J135
J136 Make (0) local, leave (0)
J137 Mark list (0) processed, leave (0)
J138 Make (0) internal, leave (0)
J139

*J140 Read list structure to (0)
*J141 Read symbol from console to (0)
J142 Write list structure (0)
J143 Rewind tape (0)
J144 Skip to next tape file
J145 Write end of file
J146 Write end of block

J147 Mark routine (0) to trace
J148 Mark routine (0) to propagate trace
J149 Mark routine (0) to not trace

J150 Print list structure (0)
J151 Print list (0)
J152 Print symbol (0)
J153 Print data term (0) w/o name or type
J154 Clear print line
J155 Print line
*J156 Enter symbol (0) left-justified
*J157 Enter data term (0) left-justified
*J158 Enter symbol (0) right-justified
*J159 Enter data term (0) right-justified
J160 Tab to column (0)
J161 Increment column by (0)
J162
J163
J164

J165 Load routines and data
J166 Save on unit (0) for restart
*J167 Skip list structure
J168
J169

J170 Trap on (0)

IPL INSTRUCTION: PQ SYMB LINK

P is operation code
- P=0 Execute S
- P=1 Input S (after preserving H0)
- P=2 Output to S (then restore H0)
- P=3 Restore (pop up) S
- P=4 Preserve (push down) S
- P=5 Replace (0) by S
- P=6 Copy (0) in S
- P=7 Branch to S if H5 -

Q is designation code
- Q=0 S=SYMB
- Q=1 S=symbol in cell named SYMB
- Q=2 S=symbol in cell named in cell named SYMB
- Q=3 S=SYMB; start selective trace
- Q=4 S=SYMB; continue selective trace

SYMB is symbol operated on by Q
LINK is address of next instruction (0 for end of routine)

SYSTEM STORAGE CELLS

H0	Communication cell
H1	Current instruction address cell
H2	Available space list
H3	Tally of interpretation cycles
H4	Current auxiliary routine cell
H5	Test cell
W0-W9	Common working storage
W10	Random number control cell
W11	Integer division remainder
W12	Monitor start cell (Q=3)
W13	Monitor end cell (Q=3)
W14	Monitor terminate cell
W15	Monitor save for restart cell
W16	Input mode cell
W17	Output mode cell
W18	Read unit cell
W19	Write unit cell
W20	Print unit cell
W21	Print column cell
W22	Print spacing cell
W23	Post mortem dump cell
W24	Print line cell
W25	Print entry column cell
W26	Error trap cell
W27	Trap address cell
W28	Trap symbol cell
W29	Monitor point address cell

IPL DATA: PQ SYMB LINK

Q=0 Standard list cell
- P is irrelevant
- SYMB is symbol
- LINK is address of next list cell (0 for end of list)

Q=1 Data term

	± PQ	SYMB	LINK
Decimal integer	1	dddd	dddd
Floating point	11	ddddd	d ±ee
Alphanumerical	21	aaaaa	
Octal	31	ddddd	ddddd

TYPE CARDS

0 (blank) Routines and data
1 Comments
2 Region definition
- NAME=regional symbol
- SYMB=origin (if given)
- LINK=size

3 Block reservation
- Q=0 Reserve regional symbols
- Q=1 Reserve print line
- Q=2 Reserve primitive block
- Q=3 Reserve auxiliary buffer
- Q=4 Specify available space

4 Listing cards
5 Main storage header
6 Fast auxiliary storage header
7 Slow auxiliary storage header
- NAME=name of storage unit
- P =input mode
  - P=0 IPL standard
  - P=1 IPL compressed
  - P=2 IPL binary
- Q =type of input
  - Q=0 Routines; internals symbolic
  - Q=1 Data; internals symbolic
  - Q=2 Routines; internals symbolic; reset internal symbol table
  - Q=3 Data; internals symbolic; reset internal symbol table
  - Q=4 Routines; internals absolute
  - Q=5 Data; internals absolute
- SYMB=input unit
  - 0 = "normal" for installation
  - 1-10 = external tapes
- Regional SYMB names first routine (terminate loading)
- LINK=output mode (of form bbcd)
  - b=blank (columns 57-59)
  - c=non-zero if assembly listing
  - d=1 output in IPL compressed
  - d=2 output in IPL binary

9 First card