MEMORANDUM
RM-3739-RC
JUNE 1963

IPL-V PROGRAMMERS' REFERENCE MANUAL

Edited by Allen Newell

The RAND Corporation
1700 MAIN ST. • SANTA MONICA • CALIFORNIA

MEMORANDUM
RM-3739-RC
JUNE 1963

IPL-V PROGRAMMERS' REFERENCE MANUAL

Edited by Allen Newell

The RAND Corporation
1700 MAIN ST. • SANTA MONICA • CALIFORNIA

PREFACE

This Memorandum is a reference manual for the computer language INFORMATION PROCESSING LANGUAGE-V (IPL-V). It is a revision of Section II of the Information Processing Language-V Manual published by Prentice-Hall, Inc.,* and contains the official extensions to the language.

IPL-V is designed to deal with problems which impose memory requirements which change in an unpredictable fashion during the course of computation. It has been used to write programs for theorem proving, problem solving, information retrieval, natural language processing, assembly line balancing, simulation of cognitive processes, etc.

IPL-V was originally developed at The RAND Corporation, under U.S. Air Force Project RAND, and at Carnegie Institute of Technology, with collaboration of scientists at a number of institutions. This revision of the Manual was sponsored by The RAND Corporation.

---

*Newell, Allen, ed., Information Processing Language-V Manual, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1961.

SUMMARY

This Memorandum sets out the complete rules for coding in Information Processing Language-V (IPL-V), and documents extensions incorporated since publication of the Information Processing Language-V Manual.* A summary of extensions and the minor modifications to the language is contained in the final section (§25.0).

IPL-V processors are available for the IBM 650, 704, 709, 7090, 7094, Philco 2000, Bendix G-20, CDC 1604, UNIVAC 1105, and the AN/FSQ-32. A system for the Burroughs 220 is under development. Machine system write-ups are available for the various machines on which IPL-V is being used. These write-ups contain differences in the language peculiar to each machine, and must be consulted before using IPL-V.

An index, a list of the basic IPL-V processes, and a full-scale copy of the coding sheet, suitable for photo-reproduction, appear at the end of the Memorandum.

---

*Ibid.

ACKNOWLEDGMENTS

This revision to Section II of the Information Processing Language-V Manual was made by Hugh S. Kelly and Allen Newell. Of the extensions to the IPL-V language incorporated in this version, the line read processes are due to Fred. M. Tonge and the block handling processes to Einar Stefferud. The latter, needed for dealing with extremely large programs, were developed external to RAND by MESA Scientific Corporation, under contract to Hughes Aircraft Company. They are reported here with the kind permission of Hughes Aircraft Company.

The following contributed to the preparation of the original IPL-V Manual: Charles L. Baker, Edward A. Feigenbaum, Bert F. Green, Jr., Hugh S. Kelly, George H. Mealy, Nicholas Saber, Fred M. Tonge, Alice K. Wolfe, and Ted Van Wormer.

CONTENTS

PREFACE iii

SUMMARY v

ACKNOWLEDGMENTS vii

Section

1.0 GENERAL DEFINITIONS 1

2.0 DATA LIST STRUCTURES 12

3.0 ROUTINES AND PROGRAMS 20

4.0 BASIC SYSTEM OF PROCESSES 32

5.0 GENERAL PROCESSES, J0 to J9 36

6.0 DESCRIPTION PROCESSES, J10 to J16 38

7.0 GENERATOR HOUSEKEEPING PROCESSES, J17 to J19 40

8.0 WORKING STORAGE PROCESSES, J20 to J59 45

9.0 LIST PROCESSES, J60 to J104 46

10.0 AUXILIARY STORAGE PROCESSES, J105 to J109 56

11.0 ARITHMETIC PROCESSES, J110 to J129 61

12.0 DATA PREFIX PROCESSES, J130 to J139 64

13.0 INPUT-OUTPUT CONVENTIONS 68

14.0 READ AND WRITE PROCESSES, J140 to J146 ... 71

15.0 MONITOR SYSTEM, J147 to J149 73

16.0 PRINT PROCESSES, J150 to J162 79

17.0 BLOCK HANDLING PROCESSES, J171 to J179 ... 83

18.0 INITIAL LOADING 86

19.0 IN-PROCESS LOADING 97

20.0 SAVE FOR RESTART 98

21.0 ERROR TRAP, J170 100

22.0 LINE READ PROCESSES, J180 to J189 102

23.0 PARTIAL WORD PROCESSES, J190 to J197 105

24.0 MISCELLANEOUS PROCESSES, J200 to J209 106

25.0 CHANGES AND EXTENSIONS 107

LIST OF IPL-V BASIC PROCESSES 111

INDEX 117

1.0 GENERAL DEFINITIONS

1.1 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.

1.2 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 the 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 remainder--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.

1.3 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 § 2.0, DATA 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.

1.4 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.

1.5 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 data term. The P prefix specifies the type of date. (Q = 1 is also used in routines with a different meaning; program and data are kept separate by context.)

1.6 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 Fig. 1). For standard IPL words, the columns labeled NAME, P, Q, SYMB, and LINK are used. Type is 0 or blank, Sign (+ -) is irrelevant (but see §18.0, INITIAL LOADING), and all other columns are ignored by the IPL computer. (Certain columns are excluded from use.) P and Q may each contain

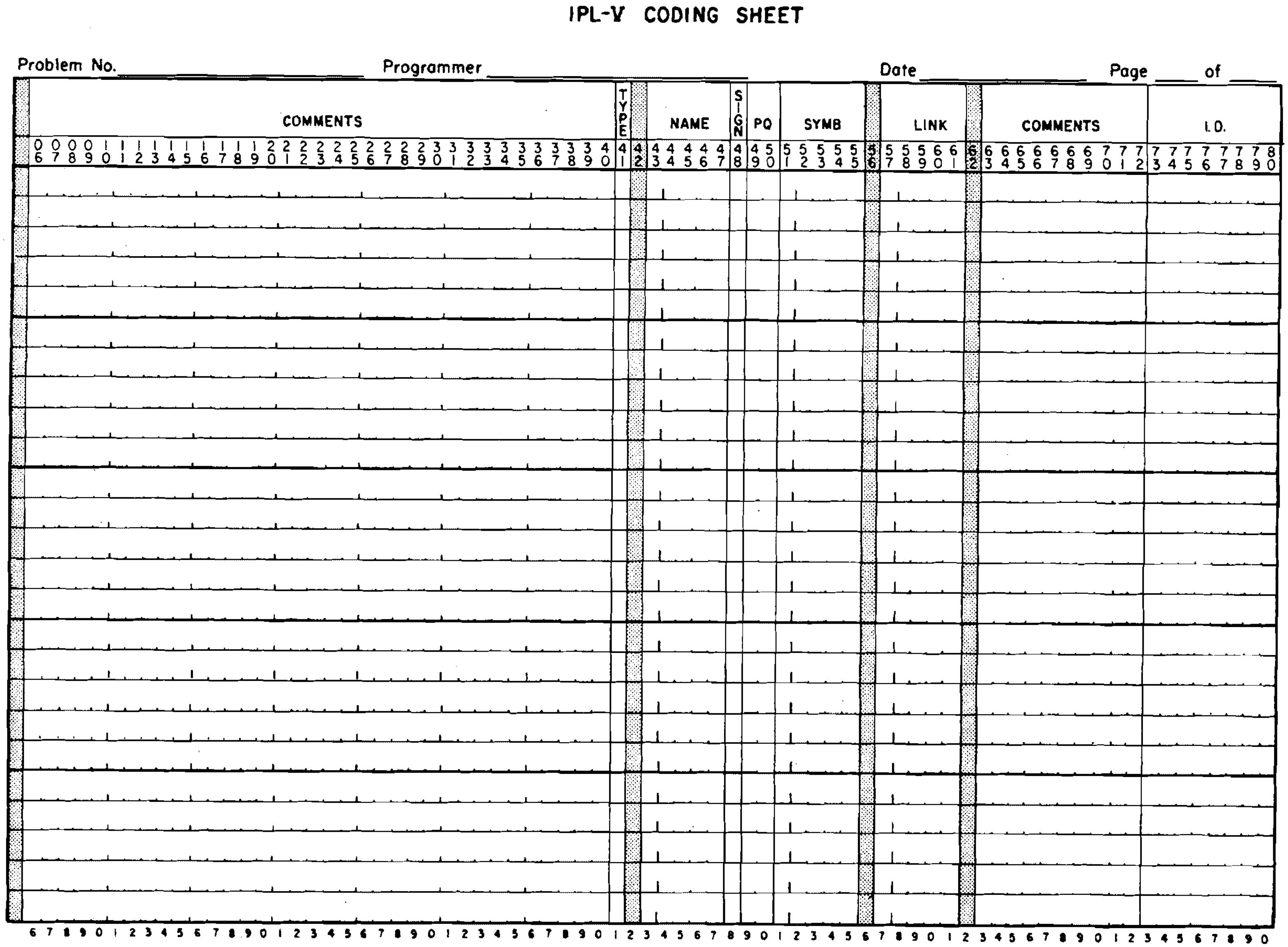

Fig. 1--IPL-V Coding Sheet

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. (In the Manual, we shall use "9-" to indicate a local symbol.) 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.

1.7 DATA TYPE CODE P

The format for data terms is shown in Fig. 2. Data terms have been defined for P = 0, 1, 2, 3, only. The other

IPL-V CODING SHEET

Problem No. ______ Programmer ______ Date ______ Page ____ of ____

COMMENTS	TYPE	NAME	SIGN	PO	SYMB	LINK	COMMENTS	I.D.
P = 0 DECIMAL INTEGER								
General Format --			±	01	dddd	dddd		
Examples: -12,192			−	1	1	2192		
15				1		15		
P = 1 FLOATING POINT								
General Format --			±	11	ddddd	d -ee		
(Exponent ee ranges from +50 to								
-50. Any character in col. 59								
is taken as "-".)								
Examples: $-.15 \times 10^3$			−	11	15	3		
$.732 \times 10^{-2}$				11	132	− 2		
P = 2 ALPHANUMERIC								
General Format --				21	aaaaa			
Example: IPL V				21	IPL 5			
P = 3 OCTAL								
General Format --			±	31	ddddd	ddddd		
Example: 7777777777				31	77777	77777		

Fig. 2--Format for Data Terms

four values, 4 through 7, are available for private use (see machine system write-ups).

1.8 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.

1.9 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 are 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.

1.10 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 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.

1.11 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.

1.12 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.

1.13 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 internal 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 § 9.4, DELETE). Any cell containing 0--i.e., SYMB = 0--is called empty.

1.14 STORAGE CELLS

A storage cell is one whose purpose is to hold symbols. A storage cell is created 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).

1.15 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.

2.0 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.)

2.1 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. The following rules apply to all data lists:

- Only the 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 a termination symbol for the LINK of the head is called an empty list. Cells containing data terms (cells with Q = 1), while not subject to the above rules, are considered to be the heads of empty data lists when manipulating data list structures.

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 on 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 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.

2.2 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

2.3 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 is in the first 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.)

2.4 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.

2.5 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.

2.6 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.

2.7 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 empty 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 § 2.9, 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

2.8 DATA LIST STRUCTURES

A list structure is a set of lists connected together by the names of the lists occurring 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 or internal name.
- 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.

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, a list, and a 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.

2.9 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.

2.10 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:	NAME	PQ	SYMB	LINK
A single list can be a data list structure:	L1		0	
			S1	
			S2	
			S3	0
A single data term can be a data list structure:	B5	21	BILL	
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):	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
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:	L3		0	
			9-3	
			N3	
			9-1	0
	9-1	1		15
	9-3	- 1		19

	NAME	PQ	SYMB	LINK
A list can have multiple occurrences of sublists, as well as mutual references and self references:	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.	L5		0	
			L5	
			L5	
			L5	0
The algebraic expression, (X1+X2)·(X3-X4) can be written as a list structure where the sublist arrangement indicates the parenthetical structure:	X0		0	
			9-1	
			·	
			9-2	0
	9-1		X1	
			+	
			X2	0
	9-2		X3	
			-	
			X4	0

2.11 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 § 18.0, 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.

3.0 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.

3.1 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 machine system write-ups).

3.2 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.

3.3 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 re-entrant, or merge; they may have regional symbols as LINKs, and names of list cells as SYMB.

3.4 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 or internal 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.

3.5 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 in routines 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.)

3.6 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 into which it can put a symbol, then execute a subroutine, and expect to find the symbol still in the cell afterwards.

3.7 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 implicit 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, HO, called the communication cell. If there are multiple inputs, they are placed in the push down list of HO in a sequence determined by the definition of the routine. All outputs from a routine are also placed in the communication cell, HO. If there are multiple outputs, they are placed in the push down list of HO in a sequence determined by the definition of the routine. In the Manual we will let (0), (1),..., represent, respectively, the symbols in HO 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 HO 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.)

3.8 EXPLICIT STATEMENT OF INPUTS AND OUTPUTS

The safety of HO implies that a routine must remove all its input symbols from HO. 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.

3.9 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.)

3.10 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:

	NAME	PQ	SYMB	LINK
S = B1		0	B1	
S = C1		1	B1	
S = D1		2	B1	

3.11 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 clear 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. H0 is preserved; then a copy of S is put in H0.

P = 2 OUTPUT TO S. A copy of (0) is put in cell S; then H0 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 H0; 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, H0; P = 3 and 4 are used in connection with safe cells; and P = 7 is a centralized transfer of control.

Examples: On the right we give small segments of program lists--i.e., sequences of instructions. On the left we give a verbal statement of the action.

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.

NAME	PQ	SYMB	LINK
	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.

NAME	PQ	SYMB	LINK
	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 preceded by 40Y5, which preserves Y5.

NAME	PQ	SYMB	LINK
	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.

NAME	PQ	SYMB	LINK
	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.

NAME	PQ	SYMB	LINK
	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.

NAME	PQ	SYMB	LINK
	11	W0	
		P30	
	30	W0	

3.12 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 alternative ways: first, as a series of rules, by the RULES OF INTERPRETATION; and second, as a step-by-step sequence of actions, by the INTERPRETATION CYCLE, similar to a flow diagram.

3.13 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 in-process 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.)

3.14 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 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.

3.15 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 § 15.0, MONITOR SYSTEM); take S = SYMB; go to INTERPRET P.
- Q = 5: Transfer machine control to SYMB (executing primitive); to 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 subprogram 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. (The various machine systems may not correspond exactly to this diagram--see machine system write-ups for details.)

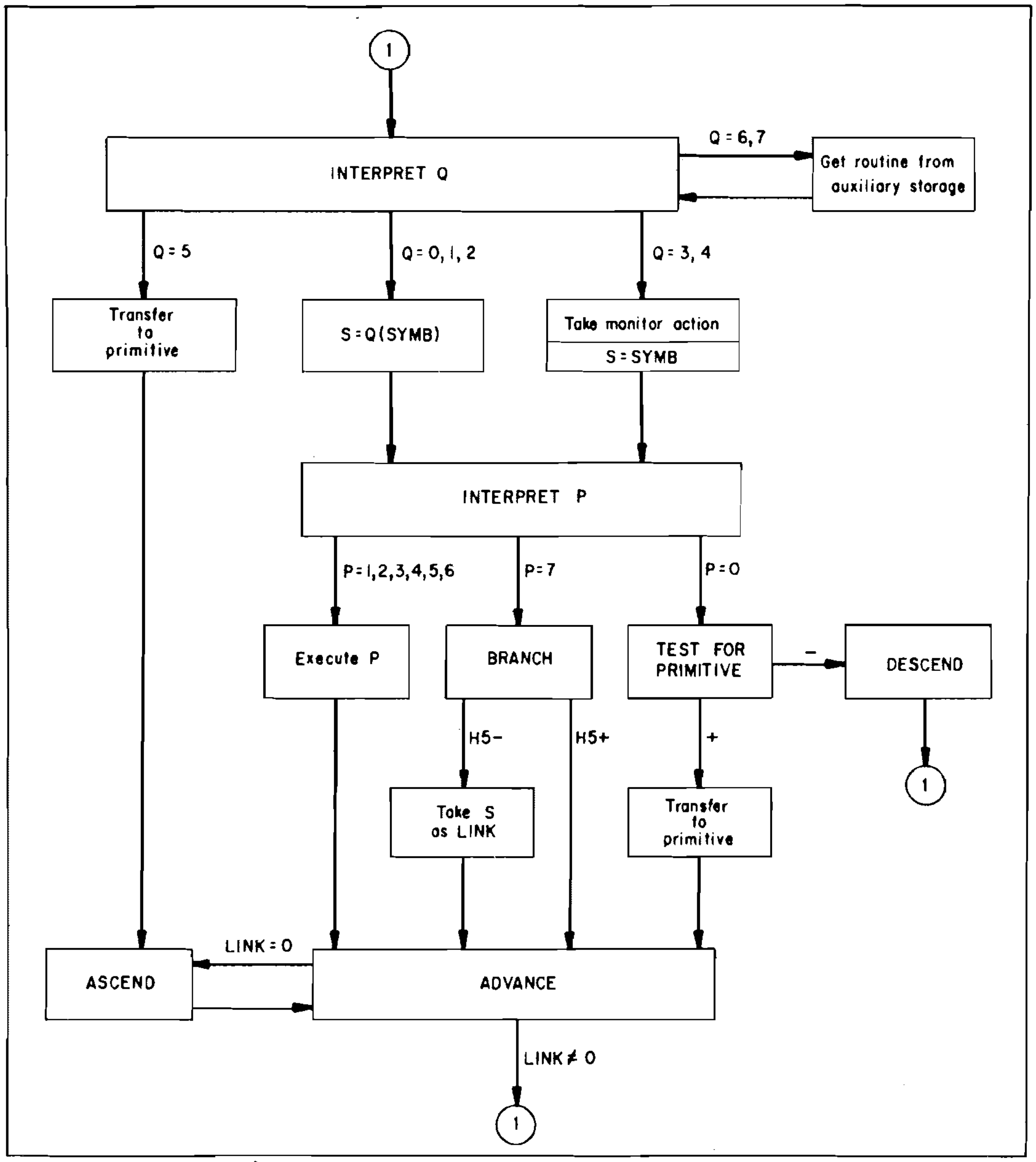

Fig. 3--The Interpretation Cycle

3.16 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.

4.0 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 focused 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.

4.1 SYSTEM REGIONS (EXCLUDED FROM OTHER USE)

The regions H, J, and W are used by the system, and no new symbols in these regions may be defined by the programmer.

The $ region is set aside to be used by individual installations for their own system routines and data. The need for this arises because each installation eventually creates a few routines which it makes commonly available to its users. The designation of a single region for these prevents unnecessary conflicts, since users everywhere can avoid using the $ region. Similarly, it is unnecessary for an installation to use J-routines and W-cells for its unique system routines and data.

4.2 SYSTEM CELLS

The following cells have special functions. They are all storage cells and safe, except H3, W11, and W33, which are integer data terms.

H0 The communication cell.

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

H2 Available space list; never manipulated 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-W9 Ten cells for common working storage (see § 8.0, WORKING STORAGE PROCESSES, and § 7.0, GENERATOR HOUSEKEEPING PROCESSES).

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

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

(See § 15.0, MONITOR SYSTEM, for W12 through W15, W23, W29.)

W12 Monitor start cell; holds name of routine executed at start of trace (0 = 3).

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

W14 External interrupt cell; holds name of routine executed at signaled interruption.

W15 Post mortem routine cell; holds name of routine executed after the post mortem lists have been printed.

(See § 13.0, INPUT-OUTPUT CONVENTIONS, 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 list 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 § 21.0, 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.

W30 Field length cell; holds name of integer specifying the number of columns in the current input field for the line read primitives.

W31 Trace mode cell; holds the name of an integer specifying NO TRACE if 0, FULL TRACE if 1, and SELECTIVE TRACE if 2.

W32 Reserved available space cell; holds the name of an integer specifying how many cells of available space will be withheld from H2, to be returned when H2 is exhausted.

W33 Cycle count for trap cell; an integer data term. When H3 equals W33, the trap action routine associated with H3 on W26 is executed.

W34 Current available space cell; holds the name of the available space list used by the loading processes, initially H2.

W35 Slow-auxiliary obsolete structure cell; holds the name of an integer that tallies the number of obsolete data structures occupying space in the slow-auxiliary data system.

W36 Used slow-auxiliary space cell; holds the name of an integer that tallies the total number of data structures, both current and obsolete, occupying space in the slow-auxiliary data system.

W37 Slow-auxiliary storage density cell; holds the name of an integer specifying the percentage of used slow-auxiliary space that may be occupied by obsolete structures.

W38 Slow-auxiliary storage compacting routine cell; holds the name of the routine which tests whether slow-auxiliary storage should be compacted at this time, and compacts if yes.

W39 Fast-auxiliary obsolete structure cell; same as W35, but for fast-auxiliary.

W40 Used fast-auxiliary space cell; same as W36, but for fast-auxiliary.

W41 Fast-auxiliary storage density cell; same as W37, but for fast-auxiliary.

W42 Fast-auxiliary storage compacting routine cell; same as W38, but for fast-auxiliary.

W43 Format cell; holds the name of an integer data term specifying the format for J162.

5.0 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 OPERATION. 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 <u>SET H5-</u>. The symbol in H5 is replaced by the symbol J3.

J4 <u>SET H5+</u>. The symbol in H5 is replaced by the symbol J4.

J5 <u>REVERSE H5</u>. If H5 is + , it is set - ; if H5 is - , it is set + .

J6 <u>REVERSE (0) and (1)</u>. Permutes the symbol in H0 with the first symbol down in the H0 push down list.

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

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

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

6.0 DESCRIPTION PROCESSES, J10 to J16

As described earlier (§ 2.3, DESCRIBABLE LISTS), 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) empty). 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 OF 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 set empty.

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 $N_i > 0$:

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

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.

7.0 GENERATOR HOUSEKEEPING PROCESSES, J17 to J19

7.1 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 subprocess 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 name of 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, thereby preserving the superroutine-subprocess context;
- 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, and records it in one of the hideout cells (see § 15.0, MONITOR SYSTEM).

J18 EXECUTE SUBPROCESS. Has no input. It does six things:
- Removes the symbols in W0 through Wn (generator context), thereby returning the previous context of symbols to the top of the W's (superroutine-subprocess context);
- Stacks the generator context in a hideout cell;
- Sets the trace mode of the subprocess to be that of the superroutine (see § 15.0, 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, thereby preserving the superroutine-subprocess context;
- 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 -).

7.2 CONVENTIONS FOR USING GENERATORS

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

- A generator is executed like any other routine. Its inputs are placed in H0:
  - (0) is always the name of the subprocess;
  - (1), (2), ..., are inputs to the generator.
- The subprocess sets H5 upon termination: + if the generator is to produce the next number of the sequence; - if the generator is to terminate.
- 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.

7.3 CONVENTIONS FOR CONSTRUCTING GENERATORS

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

- 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 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.
- 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 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 § 10.0, AUXILIARY STORAGE PROCESSES).

8.0 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 through 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 through J39.

J4n PRESERVE W0, W1, ..., Wn. Ten routines, J40 through J49.

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

9.0 LIST PROCESSES, J60 to J104

9.1 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.

We are given, initially, the available space list, H2, and a cell, WO, with a list proceeding from its LINK:

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

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

NAME	PQ SYMB	LINK
H2	0	1050
1050	0	1020
1020	0	

Notice that all words in the list except WO 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.

NAME	PQ SYMB	LINK
WO	B2	1000
1000	B2	500
500	C1	505
505	C2	....

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

NAME	PQ SYMB	LINK
WO	D1	1000
1000	B2	500
500	C1	505
505	C2	....

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

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

Restoring WO 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.)

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

9.2 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, and J177, the output is the name of the last list cell. (A private termination cell is not a list cell.)

9.3 INSERT

In an insert, two symbols are specified, either by the 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 H0.

	NAME	PQ	SYMB	LINK
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:	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.

9.4 DELETE

In a delete, a symbol in a list is specified, either by the input or as a 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 + .

	NAME	PQ	SYMB	LINK
Suppose the designated symbol is A1 and it is in list cell 1000:	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 <u>private termination</u> 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

9.5 POLICY ON PRIVATE TERMINATION CELLS

Private termination cells are introduced by the IPL system 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 machine system write-ups).

9.6 ERASE

To erase a structure of any kind is to return all the cells comprising it to available space. There is no output in H0.

9.7 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).

9.8 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 is 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 is 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).

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 symbol is obtained first, prior to inserting.

J66 INSERT (0) AT END OF LIST (1), IF NOT ALREADY ON IT. A search of list (1) is made, testing each symbol against (0) (starting with the cell after cell (1)). If (0) is found, J66 does nothing further. If (0) is not found, it is inserted at the end of the list, as in J65.

J67 REPLACE (1) BY (0) ON LIST (2) (FIRST OCCURRENCE ONLY). A search of list (2) is made, testing each symbol against (1) (starting with the cell after cell (2)). If (1) is found, (0) is placed in that cell. If (1) is not found, 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, the input (0) (which is no longer in H0) is a termination cell (see discussion in § 9.4, DELETE).

J69 DELETE SYMBOL (0) FROM LIST (1) (FIRST OCCURRENCE ONLY). A search of list (1) is made, testing each symbol against (0) (starting with the cell after cell (1)). If (0) is found, it is deleted, as in J68, and H5 is set + . If (0) is not found, H5 is set - .

J70 DELETE LAST SYMBOL FROM LIST (0). The last symbol on list (0) is located. If a last symbol is found, it is deleted and H5 is set + . If no last symbol exists (list (0) is empty at input), H5 is set - .

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, if 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.

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 that 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). Assume (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. (Q of (0) is ignored; thus, both cells holding internal zero and termination cells give H5-.)

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. Assume 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. J100 sets H5+ to the initial occurrence of the subprocess and 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). J100 will move in list (1) if it is on auxiliary.

J101 <u>GENERATE CELLS OF LIST STRUCTURE (1) FOR SUBPROCESS (0)</u>. 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 <u>print order</u>) 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 an internal zero. 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 empty heads created by marking processed, are not erased by the generator. (Note that J101 cannot be used in a subprocess to itself on the <u>same</u> list, because of the process marks.)

J101 will move in list structure (1) 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 before 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 (1) if it is on auxiliary.

J103 GENERATE CELLS OF BLOCK (1) FOR SUBPROCESS (0). (1) is assumed to be a block control word. The subprocess named (0) is performed successively with each of the names of the cells of the block (1) as input, generated in ascending order. J103 sets H5+ to the initial occurrence of the subprocess; and 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). (See § 17.0, BLOCK HANDLING PROCESSES.)

10.0 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.

10.1 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 that 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 head of the list structure, which acts as a control word for the filed structure. The head remains in main memory. 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 head of the filed structure 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; its image, still occupying space in the auxiliary system, is considered an obsolete structure. A move can be done any time the name of the filed 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 § 9.8, LIST PROCESSES.) J72 leaves an obsolete structure occupying auxiliary storage.

J74 COPY LIST STRUCTURE (0). (See definition in § 9.8, LIST PROCESSES.)

(See also J101 and J102.)

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). J105 leaves an obsolete structure image occupying space in auxiliary storage.

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 storage (Q = 7). (If there is no space in the slow-auxiliary, an error signal occurs; see § 21.0, ERROR TRAP.)

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.

J109 COMPACT THE AUXILIARY DATA STORAGE SYSTEM (0). J109 purges the obsolete data structures from the auxiliary data storage system specified by the integer data term (0). The slow-auxiliary system is compacted if (0) = 0, the fast-auxiliary system if (0) = 1.

The system will automatically compact both fast- and slow-auxiliary data storage when they become full or inefficient. However, since compacting may become a time-consuming operation in some applications, the programmer has the option of assuming partial or complete responsibility for specifying when and how frequently it shall occur. The following system cells are relevant to compacting slow-auxiliary storage. Cells W39 through W42 perform the same function for the fast-auxiliary data system.

- -W35 Holds the name of an integer data term which gives the number of obsolete structures currently occupying space in the slow-auxiliary data storage system.
- -W36 Holds the name of an integer data term which gives the total number of structures, current and obsolete, occupying space in the slow-auxiliary data storage system.
- -W37 Holds the name of an integer data term which the system interprets as the numerator of a fraction whose denominator is 100. When the ratio of obsolete to total structures in slow-auxiliary exceeds the above fraction, compacting will occur and all obsolete structures will be eliminated. 1W37 is initially 25, so compacting will occur when the number of obsolete structures is greater than 25 per cent of the total number of structures on slow-auxiliary storage.
- -W38 Holds the name of the routine which compacts when necessary. W38 initially names a system routine which performs the test described above under W37, but may be replaced by the name of a programmer's IPL routine which determines when compacting should occur. 1W38 compacts by executing J109. 1W38 is executed automatically after every execution of J105 (Move List Structure (0) in from Auxiliary).

10.2 AUXILIARY STORAGE FOR ROUTINES

The auxiliary system for routines is used by the interpreter to bring routines into main storage for execution. It uses an auxiliary buffer 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 buffer during loading, so that no further assembly is needed to execute them once they have been brought in (see § 3.12, INTERPRETATION). Since all auxiliary routines use the same buffer, 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 its predecessor in the auxiliary buffer back in. It is necessary to use a storage cell, the current auxiliary routine cell, H4, to keep track of the routines in the auxiliary buffer, 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. When the routines in auxiliary storage are highly interdependent, the "two call" system is quite inefficient during execution. Much of this inefficiency can be eliminated by grouping those auxiliary routines which call on one another frequently into the same buffer-load. A buffer-load of auxiliary routines is created at loading time by preceding a set of routines with a single header card (TYPE = 6 or 7). The entire set of routines is loaded into consecutive cells of the buffer and written to auxiliary storage as a unit. The first call on any one routine in the set causes the entire buffer-load to be brought into main memory. Mutual calls between the routines in this buffer-load do not result in accesses to auxiliary storage; a call on a routine in a different buffer-load does. Any number of buffer-loads can be created while loading. A routine or group of routines too large for the buffer overflows into main memory via H2,

with no ill effects other than the expenditure of cells in main memory. The above considerations lead to the following restrictions:

- No auxiliary routine shall modify itself in any way 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 the 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 supperroutine--and with it, the subprocess--and chaos would result when the generator tried to execute the subprocess (see § 7.1, GENERATORS).

11.0 ARITHMETIC PROCESSES, J110 to J129

All the input and output symbols in this section are the names of 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 both 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 § 21.0, 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 low-order digits of 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 data term = 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 machine system write-ups.)

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 machine system write-ups).

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.

12.0 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 with each occurrence. Upon loading data list structures (see § 13.0, INPUT-OUTPUT CONVENTIONS), 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	P = 1: List structure is in fast-auxiliary storage.
Q = 7	P = 1: List structure in is 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 machine system write-ups.

12.1 RECURSIONS

Besides the processes mentioned 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.

12.2 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 empty (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 empty 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 structures 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 H0.

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 H0.

J133 TEST IF LIST (0) HAS BEEN MARKED 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 marked processed.

J134 TEST IF (0) IS INTERNAL SYMBOL. Tests if Q = 4 in H0.

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 empty (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 <u>MAKE SYMBOL (0) INTERNAL</u>. The output (0) is the input (0) with Q = 4. Best considered as "unmake local symbol."

13.0 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.

13.1 EXTERNAL 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. External tapes can be generated in one run and used in a different run. External tapes are not generally compatible across different types of machines (but see machine system write-ups for details). Tapes can be used as intermediate storage, since tapes written by the <u>write</u> processes can be read back in by the <u>read</u> processes. An external tape can hold information in any of the representations defined below. (External tapes are also used as intermediate storage of blocks of information; see § 17.0, BLOCK HANDLING PROCESSES.)

13.2 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 machine system write-ups should be consulted for more information.

13.3 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	=	Machine code.
4	=	Restart mode (see § 20.0, SAVE FOR RESTART).
5, 6, 7	=	Machine language for various object machines. See machine system write-ups for further details.

13.4 IPL COMPRESSED REPRESENTATION

See machine system write-ups for information.

13.5 IPL BINARY REPRESENTATION

(See machine system write-ups 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.

14.0 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 sets (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. A structure may be loaded into a specific block of main storage, however, (see § 18.5, TYPE = 5, 6, 7, 8: HEADER CARDS). 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 § 13.2, INPUT-OUTPUT UNIT CODE).

J140 <u>READ LIST STRUCTURE</u>. A list structure on cards (or external tape) in any of the admissible forms (IPL, compressed, binary) is read into the main storage cells taken from the available space list 1W34, its name input to (0), and H5 set + . Blank records are treated as end-of-list-structure marks. (End-of-list-structure is also signaled by an input end-of-file condition or by the start of a new list structure, with a regional or internal name.) 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 machine system write-ups 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. 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 SET. A blank record (appropriate to mode 1W17) is written on the external tape named in W19. (See § 18.0, INITIAL LOADING, for use of blank records.)

15.0 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 "snapshot" of the program to see what it is doing. Second, it is possible to get "post morten" information after a program has stopped. Third, it is possible to trace the program, printing information on each instruction as it is executed. The machine system write-ups should be consulted on the conventions for using the console to accomplish the features described below.

15.1 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 whether the number of cells of reserved available space is equal to 1W32. If unequal, it adjusts the supply of cells to equal 1W32.
- It checks the console for the following signals:
    - External interrupt: if the external interrupt signal is present, the routine named in the safe storage cell W14 is executed and the program continues.
    - External trace mode: no trace, selective trace, full trace. (If there is no external trace signal from the console, the external trace mode is set according to 1W31.)
    - Finally, 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.

15.2 SNAPSHOTS

W12 and W13 hold snapshot routines. As seen above, they will be executed under various conditions associated with the monitor points, Q = 3. There is no restriction on the routine that may be executed, although the normal use is to print out various lists to see how the program is progressing.

The snapshot mechanism is operative at monitor points regardless of the trace mode or external trace conditions. The snapshot cells (W12 and W13) initially contain J0, meaning "no operation."

15.3 EXTERNAL INTERRUPT

The system checks for the presence of an external interrupt signal at each monitor point. If the signal is present, the routine named in the safe storage cell W14 is executed and the program continues. Setting a console switch manually is the normal way of providing an external interrupt signal, but see the machine system write-ups for additional or alternative methods.

There is no restriction on the nature of the routine 1W14. In particular, terminating 1W14 with J166 will save for restart and continue with the program. Terminating 1W14 with J7 will terminate the program without providing for restart. To terminate the program and provide for restart, see the example in § 20.0, SAVE FOR RESTART.

15.4 POST MORTEM

In the event the system detects some internal error while executing a program, it automatically prints out information about the terminating condition of the machine via J202 and then stops. J202 may also be executed directly by the programmer any number of times during a run. W23 holds the name of the list specifying information to be printed by J202. This list may be modified by the programmer. W15 holds the name of a routine that J202 executes after the other information has been printed. Any routine may be executed except J202. Its primary use is to select and print debugging information that cannot be specified on the 1W23 list. W15 initially holds J0.

J202 PRINT POST MORTEM AND CONTINUE. Print as defined for the particular machine system.

15.5 TRACING

There are two internal 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.

The three externally imposed trace conditions (no trace, full trace, and selective trace), may also be imposed internally by the integer data term named by W31. The code for W31 is:

0 = No trace;
1 = Full trace;
2 = Selective trace.

1W31 is set for selective trace initially. The programmer may change 1W31 anytime. The change becomes effective when the next monitor point is encountered. 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.

15.6 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, H1 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 $\neq$ 3 or 4 in any instruction.

Where generators are involved, the superroutine and subprocess are on the same level; the subprocess will trace without being marked, provided the superroutine is tracing. The generator is down one level from the superroutine; hence, if marked with Q = 4, the generator will trace when the superroutine is tracing.

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 = J0, 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 and P = 0, in which case J149 restores (0). If Q $\neq$ 3 or 4, does nothing.

16.0 PRINT PROCESSES, J150 to J162

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 § 13.2, 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 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 machine system write-ups should be consulted.

16.1 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. All data terms are translated to their external form. If input (0) is a block control word or the head of a structure on auxiliary, only the word (0) itself is printed. 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	111	11112	2222	22	22233333333
			012	345	67890	1234	56	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. If input (0) is a block control word or the head of a structure on auxiliary, only the word (0) itself is printed.

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.

16.2 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 § 18.3, 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, H5 is set - , and H0 no longer holds the input.

Symbols are entered in the print line compactly; i.e., as A1, B10, etc. (A0 is entered as A). Data terms are entered as follows:

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

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 cols.); ".14x10^{-16}" entered as "140000 -16" (10 cols.).

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, no entry is made 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 is 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 is 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 is 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).

J162 ENTER (0) ACCORDING TO FORMAT 1W43. The name and contents of cell (0) are entered after having been converted to an appropriate external representation (e.g., octal), specified by the data term in W43. 1W25 and H5 are treated as in J156. J162 is intended primarily to provide dumps of blocks when used with J103. (See machine system write-ups for the various formats and conversion schemes available.)

In addition to lines composed using these primitives, complete headings and partial lines can be specified at loading (see § 18.3, TYPE = 3: BLOCK RESERVATION CARDS).

17.0 BLOCK HANDLING PROCESSES, J171 to J179

In order to deal effectively with programs which exceed the main storage capacity of the computer several times over, it is necessary to have techniques for dealing with blocks of main storage. A block control word is a cell with P = 7 and Q = 7, whose SYMB specifies the origin of a continuous block of memory cells, and whose LINK specifies the number of cells in the block.

Since a region is represented in the computer by a block of cells, there is a block control word for each of the 36 possible regions definable by the IPL-V programmer. We will refer to a block control word for a region as a region control word hereafter. The 36 region control words are a permanent part of the system; their names are only obtainable via J175. They are used by the system to translate the external representation of regional symbols (e.g., R15) into computer addresses during loading, and to translate regional cell names back into their external regional representation during output. The programmer may copy region control words, but should never modify them; changing the contents of a region control word effectively redefines the region it controls.

The programmer may define and name blocks of space, other than regions, with Type-3 cards; the symbol which names the block is made the control word for the block in this case.

A block of space, including a region, may be turned into a list, which may be used by the loading processes as an available space list. This ability to load into specific blocks, coupled with fast processes to read and write the contents of blocks on tape, allows overlay techniques for problems too large to be performed economically in a single phase.

J171 RETURN UNUSED REGIONAL CELLS TO H2. J171 scans all region blocks for unused cells and returns them to the end of H2. "Unused" means that the regional symbol has not appeared in any NAME, SYMB, or LINK field of the input deck and the cell does not lie within a block reserved by any Type-3 card. J171 modifies the region control word so that the highest symbol used becomes the last cell of the region after J171; unused symbols higher than this symbol lose their regional status; unused cells lower than this symbol retain their regional status for the input and output processes and for J175 and J201.

J172 MAKE BLOCK (0) INTO A LIST. Input (0) is assumed to be a block control word. Output (0) names the head of the list and is the name of the first cell of the block. The cells of the block are linked in ascending order; their P, Q, and SYMB are unchanged. (J172 provides a way to turn a block into a list. The list may then be added to H2 by J71, or used as special available space for loading, by putting its name in W34.)

J173 READ NEXT BLOCK FROM TAPE 1W19 INTO BLOCK (0). Sets H5+ if successful. Traps on trap attribute 'J173' with first word of block named in H0 if first word failed to match the contents of (0). Sets H5-, gives message, and traps on attribute 'H0' if (0) is not a block control word; i.e., P and Q not equal to 7.

J174 WRITE BLOCK (0) ONTO TAPE 1W19. Sets H5+ if successful. Same procedure as J173 if (0) is not a block control word. The first word written on tape is not the first word of the block, but is the contents of the input block control word (0). This is used by J173 to detect reading of information into a block other than that from which it was written.

J175 FIND REGION CONTROL WORD OF REGIONAL SYMBOL (0). If (0) is a regional symbol, H5 is set + , and output (0) is the name of the block control word for the region. H5 is set - and there is no output if input (0) is not a regional symbol. A symbol is a regional symbol to J175 if the cell which it names lies within one of the blocks defined by the 36 region control words.

J176 SPACE (0) BLOCKS ON UNIT 1W19. (0) is assumed to be a signed integer data term. Tape 1W19 is spaced (0) blocks in the direction indicated by the sign of (0). Plus indicates forward spacing, minus specifies backspacing.

18.0 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.

18.1 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.

18.2 TYPE = 2: REGION CARDS

All the regional symbols with the same initial letter constitute a region. Each region is represented in the computer by a block of consecutive cells. For example, the R-region might correspond to the block of cells 1000 to 1018: then R0 would correspond to 1000, R1 to 1001, and R18 to 1018. The size of 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 block is put in LINK. The initial loader assigns the next available block of

contiguous cells to this region and records the origin and size of the block in the region control word. Thus, the origin of a region block 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 absolute address of the origin in SYMB. The origin can also be specified symbolically in terms of another region, provided the other region is first defined. (See machine system write-ups for further details.)

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

There are 36 possible regions:

A B C D E F G H I J K L M N O P Q R

S T U V W X Y Z + - / = . , $ *) (

Three regions, H, J, and W, have already been permanently specified for the basic sysbem. Also, the $ region is to be used for system routines unique to particular installations (see § 4.1, SYSTEM REGIONS). The first symbol of all those regions which the programmer does not define with Type-2 cards is automatically defined and reserved by the initial loader when the first header card (TYPE = 5, 6, 7, or 8) is encountered. This allows the programmer to use the line read primitives on English text without having to define all 36 regions explicitly (see § 22.0, LINE READ PROCESSES). 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 and have not been read by J181 (Input Line Symbol) during processing--may be made

part of the available space for the IPL computer by executing J171. All regional symbols mentioned (in SYMB or LINK) but not defined (in NAME) are used but empty. If the exact limits of regions are specified, then the blocks of cells corresponding to different regions may overlap and need not be contiguous. If origins are assigned by the IPL computer, the region blocks are adjacent and disjoint. The block control word for a region may be obtained by J175.

18.3 TYPE = 3: BLOCK RESERVATION CARDS

It is necessary to create blocks of space for various purposes, and sometimes desirable to set a number of regional symbols to be empty without mentioning them. Type-3 cards are used to accomplish this. As in Type-2 cards, SYMB indicates the base, if appropriate, and LINK indicates the size of the block. The initial loader creates a block control word in the cell mentioned in the NAME field of all Type-3 cards. 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 empty, and will not be put back on available space by J171. If NAME is B20, then B20 is a block control word for the block A5-A14. The symbols reserved must have previously been covered by a Type-2 card.

Q = 1 RESERVE PRINT LINE. NAME is the regional symbol naming the line (i.e., the block control word). 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 machine system write-ups for details 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 BCD 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 BLOCK. The regional symbol appearing in the NAME field is the name of the block, (i.e., the block control word). LINK is an integer specifying the number of cells in the block. The size of any one block is limited by the size of the machine. Blocks may overlap one or several other blocks, completely or partially, including blocks of regional cells or even blocks of code which make up the IPL-V Loader, Interpreter, or Monitor, if this is useful. Any number of blocks may be reserved. In general, a block control word should not lie within the block which it controls.

Q = 3 RESERVE AUXILIARY ROUTINES BUFFER. This reserves a block of size LINK (starting at SYMB, if given). LINK is the number of cells, and SYMB, if given, specifies the origin of the block. NAME is optional. Only one such buffer may be reserved. This buffer is used by all the routines on auxiliary storage. Its size limits the maximum size of a routine on auxiliary (but see § 10.0 AUXILIARY STORAGE PROCESSES).

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 to H2. This includes all interstices between blocks, if any, which would go at the end of available space. LINK, when present, specifies the number of cells that will be provided, in one continuous block if possible. NAME, specifying a block control word, is optional. If a large enough block is not available, a message is given, but the block control word is not modified.

18.4 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.

18.5 TYPE = 5, 6, 7, 8: HEADER CARDS

Data or routines are loaded in a series of separate sets, each of which is preceded by a header card that governs the loading process. The input set may be in one of 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 header. 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 set may contain either routines or data, and it is necessary to specify which, as the P and Q codes are treated differently. Also, the set may go into main storage (TYPE = 5), may go to one of the auxiliary storages (TYPE = 6 for fast, TYPE = 7 for slow), or may be skipped (TYPE = 8). Structures going into main storage may be loaded into cells from the standard available space list, H2, or may be loaded into a specific block. Data list structures are loaded to auxiliary in relocatable form; auxiliary routines are assembled into the single auxiliary routines buffer and written to secondary storage in non-relocatable form.

Loading into a specific block of main storage is accomplished by the use of the safe storage cell W34. W34 holds the name of the available space list used by the loading processes (initial loading, J140 and J165) and initially holds H2. To load into a specific block it is necessary to make the cells of the block into a list with J172 and put the name of this list into the safe storage cell W34. Sets of data or routines preceded by

Type-5 cards with NAME = blank will then be loaded into the cells of the block. The first cell of the block is never loaded into since, like H2, it is the head of the available space list. If the list 1W34 becomes exhausted, H2 is placed in W34 without push down, an error message is given, and loading continues from H2.

The loader will automatically set the name of the desired available space list into W34 if it encounters a Type-5 card with NAME = name of block. It preserves W34 and places the name of the first cell of the block into W34; the associated set of routines or data is loaded and W34 is restored when the next header card (TYPE = 5, 6, 7, or 8) is encountered. The block mentioned in the NAME field must have been made into a list (J172) at some previous time.

A Type-8 <u>editing header</u> inhibits the loading of its associated set of routines or data, but allows the listing and output options. It is intended for use on the controlling unit to skip over unwanted sets on an alternate unit. (See § 18.7, CONTROLLING AND ALTERNATE INPUT UNITS.)

Finally, a Type-5 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 storage
- 7 = Slow-auxiliary storage
- 8 = Inhibit loading--permits listing and output options.

NAME: Name of storage block:
NAME = Blank, TYPE = 5: Load into the main memory cells taken from the current available space list, 1W34. 1W34 is initially H2.

NAME = Regional symbol, TYPE = 5: Name is assumed to be a block control word whose SYMB names a previously constructed available space list. Preserve and set W34 = SYMB, and load the set into the main memory cells taken from the list named SYMB, and restore W34 when the next header is encountered.

NAME = Anything, TYPE = 6: (NAME is ignored.) Load each data list structure to fast- or slow-auxiliary in relocatable form. Load routines to fast- or slow-auxiliary in non-relocatable form, each routine origined one cell beyond the end of the immediately preceding routine. The first routine in the set is origined at the first cell of the auxiliary routines buffer. (See § 10.2, AUXILIARY STORAGE FOR ROUTINES.)

P: Input Mode:
0 = IPL standard (1 word per card)
1 = IPL compressed
2 = IPL binary
3 = Machine code
4 = Restart mode
5, 6, 7 = Machine dependent modes for various object machines. See machine system write-ups 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 (0-9 are always defined and absolute).
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). Such internal symbols do <u>not</u> start a new list structure.

5 = Data list structures. Internal symbols are considered machine addresses and do not start new list structures.

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 (see § 18.7, CONTROLLING AND ALTERNATE INPUT UNITS).
If SYMB of a Type-5 card contains a regional symbol, this start card terminates loading and the program begins at the routine named in SYMB.

LINK: Output mode: of form bbbcd
b = Output unit: blank = unit 1W19; 1-10 means unit 1-10.
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.
  = 3 if output in machine code.
  = 9 if output in IPL standard.
The output unit is the one given in W19.
Each set of IPL compressed or IPL binary output ends with a blank record appropriate to that mode (see § 18.7, CONTROLLING AND ALTERNATE INPUT UNITS).

18.6 TYPE = 9: FIRST CARD

The very first card of each program to be loaded must be a Type-9 card. The use of Type-9 cards allows several programs to be stacked on an external tape for batch execution. SYMB of the Type-9 card specifies the controlling unit for initial loading. If SYMB is blank or 0, the standard input unit is the controlling unit.

18.7 CONTROLLING AND ALTERNATE INPUT UNITS

Generally all sets exist in sequence on a single input unit. However, it is possible to have more complex arrangements. In any case, there will be a single

controlling input unit which contains the header cards of all sets in order. (This unit is specified by SYMB of the first Type-9 card.) If SYMB of a particular header card is blank, then the associated set follows immediately on the controlling unit. If SYMB of the header card refers to an alternate input unit, then the set associated with the header card is read from the alternate unit. The header card on the controlling unit completely specifies the input mode, type of input, destination in storage, output mode and unit; header cards on the alternate unit are ignored. Discrepancy between the header card on the controlling unit and the actual information on the alternate input unit causes a loading error. The set on the alternate unit is terminated by a blank record or by a header card, at which time the next header on the controlling unit is read. Any non-Type-0 cards on the alternate unit are printed like Type-1 cards.

18.8 ASSEMBLY LISTING

It is possible to obtain an assembly listing of the program being loaded when specified by LINK of the Type-5, 6, 7, or 8 header card. 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 Type-1 cards can be suppressed for any set by a signal in the LINK of the header card. Other Type cards are printed under all conditions.

18.9 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. <u>All Type-3 cards with exact limits, if any, in any order</u>.

4. <u>All Type-2 cards giving only region size, if any, in any order</u>.

5. <u>All Type-3 cards giving only block size, if any, in any order</u>.

   Only regions and blocks defined by these cards (plus the H, J, W, and $ 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. <u>Sets of data and routines, in any order</u>.

   Each set is preceded by an appropriate Type-5, 6, 7, or 8 card. For IPL standard and IPL compressed cards, the end of the set is signaled by the next Type-5, 6, 7, or 8 card. For binary and machine modes, a special termination signal is required in the last card (see machine system write-ups for details).

   The input unit named by SYMB of the Type-9 card is the controlling unit for initial loading. If a Type-5, 6, 7, or 8 card on the controlling unit indicates in SYMB that a set is to come from an alternate input unit, then after that set is loaded from the alternate unit, the next header card is picked up from the controlling unit.

7. <u>The start card: A final Type-5 card on the controlling unit with a regional symbol for SYMB to start the program at SYMB</u>.

   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 declare regions at the beginning.)

In loading Type-0 cards, the IPL computer assigns locations from available space to represent 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).

When internal symbols are treated as pure symbolics rather than as absolute machine locations, they are likewise represented by locations assigned 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 (see Q of Type-5, 6, 7, 8 cards). 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. (Internals 0 through 9 are always defined and absolute, however.)

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.)

19.0 IN-PROCESS 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 in-process loading. No new regions or blocks can be specified during in-process loading. (Not all object machines have full flexibility, so the machine system write-ups 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, 7, or 8), each followed by a set of routines or data (except when the headers specify a set from an alternate input unit), and terminated by a start card (a Type-5 card with a regional SYMB). The routine named as SYMB on the start card is taken as the next routine to be interpreted. If there are no routines or data, or if there is no start card following the sets, then interpretation continues with the instruction following J165.

20.0 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 initiated externally at a monitor point (see § 15.0, MONITOR SYSTEM) or may be put in the program at any point.

J166 SAVE ON UNIT (0) FOR RESTART. The entire contents of main and auxiliary storage are written onto a single external tape (or punched on cards, according to the unit named by data term (0)). Identification of the auxiliary units and external tapes being used by the IPL computer are printed out. Then H5 is set + and the program continues. If the specified auxiliary units and external tapes are provided, and the tape (deck) is loaded under control of a Type-5 card with P = 4 (restart mode), subsequent runs will commence at the instruction following J166, with H5 set - .

Since J166 sets H5+ and the restart process sets H5-, the instruction following J166 can take different action depending on whether this is the original run (H5+) or a restart run (H5-). For example, if the external interrupt cell, W14, named the routine X1, below, and the console signaled an external interrupt, then the run would save for restart and terminate when the next monitor point occurred. Restart runs, since H5 is set - , would restore the original sign of H5 and resume execution at the monitor point.

NAME	SIGN	PQ	SYMB	LINK		COMMENTS
X1		40	H5			Save Current Sign of H5
		10	9-1			
			J166			Save for Restart on Unit 3,
		70		J7		Then Terminate this Run
		30	H5	0		Restore H5 on Restart Runs
9-1	+	01			3	Integer 3

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 external 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.

Save for restart is used to provide a fast-loading version of checked-out routines, to which additional routines to be debugged can be added by J165.

21.0 ERROR TRAP, J170

Many different error conditions can occur during processing by the IPL computer--for example: available space exhausted; specifying other than a data term as operand for an arithmetic process; etc. 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. 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 by 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 machine system write-ups should be consulted for the normal error condition and trap actions. However, the traps described below are standard for all machine systems.

TRAP ON AVAILABLE SPACE EXHAUSTED: 'H2'.

W32 holds the name of an integer data term which specifies the number of cells to be removed from H2 before execution of the program begins. If available space becomes exhausted during execution, these cells are returned to H2 to enable trapping on the attribute H2. The trap action routine to regain space must be provided by the programmer, and may include erasing data structures or routines (J72, J201), or filing data on auxiliary space (J106 or J107). Upon return from the trap, 1W32 cells are again removed from H2 and the program continues. 1W32 initially specifies ten cells, but may be changed by the programmer at any time. The change becomes effective at the next monitor point, after the next H2 trap has been executed, or whenever a start card is encountered by the loader.

TRAP ON INTERPRETATION CYCLE COUNT: 'H3'.

Traps when H3 (cycle count) is equal to W33. W33 is an integer data term that is compared to H3 each interpretation cycle, after H3 has been incremented. When H3 is equal to W33, the action associated with the symbol H3 on 1W26 is executed and the program continues. W33 is initially zero, so no trapping will occur until the programmer sets W33 to a non-zero value.

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.

22.0 LINE READ PROCESSES, J180 to J189

The line read primitives provide a means of reading a BCD card under control of an IPL-V program and translating selected fields into IPL symbols or data terms.

Control Cells:

- 1W18 names the input unit for J180. 1W18 = 0 means the normal input tape.
- 1W24 names the current read line. ("Read lines" and "print lines" are identical and interchangeable. Lines for either or both purposes are specified by Type-3 cards with Q = 1.)
- 1W25 is a decimal integer data term specifying the left column of the current input field.
- 1W30 is a positive decimal integer data term specifying the size (number of columns) of the current input field.

J180 READ LINE. The next record on unit 1W18 is read to line 1W24. (The record is assumed to be BCD, 80 cols.) Column 1 of the record is read into column 1 of the read line, and so forth. H5 is set + . If no record can be read (end-of-file condition), the line is not changed and H5 is set - .

J181 INPUT LINE SYMBOL. The IPL symbol in the field starting in column 1W25, of size 1W30, in line 1W24, is input to H0 and H5 is set + . The symbol is regional if the first (leftmost) column holds a regional character; otherwise, it is absolute internal. All non-numerical characters except in the first column are ignored. If the field is entirely blank, or ignored, there is no input to H0, and H5 is set - . In either case, 1W25 is incremented by the amount 1W30. (J181 turns unused regional symbols into empty but used symbols.)

J182 INPUT LINE DATA TERM (0). The field specified as in J181 is taken as the value of a data term. Input data term (0) is set to that value and left as output (0). H5 is set + . The data type of input (0) determines the data type of the output. If the input (0) is a decimal or octal integer, or BCD, the read line field is interpreted as that type. Any other data type is treated as BCD. In composing BCD data terms,

the field is left-justified and the full data term completed with blanks on the right, if necessary. If the specified field exceeds five columns, the rightmost five columns are taken as the field. In composing decimal and octal integer data terms, non-numerical characters are ignored. If the resulting information exceeds the capacity of the data term, the rightmost digits are retained. If the read line field is entirely blank (or non-numerical, for integer data types), (0) is cleared (to blanks for BCD; to zero for integer) and H5 is set - . In either case, 1W25 is incremented by the amount 1W30.

J183 <u>SET (0) TO NEXT BLANK</u>. (0) is taken as a decimal integer data term. Line 1W24 is scanned, left to right, starting with column 1W25+1, for a blank. One is added to (0) for each column scanned, including that in which the scanned-for character ('blank' in J183) is found. (0) is left as output (0). H5 is set + if the character is found in the line, and - if it is not. (Thus, if input (0) = 1W25, after scanning, output (0) will specify the column holding the scanned-for character. If input (0) = decimal integer 0, after scanning, output (0) will be the size of a field beginning in column 1W25 and delimited on the right by the next occurrence of the scanned-for character.)

J184 <u>SET (0) TO NEXT NON-BLANK</u>. Same as J183, except scans for any non-blank character.

J185 <u>SET (1) TO NEXT OCCURRENCE OF CHARACTER (0)</u>. Same as J183, except scans for character (0), counting into decimal integer data term (1). Input (1) is left as output (0). If input (0) is a regional symbol, its region character is the character scanned for, if input (0) is internal, its last (low-order) digit is the character scanned for.

J186 <u>INPUT LINE CHARACTER</u>. The character in column 1W25 of line 1W24 is input to H0, H5 is set + . If the character is numerical, that internal symbol is input; if the character is non-numerical, the zeroth symbol in the region designated by that character is input; i.e., A → A0, 3 → 3. If the character is a blank, there is no input and H5 is set - . In either case, 1W25 is not advanced.

J189 TRANSFER FIELD. The field in line 1W24, starting in column 1W25, and of size 1W30, is transferred to line (0), starting in column 1W21. H5 is set + . If the entire field cannot be transferred (line (0) is too short), as much is transferred as can be, and H5 is set - . In either case, 1W25 is set to the last column transferred plus one.

23.0 PARTIAL WORD PROCESSES, J190 to J197

These primitives allow manipulation and testing of the P, Q, SYMB, or LINK of IPL words. The words are assumed to be standard words, not data terms. The P, Q, SYMB, or LINK is input to, or output from, the symbol portion of H0, and may be treated as any other IPL symbol.

J190 INPUT P OF CELL (0) TO H0. After J190, the symbol in H0 will be an absolute internal symbol between zero and seven.

J191 INPUT Q OF CELL (0) TO H0. After J191, the symbol in H0 will be an absolute internal symbol between zero and seven.

J192 INPUT SYMB OF CELL (0) TO H0. The symbol input will be regional if covered by a region control word; otherwise, it will be internal. That is, the Q of the cell (0) is not used to determine the type of symbol.

J193 INPUT LINK OF CELL (0) TO H0. Q of H0 will be regional. The symbol input will be regional if covered by a region control word; otherwise, it will be internal.

J194 Set (1) TO BE THE P OF CELL (0).

J195 SET (1) TO BE THE Q OF CELL (0).

J196 SET (1) TO BE THE SYMB OF CELL (0). Q of cell (0) is unchanged.

J197 SET (1) TO BE THE LINK OF CELL (0).

24.0 MISCELLANEOUS PROCESSES, J200 to J209

J200 LOCATE THE (0)th SYMBOL ON LIST (1). (0) is an integer data term whose sign is ignored, and whose value, n, specifies that the name of the nth list cell of list (1) be output in H0, with H5 set + . Output (0) names the last cell if H5 is set - , indicating that less than n symbols exist on list (1). (Note that private termination cells are not list cells.)

J201 ERASE ROUTINE (0). Return the space to the available space list, 1W34. (0) is assumed to be a regional cell and is set empty rather than being returned to available space. If (0) contains Q = 6 or 7, it is assumed to be an auxiliary routine and J201 does nothing.

All non-regional symbols appearing in the SYMB of a routine are treated as sublists to be erased. Thus, mentioning local or internal data terms, working cells, or data lists in the routine will cause unpredictable erasure. A regional LINK is equivalent to LINK = 0, signaling the end of the sublist. If a routine is loaded after J171 has been executed, an unused regional cell from the middle of a regional block may be used in its construction. Since J201 considers this cell to be regional and hence the termination of a sublist, a portion of the routine may not be returned to available space.

J202 PRINT POST MORTEM AND CONTINUE. (See § 15.4, POST MORTEM, for complete definition of J202.)

25.0 CHANGES AND EXTENSIONS

The modifications described in this section have originated from users' experience with IPL-V in the two years since publication of the first edition of the Manual. Sections 25.1 through 25.3 describe changes to previously defined features of the system; they are reported separately because in some cases they may impose minor modifications to previously checked out programs. The extensions of IPL-V are described in Sections 25.4 through 25.8; they impose no modifications to existing programs. The modifications are not described in full in this section, but are simply listed with references to the appropriate sections of the Manual.

25.1 SYSTEM CELL CHANGES (see § 4.2)

W14 External interrupt cell; holds name of routine executed at return to Q = 3 point. (See § 15.3, EXTERNAL INTERRUPT.)

W15 Post mortem routine cell; holds name of routine executed after the post mortem lists have been printed. (See § 15.4, POST MORTEM.)

25.2 PRIMITIVE PROCESS CHANGES

J166 SAVE ON UNIT (0) FOR RESTART. The program does not terminate when J166 is executed. J166 sets H5+, and restarting causes H5 to be set - . (See § 20.0, SAVE FOR RESTART.)

25.3 CHANGES IN LOADING CONVENTIONS

TYPE = 3: BLOCK RESERVATION CARDS (See § 18.3)

Name is the regional symbol naming the line for Q = 1. (The earlier edition of the Manual erroneously stated that SYMB specified the name.)

TYPE = 6 or 7: HEADER CARDS (See § 10.2, AUXILIARY STORAGE FOR ROUTINES)

When a single Type-6 or Type-7 header precedes several routines, the entire set of routines is loaded into consecutive cells of the buffer and written to auxiliary as a single unit when the next header is encountered. A set of routines too large for the buffer overflows into main memory, using cells from H2. The entire set of routines is brought into main memory when any one of them is executed. Mutual calls between routines in the same set do not result in accesses to auxiliary.

TYPE = 9: FIRST CARD (See § 18.6, 18.7)

SYMB of the first Type-9 card specifies the controlling unit; comments on Type-9 cards are restricted to the COMMENTS field of the coding form.

J171 RETURN UNUSED REGIONAL CELLS TO H2. (See § 17.0, BLOCK HANDLING PROCESSES)

Unused regional cells are not automatically returned to available space at the end of initial loading; they are returned only when J171 is executed.

25.4 EXTENSIONS TO LIST OF SYSTEM CELLS

The cells W30 through W43 have been assigned system functions as described in Sec. 4.2.

25.2 EXTENSIONS TO THE LIST OF BASIC PROCESSES

The following processes have been added; their full descriptions are found in the indicated sections:

LIST PROCESSES (§ 9.8)
J103 Gen cells of block (1) for (0).

AUXILIARY STORAGE PROCESSES (§ 10.1)
J109 Compact auxiliary data storage system (0).

PRINT PROCESSES (§ 16.2)
*J162 Enter (0) according to format W43.

BLOCK HANDLING PROCESSES (§ 17.0)
- J171 Return unused regionals to H2.
- J172 Make block (0) into a list.
- *J173 Read into block (0).
- *J174 Write block (0).
- *J175 FIND region control word of regional symbol (0).
- J176 Space (0) blocks on unit 1W19.

LINE READ PROCESSES (§ 22.0)
- *J180 Read line.
- *J181 Input line symbol.
- *J182 Input line data term (0).
- *J183 Set (0) to next blank.
- *J184 Set (0) to next non-blank.
- *J185 Set (1) to next occurrence of character (0).
- *J186 Input line character.
- *J189 Transfer field to line (0).

PARTIAL WORD PROCESSES (§ 23.0)
- J190 Input P of cell (0).
- J191 Input Q of cell (0).
- J192 Input SYMB of cell (0).
- J193 Input LINK of cell (0).
- J194 Set (1) to be P of cell (0).
- J195 Set (1) to be Q of cell (0).
- J196 Set (1) to be SYMB of cell (0).
- J197 Set (1) to be LINK of cell (0).

MISCELLANEOUS PROCESSES (§ 24.0)
- *J200 LOCATE (0)th symbol on list (1).
- J201 ERASE routine (0).
- J201 Print post mortem and continue.

25.6 EXTENSIONS TO THE LOADER

TYPE = 2: REGION CARDS (See § 18.2, 17.0)

A block control word for a region is created by a Type-2 card, and this region control word is accessible by J175. The loader defines the first symbol of those regions the programmer did not define. The $ region is reserved for system routines and data unique to local installations.

TYPE = 3: BLOCK RESERVATION CARDS (§ 18.3, 17.0)

The loader creates a block control word in the cell appearing in NAME of all Type-3 cards.

TYPE = 5, 6, 7, 8: HEADER CARDS (See § 18.5, 19.0, 14.0, 17.0)

The loading processes load into the available space list 1W34. Sets of data or routines going into main storage may be loaded into cells from the standard available space list H2 (by NAME = blank, 1W34 = H2) or into specific blocks of cells (by NAME = name of block).

A Type-8 editing header inhibits loading of its associated set of routines or data but allows output and listing options; it is intended for skipping sets on an alternate input unit.

INPUT MODE (§ 18.5, 20.0, 13.3)

Header cards with P = 3 indicate machine code; headers with P = 4 indicate restart mode.

OUTPUT MODE (See § 18.5, 13.3)

The integer 3 indicates machine code output; the integer 9 indicates output in IPL standard form.

25.7 EXTENSIONS TO THE MONITOR SYSTEM

The three externally imposed trace conditions may also be imposed internally by setting the data term 1W31 appropriately. (See § 15.5, TRACING.)

A post mortem may be printed at any point in the processing by J202, without terminating the program. A terminal post mortem is still given automatically. (See § 15.4, POST MORTEM.)

25.8 EXTENSIONS TO THE INTERPRETIVE SYSTEM

The interpretation cycle count in H3 is compared each cycle to the number set by the programmer in cell W33. Trapping on the attribute H3 occurs on equality. (See § 21.0, ERROR TRAP.)

When available space is exhausted, a number of cells of reserved space is added to H2 and trapping on the attribute H2 occurs. (See § 21.0, ERROR TRAP.)

LIST OF IPL-V BASIC PROCESSES

* Indicates processes which set H5

General Processes (§ 5.0)
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)

Description Processes (§ 6.0)
*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

Generator Housekeeping Processes (§ 7.1)
J17 Gen set up: context (0), subprocess (1)
*J18 Execute subprocess of Gen
*J19 Gen clean up

Working Storage Processes (§ 8.0)
J2n MOVE (0)-(n) into W0-Wn
J3n Restore W0-Wn
J4n Preserve W0-Wn
J5n Preserve W0-Wn; MOVE (0)-(n) into W0-Wn

List Processes (§ 9.8)
*J60 LOCATE next symbol after cell (0)
*J61 LOCATE last symbol on list (0)
*J62 LOCATE (0) on list (1) (1st occurrence)
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) (1st occur.)
*J68 DELETE symbol in cell (0)
*J69 DELETE (0) from list (1) (1st occurrence)
*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 symbol
*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 Gen cells of block (1) for (0)
J104

Auxiliary Storage Processes (§ 10.1)
*J105 MOVE list structure (0) in from auxiliary
J106 File list structure (0) in fast-auxiliary
J107 File list structure (0) in slow-auxiliary
*J108 TEST if list structure (0) is on auxiliary
J109 Compact auxiliary data storage system (0)

Arithmetic Processes (§ 11.0)
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)

Data Prefix Processes (§ 12.2)
*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 marked 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

Read and Write Processes (§ 14.0)
*J140 Read list structure
*J141 Read symbol from console
J142 Write list structure (0)
J143 Rewind tape (0)
J144 Skip to next tape file
J145 Write end-of-file
J146 Write end-of-set

Monitor System (§ 15.6)
J147 Mark routine (0) to trace
J148 Mark routine (0) to propagate trace
J149 Mark routine (0) to not trace

Print Processes (§ 16.1, 16.2)
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 Enter (0) according to format W43
J163
J164

In-process Loading (§ 19.0)
J165 Load routines and data

Save for Restart (§ 20.0)
*J166 Save on unit (0) for restart
*J167 Skip list structure
J168
J169

Error Trap (§ 21.0)
J170 Trap on (0)

Block Handling Processes (§ 17.0)
J171 Return unused regionals to H2
J172 Make block (0) into a list
*J173 Read into block (0)
*J174 Write block (0)
*J175 FIND region control word of regional symbol (0)
J176 Space (0) blocks on unit 1W19
J177
J178
J179

Line Read Processes (§ 22.0)
*J180 Read line
*J181 Input line symbol
*J182 Input line data term (0)
*J183 Set (0) to next blank
*J184 Set (0) to next non-blank
*J185 Set (1) to next occurrence of character (0)
*J186 Input line character
J187
J188
*J189 Transfer field to line (0)

Partial Word Processes (§ 23.0)
J190 Input P of cell (0)
J191 Input Q of cell (0)
J192 Input SYMB of cell (0)
J193 Input LINK of cell (0)
J194 Set (1) to be P of cell (0)
J195 Set (1) to be Q of cell (0)
J196 Set (1) to be SYMB of cell (0)
J197 Set (1) to be LINK of cell (0)
J198
J199

Miscellaneous Processes (§ 24.0)
*J200 LOCATE (0)th symbol on list (1)
J201 ERASE routine (0)
J202 Print post mortem and continue

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
  Q = 5  Machine language routine
  Q = 6  Routine in fast-aux. storage
  Q = 7  Routine in slow-aux. storage
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  External interrupt cell
W15  Post mortem routine 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 list 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
W30  Field length cell
W31  Trace mode cell
W32  Reserved available space cell
W33  Cycle count for trap cell
W34  Current available space cell
W35  Slow-aux. obsolete structure cell
W36  Used slow-auxiliary space cell
W37  Slow-auxiliary storage density cell
W38  Slow-auxiliary storage compacting
     routine cell
W39  Fast-aux. obsolete structure cell
W40  Used fast-auxiliary space cell
W41  Fast-auxiliary storage density cell
W42  Fast-auxiliary storage compacting
     routine cell
W43  Format 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
       Alphanumeric       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
     NAME = Block control word (if given)
     SYMB = Origin (if given)
     LINK = Size
     Q = 0  Reserve regional symbols
     Q = 1  Reserve print line
     Q = 2  Reserve 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
8  Editing header; inhibits loading
     NAME = Name of storage block
     P    = Input mode
            P = 0  IPL standard
            P = 1  IPL compressed
            P = 2  IPL binary
            P = 3  Machine code
            P = 4  Restart mode
     Q    = Type of input
            Q = 0  Routines; internals
                   symbolic
            Q = 1  Data; internals
                   symbolic
            Q = 2  Routines; internals
                   symbolic; reset inter-
                   nal symbol table
            Q = 3  Data; internals sym-
                   bolic; reset internal
                   symbol table
            Q = 4  Routines; internals
                   absolute
            Q = 5  Data; internals
                   absolute
     SYMB = Alternate input unit
            0 (blank) = controlling unit
            1-10 = Internal tapes
            Regional SYMB names first
              routine (terminate loading)
     LINK = Output mode:  of form bbbcd
            b = Output unit:  blank = unit
                1W19; 1-10 = unit 1-10
            c = 0  (blank) if assembly
                listing
              = 1  or any character if no
                assembly listing
            d = 0  (blank) if no output
              = 1  IPL compressed output
              = 2  IPL binary output
              = 3  Machine code output
              = 9  IPL standard output
9  First card
     SYMB = Controlling unit (0 or blank
              = normal input unit)
```

LIST OF IPL-V BASIC PROCESSES

* Indicates processes which set H5

General Processes (§ 5.0)
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)

Description Processes (§ 6.0)
*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

Generator Housekeeping Processes (§ 7.1)
J17 Gen set up: context (0), subprocess (1)
*J18 Execute subprocess of Gen
*J19 Gen clean up

Working Storage Processes (§ 8.0)
J2n MOVE (0)-(n) into W0-Wn
J3n Restore W0-Wn
J4n Preserve W0-Wn
J5n Preserve W0-Wn; MOVE (0)-(n) into W0-Wn

List Processes (§ 9.8)
*J60 LOCATE next symbol after cell (0)
*J61 LOCATE last symbol on list (0)
*J62 LOCATE (0) on list (1) (1st occurrence)
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) (1st occur.)
*J68 DELETE symbol in cell (0)
*J69 DELETE (0) from list (1) (1st occurrence)
*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 symbol
*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 Gen cells of block (1) for (0)
J104

Auxiliary Storage Processes (§ 10.1)
*J105 MOVE list structure (0) in from auxiliary
J106 File list structure (0) in fast-auxiliary
J107 File list structure (0) in slow-auxiliary
*J108 TEST if list structure (0) is on auxiliary
J109 Compact auxiliary data storage system (0)

Arithmetic Processes (§ 11.0)
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)

Data Prefix Processes (§ 12.2)
*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 marked 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

Read and Write Processes (§ 14.0)
*J140 Read list structure
*J141 Read symbol from console
J142 Write list structure (0)
J143 Rewind tape (0)
J144 Skip to next tape file
J145 Write end-of-file
J146 Write end-of-set

Monitor System (§ 15.6)
J147 Mark routine (0) to trace
J148 Mark routine (0) to propagate trace
J149 Mark routine (0) to not trace

Print Processes (§ 16.1, 16.2)
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 Enter (0) according to format W43
J163
J164

In-process Loading (§ 19.0)
J165 Load routines and data

Save for Restart (§ 20.0)
*J166 Save on unit (0) for restart
*J167 Skip list structure
J168
J169

Error Trap (§ 21.0)
J170 Trap on (0)

Block Handling Processes (§ 17.0)
J171 Return unused regionals to H2
J172 Make block (0) into a list
*J173 Read into block (0)
*J174 Write block (0)
*J175 FIND region control word of regional symbol (0)
J176 Space (0) blocks on unit 1W19
J177
J178
J179

Line Read Processes (§ 22.0)
*J180 Read line
*J181 Input line symbol
*J182 Input line data term (0)
*J183 Set (0) to next blank
*J184 Set (0) to next non-blank
*J185 Set (1) to next occurrence of character (0)
*J186 Input line character
J187
J188
*J189 Transfer field to line (0)

Partial Word Processes (§ 23.0)
J190 Input P of cell (0)
J191 Input Q of cell (0)
J192 Input SYMB of cell (0)
J193 Input LINK of cell (0)
J194 Set (1) to be P of cell (0)
J195 Set (1) to be Q of cell (0)
J196 Set (1) to be SYMB of cell (0)
J197 Set (1) to be LINK of cell (0)
J198
J199

Miscellaneous Processes (§ 24.0)
*J200 LOCATE (0)th symbol on list (1)
J201 ERASE routine (0)
J202 Print post mortem and continue

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
  Q = 5 Machine language routine
  Q = 6 Routine in fast-aux. storage
  Q = 7 Routine in slow-aux. storage
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 External interrupt cell
W15 Post mortem routine 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 list 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
W30 Field length cell
W31 Trace mode cell
W32 Reserved available space cell
W33 Cycle count for trap cell
W34 Current available space cell
W35 Slow-aux. obsolete structure cell
W36 Used slow-auxiliary space cell
W37 Slow-auxiliary storage density cell
W38 Slow-auxiliary storage compacting routine cell
W39 Fast-aux. obsolete structure cell
W40 Used fast-auxiliary space cell
W41 Fast-auxiliary storage density cell
W42 Fast-auxiliary storage compacting routine cell
W43 Format 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
Alphanumeric	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
    NAME = Block control word (if given)
    SYMB = Origin (if given)
    LINK = Size
    Q = 0 Reserve regional symbols
    Q = 1 Reserve print line
    Q = 2 Reserve 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
8 Editing header; inhibits loading
    NAME = Name of storage block
    P = Input mode
        P = 0 IPL standard
        P = 1 IPL compressed
        P = 2 IPL binary
        P = 3 Machine code
        P = 4 Restart mode
    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 = Alternate input unit
        0 (blank) = controlling unit
        1-10 = Internal tapes
        Regional SYMB names first routine (terminate loading)
    LINK = Output mode: of form bbbcd
        b = Output unit: blank = unit 1W19; 1-10 = unit 1-10
        c = 0 (blank) if assembly listing
          = 1 or any character if no assembly listing
        d = 0 (blank) if no output
          = 1 IPL compressed output
          = 2 IPL binary output
          = 3 Machine code output
          = 9 IPL standard output
9 First card
    SYMB = Controlling unit (0 or blank = normal input unit)

LIST OF IPL-V BASIC PROCESSES

* Indicates processes which set H5

General Processes (§ 5.0)

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)

Description Processes (§ 6.0)

*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

Generator Housekeeping Processes (§ 7.1)

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

Working Storage Processes (§ 8.0)

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

List Processes (§ 9.8)

*J60	LOCATE next symbol after cell (0)
*J61	LOCATE last symbol on list (0)
*J62	LOCATE (0) on list (1) (1st occurrence)
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) (1st occur.)
*J68	DELETE symbol in cell (0)
*J69	DELETE (0) from list (1) (1st occurrence)
*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 symbol
*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	Gen cells of block (1) for (0)
J104	

Auxiliary Storage Processes (§ 10.1)

*J105	MOVE list structure (0) in from auxiliary
J106	File list structure (0) in fast-auxiliary
J107	File list structure (0) in slow-auxiliary
*J108	TEST if list structure (0) is on auxiliary
J109	Compact auxiliary data storage system (0)

Arithmetic Processes (§ 11.0)

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)

Data Prefix Processes (§ 12.2)

*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 marked 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	

Read and Write Processes (§ 14.0)

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

Monitor System (§ 15.6)

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

Print Processes (§ 16.1, 16.2)

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	Enter (0) according to format W43
J163	
J164	

In-process Loading (§ 19.0)

J165	Load routines and data

Save for Restart (§ 20.0)

*J166	Save on unit (0) for restart
*J167	Skip list structure
J168	
J169	

Error Trap (§ 21.0)

J170	Trap on (0)

Block Handling Processes (§ 17.0)

J171	Return unused regionals to H2
J172	Make block (0) into a list
*J173	Read into block (0)
*J174	Write block (0)
*J175	FIND region control word of regional symbol (0)
J176	Space (0) blocks on unit 1W19
J177	
J178	
J179	

Line Read Processes (§ 22.0)

*J180	Read line
*J181	Input line symbol
*J182	Input line data term (0)
*J183	Set (0) to next blank
*J184	Set (0) to next non-blank
*J185	Set (1) to next occurrence of character (0)
*J186	Input line character
J187	
J188	
*J189	Transfer field to line (0)

Partial Word Processes (§ 23.0)

J190	Input P of cell (0)
J191	Input Q of cell (0)
J192	Input SYMB of cell (0)
J193	Input LINK of cell (0)
J194	Set (1) to be P of cell (0)
J195	Set (1) to be Q of cell (0)
J196	Set (1) to be SYMB of cell (0)
J197	Set (1) to be LINK of cell (0)
J198	
J199	

Miscellaneous Processes (§ 24.0)

*J200	LOCATE (0)th symbol on list (1)
J201	ERASE routine (0)
J202	Print post mortem and continue

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
- Q = 5 Machine language routine
- Q = 6 Routine in fast-aux. storage
- Q = 7 Routine in slow-aux. storage

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	External interrupt cell
W15	Post mortem routine 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 list 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
W30	Field length cell
W31	Trace mode cell
W32	Reserved available space cell
W33	Cycle count for trap cell
W34	Current available space cell
W35	Slow-aux. obsolete structure cell
W36	Used slow-auxiliary space cell
W37	Slow-auxiliary storage density cell
W38	Slow-auxiliary storage compacting routine cell
W39	Fast-aux. obsolete structure cell
W40	Used fast-auxiliary space cell
W41	Fast-auxiliary storage density cell
W42	Fast-auxiliary storage compacting routine cell
W43	Format 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:

Data term:	±PQ	SYMB	LINK
Decimal integer	1	dddd	dddd
Floating point	11	ddddd	d ±ee
Alphanumeric	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
- NAME = Block control word (if given)
- SYMB = Origin (if given)
- LINK = Size
- Q = 0 Reserve regional symbols
- Q = 1 Reserve print line
- Q = 2 Reserve 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
8 Editing header; inhibits loading
- NAME = Name of storage block
- P = Input mode
  - P = 0 IPL standard
  - P = 1 IPL compressed
  - P = 2 IPL binary
  - P = 3 Machine code
  - P = 4 Restart mode
- 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 = Alternate input unit
  - 0 (blank) = controlling unit
  - 1-10 = Internal tapes
  - Regional SYMB names first routine (terminate loading)
- LINK = Output mode: of form bbbcd
  - b = Output unit: blank = unit 1W19; 1-10 = unit 1-10
  - c = 0 (blank) if assembly listing
    - = 1 or any character if no assembly listing
  - d = 0 (blank) if no output
    - = 1 IPL compressed output
    - = 2 IPL binary output
    - = 3 Machine code output
    - = 9 IPL standard output

9 First card
- SYMB = Controlling unit (0 or blank = normal input unit)

INDEX

(Index references are to section numbers, except where noted by the letter "p", in which case it is a particular page being referenced.)

Alternate input units (see also: Input units) .. 18.7, 18.9, 19.0

Arithmetic processes, J110 to J129 .... 11.0

Assembly listing 18.8, 18.5

Attributes 2.5, 2.3-2.7, 6.0

Auxiliary storage 10.0, 1.11

processes, J105 to J109.............................. 10.0

for data structures 10.1, 18.5, 9.8

for routines 10.2, 18.3, 18.5

loading into 18.5, 19.0, 18.3

Available space 1.9

amount of 18.3

counting, J126 .... 11.0

for loading, 1W34 .... 18.5, 19.0, 4.2

private blocks of 17.0, 18.5, 19.0

reserved, 1W32 .... 4.2, 18.0, 15.1

returning regionals to, J171 .... 17.0, 24.0

trap when exhausted 21.0

Blocks 17.0

control words for 17.0, 18.2, 18.3

generator for, J103 .... 9.8

loading into 18.5, 19.0, 4.2

processes for handling 17.0

regions 18.2, 17.0

reserving, Type-3 cards 18.3

Buffer for auxiliary routines 10.2, 18.3, 17.0

Cells

CIA, H1 .... 3.13, 4.2

communication, H0 .... 3.7, 3.8

head 1.13, 2.1

list 1.13, 2.1, 2.2

Cells (continued)
names ... 1.12
push down ... 1.14, 1.15
private termination ... 9.5, 1.13, 9.4
safe ... 3.6, 4.2
storage ... 1.14, 1.15
system ... 4.2
termination ... 1.13, 9.5
test, H5 ... 3.9, 4.2, 5.0
Changes ... 25.1-25.3
CIA cell, H1 ... 3.13, 15.6, 4.2
Coding form ... 1.6
example (blank) ... 1.6
example (showing data terms) ... 1.7
Comment cards, Type-1 ... 18.1, 18.8
Communication cell, H0 ... 3.7, 3.8, 4.2
Context ... 7.0
Copy ... 9.7
cell of any kind, J120 ... 11.0
data term, J120 ... 11.0
list, J73 ... 9.8, 10.1
list structure, J74 ... 9.8, 10.1
parts of words ... 23.0
Current auxiliary routine ... 10.1, 4.2
Current available space list, 1W34 ... 18.5, 4.2, 19.0
Current column, 1W25 ... 16.2, 4.2
Current instruction address cell, H1 ... 3.13, 15.6, 4.2
Cycles, interpretation
count for trap cell, W33 ... 21.0, 4.2
explanation of ... 3.12-3.16
flow chart of ... 3.16
rules of ... 3.14
tally of, H3 ... 3.16, 21.0, 4.2
trap on count of ... 21.0, 4.2

Data
 header cards for loading 18.5
 in auxiliary storage 10.1, 18.5
 in routines 3.5, 24.0
 initial loading of 18.5
 in-process loading of 19.0
 read-write processes for 14.0, 22.0
 sets of 18.5
Data list (see also: Data list structure) 2.1
Data list structure 2.0, 2.8
 auxiliary storage for 10.1, 18.5
 formation rules for 2.8
 loading of 18.5, 19.0, 14.0
 obsolete 10.1, 4.2
 printing of 16.0
 processes for manipulating, J60 to J104 .... 9.0
  copy .... p. 52
  erase .... p. 52
  generate .... pp. 54, 55
Data prefix processes, J130 to J139 .... 12.0, 12.2, 23.0
Data terms 1.5
 example on coding form 1.7
 P, data type of 1.7, 11.0
 processes for, J110 to J129 .... 11.0
 Q, prefix of 12.0
 reading and printing of 22.0, 16.1, 16.2
Delete 9.4, 9.5
Describable lists 2.3-2.7, 6.0
Description list 2.6, 2.3-2.7
 processes for, J10 to J16 .... 6.0
Description processes, J10 to J16 .... 6.0
Designated symbol, S 3.10
Designation operation, Q 3.10
Editing header, Type-8 .... 18.5

Enter into printline, J156 to J162 .... 16.2
Erase 9.6
block, J172 .... 17.0
cell, J9 .... 5.0
list, J71 .... 9.8
list structure, J72 .... 9.8
unused regionals, J171.................................... 17.0
routine, J201 .... 24.0
Error trap, J170 .... 21.0
Extensions 25.4-25.8
External interrupt 15.3, 15.1, 4.2
External tapes 13.1, 14.0
External trace mode, W31 .... 15.1, 4.2
Filed list structure 10.1
Find 5.0
attribute randomly, J16 .... 6.0
nth symbol, J8n 9.8
region control word, J175 .... 17.0
value of attribute, J10 .... 6.0
First card, Type-9 .... 18.6, 18.7, 18.9
Generators
conventions for constructing 7.3
conventions for using 7.2
housekeeping processes 7.1
of block, J103 .... p. 55
of list, J100 .... p. 53
of list structure, J101 .... p. 54
of tree, J102 .... p. 55
General processes, J0 to J9 .... 5.0
H0, communication cell 3.7, 3.8, 4.2
H1, CIA cell 3.13, 15.6, 4.2
H2, available space list (see also: Available space) .. 1.9, 4.2
H2 trap 21.0, 4.2
H3, interpretation cycle tally 3.16, 4.2, 21.0
H3 trap 21.0, 4.2
H5, test cell 3.9, 4.2, 5.0

Header cards, Type-5, 6, 7, or 8 .... 18.5
Heads of lists 1.13
Initial loading 18.0
   order of deck for 18.9
In-process loading 19.0
Input
   a symbol, P = 1 .... 3.11
   deck for loading 18.9, 18.0
   line symbols, data terms, and characters, J180 to J189 .. 22.0
   partial words, J190 to J193 .... 23.0
Input mode 13.3
   cell, W16 .... 4.2, 14.0
   on header cards 18.5
Input-output 13.0
   conventions 13.0
   processes for
      block handling, J171 to J176 .... 17.0
      initial loading 18.0
      in-process loading, J165 .... 19.0
      line read 22.0
      print, J150 to J162 .... 16.0
      read and write, J140 to J146 .... 14.0
      save for restart, J166 .... 20.0
   representation mode 13.3
   unit code 13.2
Inputs of routines 3.7, 3.8
Input unit (see also: Input-output)
   alternate 18.7, 18.9, 19.0
   cell, W18 .... 4.2, 19.0, 22.0
   code for 13.2
   controlling 18.7, 19.0, 18.6, 18.9
   normal 18.5
Insert 9.3
   processes, J63 to J66 .... 9.8
Instructions 3.2, 23.0

Internal symbols 1.3, 2.2
   detecting, J130 to J139 .... 12.0
   symbolic or absolute for loading 18.5, 18.9, 14.0, 22.0
   rules for writing 1.6
   table of definitions 18.9
Interpretation 3.12
   cycles of (see: Cycles, interpretation)
   explanation of 3.12-3.16
   flow chart of 3.16
   rules of 3.14
Interpretive system, IPL-V 1.2
Interrupt, external 15.3, 15.1, 4.2
IPL binary representation 13.5
IPL compressed representation 13.4
Levels
   in data list structures 2.10
   in routines 3.4
Lines, print and read
   naming and reserving 18.3
   printing 16.2
   reading 22.0
   loading 18.3
LINK
   of block control words 17.0, 18.2, 18.3
   of cells 1.8
   of coding form 1.6
   of data lists 2.1
   of header cards 18.5
   of instructions 3.2, 3.3
   of program lists 3.3
   of Type-2 cards 18.2, 17.0
   of Type-3 cards 18.3, 17.0
List cells 1.13, 2.1, 2.2
List processes, J60 to J104 .... 9.0

List structure, data 2.0, 2.8
auxiliary storage for 10.1, 18.5
formation rules for 2.8
loading of 18.5, 19.0, 14.0
obsolete 10.1, 4.2
printing of 16.0
processes for manipulating, J60 to J104 .... 9.0
copy .... p. 52
erase .... p. 52
generate .... pp. 54, 55
List structure, other 2.11
List structure, routine 3.0, 3.4
erasing, J201 .... 24.0
Listing, assembly 18.8, 18.5
Listing cards, Type-4 .... 18.4, 18.8
Lists
data (see also: List structures, data) 2.1
describable 2.3-2.7, 6.0
description 2.6, 2.3-2.7, 6.0
program 3.3
push down 1.15
Loading 18.0, 19.0
from alternate units 18.7
initial 18.0
order of deck for 18.9
in-process 19.0
into specific blocks 18.5
of data only, J140 .... 14.0
to auxiliary storage 10.0, 18.5
Local symbols 1.3
detecting, J130 to J139 .... 12.0
domain of definition 2.9
in data list structures 2.8
rules for writing 1.6
table of definitions 18.9

Locate 9.2
processes, J60 to J62, J200 .... 9.8, 24.0
Marking processed 12.2
Monitor point, Q = 3 .... 15.1
Monitor system, J147 to J149 .... 15.0
Move
definition 5.0
structure in from auxiliary, J105 .... 10.1
from HO to working storage, J2n and J5n 8.0
NAME 1.12, 2.2, 1.6, 3.4, 2.1
of blocks 18.3, 18.5, 17.0
of coding form 1.6
of data list structures 2.8
of data lists 2.1
of header cards 18.5, 17.0
of list cells 2.2
of regions 18.2
of routines 3.4
of Type-2 cards 18.2, 17.0
of Type-3 cards 18.3, 17.0
Operation code, P 3.11, 23.0
Output (see also: Enter; Store)
a symbol, P = 2 .... 3.11
during loading 18.5
partial words 23.0
Output mode 13.3
cell, W17 .... 4.2, 14.0
on header cards 18.5
Outputs of routines 3.7, 3.8
Output unit (see also: Input-output)
cell for printing, W20 .... 16.0, 18.5, 4.2
cell for writing, W19 .... 14.0, 18.5, 4.2
code for 13.2
for save for restart 20.0

P, data type code 1.7, 11.0
P, operation code 3.11, 23.0
Pop up 1.15, 8.0, 9.0
Post mortem, J202 .... 24.0, 15.4
Preserve 1.15, 8.0, 9.0
Primitive processes 1.2, 3.1, 13.3, 18.3, 18.5
Print lines 16.0, 16.2, 18.3, 22.0
Print processes, J150 to J162 .... 16.0
Private termination cells 9.5, 1.13, 9.4
Processes
  arithmetic, J110 to J129 .... 11.0
  auxiliary storage, J105 to J109 .... 10.1
  basic system of 4.0
  block handling 17.0
  data prefix, J130 to J139 .... 12.0
  description, J10 to J16 .... 6.0
  error trap, J170 .... 21.0
  general, J0 to J9 .... 5.0
  generator housekeeping, J17 to J19 .... 7.0
  in-process loading, J165 .... 19.0
  line read, J180 to J189 .... 22.0
  list, J60 to J104 .... 9.8
  miscellaneous, J200 to J209 .... 24.0
  monitor system, J147 to J149 .... 15.0
  partial word, J190 to J199 .... 23.0
  print, J150 to J162 .... 16.0
  read and write, J140 to J146 .... 14.0
  save for restart, J166 to J167 .... 20.0
  working storage, J20 to J59 .... 8.0
Program lists 3.3, 3.4
Programs 3.4
Program, rules for 3.4
Push down 1.15, 8.0, 9.0
Push down lists 1.15

Q, data 12.0, 23.0
Q, of Type-3 and header cards 18.3, 18.5
Q, routines 3.10, 23.0
Read and write processes, J140 to J146 .... 14.0
Recursions 12.1
Region cards, Type-2 .... 18.2, 17.0
Region control word 17.0, 18.2
Regional symbols 1.3, 1.6, 18.2, 17.0
Reserved available space 4.2, 18.0, 15.1
Restart 20.0
Restore 1.15, 8.0, 9.0
Routines 3.0, 3.4
   auxiliary storage for 10.2
   data in 3.5, 24.0
   erasing 24.0
   inputs and outputs of 3.7, 3.8
   rules for 3.4
S, designated symbol 3.10
Safe cells 3.6
   list of 4.2
Save for restart, J166 to J167 .... 20.0
Set of input routines or data 18.5
Set full word, J121 .... 11.0
Set partial word, J194 to J197 .... 23.0
Skip
   block on unit 1W19, J176 .... 17.0
   list structure, J167 .... 20.0
   set during loading, Type-8 card 18.5
   to next file, J144 .... 14.0
Start card 18.9, 18.5
Superroutine 7.0, 15.6
Subprocess 7.0, 15.5
Snapshots 15.2, 15.1

Storage
auxiliary 1.11
for data list structures 10.1, 18.5, 9.8
for routines 10.2, 18.3, 18.5
intermediate (tapes) 13.1, 17.0
main (see also: Available space) 1.2, 1.11
working, W0 to W9 .... 8.0, 4.2
Storage cells 1.14, 1.15
SYMB
of block control words 17.0, 18.2, 18.3
of cells 1.4, 1.8, 1.14
of coding form 1.6
of data list heads 2.3-2.7
of header cards 18.5
of instructions 3.2
of Type-2 cards 18.2, 17.0
of Type-3 cards 18.3, 17.0
of Type-9 cards 18.6, 18.7
Symbols (see also: Internal; Local; Regional) 1.3
termination 1.13, 2.1, 2.2
System
cells 4.2
IPL-V, components of 1.2
interpreter 3.12-3.16
loader 18.0, 19.0
monitor 15.0
primitive processes, list of .... p. 111
regions 4.1, 18.2
Tally of interpretation cycles, H3 (see also:
Cycles, interpretation) 3.16, 21.0, 4.2
Tapes, external 13.1, 14.0
Terminate program
after n interpretation cycles, W33 .... 21.0
because of internal errors 15.4
for restart, J166 .... 20.0, 15.3

Terminate program (continued)
normally 3.14
via external interrupt, 1W14 .... 15.3
Termination
cells 1.13, 9.5, 9.4
of data lists 2.1, 2.2
of each set during loading 18.9
of each structure during loading 14.0
of loading 18.9
of processing (see: Terminate program)
of routines 3.3, 3.4
symbols 1.13, 2.1, 2.2
Test, definition of 5.0
Test cell, H5 .... 3.9, 3.11, 5.0
Trace 15.0
external mode of, 1W31 .... 15.1, 15.5, 4.2
internal mode of 15.1, 15.5, 15.6
format of 15.5
mark carried in H1 .... 15.6
of generators, subprocesses, superroutines 15.6
primitives, J147 to J149 .... 15.6
Trap, error 21.0
arbitrary trap conditions, J170 .... 21.0
cells involved
W26 to W29 .... 4.2
W32, W33 .... 4.2
standard trap conditions
available space exhausted 21.0
interpretation cycle count 21.0
Type cards 18.0
order of in input deck 18.9
Type-1: comment cards 18.1
Type-2: region cards 18.2, 17.0
Type-3: block reservation cards 18.3, 17.0
Type-4: listing cards 18.4
Type-5, 6, 7, 8: header cards 18.5
Type-9: first card 18.6, 18.7

Unit
code 13.2, 18.5
input (see: Input unit)
output (see: Output unit)
Values of attributes 2.5, 2.3-2.7, 6.0
Words
IPL standard 1.4
IPL special 1.5
block control 17.0
Working storage processes, J20 to J59 .... 8.0

IPL-V CODING SHEET

Problem No.________ Programmer ________ Date ________ Page ____ of ____

COMMENTS	TYPE		NAME	SIGN	PQ	SYMB		LINK		COMMENTS	I.D.
06–40	41	42	43–47	48	49–50	51–55	56	57–61	62	63–72	73–80