TEMAC:

A Machine Language for Processing Text

PART I
Programming Manual

M. D'Imperio
R405

CONTENTS

4. Definition Statements.

4.1 Alphabet Definitions.

4.2 Data Object Definitions.

4.2.1 The Paragraph.
4.2.2 The Sentence.
4.2.3 The Word.
4.2.4 The Master-Block.
4.2.5 The Cell-List.
4.2.6 The Cell.

5. Operation Statements.

5.1 Structural Statements.
(SEG, SEL, COL, OMT, INC, MTY, REF)
5.2 Data Moving Statements: Textual.
(EXT, EDT, CVT)
5.3 Data Moving Statements: Cellular.
(MOVE, REPT, CVN, ENTER, LKP)
5.4 Data Testing Statements: Textual.
(LOC, COM)
5.5 Data Testing Statements: Cellular.
(FIND)
5.6 Reading and Writing Statements.
(READ, WRITE, OPEN, CLOSE, WEF, REW, BSR, BSF)
5.7 Control Statements: Decisions.
(STEP, MOVAD, BRAN, IF)
5.8 Control Statements: Transfers of Control, Subroutines.
(DO, EXIT, RETURN, FINISH, START, GOTO)
5.9 Debugging Statements.
(Tracing and Displaying)

6. The Object Program.

6.1 The Job and the Deck.

6.1.1 The Job.
6.1.2 The TEMAC Deck; Its Relation to Other Decks in the Job.
6.1.3 The Arrangement of Cards in the Job.
6.1.4 Control Cards (see also Appendix I).
6.1.5 File Cards (see also Appendix I).

6.2 Compilations, Assemblies, Runs.

6.2.1 The TEMAC Program at Compile and Assembly Time.
6.2.2 The TEMAC Program at Run Time.
6.2.3 The [illegible] and [illegible] (FORTRAN) Character Sets.
6.2.4 [illegible] of Messages at Compile Time.
6.2.5 [illegible] Times at Run Time.

1. Introduction.

The TEMAC (Text Macro Compiler) language is a problem-oriented programming language specifically designed for the processing of unedited text, whose contents and form may be greatly garbled or widely-variable. Programs written in this language will be capable of applying operations such as word-bank lookups, scanning, extracting, and editing directly to raw text as recorded on magnetic tape without the need for hand editing or key-punching from hard copy. The TEMAC system operates under the "IBJOB" monitor for the 7094; it includes the TEMAC Compiler Program and the TEMAC Processor. The compiler program accepts statements in the TEMAC language and produces object coding in 7094 "IBMAP" (assembler) language, convertible directly into 7094 machine code. The TEMAC Processor is a package of subroutines in IBMAP language which executes the macro-instructions of the object program at "run-time" and provide user-oriented debugging and tracing facilities.

2. The TEMAC Language.

TEMAC, as a problem-oriented language for handling unedited text, provides certain specialized ways of describing data and certain kinds of actions to be performed on the data thus described. The kinds of descriptions for data will be called "data types", and the objects defined or described by means of them will be called "data objects". The actions to be carried out upon various data objects, having given data types, in statements of the TEMAC language, will be called "operations". The data objects and operations of TEMAC are like the nouns and verbs of the English language. Just as nouns in English enter into phrases with articles and adjectives in a sentence, data object names in TEMAC enter into special phrases in the statement called "operands", which permit patterns of components within a data object to be selected and acted upon by the operation of the statement.

2.1 TEMAC Data Objects and Operands.

There are two major ways in which data may be handled by a TEMAC program; as textual data - i.e., in a continuous stream of bytes or characters - or as cellular data - i.e., stored one item to a machine register or "cell". All data that is not textual is cellular. Some TEMAC statements require textual data objects in their operands, while others require cellular data objects.

2.1.1 Textual Data Objects: Words, Sentences, Paragraphs.

Text is made up of characters, arranged in strings which may be either free-standing words (i.e., surrounded by word-boundary-characters), or arbitrary character sequences chosen for some purpose (for example, the third through the seventh characters of a stream). Words may, in turn, be arranged in strings, which may be set off by sentence-boundary-characters or arbitrarily assembled; the word-string is called a "sentence". Sentences themselves may, finally, be arranged in strings, set off by paragraph-boundary-characters or arbitrarily

assembled; the sentence-string is called a "paragraph". It should be noted that the words "paragraph","sentence", "word" as used for TEMAC data object types, are not synonymous in meaning with the conventional English words, though they frequently refer to similar things. Larger units then the paragraph, or smaller units than the character do not appear to be required in most text-handling applications. TEMAC provides facilities for defining and manipulating individually-named variables or constants of all three data types described above.

2.1.2 Cellular Data Objects: Cells, Cell-Lists, Master-Blocks.

TEMAC provides three data types for describing and handling items such as numbers or names of data objects and statements, which need not be "textual", and are stored in individual machine registers in a format that is much more quickly and cheaply handled in the 7094 than text. One number or name resides in a single "cell"; a string of numbers or names is stored in a "cell-list" or "list"; and a string of such lists (for example, a matrix of numeric values), occupies a "master-block". Numeric values in TEMAC may be decimal signed numbers or octal numbers, but not floating point numbers.

2.1.3 Operands. The data object names may be called upon to provide all or some selection of their components for processing by the operation, or "verb" as it will frequently be called below. The data object name is accompanied by other special words and variable names - the "modifier" and "modifier type code" - to form the operand in a statement of the TEMAC language. Various types of operands offer many ways of selecting components within an object, or stringing together several objects to form a compound object to which the action of the verb is applied. The special words called modifiers are a kind of "index" values, which may be of types called "pointers" or "pointer-sets"; these objects function like arrows or sliding markers attached to components in a string, and specify elements of the string by their relative position from left to right within it.

2.1.4 Conversion. Data coming into the machine from tape, cards, etc., is always in textual form, and must be converted to cellular form before it can be acted on by those TEMAC verbs which require cellular data. Similarly, data objects developed within the machine in the form of counts, results of computation, position numbers in pointers or pointer-sets, etc., must be converted to textual form before they may be written out for printing.

2.2 Alphabets, Sorting Sequences, and Character-Categories.

TEMAC provides the capability of setting up and calling by name any number of different alternative and concurrently available alphabets and associated sorting sequences; these may be composed entirely at the discretion of the programmer, and need not follow any standard set or sequence. Characters within an alphabet may be chosen from the FORTRAN or IBMAP binary-coded decimal ("BCD") character set, which may be punched on the key-punch or printed by the peripheral printers using the "FORTRAN" print chain. Alternatively, characters may be represented by a special "octal-coded" method which allows any six-bit byte (or octal character between "ØØ" and "77") to be represented and utilized as an alphabet character regardless of its validity as a printable or punchable IBM character. Characters that are "invalid" for printing or punching may not, of course, be printed in the output of the program. The alphabet definition statement allows the programmer to specify those bytes, or characters within the data, in which he is interested for a given purpose. The characters are assigned by the programmer to certain character-categories, which may be up to nine in number in any one alphabet; examples of some useful categories are "even numbers", "word-boundary-characters", "invalid characters", "vowels". The categories are used in special forms of the data testing statements for textual data, and boundary categories are used in certain other statements as well. Several different alphabets may refer to the same data stream and be used alternatively in different statements.

2.3 TEMAC Operations.

In processing text, two major kinds of actions are required: data manipulation actions, and data testing actions. A program to process text alternately examines, or scans the text by means of testing operations, and selects, rearranges, or alters the text by means of manipulation actions in accordance with results of the tests. In addition, any programming language requires reading and writing operations for bringing material into memory and putting it out, and control actions for determining the path of control through the program. Finally, debugging operations are helpful, and almost mandatory with a compiler language, to aid in checking out the program.

2.3.1 Data Manipulation Operations. TEMAC does not require that data be physically moved unless this is actually intended or required; if all that is needed is a temporary record of changed relationships among parts of a data object, as an intermediate result to be used later as a basis for moving or altering the data stream, it is much more economical (especially for textual objects) to rearrange or record symbols standing for the data or for some part of it, leaving the data itself unchanged in its original location. TEMAC provides a large set of Structural Operations to act upon symbols, tags, or "descriptors" that stand for data objects and components. When it is desirable

or necessary to alter data or copy it into a new location, the appropriate Data Moving Operations provided by TEMAC are used. TEMAC provides a powerful facility for editing textual data character by character under the control of editing "pictures" or specifications called "edit control strings", and consisting of strings of edit control characters. These may be set up by the programmer to carry out just those actions he wishes in editing the data. For copying data or parts of it into a new location, an "Extracting" statement is provided, with many helpful options and special features.

2.3.2 Data Testing Operations. TEMAC includes a powerful facility for "matching" or "comparing" textual or cellular data objects to single "comparand" objects or to strings of comparand objects. The textual data testing statements may make use of character categories in alphabets, or sorting sequences, and their actions may be carried out with the aid of a special pattern called a "match control string", made up of "match control characters", similar to the control patterns for editing, to specify requirements character by character in terms of literal values or character categories desired at a given position for a match. The data testing statement is a "conditional" statement, providing a yes or no exit path, and in addition, it provides the position number(s) of those objects or that object which passed the test, in the form of a pointer or pointer set; this result object may subsequently be used in an operand of some other statement which applies an operation to just those object(s) that passed the test in the testing statement.

2.3.3 Reading and Writing Operations. TEMAC uses the IOCS (Input-Output Control System) facility of the 7094 Operating system, under IBSYS, to read data from tape or cards and to write data out to tape.

2.3.4 Control Operations. Each TEMAC statement may specify which other statements are to be executed upon completion of its own actions. The names of successor statements are specified in the "routing fields" of the statement; if one of the exits allowed for a statement is not supplied with a routing field, it is assumed that control is to proceed to the statement that follows on the coding sheet. In addition to this built-in sequencing for each statement, there are special "go to" or "branching" operations to allow decisions among many alternative paths and also facilities for entering and leaving subroutines or executing individual statements "at a distance" from various points in the program.

2.3.5 Debugging Operations. TEMAC offers a wide variety of statements for displaying contents of variables, labelled with their names if desired, or other helpful information during debug runs. The "displays", "dumps", and "trace" outputs are placed on a special "Run Log" tape which may be used to produce a detailed and fully annotated record of all events during a debug run, or for diagnostic purposes, during operational runs.

3. The Statements of the Language.

There are two main classes of TEMAC statements: Definition Statements and Operation Statements. Definition statements set up named data objects and alphabets. Operation statements apply the actions specified by TEMAC verbs to operands formed around the defined alphabets and data objects. In the following paragraphs, and Sections 4 and 5, detailed descriptions of the fields within definition and operation statements will be provided, along with examples of their use.

3.1 How to Write TEMAC Statements.

Every TEMAC statement must begin with the letters *T, which tell the TEMAC compiler program that the statement is in the TEMAC language; every statement must also have a name, a verb, and must end in a dollar sign. The dollar sign, called the "statement end marker", or "SEM", marks the end of material to be compiled in the statement. Comments or explanatory material, which will not be compiled but will appear on the listing, may be punched to fill out the last card or cards of the statement after the dollar sign. Each TEMAC statement begins on a new card; more than one statement cannot be placed on a single card. TEMAC statements are written "free-field," and punched into cards as if they were running text, with the exception of the initial *T, which must appear in columns 1 and 2 of the first card. Each subsequent card of the statement after the first must have a blank in column 1. Some definition statements may have two parts; the first part, containing the statement proper, is called the "header,"and the second part is the "data." In statements having only a header, this may overflow one card (72 positions) and occupy from one to seventeen additional cards after the first. When there is data, it may occupy from one to seventeen cards, and the header must be restricted to one card. In either case, the number of overflow cards, over and above the header, must be placed immediately after the *T, with no intervening blank. No single TEMAC statement may occupy a total of more than eighteen cards. If a word of data or header threatens to be broken at the end of a card, it should all be placed on the next card after the initial blank, and blanks left in the previous card in its place. A good practice to follow in preparing TEMAC statements for keypunch is to leave both positions 3 and 4 of each header card blank, so that, if the statement runs over onto another card or so, an overflow card number 1 to 9 may be inserted after the *T without requiring that the card be repunched, or the line be recopied on the keypunch sheets. All fields within the header must be separated by at least one blank, slash, or parenthesis, and fields may not be run together. Any number of these field separator characters (word end markers, or "WEM"s) may be placed between fields, and they may be used interchangeably; certain usages are strongly urged upon the programmer here, for reasons of readability, but departures from these will not usually cause compile errors. In data, only blanks must appear between words or groups, and any number of consecutive blanks may be used. The crucial rule for writing TEMAC statements is that at least one "WEM" must appear between fields that are shown so separated in statement formats and examples below; run-together fields will always be erroneously interpreted by the compiler.

3.2 Definition Statements.

3.2.1 Kinds of Definition Statements.

Definition statements are of two main kinds: Alphabet Definitions (verb "ALF") and Data Object Definitions (verb "DEF"). Alphabet definitions have only one form, which will be described in Section 4.1. Data objects may be textual (paragraphs, sentences, words), or cellular (master-blocks, cell-lists, cells); they may either be defined as constants, when actual values are inserted into them ahead of time, or as variables, when no actual values are entered into storage, but space is saved and descriptors are set up for the data object structure desired. Definitions of constants are called Immediate Definitions in TEMAC, and definitions of variables may be Dummy, Padded, or Fielded Definitions.

A Dummy Definition sets up a structural description (i.e., a set of short tags or descriptors standing for data) associated with the data object name. No actual space is reserved for the data itself, but only for descriptors, which usually take up far less room. The same stream of actual data, either read or moved into an area or entered into storage in an immediate definition, may be referred to by many different dummy definitions, when in the course of the program the descriptors for the data stream are moved into various dummy spaces. Similarly, one dummy definition may refer to different data streams at different times, as their descriptors are moved into it. The way in which this is accomplished should become evident in the discussion of the Operation statements in Section 5.

Padded Definitions are used for cell-lists only. The Padded definition sets up a structural description and also an empty storage reservation, which is preset to zero (e.g., for count cells), blank (e.g., for input and output formation areas), or some other convenient "pad" value. Data will later be read or moved into such a list by the program.

Fielded Definitions are used for sentences and words only, and set up arbitrary descriptions for text fields in terms of character positions and character-length within a cell-list (for example, fields within an input or output formation area or "buffer"). Any textual material currently occupying the buffer may then be acted upon uniformly in data moving or data testing operations by means of the field definitions, regardless of word, sentence, or paragraph boundary characters in it.

3.2.2 The General Form of Definition Statements.

*T NAME VERB DATA TYPE DESCRIPTION /COMPONENTS $

For those Data objects that may have components on a lower level, the number of these or else a list of their names must appear after the slash. Alphabet definitions require no data type. Names of data objects and alphabets may have from one to six characters, the first of which must be one of the letters A through Z, and the remainder letters or numbers. Special characters (comma, blank, apostrophe, .,/,(,),=,-,+,*,$) must not be used within names.

3.3 Operation Statements.

3.3.1 The General Form of Operation Statements.

*T NAME VERB DATA TYPE(S) PRIMARY OPERANDS and AUXILIARY OPERANDS /

RESULT OPERANDS / EXIT ROUTINGS $

There are many varieties of operation statements, each having a different verb. Every operation statement must have a name, formed in accordance with the rules given in paragraph 3.2.2 for data object and alphabet names; it must also have at least one primary operand, which names the data object to which the action of the verb is to be applied, and an implicit or explicit exit routing, which tells the names or name of other statements to be executed after the present one, or in the event of certain conditions arising during execution of the present one. The TEMAC statement is, in many ways, similar to an English sentence, and can usually be "read" aloud with the addition of a few transitional words here and there. Some of the parts of the operation statement will be discussed in more detail in the following paragraphs.

3.3.2 The Data Types and Primary Operands.

Some operation statements have one primary operand, the Source Operand, whereas others have two primary operands. In the latter case, if the statement is a data manipulation statement, it has a source operand, and a target operand (the source being the place of origin of the data, and the target being the destination, or receiving location for the data); if the statement is a data testing statement, it has a source and a comparand, providing data to be compared to the data obtained from the source. A preposition separates the primary operands, for readability. There is usually a data type code, consisting of one letter (P, S, W, M, L, C for the main types of data object) attached to the verb for each primary operand. The data type on the verb

describes the kind of unit designated by the operand in the statement; it is often but not necessarily the same as the data type assigned to the data object whose name enters into the operand. There are five main kinds of operands, and two additional special-purpose operand types which will be described below.

3.3.3 Kinds of Primary Operands: One-Component, Sequence, Whole, Left-Boundary, Set, Character-Stream, Field.

The five main types of operands, used in all types of operation statements, may be illustrated with reference to an artificial "data object" consisting of a group of five men sitting in a row in an auditorium, who are being called up one at a time or in subgroups to receive awards or diplomas. Let the row as a whole be called "A", and the men, in order as they are sitting, be called Joe, Jack, Bill, Jim, and Tom. Then TEMAC operands might be written that would select certain ones alone or in combination with others from the row, as follows:

One-Component Operand ONE NAME(POINTER)	Jim, the fourth man	ONE$\wedge$A(4)
Whole Operand NAME	All of Row A One man: Joe.	A JOE
Sequence Operand SEQ N NAME(POINTER)	Three men sitting next to each other; Jack, Bill, Jim	SEQ$\wedge$3$\wedge$A(2) SEQ$\wedge$N$\wedge$A(I) N=3, I=2
Left-Boundary Operand NAME(POINTER)	All men from Bill to the end of Row A	A(3) A(I), I=3
Set Operand SET N NAME(POINTER-SET)	A selection of men, pointed to one at a time. Joe, Bill, Joe again, Tom, Bill again.	SET$\wedge$5$\wedge$A(1$\wedge$3$\wedge$1$\wedge$5$\wedge$3) SET$\wedge$N$\wedge$A(HITS) N=5, HITS=1,3,1,5,3

A pointer may be a positive, non-zero decimal or octal number (octal distinguished by an attached = sign), or the name of a cell. A pointer-set may be a string of octal or decimal numbers separated by blanks, or the name of a cell-list. Pointers and pointer-sets must always be enclosed in parentheses. The set or sequence length, "N", may be similar to a pointer. In addition to the five operand types above, there are two "special purpose" operand types, the character-stream and the field, which are used only in data testing and data moving statements. The character-stream allows only certain characters of a word, or within a cell-list to be processed. Character-streams are specified in an operand of the form "NAME START LENGTH", or ONE NAME(POINTER) START LENGTH", where START and LENGTH may be numbers or cell names. For example, in a word named "WD", containing the value "characters", the operand "WD 3 5" selects the characters "aract"; if "WD" is word 3 of the sentence "S1", the operand may be written "ONE$\wedge$S1(3)$\wedge$3$\wedge$5". If the letters "characters" are packed into a cell-list named "BUFR", which is the second of two buffers in a master-block "BUFRS", the

character stream "aract" may be obtained by writing "BUFR2 3 5" or "ONE BUFRS (2) 3 5". The character-stream based on a word is called a "word-based character-stream", while that based on a list is called a "list-based character-stream". Character-stream operands are also distinguished by the data type letter "H" in the position for the operand on the verb of the statement. The field operand is similar to a word, but the number of characters processed is determined by the length of the other operand in the statement. In other words, only as many characters are used from the "field" as are required to occupy or to match positions in the target or comparand. The field operand is distinguished only by the use of the letter "F" in the position for that operand on the verb of the statement.

3.3.4 Auxiliary Operands. Some statements may have a third or auxiliary operand, which aids in the action of the statement, for example a match or edit control string, or a number to be added or subtracted. Auxiliary operands are linked to other operands in the statement by a preposition or by a "function" word like "+" or "-".

3.3.5 Result Operands. These are the names of cells or cell-lists which store certain counts or other information resulting from the action of the verb. They are set off from the primary and auxiliary operands by a slash. Some statements have none, some one, some two, and some three result operands. The information stored in the result operands is frequently of primary aid in debugging the program, but it may also be a major product of the operation, and may be referred to by later statements as a data object. In data testing statements, the result operands contain the record of "hits" found during testing. In many data manipulation statements, the result operands store the number of components selected from the source or moved into the target.

3.3.6 Exit Routings. Following the result operands and a second slash, there may be one or more names accompanied by standard labelling words; these are the names of statements to be executed on completion of the present statement, or if certain conditions arise in its execution. Exit routings may be Unconditional Routings or Conditional Routings. Certain of the exit names and labels may be dropped, in which case control will proceed to the next statement in the program when the conditional associated with that exit arises.

3.3.6.1 Unconditional Routings. There may be either one or two exits in the unconditional routing: the Error Exit and the Normal Exit. Some statements do not require an error exit. The label for the error exit is "ERR", before the exit name, and the label for the normal exit is "THEN". If the error exit is dropped, control goes to the normal exit for both error and normal conditions; it should be remembered that, if an error is likely to occur and the "ERR" exit has been dropped, a test must be made to see if an error took place in the next statement. If both exits are dropped, control proceeds to the next statement in case of normal completion as well as an error, so that a test for possible errors is even more important. Each error condition arising in a statement is signalled by a numeric code in a cell called "DBUG", which may be interrogated by the programmer. (See Section 6.2.4)

Possible forms of unconditional routing are:

/ERR X THEN Y $	(fully specified; X and Y are statement names)
/ERR X$	(normal exit implied to next statement)
/THEN Y$	(error exit and normal both go to Y)
$	(both error and normal exits proceed to next statement)

3.3.6.2 Conditional Routings. The fully specified conditional routing includes three labels and names: the Error Exit (label "ERR"), the Positive Exit (label "YES"), and the Negative Exit (label "ELSE"). The positive exit may never be dropped, but one or both of the others may be omitted.

Possible forms of Conditional routing are:

/ERR X YES Y ELSE Z $	(fully specified)
/ERR X YES Y $	(negative exit goes to next statement)
/YES Y ELSE Z $	(error exit and negative both go to Z)
/YES Y $	(error exit and negative go to next st.)

3.3.6.3 Error Exits. An error can arise in a TEMAC statement if one of the following conditions holds:

a) A component is requested that does not exist within an object at the time it is requested, for example, ONE S2(9), when S2 has only seven words.

b) An object or component is requested as a source when it is empty, or has been emitted by an OMIT statement.

c) An attempt has been made to store too many components into a target object, or to store something into a component that does not exist in a target object.

d) Some special error condition peculiar to a given statement has arisen; these will be mentioned in the description of each statement.

4. Definition Statements.

4.1 Alphabet Definitions. An alphabet definition sets up a named TEMAC alphabet and sorting sequence. Up to ten different alphabets may be defined in any one object program, and each alphabet may have from one to nine programmer-selected character-categories (named 1, 2, ..., 9), three boundary character-categories (named W, S, and P for word-boundaries, sentence-boundaries, and paragraph boundaries), and an "unassigned" or wastebasket category named U. The order in which the characters are written in the definition establishes a sorting sequence. Characters within the alphabet may be represented either by punching them directly on the card (character-coded representation) or by punching an equal sign followed by two octal digits standing for the "BCD" value (octal-coded representation). The alphabet definition statement has the following general form: (the symbol ^ stands for a blank in all examples below)

*T (n) NAME ALF Category Descriptions $

Example: *T^AB^ALF^1^A**Z^2^0**9^W=60,/()^S^.*^P =53^$

*T - indicates a TEMAC statement; no overflow cards in this example.
AB - name of alphabet
ALF - verb of alphabet definition statement
1 A**Z - designation and members of the first programmer-selected category: letters A through Z.
2 0**9 - designation and members of second programmer's category: whole numbers 0 through 9.
W =60,/() - word boundary category; =60 is octal coded representation of "blank".
S .* - sentence boundary category;
P =53 - paragraph boundary category: octal-coded representation of $.

Blank and dollar sign, if they are to be used as members of a category, must be represented octally since the characters are used as boundary characters in the TEMAC statement. The special category assignments A**Z and 0**9 are available to describe the normal English alphabet and numbers. Any other arrangements, or any portions (e.g. A to F, 0 to 4) must be written out in full in the desired order.

4.2. Data Object Definitions.

4.2.1 The Paragraph. There are two main varieties of paragraph definitions: the general paragraph definition and the dummy paragraph definition.

4.2.1.1 The General Paragraph Definition. The statement follows the form:

*T (n) PNAME DEFP / $SNAME_1$ ... $SNAME_m$$

Where n may be from 1 to 17, and tells the number of extra cards occupied by the statement after the first; PNAME is the name of the paragraph being defined; DEF is the verb; and P after DEF is the data type code for the data object (paragraph). $SNAME_1$, $SNAME_m$ are names of component sentences in the paragraph. There must always be as many such names as there are to be components in the paragraph, and they must be in the same order as they are to appear. The paragraph definition must always be followed immediately

by the sentence definitions for each SNAME, arranged in the same order as the names. The sentence definitions may be of any kind allowable for sentences - immediate, dummy, or fielded. Different kinds of sentences may be mixed in any combination within a general paragraph. The sentence definitions are not a part of the paragraph definition statement, but must follow it in the program. Examples of the general paragraph definition and component sentence definitions may be seen after section 4.2.2 and in paragraph 4.2.2.3.

4.2.1.2 The Dummy Paragraph Definition.

*T PNAME DEFP N M $

where N is the number of component sentences and M the number of words in each sentence. These numbers set the maximum of places allowed within the structure, which represents a variable. If each sentence is to have a different maximum length, a slightly different form is used:

*T PNAME DEFP N / $M_1 \ldots M_n$ $

In order to set up a dummy paragraph with named sentences, the "general" paragraph definition form should be used, with each sentence name defined as a dummy sentence.

Examples of Dummy Paragraphs:

*T^VP^DEFP^3^7$

*T^VP1^DEFP^3/5^4^3$

4.2.2 The Sentence. Sentence definitions may be Immediate, Dummy, or Fielded. Their component words may have names, in which case the names must all appear after the slash, in order as the words are to be in the sentence. Every component word must have a name, if any word is to have one. If no names for words are needed, the number of words should appear as the last field of the statement.

4.2.2.1 The Immediate Sentence Definition.

*T (n) SNAME DEFS I / $WNAME_1 \ldots WNAME_n$

*T (n) SNAME DEFS I M $ (no word names).

"n" data cards (from 1 to 17) may follow the header, which must not overflow the first card. The words to be entered into the sentence should be punched into the data cards like running text, separated by blanks, with a blank in column 1 of each card, and no broken words at the end of a card. (In all examples, the symbol "^" will stand for "blank".)

Examples of Immediate Sentences:

*T1^S1^DEFS^I/W1^W2^W3^W4$
^HERE^IS^A^SENTENCE^$
*T1^S3^DEFS^I^5$
^A^SENTENCE^WITH^FIVE^WORDS^$

4.2.2 The Dummy Sentence Definition.

The dummy sentence definition statement may not occupy more than one card, even though it has only a "header" and no data.

*T (n) SNAME DERS D / $WNAME_1 \ldots WNAME_m$$

*T SNAME DEFS D M $

Examples of Dummy Sentences:

*T^SVAR^DEFS^D^/WV1^WV2^WV3$

*T^VS^DEFS^D^10$

4.2.2.3 The Fielded Sentence Definition.

*T (n) SNAME DEFS F / $FNAME_1 \ldots FNAME_m$$

*T (n) SNAME DEFS F M $

"n" data cards may follow the first card; these cards contain a string of field definitions, as follows:

$$BASENAME_1 \; S_1 \; L_1 \; [BASENAME_2] \; S_2 \; L_2 \ldots [BASENAME_m] \; S_m \; L_m \; \$$$

where BASENAME is the name of any cell-list, cell, sentence, or word which contains or is intended to contain characters (the base stream); S_1 is the starting character position of the first field, counting characters from left to right in the base stream, and L_1 is the length of the first field in characters. If all the fields of the sentence are intended to refer to the same base stream, only the first basename need be present, and the remaining fields may be represented only by their S and L values, but the first basename at least must be present at the beginning of the string. Any number of different base streams may be referred to, by entering a new basename for each field. Fields need not be contiguous or exhaustive within any stream - i.e., gaps can be left, and the starting positions may "jump around" in the stream, as may be seen from the examples below.

Examples of Fielded Sentences Within a General Paragraph Definition.

*T^FLDS^DEFP^/^ FS1^FS2$
*T^FS1^DEFS^F^3^$
^BUFR^3^7^^ STREM^12^5^^S2^9^6^$
*T^FS1^DEFS^F^/F1^F2^F3$
^BUFR^15^9^^6^5^^1^3$

(BUFR, STREM, and S2 are defined in other examples: see paragraphs 4.2.5.6, 4.2.5.5 and below)

Examples of a General Paragraph Definition and Sentences

```
*T^P1^DEFP/S1^S2^S3$
*T^S1^DEFS^1/W1^W2^W3^W4$
^HERE^IS^A^SENTENCE^$
*T1^S2^DEFS^1^7$
^THIS^IS^A^SENTENCE^OF^SEVEN^WORDS$
*T1^S3^DEFS^1^5$
^A^SENTENCE^WITH^FIVE^^WORDS $
```

4.2.3 The Word. Word definitions may be immediate, dummy, or fielded, like sentence definitions.

4.2.3.1 The Immediate Word Definition.

```
*T WNAME DEFW I$
(followed by a single data card only, containing any punchable
 character except blank or dollar sign within the word, blanks
 surrounding the word, and a dollar sign after the word.)
```

Example of an Immediate Word:

```
*T1^IW^DEFW^I$
^ANTIDISESTABLISHMENTARIANISM^^$
```

4.2.3.2 The Dummy Word Definition.

```
*T WNAME DEFW D $    (no data cards or overflow cards)
```

Example of a Dummy Word:

```
*T^DW^DEFW^D$
```

4.2.3.3 The Fielded Word Definition.

```
*T WNAME DEFW F BASENAME S L $  (no additional cards)
```

Example of a Fielded Word:

```
*T^FW^DEFW^F^BUFR^5^6^$
```

4.2.4 The Master-Block.

Master-blocks are defined in the same ways as paragraphs - by a general or a dummy definition.

4.2.4.1 The General Master-Block Definition.

*T (n) MNAME DEFM/ $LNAME_1$ ... $LNAME_m$$

List definitions for the LNAME's must follow immediately, and may be of any kind allowed for lists (immediate, dummy, or padded). Examples appear after section 4.2.5.

4.2.4.2 The Dummy Master-Block Definition.

*T MNAME DEFM N $ (no extra cards).

N is the number of lists in the master-block; no lengths need be given for the lists. If named dummy lists are desired, the general form of definition should be used, followed by dummy list definitions.

Example of a Dummy Master-Block:

*T$_\Lambda$VM$_\Lambda$DEFM$_\Lambda$3$

4.2.5 The Cell-List.

Cell-lists may be immediate, Padded, or Dummy. The immediate definition may have five different sub-types, depending upon the type of data to be entered into the list; these are symbolic, numeric, edit control string (ECS), match control string (MCS), and Literal. Symbolic definitions set up lists of names, one to a cell; numeric definitions set up lists of positive or negative decimal numbers (integers only), or octal values. Edit control string and match control string definitions cause a designated number of edit or match control characters to be entered into a cell-list from a single data card in the statement; these control strings are used in data manipulation and data testing statements of the program. Literal definitions cause a stream of from one to fifty-four characters from a single data card to be packed (six to a cell) into a list of from one to nine cells.

4.2.5.1 Immediate Symbolic Definition.

*T (n) LNAME DEFL I SYM / CNAME$_1$ ... CNAME$_m$ $

*T (n) LNAME DEFL I SYM M $

where n may be from 1 to 17, and refers to data cards; the CNAME's may not overflow the first card. Examples are together at the end of section 4.2.5.5.

4.2.5.2 Immediate Numeric Definition.

*T (n) LNAME DEFL I NUM / CNAME$_1$... CNAME$_m$$

*T (n) LNAME DEFL I NUM M $

Data cards may contain decimal numbers with or without minus signs, octal numbers preceeded by an equal sign, or the word =BLANK, referring to a cell full of blanks (six blanks).

4.2.5.3 Immediate Edit Control String Definition.

*T1 LNAME DEFL I ECS K $ (no names of cells)

Followed by one data card only; "K" tells the number of characters to be used from the data card after the initial blank. The characters must

be edit control characters, which are as follows: T (transmit), D (delete), E (end or stop), I (insert a single character-coded or octal-coded value to follow), and R (replace current source character by a single character-coded or octal-coded value to follow).

4.2.5.4 Immediate Match Control String Definition.

```
*T⊥ LNAME DEFL I MCS K $
```

Followed by a single card containing "K" match control characters chosen from the following: M (source and comparand characters must match), ⊥ through 9 (character must belong to the designated programmer-selected category), W, S, P, U (character must belong to the designated boundary category, or the wastebasket category in some alphabet,) * ("skip"; character position is skipped, not tested), E (end or stop), L (character must match the following octal-coded or character-coded value). The meanings of these control characters and also of the edit control characters will be explained further in the section dealing with the Locate and Compare statements and the Edit statement.

4.2.5.5 Immediate Literal Definition.

```
*T⊥ LNAME DEFL I LIT K $
```

Followed by one data card, from which "K" characters after the initial blank are chosen and packed into a cell-list of from one to nine cells. K must be less than or equal to 54, and the characters may be any punchable characters, including blank and dollar sign.

Examples of Immediate Cell-List Definitions.

```
*T1^JUMPS^DEFL^I^SYM/J⊥^J2^J3^J4$
^ST⊥^ST2^ST3^ST4$      (these are statement names)

*T1^NUMS^DEFL^I^NUM/N⊥^N2^N3^N4$
^5^2^⊥^3$

*T1 ^LIST^DEFL^I^NUM^5$
^Ø^-5^=77^=BLANK^⊥ØØ$

*T1^ECS⊥^DEFL^I^ECS^⊥3$
^TTTDIØR/R=77E

*T1 ^CMCS^DEFL^I^MCS^⊥Ø$      (Match control string for a compare statement)
^M⊥⊥2WW**ME

*T1 ^LMCS^DEFL^I^MCS^⊥5$      (Match control string for a locate statement)
^⊥⊥LAL,**WL=77SE

*T1^STREM^DEFL^I^LIT^3Ø$
^ABCDEFGZØ123456789^.,=/()'+-$*
```

4.2.5.6 The Padded Cell-List Definition.

*T LNAME DEFL P Padvalue / $CNAME_1 \ldots CNAME_m$ $

*T LNAME DEFL P Padvalue / M $

The "pad value" may be a positive or negative decimal number, an octal number preceded by an equal sign, or the value =BLANK representing a cell of blanks. As was true for the dummy sentence definition, the added definition statement may not have overflow cards, and can occupy only one card.

Examples of Padded List Definitions:

*T^VCLS^DEFL^P^Ø^/^I^J^K^TAB^STAB^TTAB^CT^SLS^SN$

*T^HITS^DEFL^P^Ø^5$

*T^BUFR^DEFL^P^=BLANK^2Ø$

4.2.5.7 The Dummy Cell-List Definition.

*T LNAME DEFL D $ (no additional cards or names)

Example of Dummy List:

*T^VL^DEFL^D $

Example of a General Master-Block Definition with its Component Lists:

```
*T^M1^DEFM/BUFR^STREM^JUMPS$
*T^BUFR^DEFL^P^=BLANK^2Ø$
*T1^STREM^DEFL^I^LIT^3Ø$
^ABCDEFGZØ123456789^.,=/()'+-$*
*T1^JUMPS^DEFL^I^SYM/J1^J2^J3^J4$
^ST1^ST2^ST3^ST4$
```

4.2.6 The Cell.

There is only one kind of cell definition -- the immediate definition; a single decimal number, with or without minus sign, an octal number preceded by an equal sign, the value BLANK, or a single name is entered into a named cell.

*T CNAME DEFC Value $

Examples of Cell Definitions.

```
*T^NC^DEFC^-5$
*T^NB^DEFC^=BLANK$
*T^NS^DEFC^BUFR$
*T^CELL^DEFC^Ø$
```

5. Operation Statements.

5.1 Structural Statements.

Structural statements do not cause data objects to be moved or altered in memory, but instead cause changes in the TEMAC system tables that describe the structure of the objects. It is important to remember that the cell is the only entity that possesses any operational reality in the 7094 computer; the TEMAC objects called paragraph, sentence, word, character-stream, master-block, and cell-list are artificial objects which are accessible through the system descriptor tables that describe them in terms of cells and character-positions within cells. The tables are formed and associated with the data object names by the Definition Statements, and their contents are filled in, altered, or moved from one table to another by the structural operations. No operation statement, even a structural statement, may create a new name or descriptor table, but all must refer to previously defined names and tables set up by the DEF statements ahead of time. The descriptor tables form a part of the object program, and are printed out in the listing so that the programmer may see them. They have names which the programmer may refer to as machine addresses, if he is familiar with some simple aspects of 7094 machine language and assembly language; more will be said about these matters in Part II.

5.1.1 The SEGMENT Statement: verb SEG.

The SEGMENT statement scans a source string of characters for those word, sentence, and paragraph boundary characters specified in an alphabet, and creates a description for the sentence or paragraph in a previously defined sentence or paragraph variable. The verb may have data type combinations LP, LS, or HS, allowing for either a cell-list containing characters or a character-stream operand as a source, and a paragraph or sentence variable as target.

*T (n) NAME SEGT_1T_2 SOURCE ALPHABET AS TARGET / [RL] RC TAB / ERR X THEN Y $

*T(n) - TEMAC statement indicator; statement may overflow onto "n" (1 to 17) cards after the first.

NAME - name of the statement.

SEGT_1T_2 - Verb SEG and data type codes for source and target; Allowable data type combinations are: LP, LS, HP, HS.

SOURCE - The operand containing the characters to be scanned.

ALPHABET - The name of a previously defined alphabet containing the desired boundary character-categories.

AS - preposition used in all structural statements having more than one operand.

TARGET - The operand naming the sentence or paragraph variable being formed.

RL - The name of a result cell-list to contain the lengths of all the sentences found in a paragraph. (SEGLP, SEGHP)

RC - The name of a cell to contain the count of sentences found (SEGLP, SEGHP), or of words found (SEGLS, SEGHS).

TAB - The name of a result cell which serves a special purpose described in paragraph 5.1.1.1 below.

Error exit- Chosen when the stream scanned has too many words or sentences to fit into the limits in the definition of the target variable, or when no words (i.e., no non-boundary characters) are found in the stream, or when the source or target operand is unobtainable.

5.1.1.1 The TAB Result Cell. This cell serves to store a special value representing the position of the next character to be scanned after scanning left off for one execution of the statement. When the source is a cell-list, the TAB cell serves another purpose as well, in that it contains a value that tells that statement where to start scanning in the cell-list for each execution. If the TAB cell is zero, scanning begins at the first (left-most) character of the cell-list. After the statement has been executed once, TAB will contain a value placed there after that execution, and the next scan will start where the last one left off (with the next unscanned character). The programmer need not concern himself with the format of the value in TAB unless he wishes to; it will be described in Part II.

Examples of SEG Statements:

Given, a stream read into cell-list "BUFR", as follows:

THIS^^SENTENCE/^HAS,,(BEEN)READ^/,/IN*^ANOTHER^SENTENCE..$

TAB, the TAB cell, starts with contents zero for each statement.

*T^ST1^SEGLP^BUFR^AB^AS^VP/SLX^SN^TAB/ERR^STX^THEN^STY$

*T^ST2^SEGLS^BUFR^AB^AS^VS^/^SN^TAB/THEN^STY$

*T^ST3^SEGHS^BUFR^38^20^AB^AS^VS^/^SN^TAB$

(In this and future examples, data object names are as defined in examples for Definition Statements in Section 4).

*T^SLX^DEFL^P^0^5$ (a cell-list for lengths of sentences in VP.)

5.1.2 The SELECT Statement: verb SEL.

This statement forms a paragraph, sentence, word, or master-block variable by selecting one component or a specified string of components from a single previously existing data object. The components may be chosen from the source by a left-boundary, whole, sequence, set, or one-component operand.

*T (n) NAME SELT$_1$T$_2$ SOURCE AS TARGET / RC / ERR X THEN Y $

SELT$_1$T$_2$ - verb SEL and data types of source and target; possible data type combinations are: PP, SS, WW, MM, LL.

SOURCE - the source of the components being chosen to form the target variable.

TARGET - a single paragraph or master-block name, or else a sentence, word, or list referred to by name or as one component of a larger object.

RC - the name of a result cell which stores the number of components in the variable object after its formation is complete.

Error Exit - chosen when the source or target is unobtainable, or when the source has too many components to fit into the target.

Examples of SEL Statements:

*T$\wedge$ST4$\wedge$SELPP$\wedge$P1$\wedge$AS$\wedge$VP$\wedge$/CT/ERR$\wedge$STX$\wedge$THEN$\wedge$STY$

*T$\wedge$ST5$\wedge$SELSS$\wedge$SEQ$\wedge$3$\wedge$S2(5)$\wedge$AS$\wedge$ONE$\wedge$VP1(2)$\wedge$/$\wedge$CT/THEN$\wedge$STY$

*T$\wedge$ST6$\wedge$SELWW$\wedge$ONE$\wedge$S1(3)$\wedge$AS$\wedge$DW/CT$

*T$\wedge$ST7$\wedge$SELMM$\wedge$M1(2)$\wedge$AS$\wedge$VM/CT$

*T$\wedge$ST8$\wedge$SELLL$\wedge$BUFR$\wedge$AS$\wedge$ONE$\wedge$VM(2)$\wedge$/CT/ERR$\wedge$STX$

5.1.3 The COLLECT Statement: verb COL.

The COLLECT Statement forms a paragraph, master-block, or sentence by stringing together scattered components or objects previously defined under various names. Its source operand consists of a string of names or one-component operands formed around names, and its target operand is a single name or one component of a named object.

*T (n) NAME COLT_1T_2 SOURCE / SOURCE / .../SOURCE AS TARGET/

RC/ERR X THEN Y $

$COLT_1T_2$ – COL verb and data types; allowable combinations are SP, WS, LM.

RC – name of a result cell storing the number of components in the completed target variable.

Error Exit – chosen if a source or target is unobtainable or if the source components overflow the target.

Examples of COL Statements:

*T ST9 COLSP S1 + S1 + ONE P1(3) AS VP/CT/THEN STY$

*T ST10 COLWS ONE S2(1) + W2 + W AS ONE VP(1) / CT$

*T ST11 COLLM NUMS + ONE M1(2) + NUMS AS VM/CT$

5.1.4 The OMIT and INCLUDE Statements: Verbs OMT, INC.

These statements have only one operand, which must be textual. The OMIT statement causes objects or components to be skipped over if they are referred to by a later statement as a part of a source operand or comparand. If the entire object whose name is used in an operand has been omitted, a statement containing that operand as a source or comparand will not be executed, but an error exit will be taken. The INCLUDE statement causes previously omitted objects or components to be made available again as they were before omission. No change is made in the data, but the descriptor table entries are flagged for omission by OMIT and the flags removed by INCLUDE. These operations have no effect when applied to objects which have already been omitted or included, and are equivalent in that case to "no operation".

*T(n) NAME $\left\{\begin{matrix}\text{OMT}\\\text{INC}\end{matrix}\right\}T_1$ SOURCE / ERR X THEN Y $

T_1 – one data type code only, which may be P, S, W

SOURCE – a whole, single-component, set, sequence, or left-boundary operand specifying the object or components to be omitted or included.

Error Exit – chosen if the source is unobtainable

Examples of OMIT and INCLUDE Statements:

*T ST12 OMTP VP/THEN STX$

*T ST13 OMTS SET 3 S2(1 2 6)$

*T ST14 OMTW ONE S3(1)/ERR STX$

5.1.5 The EMPTY Statement: Verb MTY.

This statement, like OMIT and INCLUDE, has only one operand. It causes an existing paragraph, sentence, or master-block to be marked as empty of components, so that it may be refilled.

*T NAME MTYT$_{\perp}$ SOURCE / ERR X THEN Y $

T$_{\perp}$ - single data type code, may be P, S, or M (words, lists, and cells are not "emptied" in this way.)

SOURCE - a paragraph or master-block name, a sentence name, but not one component of a paragraph.

Error Exit - chosen if the source is unobtainable or omitted.

Examples of MTY Statements:

*T$_{\wedge}$ST$\perp$5$_{\wedge}$MTYP$_{\wedge}$VP$_{\wedge}$/$_{\wedge}$ERR$_{\wedge}$STX$_{\wedge}$THEN$_{\wedge}$STY$

*T$_{\wedge}$ST$\perp$6$_{\wedge}$MTYS$_{\wedge}$S1$_{\wedge}$/$_{\wedge}$ERR$_{\wedge}$STX$

5.1.6 The REFERENCE or REFER Statement: Verb REF.

The REF statement enters a value into a single target cell which describes the location of the beginning (left boundary) or end (right boundary) of an object, or its length in components. The beginning or end of a textual object means the position of the first character of its first word, or of the last character of its last word. The beginning or end of a cellular object means the address of its first or last cell. Lengths are in terms of components on the next lower level (sentences in a paragraph, words in a sentence, characters in a word, lists in a master-block, cells in a list).

*T (n) NAME REFT$_{\perp}$ TYPE SOURCE AS TARGET / ERR X THEN Y $

T$_{\perp}$ - data type code for source (target is always a cell). may be P, S, W, M, L (not C, since a cell is an indivisible unit having no beginning, end, or length).

TYPE - specifies the type of information desired; may be "BGN" (beginning), "END", or "LTH" (length).

SOURCE-a single object name, or one component of a larger object about which information is desired.

TARGET-a cell name, or one component of a list.

Error Exit - chosen if the source is unobtainable.

Examples of REF Statements:

```
*T/ST17/REFP/BGN/P1/AS/TAB/THEN/STY$
*T/ST18/REFS/END/S2/AS/TAB/ERR/STX$
*T/ST19/REFM/BGN/M1/AS/CELL$
*T/ST20/REFW/LTH/ONE/S1(4)/AS/CELL$
*T/ST21/REFL/LTH/NUMS/AS/CT/THEN/STY$
```

5.2 Data Moving Statements: Textual.

The textual data moving statements cause data to be copied, or moved in altered form from a source to a target area in storage. If the target object is a "fielded" sentence or word, or can be referred to accurately in terms of a previous segmentation as a word or sentence variable, it may immediately be treated as a TEMAC word or sentence after material has been moved into it. In general, however, the target object created by an EXTRACT, EDIT, or CONVERT statement has the status of a character-stream or a cell-list, and must be segmented before it can be treated as a word, sentence, or paragraph. Frequently, the extracting, editing, or converting are performed for the purpose of formatting material for output, and segmentation is not necessary.

5.2.1 The EXTRACT Statement: Verb EXT.

The EXT statement copies a source word, character-stream, or sentence into a target cell-list, word, character-stream, or sentence. A specified boundary-character may be inserted before, after, or both before and after one word or character-stream, or each word of a sentence in the target; (only the characters of the word or stream itself are copied, and not the boundary-characters of the original sentence, which are considered to be outside of the words). Alternatively, leading zeroes or blanks may be inserted to fill out a field. Copying may proceed from left to right ("left-justified" extracting), or from right to left ("right-justified" extracting.) Right-justified extracting does not cause the source to be inverted or moved "backwards," but rather causes it to appear positioned to the right within the target area or field, with overflow characters chopped off on the left. When leading character insertion is requested with right-justified extracting, the leading zeroes or blanks are placed to the right of the extracted material; the result is no longer in any sense "right-justified," but might be useful for some special purpose. If the first data type code on the verb is "F", the source is a "field operand" as described in Section 3.3.3, and only as many characters as will fill the target operand are moved from the source; if the target is full, and there are still characters in the source to be moved, copying stops but no error exit results as would usually be the case. Copying that has been interrupted by an error exit because of a target overflow in the usual case (where neither data type code is "F") may be resumed where it left off in the source by using the "continuous" form of the statement and re-entering the statement after the full target area has been processed and cleared.

*T (n) NAME EXTT$_1$T$_2$ SOURCE [WB x] [C] RIGHT TO TARGET/ ST TT/

ERR X THEN Y$

T$_1$T$_2$ - data type codes for source and target; possible combinations are WL, HL, HW, HH, WW, WH, FW, FH, SL, SH, SS.

SOURCE - operand containing the textual data to be moved.

WB x - word-boundary or leading character insertion option; "WB" may be A(after), B(before), BA(Before and after), LZ(leading zeroes), or LB (leading blanks). "x" is present only for B, A, and BA, and may be an octal-coded or character-coded value for a boundary character to be inserted.

C - If the letter "C" is present, the "continuous" form of the statement is intended.

RIGHT - If the word "RIGHT" is present, extracting is right-justified, otherwise it is left-justified.

TO - preposition used with most data moving statements. (INTO is used for the ENTER statement for reasons of readability)

TARGET - operand into which the source characters are to be moved.

ST - Name of a result cell called the "source tab" cell, which records the position in the source where copying left off on completion of the operation.

TT - Name of a result cell called the "target tab" cell, which performs a service for the target similar to that of the source tab cell for the source. If the target overflows, this cell is set negative. When the target is a cell-list, the tab cell, if non-zero, tells where copying into the target list is to begin, as well as recording where it leaves off.

Error Exit-chosen if the target overflows, or if source or target is unobtainable.

Examples of EXTRACT Statements:

*T∧ST22∧EXTWW∧ONE∧S2(7)∧TO∧F2/STAB∧TTAB/THEN∧STY$

*T∧ST23∧EXTFW∧W4∧B∧,∧TO∧FW/STAB∧TTAB$ (TTAB starts at beginning of BUFR)

*T∧ST24∧EXTHH∧IW∧1∧4∧BA∧=6Ø∧RIGHT∧TO∧BUFR∧5∧1Ø/STAB∧TTAB/ERR∧SEX$

*T∧ST25∧EXTSL∧S2(4)∧A∧=6Ø∧TO∧BUFR/STAB∧TTAB$

*T∧ST26∧EXTWH∧IW∧C∧TO∧BUFR∧1∧18/STAB∧TTAB/ERR∧STX∧THEN∧STY$

*T∧ST26A∧EXTSS∧SEQ∧3∧S2(1)∧LB∧TO∧FS2/STAB∧TTAB$

*T∧ST26B∧EXTHW∧STREM∧9∧5∧LZ∧TO∧ONE∧FS2(1)∧/STAB∧TTAB$

5.2.2 The EDIT Statement: Verb EDT.

The EDIT statement alters the characters of a source word or character-stream as they are moved into a target cell-list, word, or character-stream. Editing is carried out in accordance with an auxiliary operand called the edit control string, or "ECS", which is a cell-list predefined by a "DEFL 1 ECS" definition as described in Section 4.2.5, and contains edit control characters. (These are "T" - transmit source character unchanged; "D" - delete source character; "R" - replace source character by character to follow in ECS; "I" - insert character to follow in ECS into target before next source character; "E" - stop). In other ways, the EDIT statement is like the EXTRACT statement for single words or character streams, and provides similar options (right or left-justified editing, continuous editing, insertion of boundary characters). If the ECS ends without the stop character "E" before all of the source has been moved, the remaining source characters are transmitted unchanged as in EXTRACT. The EDT statement does not allow leading character insertion.

*T (n) NAME EDTT_1T_2 SOURCE [WB x] [C] [RIGHT] TO TARGET BY ECS/

ST TT/ ERR X THEN Y $

T_1T_2 - data type codes for source and target; allowable combinations are WL, HL, HW, HH, WW, WH, FW, FH.

BY - preposition that links target operand and ECS

ECS - the name of a cell-list containing edit control characters, or one component of a master-block of control strings.

Error Exit - chosen for the usual reasons, or because the ECS contained a character that was not an edit control character.

Examples of EDIT Statements:

```
*T^ST27^EDTWL^W4^A^=60^TO^BUFR^BY^ECS↓/STAB^TTAB/THEN^STY$
*T^ST28^EDTHH^STREM^↓^8^RIGHT^TO^BUFR^↓^8^BY^ECS↓/STAB^TTAB$
*T^ECS↓^DEFL^I^ECS^↓3$
^TTTDIØR/R=77E
```

5.2.3 The Textual Conversion Statement: Verb CVT.

This statement converts a single unsigned cellular value or list of such values to decimal form, and represents it in a stream of BCD numeric characters. The characters are then extracted to a word, character-stream, or sentence target. A boundary character may be inserted after each converted value as for EXT and EDT. The continuous option is similarly available in case the extracted numbers overflow the target area. There is no right-justified conversion, however, but instead a "leading blank or zero" option, which allows blanks or zeroes to be inserted to fill out the target before the number. Either word-boundary insertion or leading blank or zero may be used, but not both. If the source is negative, it will be converted as positive, the sign being ignored.

*T (n) NAME CVTT$_{1}$T$_{2}$ SOURCE [WB x] [C] TO TARGET/ RC TT /

ERR X THEN Y $

T_1T_2 - data type codes for source and target; allowable combinations are CW, CH, CL, LS, LH, LL.

SOURCE - always a cellular object, a cell or cell-list; may be any type of operand involving a cell-list name or cell name.

TARGET - a word, cell-list or character-stream for a single number; a sentence, cell-list, or character-stream for a list of numbers.

RC - the name of a result cell which stores the count of numbers converted from cells and moved to the target.

TT - the name of a target tab cell like that in EXT and EDT.

Error Exit - chosen if source or target unobtainable, or if target overflows.

Examples of CVT Statements:

*T^ST29^CVTCW^ONE^LIST(N1)^LZ^TO^F2/CT^TTAB/ERR^STX^THEN^STY$

*T^ST3Ø^CVTLH^NUMS^A^,^TO^BUFR^1^12/CT^TTAB/THEN^STY$

*T^ST31^CVTLS^SEQ^N4^NUMS(2)^LB^TO^FS2/CT^TTAB/THEN^STY$

5.3 Data Moving Statements: Cellular.

Cellular data moving statements move contents of cells, cell-lists, or master-blocks. No "editing" is needed for cellular objects, as this operation is usually understood, but they may be altered by adding or subtracting, multiplying or dividing with some value or else by replacing their contents by other cellular objects.

5.3.1 The MOVE Statement: Verb MOVE.

The contents of a cell-list are moved to the cells of another cell-list, or the contents of a master-block to the cells of the lists in another master-block. The value in each cell may be incremented or decremented as it is moved by a single auxiliary value, or each cell may be copied unchanged.

*T (n) MOVET$_{\perp}$T$_2$ SOURCE [± AUXILIARY] TO TARGET / RC / ERR X THEN Y $

T$_{\perp}$T$_2$ - data type codes for source and target; allowable combinations LL, MM.

SOURCE - a whole list or master-block, or a left-boundary, sequence, set, or one-component operand.

± - a plus or minus that must be surrounded by blanks, and not attached to the following word; it must precede the auxiliary if one is present.

AUXILIARY-a cell name, one component of a cell-list, or a "literal" number (i.e., a decimal number or an octal number with an equal sign attached to it).

TARGET - similar to the source; source and target must describe objects having the same number of components, so that there is a source cell for every target cell.

RC - a result cell to store the number of cells or lists moved into the target list or master-block.

Error Exit - chosen if source or target is unobtainable, or if the target overflows.

Examples of MOVE Statements:

*T^ST32^MOVEMM^SEQ^2^ML(1)^TO^VM/CT/ERR^STX^THEN^STY$

*T^ST33^MOVELL^SET^3^NUMS(3^2^4)^-^50^TO^LIST(N4)/CT/THEN^STY$

*T^ST34^MOVELL^ONE^ML(3)^TO^VL/CT$

5.3.2 The REPEAT Statement: Verb REPT.

A single cell, a signed decimal number, or an octal number is copied repeatedly into specified cells in a list. A single list, one component of a master-block, or a literal string of octal or signed decimal numbers is copied repeatedly into specified lists of a master-block. No change is made in the source values as they are moved.

*T (n) REPTT$_{\perp}$T$_2$ SOURCE TO TARGET / RC / ERR X THEN Y $

T$_{\perp}$T$_2$ - data type codes for source and target; may be CL, LM.

SOURCE - the single value or list of values to be copied.

TARGET - the name of a whole list or master-block, or a one-component, sequence, set, or left-boundary operand referring to a list or master-block name.

RC - result cell to store the total number of repetitions of the source value or list into the target.

Error Exit - chosen if source or target is unobtainable, or if the target overflows.

Examples of REPEAT Statements:

*T^ST35^REPTLM^-2^5^=777^Ø^TO^MŁ(2)/CT/THEN^STY$

*T^ST36^REPTLM^SET^3^NUMS(2^Ł^3)^TO^SEQ^2^MŁ(2)/CT$

*T^ST37^REPTCL^=BLANK^TO^BUFR/CT$

*T^ST38^REPTCL^Ø^TO^HITS/CT$

*T^ST39^REPTCL^N2^TO^SET^3^LIST(Ł^3^5)/CT/THEN^STY$

5.3.3 The Cellular Conversion Statement: Verb CVN.

A textual word or character-stream containing numeric characters is converted to binary and stored in a cell; a sentence made up of words containing numeric characters is converted and its component values stored in the cells of a cell-list. Any number read in from tape as a stream of BCD numeric characters must be converted in this way before it can be referred to in cellular data moving or data testing statements, or used as a pointer in an operand. It should be noted that number words using alphabetic characters, such as "one hundred", "five", are not what is meant by a textual word containing numeric characters, but rather "ŁØØ", "5". If the source word or character-stream is longer than eleven characters, or if it contains any non-numeric characters, the error exit from the statement is taken.

*T (n) CVNT$_Ł$T$_2$ SOURCE TO TARGET / RC ST / ERR X THEN Y $

T$_Ł$T$_2$ - data type codes for source and target; allowable combinations are WC, HC, SL.

SOURCE - a word or character-stream, one component of a paragraph, a sentence name, or any operand involving a sentence name, containing the characters to be converted.

TARGET - a cell name, a list name, or any operand involving a list name; one component of a master-block.

RC - result cell to store the number of values converted.

ST - a source tab cell, similar to that for EDIT and EXTRACT.

Error Exit - chosen for the usual reasons and also if the source contained more than 11 characters or one of its characters was non-numeric.

Examples of CVN Statements:

Given, an immediate sentence as follows:

*T1^NSENT^DEFS^I/NS1^NS2^NS3^NS4$
^15Ø^2Ø^64^2ØØ^ $
*T^ST4Ø^CVNHC^STREM^1Ø^4^TO^CELL/CT^STAB/THEN^STY$
*T^ST41^CVNSL^NSENT(2)^TO^NUMS/CT^STAB$
*T^ST42^CVNWC^NS4^TO^CELL/CT^STAB$

5.3.4 The ENTER Statement: Verb ENTER.

A string of different values specified by cell names, single components of cell-lists, octal numbers or signed decimal numbers may be entered, each into a corresponding target cell name or single list component. The source and target operands of the statement may both be compound, consisting of a string of names or one-component operands or numbers separated by slashes. There is no data type code, since ENTER always refers to single cells. No error exit is needed, because single cells, unlike other TEMAC objects, always have machine addresses and are thus always considered to be "obtainable." There must always be an equal number of source and target elements. The value "," (comma) occurring as a source means that the material obtained from the previous source value should be inserted again, into a different target name. Similarly, the use of a comma between parentheses as a pointer implies that the same pointer value is used as for the previous operand. (Needless to say, a comma cannot be used in either of these ways in the first source designation of the statement.) These features provide a saving in machine instructions in the compiled MAP coding.

*T (n) NAME ENTER SOURCE [/SOURCE/ ... / SOURCE] INTO TARGET [/TARGET/ .../ TARGET] /THEN Y$

Examples of ENTER Statements:

*T^ST43^ENTER^Ø/,/,^INTO^I/J/K/THEN^STY$
*T^ST44^ENTER^1/Ø/-5^INTO^J/ONE^NUMS(3)/CELL$

*T^ST45A^ENTER^Ø/,/,/1/,/,^INTO^A/B/C/D/E/F$
*T^ST45B^ENTER^ONE^NUMS(I)/ONE LIST(,)^INTO^ONE^HITS(,)/ONE^VCLS(,)$

5.3.5 The LOOKUP Statement: Verb LKP.

A single cell or one list-component is "looked up" (i.e., used as a pointer or position number) on a cell-list; the value in the cell thereby selected is similarly looked up on a second cell-list, etc. The final result, read out of the last cell looked up, is stored in the third operand, a single cell or one component of a list. A single cell name, list component, decimal or octal number may be added or subtracted from the source to produce the value that is looked up. The list names on which the lookups are made are arranged in a string separated by slashes, like the operands of the ENTER statement, and form a compound auxiliary operand in addition to the source, the added or subtracted number (if any), and the target. No data type codes are needed on the verb, since the statement has only cells as its primary operands (source and target). The auxiliary objects must be list names and not components of a master-block. No error exit is required.

*T (n) LKP SOURCE ON AUXILIARY [/AUXILIARY/ ... /AUXILIARY] TO

TARGET / THEN X $

Examples of LKP Statements:

Given a master-block as follows:

*T∧M2∧DEFM/LA∧LB∧LC$
*T1∧LA∧DEFL∧I∧NUM∧4$
∧2∧1∧4∧3∧$
*T1∧LB∧DEFL∧I∧NUM∧4$
∧1∧3∧4∧2∧$
*T1∧LC∧DEFL∧I∧NUM∧4$
∧4∧2∧3∧1∧$
*T∧ST46∧LKP∧ONE∧LB(4)∧ON∧LA/LB/LC∧TO∧CELL/THEN∧STY$
*T∧ST47∧LKP∧N4∧ON∧LC/LB∧TO∧ONE∧VCLS∧(3)$

5.4 Data Testing Statements: Textual.

Textual data testing statements investigate the similarity between a source operand and a comparand, according to any one of eight different match conditions, as follows: identity match (ID), or character-by-character comparison requiring that source and comparand have the same characters; tolerance match (IT THR), a match requiring that source and comparand have the same characters, but allowing a number of mismatches not to exceed that specified as a cell name or number at "THR"; category match (EC ALF), accepting as matches either identical characters or characters belonging to the same category in the TEMAC alphabet named at "ALF"; length match (EL), accepting only words of the same length, without regard to the characters they contain; sort-order tests, of which there are four, referring to the sorting sequence of a named TEMAC alphabet at "ALF" (AFT ALF - source must be after comparand; BEF AlF - source must be before comparand; AFQ ALF - source must be after or equal to comparand; BFQ ALF - source must be before or equal to comparand). If the special "skip" character "*" appears in either source or comparand, the corresponding position is skipped, or accepted unconditionally.

5.4.1 The COMPARE Statement: Verb COM.

The COM statement compares two words or character-streams according to any one of the eight match conditions described above. If the two objects - the source and comparand - are accepted as a matching pair, the "YES" exit is taken from the statement, otherwise the "ELSE" exit. If either source or comparand has an apostrophe at some position, the other operand will be considered to match only if it has a number (0 through nine) at the corresponding position. This provision allows numerals to be matched as if they were a character-category, without the need for a match control string or the "EC ALF" condition. The result operands store various kinds of information about the compared objects. This information is primarily useful in case of a non-match. When the match condition is ID, IT THR, or EC ALF, the result cell stores the total number of non-matching positions, and the result list stores the character position-numbers where the discrepancies occurred. When the condition is EL, the result cell stores zero, or the signed difference between the lengths of the objects (source length minus comparand length), and the result list is not required. When the condition is a sort-order test, the result cell stores the position of the character-comparison that was decisive in assigning a relative sort order to the operands, and the result list is not used. If one data type code is "F", that operand is a "field operand", and only as many characters are used from it as are needed to pair with characters in the other operand. If neither type code is "F", the operands must have the same length to be accepted as a match, but information is recorded about non-matching positions up to the end of the shortest operand in any case. In the EC ALF condition, an auxiliary operand is required, in the form of a match control string, predefined by a "DEFL I MCS" definition, and containing match control characters. Each match control character applies to a pair of characters in the primary operands. Match control characters are as follows" "M" - source and comparand characters must be the same; "1 through 9" - both must belong to the specified character-category; "W, S, P, U" - both must belong to the specified boundary character-category, or to the wastebasket category; "*" - character position is skipped; "E" - matching is terminated, regardless of other conditions. If the MCS ends without an "E" character, before the source or comparand is done, matching continues as if the condition were ID.

*T (n) NAME COMT_1T_2 SOURCE CONDITION IN COMPARAND [BY MCS] / [RL] RC /

ERR X YES Y ELSE Z $

T_1T_2 - data type codes for source and comparand; allowable combinations are WW, WF, FW, HH, FH, HF, HW, WH.

CONDITION-match condition; may be ID, IT THR, EC ALF, EL, AFT ALF, BEF ALF, AFQ ALF, BFQ ALF, where ALF is a predefined alphabet name, and THR is a number or cell name.

IN - preposition used for all data testing statements.

[BY - preposition used to link the comparand and the MCS.

MCS] - match control string; a cell-list name or one component of a master-block.

Error Exit - chosen if source or comparand is unobtainable, or the result list is too short to hold the information being stored in it.

Examples of COM Statements:

Given a sentence:

```
*T⊥^SA^DEFS^!^5$
^B^BBC^BBB^BBCDE^BBCD^$
```

and a word:

```
*T⊥^WORD^DEFW^!^$
^BBCD^$
```

```
*T^ST48^COMWW^WORD^ID^IN^ONE^SA(J)/HITS^CT/YES^STY^ELSE^STZ$  (let J=⊥)
```

```
*T^ST49^COMFW^WORD^IT^⊥^IN^ONE^SA(J)^/HITS^CT/YES^STY^ELSE^STZ$
```

Given a sentence:

```
*T⊥^ENDS^DEFS^!^3$
^IONALLY^ALLY^LY^$
```

and a word:

```
*T⊥^WORD2^DEFW^!$
^INTENTIONALLY^$
```

Note: There is no right-justified match option for COM; if a right-to-left match of two single words is desired, a LOC statement should be used, having the following form:

```
*T^NAME^LOCWS^WORD^ID^RIGHT^IN^SEQ^1^SA(J)/HIT/YES^STX$
```

5.4.2 The LOCATE Statement: Verb LOC.

The LOCATE Statement compares a single source word or character-stream successively to each word of a comparand sentence, or each offset within a comparand character-stream. If the "ALL" option is requested, all hits are found in the comparand; the result list stores the position numbers of words, or the character positions of offsets in the comparand that matched the source, and the result cell then stores the number of hits, or zero if there were none. If the word "ALL" is not present in the statement, only the first (left-most) hit is found, and the position number is stored in the result cell; the result list is not required. The match conditions are the same as for COM. If at least one hit is found in the comparand sentence or stream, the "YES" exit is taken, otherwise the "ELSE" exit. A right-justified matching option is allowed with LOCATE, which provides for matching of the end of a source word against the ends of the comparand words. For the EC ALF condition, one of the primary operands must be a match control string; it is not, as in COMPARE, a skip operand. If the MCS is the source, the first data type code on the verb must be "L", and if a string of MCS patterns in a master-block is the comparand, the second data type code must be "M". Control characters state a requirement on the corresponding character-position in the other operand. The control characters are the same as for COM, with one exception: instead of "M", the character "L" is used, followed by a single octal-coded or character-coded value which must be matched by the corresponding character in the other operand. Every MCS pattern for a LOC statement must end in the stop character "E", and matching always continues in the EC ALF condition until the stop character is reached in the MCS, regardless of the length of the other operand. When endings of words are being compared by means of the EC ALF condition, and one operand contains match control characters, the control string is still read from left to right, and must be set up with that in mind. There may be an "F" for either source or comparand on the verb, meaning that it is a field operand, and resulting in the same modifications in the operation as were described for COM; it should be noted that, when the source is the field operand, a different number of characters will be matched for each comparand word. If there is no "F" on the verb for either operand, matching will not take place for any word or stream-pair that does not have the same length, but they will be considered as non-matching.

*T (n) NAME LOCT_1T_2 [ALL] SOURCE CONDITION [RIGHT] IN COMPARAND /

[RL] RC / ERR X YES Y ELSE Z $

T_1T_2 - data type codes for source and comparand: allowable combinations WS, HS, HH, WH, WF, HF, FS, LS, LH, WM, HM.

[ALL] - If this word is not present, matching will stop as soon as the first hit has been found. If "ALL" is present, the entire comparand string will be scanned and all hits found.

SOURCE - a single word, character-stream, or MCS.

RIGHT - if this word is present, right-justified matching is called for (endings of words are matched).

TARGET - a string of words in a sentence, all offsets within a character-stream, or a master-block of MCS patterns.

RL - name of a result list; not needed in statements where the word "ALL" was absent, since only one hit will be found, which is stored in the result cell.

Error Exit - chosen for the same reasons as in COM.

Examples of LOC Statements:

```
*T^ST51^LOCFS^ALL^WORD2^ID^RIGHT^IN^ENDS/HITS^CT/YES^STY^ELSE^STZ$
```

```
*T^ST52^LOCWS^ALL^WORD^ID^IN^SA/HITS^CT/YES^STY$
```

(these two statements use the definitions set up for COM)

Given an immediate sentence:

```
*T⊥^NMW^DEFS^I^2$
^ABC⊥50^89ZGQ^M^$
```

and a master-block of MCS patterns:

```
*T^PATRN^DEFM^PAT⊥^PAT2$
*T⊥^PAT⊥^DEFL^I^MCS^8$
^222LZ⊥WE
*T2^PAT2^DEFL^I^MCS^8$
^22LZ⊥⊥WE
```

```
*T^ST53^LOCWM^ ^ONE^NMW^(2)^EC^AB^RIGHT^IN^PATRN/CELL/YES^STY$
```

```
*T^ST54^LOCWM^ ^ONE^NMW^(2)^EC^AB^IN^PATRN/CELL/YES^STY$
```

(numbers may also be found by using the special control character "apostrophe" in one operand as for COM, without the need for a match control string).

5.5 Data Testing Statements: Cellular - The FIND Statement, Verb FIND.

There is only one cellular data testing verb, the FIND verb. The FIND statement allows for testing each of a list of comparand values for certain conditions or certain relationships to a given source value. If the word "ALL" does not appear in the statement, only one hit is looked for in the comparand, and only a result cell is needed to store the position of that hit. If ALL is present, all hits are found, and a result list is required to store their position numbers; the result cell then stores the number of hits. The source and comparand values are signed numbers, and all tests and comparisons take the sign into account. Due to a peculiarity of the computer, if a negative number is subtracted from a positive number of equal magnitude, the result is "plus zero", but if a positive number is subtracted from an equal negative number, the result is "minus zero". It should be noted also that the cellular value=BLANK, (corresponding to an octal number "6ø6ø6ø6ø6ø6ø") is negative in the computer. The FIND statement has, at present, only the data type combination "CL" on its verb; "LM" may later be added if a need is felt for it.

5.5.1 Finding the Largest or Smallest Cellular Value.

This form of the FIND statement has only a comparand, and produces one position number only, stored in a result cell.

*T (n) NAME FINDCL CONDITION IN COMPARAND /RC/ ERR X THEN Y $

CONDITION - may be "GST" (find the greatest value), or "LST" (find the smallest value)

COMPARAND - one component of a master-block, or any operand containing a List name.

(examples are at the end of this section.)

5.5.2 Finding Cellular Values of a Given Type.

In this form of the statement, there is only a comparand, and the exit routing is conditional, with a "YES" and an "ELSE" exit.

*T (n) NAME FINDCL [ALL] TEST CONDITION IN COMPARAND /[RL] RC /

ERR X YES Y ELSE Z $

TEST CONDITION - may be "ZRO" (zero value), "NZE" (non-zero value), "MNS" (negative value), "PLS" (positive value), "RNG L U" (value greater than or equal to a number or cell name at L and less than or equal to a number or cell name at U).

RL - required if the word "ALL" is present in the statement.

Error exit - chosen if a comparand element was unobtainable, or because the result list overflowed.

5.5.3 Finding Cellular Values With a Given Relation to Another.

In this form, there is a source as well as a comparand, and a conditional exit routing.

*T (n) NAME FINDCL [ALL] COMPARISON CONDITION SOURCE IN COMPARAND / [RL] RC / ERR X YES Y ELSE Z $

COMPARISON CONDITION - "EQU" (source equal to comparand), "NEQ" (source unequal to comparand), "GRT (source greater than comparand), "GRE" (source greater than or equal to comparand), "LES" (source less than comparand), "LSE" (source less than or equal to comparand).

SOURCE - a cell name, one component of a list, a signed decimal number, or an octal number preceded by an equal sign.

Examples of FIND Statements:

*T^ST55^FINDCL^GST^IN^NUMS^/^CELL^$

*T^ST56^FINDCL^ALL^ZRO^IN^LIST^/^HITS^CT^/^YES^STY^ELSE^STZ^$

*T^ST57^FINDCL^ALL^RNG^4^3^IN^NUMS/HITS^CT/YES^STY$

*T^ST58^FINDCL^ALL^LES^N4^IN^NUMS/HITS^CT/YES^STY$

*T^ST59^FINDCL^ALL^NEQ^=77^IN^LIST/HITS^CT/YES^STY$

5.6 Reading and Writing Statements.

The TEMAC programmer may have several input and output tapes, or "files" in his program. The number of these is limited by the unfortunately large requirements of the IBSYS system, but there are ways in which extra tapes may be borrowed from the system if they are needed. The usual text-processing program will need at least a major text input file and a major output file, both of which will probably be "multi-reel" files (i.e., allowed to run over onto more than one physical tape reel). The output file may be intended for printing or for future machine processing on the 7094 or some other machine. There may be several output files, some for printing and others for machine processing. There might also be an additional input file, to provide parameters or control information, or a large dictionary. Every TEMAC program has a special file called the "system run log," which contains debugging information produced by the debugging statements and material the programmer chooses to write on it for the purpose of recording or explaining occurrences during the run. Files are named and defined by means of "file cards", which are required for any 7094 program using IOCS, and are not peculiar to TEMAC programs. The file name in the file card is used in all TEMAC reading and writing statements that refer to the file.

Files must be "opened" at least once, before they may be read or written, and they may also require to be "closed" (these actions form a part of the IOCS system). The RUNLOG file must be opened, but should never be closed by the programmer; all other files must be opened, and may be closed. The IBJOB system closes them all if they have not been closed by the programmer when the job is terminated. All non-normal exits from reading and writing statements must be present; only the normal or "THEN" exit may be dropped. File cards are discussed in Section 6 and Part II.

5.6.1 The READ Statement: Verb READ.

A single "logical record" is read from a named source file into one or more target areas of memory. The length of the material read from the file is equal to the sum of the cell counts of material read into different memory areas. If this total count is less than the block size for the file, specified in its file card, the remaining, unread cells in the block will be skipped, and subsequent reading will start from the beginning of the next physical block. If the count is greater than the block size for the file, all cells of each block will be read in a continuous stream regardless of physical boundaries on the file, except for any cells left over in the last block read. These left-over cells will be skipped, and the next "READ" will start at the beginning of the following block. The target names must be cell names or list names; they cannot be single components of master-blocks ("ONE M(I)"). Each target name must be followed by two positive decimal numbers: a starting position, and a length in cells. The starting position tells which cell, relative to the target name, will contain the first cell-full of material read in, and the length tells how many cells are to be read into the area. For example, "BUFR 1 12" means that 12 cells are to be read into cells "BUFR" through "BUFR+11". The length can be any number, regardless of TEMAC cell-list or other object boundaries. Mandatory error exits are provided for "end-of-file" and tape reading errors ("bad records"). The programmer must specify these exits, which cannot be dropped; he should place there the names of routines or statements which handle the conditions in whatever way he desires, e.g., finishing off the program run at "end-of-file", or preparing to read a new file; accepting a "bad record" or skipping to the next record. To accept a bad record, (which is frequently the most reasonable thing to do since the record is rarely seriously in error), an immediate return is made to the same READ statement; a warning may be displayed on the RUNLOG file with a "DISPI" debugging statement (see 5.9) before returning to the READ. To skip the bad record, an additional "dummy" READ statement should be executed and then a return made, with or without displaying a message on RUNLOG, to the original READ.

*T (n) NAME READ FILENAME TO TARGET$_1$ S$_1$ L$_1$ [/ ... / TARGET$_i$ S$_i$ L$_i$]/

EOF X ERR X THEN Z $

FILENAME	-	name of a file defined in a file card as an INPUT file.
TARGET	-	a cell name, cell-list name, or IBMAP address.
S	-	a positive decimal number, not a cell name; starting position.
L	-	like S; length.
EOF	-	end of file; X is exit address for this condition.

ERR - bad record exit; Y is exit address.

Examples of READ Statements:

```
*T/ST6Ø/READ/TEXTFL/TO/BUFR/1/2Ø/EOF/STX/ERR/STY/THEN/STZ$
```

Accepting bad records:

```
*T/ST6ØA/READ/INFIL/TO/BUFR/1/14/EOF/ST6ØX/ERR/ST6ØY/THEN/STZ$
*T1/ST6ØX/DISPI/19/THEN/ST6ØA$
/BAD/RECORD/ACCEPTED

*T/ST6ØY/DISPI/2Ø/THEN/ENDUP$
/END/OF/FILE/ON/INPUT

*T/ENDUP/FINISH$
```

Multiple Target areas:

```
*T/ST6ØB/READ/TEXTFL/TO/BUFR/1/1Ø/STREM/1/5/HITS/1/5/EOF/STX/ERR/STY$
```

5.6.2 The WRITE Statement: Verb WRITE.

A single physical record (block) is written onto a named file from one or more areas of memory. The total number of cells written must equal the block size of the file in the file card. The source name or names are like the target for the READ statement. Each source name must be followed by a starting cell position and a length in cells, as for READ. No error exits are required.

*T (n) NAME WRITE FILENAME FROM SOURCE$_1$ S_1 L_1[/ ... / SOURCE$_i$ S_i L_i]/

THEN X $

Examples of the WRITE Statement:

```
*T/ST61/WRITE/PRNTFL/FROM/BUFR/1/2Ø/THEN/STX$
*T/ST61A/WRITE/PRNTFL/FROM/LIST/1/5/BUFR/11/1Ø/HITS/1/5$
```

5.6.3 The OPEN Statement: Verb OPEN.

This statement causes a file to be "opened" by the IOCS system; this means that material is read from the tape into buffers outside of the TEMAC program, and various initializing actions are carried out in the buffering system. Physical reading takes place in IOCS independently of "READ" statements given by the programmer, which serve only to move data from the system buffers to the target area and to trigger more physical reading when more data is needed; the "opening" of the file starts off this process. The file may be opened with or without rewinding the tape to the beginning before getting data from the file; it is frequently safest to rewind the tape, if the beginning of the file is desired. The RUNLOG file should be opened by the programmer along with the others as close to the beginning of his program as possible; RUNLOG should never be rewound, however.

*T NAME OPEN OPTION FILENAME / THEN X $

OPTION - may be "REW" (rewind the tape), or "NOREW" (do not rewind)

Examples of the OPEN Statement:

*T ST62 OPEN REW TEXTFL / THEN STX$

*T ST63 OPEN NOREW RUNLOG$

5.6.4 The CLOSE Statement: Verb CLOSE.

This statement causes IOCS to "CLOSE" a file; closing involves releasing the system buffers in which material was being read for the file, and carrying out other actions which render the file unavailable until it has been re-opened, or in some cases (when the file has been "unloaded") unavailable under any circumstances. Closing should, therefore, be done when reading or writing in a file is completed, and it is desired that the system buffers be free for processing another file, or when the program run is over and all files are completed. The RUNLOG file should not be closed by the programmer.

*T NAME CLOSE OPTION FILENAME / THEN X $

OPTION - may be "REWUNL" (write end-of-file on an output file, rewind and unload the tape,) "REWEOF" (write end-of-file and rewind the tape,) "NOREW" (write end-of-file, leave tape positioned as it is,) "NOEOF" (does nothing to the tape; releases buffers). (end-of-file is not, of course, written unless the file is an output file).

Examples of CLOSE Statements:

*T^ST64^CLOSE^REWUNL^TEXTFL$

*T^ST65^CLOSE^REWEOF^MACFIL/THEN^STX$

5.6.5 Other I/O Actions: REWIND, WRITE END-OF-FILE, BACKSPACE RECORD, BACKSPACE FILE; Verbs REW, WEF, BSR, BSF.

The REWIND statement allows a tape to be rewound to the beginning, e.g., preparatory to re-reading it, without closing the file or affecting the system buffers. WRITE END-OF-FILE allows an end-of-file mark to be written and action temporarily suspended on an output file, prior to writing another physical file onto the reel, under the same file name, without closing the file. BACKSPACE RECORD allows a single physical record (not a logical record, as in the READ statement) to be re-read. BACKSPACE FILE allows a specified number of files on a multi-file reel to be re-read or re-written. Most TEMAC programmers will not need to use these provisions unless they are also familiar with IOCS, or have done something out of the ordinary with respect to tape usage. As may be seen care must be used in mixing these statements with READ, WRITE, OPEN and CLOSE statements, and the programmer must have a clear idea of the physical arrangement of material on the tape reel. All error exits must be specified.

```
*T NAME REW FILENAME / THEN X $

*T NAME WEF FILENAME / EOT X THEN Y $

*T NAME BSR FILENAME / EOF X THEN Y $

*T NAME BSF N FILENAME / THEN Y $
```

EOT Exit - taken if end of tape is found while writing end-of-file.

EOF Exit - taken if an end-of-file (signalling the end of the last file and the beginning of the current one) is found while backspacing a record.

N - positive decimal number, telling how many files are to be backspaced over; not a cell name.

Examples of REW, WEF, BSR, BSF Statements:

```
*T^ST66^REW^TEXTFL/THEN^STX$

*T^ST67^WEF^PRNTFL/EOT^STX^THEN^STY$

*T^ST68^BSR^TEXTFL/EOF^STX$

*T^ST69^BSF^2^TEXTFL$
```

5.7 Control Statements: Decisions.

The control statements provide ways of determining the path of control in the program. "Decisions" allow a choice among alternative paths; if some condition is present, a given statement will be executed next, otherwise a different statement. There are three TEMAC statements whose primary function is to provide a decision among alternatives, or a choice of where to go next in the program (MOVAD, IF, BRAN); some forms only alter single cellular values without providing a test or decision. All forms have only single cellular values as operands, and none require error exits. Most forms have a conditional exit routing, of which the positive exit may not be dropped; no label "YES" is used before the positive exit in these statements, since they are more easily readable without the label. No data type codes are needed on the verb, except in the case of BRANCH, which has two special data type possibilities.

5.7.1 The MOVE-ADD Statement: Verb MOVAD.

This statement changes the value in a source cell by multiplying or dividing it by another value, adding or subtracting one or more other values, or performing several such operations, then moves the result into a target cell or list component. If multiplication or division is specified, only one of these operations can be performed, and it must come first, before any additions or subtractions. Any number of additions and subtractions may be carried out. The source and target are cells or single components of lists, and the auxiliaries may be decimal numbers, octal numbers (with = sign prefixed), cells, or list components. The unconditional form of the MOVAD statement does not provide a test, but simply alters and moves a source value, then goes to a normal exit. The conditional forms change and move the source value, then apply various tests to the result or compare it to another, comparand value and provide a positive (unlabelled) or negative ("ELSE") exit depending on the outcome. If a comma appears as the source, the current contents of the accumulator register are used (this feature saves machine instructions, but must be used with care by someone who understands the machine code generated by the compiler). If a comma appears between parentheses as a pointer to select a list component, the same value as was specified in the previous pointer will be used; a similar feature was described for the ENTER statement. A source list component or cell name may be preceded by an attached minus sign; auxiliaries should not have such a sign (and do not require it since they are preceded by plus or minus signs as arithmetic function words, surrounded by blanks).

*T (n) NAME MOVAD SOURCE F AUX_1 [F ...F AUX_i] TO TARGET / THEN X $

*T (n) NAME MOVAD SOURCE F AUX_1 [F ...F AUX_i] TO TARGET / TEST CONDITION X ELSE Y $

*T (n) NAME MOVAD SOURCE F AUX_1 [F ...F AUX_i] TO TARGET / COMPARISON CONDITION COMPARAND X ELSE Y $

AUX_1 to AUX_i - a cell name, one list component, decimal number, octal number (preceded by attached = sign).

F - Arithmetic function word; may be "+" (plus), "-" (minus), ".D" (divided by), "." (multiplied by)..

Examples of MOVAD Statements:

*T^ST73^MOVAD^I^+^J^TO^K/THEN^STX$

*T^ST74^MOVAD^J^-^1^TO^ONE^VCLS(1)/RNG^-2^3^STX$

*T^ST75^MOVAD^ONE^VCLS(3)^+^5^TO^I/GRE^10^STX^ELSE^STY$

*T^ST75A^MOVAD^-J^.D^6^+^K^-^ONE^HITS(3)^TO^I$

*T^ST75B^MOVAD^ONE^HITS(I)^.^5^+^ONE^NUMS(,)^TO^ONE^VCLS(,)$

*T_ST75C_MOVAD_,_._J_TO_K$

*T_ST75D_MOVAD_I_+_1_TO_I/GRT_SN_STX$

5.7.2 The IF Statement: Verb IF.

The IF statement does not alter or move any cellular value; it provides the same tests as the MOVAD statement (conditional forms) as if the source value had been altered in specified ways (also similar to those provided by MOVAD), but leaves the source unchanged. All IF statements have a positive (unlabelled) exit and a negative ("ELSE") exit.

*T NAME IF SOURCE F AUX_1 [F ...F AUX_i]/TEST CONDITION X ELSE Y $

*T NAME IF SOURCE F AUX_1 [F ...F AUX_i]/COMPARISON CONDITION

COMPARAND X ELSE Y $

Examples of IF Statements:

*T_ST76_IF_I_ZRO/STX$

*T_ST77_IF_J_-_2_MNS/STX_ELSE_STY$

*T_ST78_IF_I_-_J_GRT_K/STX$

*T_ST78A_IF_,_.D_6_ZRO/STX$

*T_ST78B_IF_ONE_VCLS(CELL)_._ONE_NUMS(,)_+_2_EQU_ONE_HITS(,)/STX$

5.7.3 The BRANCH Statement: Verb BRAN.

The BRANCH statement allows a many-way decision among alternatives, keyed to the value in a single source cell, which may have a single value added or subtracted. The source cell is tested for one of a specified string of values or relations, and the corresponding one of a string of exits is selected. There are two forms of the statement: the immediate form (verb BRANI), which lists in the statement itself certain key values for the source cell and the exits corresponding to them, and the cell-list form (verb BRANL), which tests all possible values of the source cell and names a cell-list, usually predefined by a "DEFL I" definition statement, containing the exits as a list of statement names. The BRANI form has an "ELSE" exit which is taken if the source does not agree with any of the specified test values; the BRANL form has no exit routing at all, since all exits are supposed to be included in the exit list outside of the statement.

*T (n) NAME BRANI ON SOURCE [± AUXILIARY] CONDITION $COMPARAND_1$/ ...

/ $COMPARAND_i$ TO $NAME_1$ / ... / $NAME_i$ /ELSE X $

*T (n) NAME BRANL ON SOURCE [± AUXILIARY] TO EXIT LIST $

SOURCE - a cell name or one list component

AUXILIARY - a cell name, list component, decimal number, or octal number (prefixed = sign)

CONDITION - may be EQU, LSE, GRE

COMPARAND - positive or negative decimal numbers, octal numbers, cell names, list components.

NAME - statement name elsewhere in program.

EXIT LIST - cell list name; not one component of master-block

Examples of BRANCH Statements:

```
*T^ST79^BRANI^ON^J^LSE^1/4/7/8^TO^STA/STB/STC/STD/^ELSE^STX$

*T^ST8Ø^BRANL^ON^J^TO^JUMPS$

*T^ST8ØA^BRANL^ON^I^+^J^TO^JUMPS$

*T^ST8ØB^BRANL^ON^,^+^2^TO^JUMPS$

*T^ST8ØC^BRANL^ON^CELL^-^=77^EQU^=76/=75/=74^TO^STA/STB/STC$
```

5.8 Control Statements: Transfers of Control and Subroutines.

Transfers of control are unconditional jumps, or specifications of one place to go next in the program. Almost all TEMAC operation statements have exit routings built into them, so that the verb "GOTO", which has no other function except to specify where to go next, will seldom be needed in most programs, and is supplied only for the sake of completeness. The verbs "FINISH" and "START" are needed for communication with the IBJOB monitor; START to tell which is the first executable instruction of the program, and FINISH to return control to the monitor at the end of a run. The verbs "DO", "EXIT", and "RETURN" are very useful for providing internal closed subroutines within a

TEMAC program deck (as distinct from "external subroutines", which are other decks "called" by the TEMAC deck; more will be said about decks and external closed subroutines in Section 6). Closed subroutines within a TEMAC deck may consist of a single statement, executed "at a distance" by a DO statement, or they may consist of many TEMAC statements, and may contain loops and even sub-subroutines of their own. Subroutines consisting of more than one statement are executed by an EXIT statement, and the program finds its way back to the place it came from by a RETURN statement within the subroutine. All closed subroutines have the common feature that control is not transferred to them permanently, but instead, when their actions are completed, the exit routings of the original EXIT or DO statement are used to tell where the program goes next. Subroutines may, however, contain direct transfers of control which go to other parts of the program without returning to the place where the subroutine was entered (for example, if an error was found during execution of the subroutine, control might be transferred to a standard error action outside of the subroutine, and never return to the original exit address). A subroutine may be entered at any statement within it, before its exits . Because all TEMAC operation statements may have explicit exit routings, the path of control is much freer and less constrained within a subroutine than in languages that follow a strict sequence from one instruction to the next.

5.8.1 The GOTO Statement: Verb GOTO.

The single exit name after the verb is taken as the name of the next statement to be executed; GOTO is an unconditional jump or transfer of control.

```
*T NAME GOTO X $
```

Example of GOTO Statement:

```
*T^ST81^GOTO^STX$
```

5.8.2 The START Statement: Verb START.

This statement should appear as the first statement after the $TEMAC control card for the program, and especially, before any of the data definition statements. It tells the IBJOB monitor which statement is the first to be executed at run time. (For a discussion of "run-time" and "compile-time" and a description of control cards see Section 6). The statement name for START should be the program name, or deck name as given in the $TEMAC card, and the name after the verb is the name of the first operation statement to be executed - i.e., where the action of the program actually begins. Every TEMAC program must have a START statement.

```
*T DECKNAME START X $
```

Example of START Statement:

```
*T^TXEDT^START^BGIN$
```

5.8.3 The FINISH Statement: Verb FINISH.

This statement tells the IBJOB monitor that the program run is complete, or is to be terminated, and the monitor is to take over, close out the job, and bring in the next job of the batched run. There is no operand or exit, but simply the verb "FINISH". A FINISH statement is not needed if the programmer has used a "DUMP" debug statement to end his program, as this automatically returns control to the monitor.

```
*T NAME FINISH $
```

Example of FINISH Statement:

```
*T^ST82^FINISH$
```

5.8.4 The DO Statement: Verb DO.

The DO statement executes another, single TEMAC statement of a special type (the "object" statement) by remote control, so that on completion of the action of the object statement, the program returns to the exit routing of the DO statement to find where to go next. The object statement must be of a form that differs from the normal statement in that is has the word ".EX" after a slash in place of the usual exit routing. The DO statement, instead, has those exit labels and names which would be appropriate to the verb of the executed statement. The DO statement is useful for executing lengthy and complex EXT, EDT, CVT, COM, LOC, and SEG statements that are to be used again and again in different parts of the program; these statements need then be written and compiled only once. It should be noted that a statement with ".EX" as its routing should only be executed through DO statements and should not be named directly as an exit in the routing of a normal statement, a START statement, or a GOTO statement.

```
*T NAME DO STA / ERR X YES Y THEN Z $          (conditional routing)

*T NAME DO STB / ERR X THEN Y $               (unconditional routing)

*T NAME DO STC / THEN X $
```

Examples of DO Statements and their Object Statements:

```
*T^ST83^DO^ST831/ERR^STX^THEN^STY$
     *T^ST831^EXTWW^ONE^S1(1)^A^,^TO^FW/STAB^TTAB/.EX$
*T^ST84^DO^ST841/YES^STX$
     *T^ST841/LOCWS^W2^ID^IN^S2/HITS^CT/.EX$
```

Note: Only the following statements may be the objects of a "DO" - SEG, SEL, COL, OMT, INC, REF, EXT, EDT, CVT, CVN, MOVE, REPT, LOC, COM, FIND. (A method of "gimmicking" DO statements for the others will be shown in Part II.)

5.8.5 The EXIT and RETURN Statements: Verbs EXIT, RETURN.

The EXIT statement causes its own name to be stored, or "remembered," in the specified one of ten special "return address boxes," and then causes an unconditional jump to a statement within an internal closed subroutine elsewhere in the program. This subroutine must contain a RETURN statement referring to the same "box" number as the EXIT statement used. When the RETURN statement is encountered in the subroutine, control returns to the EXIT statement whose name is stored in the specified return box, and the exit routines of that statement are examined to see where to go next. If a subroutine has a sub-subroutine, they must use different boxes, since whenever a box is used by an EXIT statement, the previous return address it contained is covered up by the new one. Several different conditions may arise in a subroutine, so that there may be a choice of places to go next from the subroutine. In that case, the original EXIT statement should have exit names after the slash and the label "THEN" for each of the different conditions, and there will be as many different RETURN statements at various places within the subroutine, each selecting a different exit. The RETURN statement may specify exit 1, 2, etc. up to any number that were present in the EXIT statement. If there is only one return address, and it is to the next statement in sequence in the "main" routine, the THEN exit may be dropped on both EXIT and RETURN statements. If a single return address only is specified on the EXIT statement, the RETURN statement need have no "THEN" exit. But if more than one exit address is present on the EXIT statement, the RETURN statement must specify the exit number as its "THEN" exit.

```
*T (n) NAME EXIT TO STA ON BOX K / THEN A B ... Z $
*T (n) RETURN TO BOX K / THEN EXIT M $
```

K - box number; a positive decimal number not greater than 1Ø

M - any positive decimal number

Examples of EXIT and RETURN Statements from an actual TEMAC Program:

```
*T^ENDT1^EXIT^TO^FDDAT^ON^BOX^1/THEN^S1TX1^ENDT2$
*T^ENDT2^EXIT^TO^WRITE^ON^BOX^2/THEN^R2$
*T^S1TX1^SELSS^S1^AS^S2/L$
          (Main program continues)
    *T^FDDAT^LOCFS^ONE^S1(4)^ID^IN^SEQ^2^MSCUE(25)/HIT/YES^FDDT1$
    *T^FDDX^RETURN^TO^BOX^1/THEN^EXIT^1$
    *T^FDDT1^RETURN^TO^BCX^1/THEN^EXIT^2$

*T^WRITR^SELSS^OS1^AS^S2/JK$
*T^WRTR1^EXIT^TO^LOGR2^ON^BOX^1$        (subroutine using Box 2, which
                                         has a sub-subroutine On Box 1)
*T^WRTR6^RETURN^TO^BOX^2$

      *T^LOGR2^DISPL^H^S2$
      *T^LOGR3^RETURN^TO^BOX^1$      (sub-subroutine)
```

5.9 Debugging Statements.

TEMAC debugging facilities fall into two main types: TRACING and DISPLAYING. TRACING causes the name of each statement being traced, and the contents of the DBUG error cell (if non-zero), to be placed on the RUNLOG file each time the statement is executed during the program run, so that the programmer may see a detailed picture of the path of control throughout his program. In many cases, a well chosen set of trace outputs and a printout of the input data are sufficient for debugging a TEMAC program. DISPLAYING causes a printout of the contents of a TEMAC object to be placed on the RUNLOG, in a specified format (e.g., decimal, octal, or characters), with or without its name as a label.

In addition to the TRACE and DISPLAY provisions, there are several debugging facilities which, while not properly a part of TEMAC, are available to the TEMAC programmer as a part of IBJOB. Among these are the AID debugging package, developed during the implementation of TEMAC to aid the system programmers in debugging, and generalized for use by all 7094 programmers as a part of the system library. AID allows single cells to be displayed in decimal, octal, or character format on the RUNLOG file (SYSOUl, or Bl), and it allows streams of from one to 24 characters to be placed on the RUNLOG as headings, comments, etc. There is also a system memory dump, which provides dumps in various formats for larger segments of memory than AID or the TEMAC DISPLAY facility can handle; these are familiar tools for the programmer, but may not be as easy to read for someone who has never used a "memory dump." There are two verbs for dumps, PDUMP which allows dumps to be made in the midst of the program run, and DUMP which terminates the run and returns control to IBJOB as soon as the dump is completed. There is a third system debugging facility, the SNAPSHOT program, which is called by a calling sequence like a subroutine; this program is similar to PDUMP in its provisions, but has the advantage of greater economy in time and in usage of space on the RUNLOG file. It is recommended that SNAPSHOT be used for dumps in the midst of the program, rather than PDUMP. To obtain further information about AID, DUMP, SNAPSHOT and other information on IBJOB and IBSYS facilities, refer to the relevant C44 Technical Bulletins, or see someone in C443.

5.9.1 Tracing.

```
*TRACE$

*TRACEALL$
```

If single selected statements are to be traced, a special TEMAC control card containing the single word *TRACE$, punched in columns 1 thorugh 7, should be placed immediately in front of each statement to be traced. A large number of these cards can be punched up at once, and inserted at different points in the program during debugging. Frequently, especially in early stages of debugging, it is more convenient

to trace all traceable instructions of the program, or all between given points. In this case, a single control card, containing the word *TRACEALL$ in cols 1 through 1Ø should be placed before the first instruction of the set to be traced. In addition to the control cards, it is also necessary to enter a non-zero value into a system switch cell called "ENABLT" before tracing begins. This switch "enables," or makes effective, the tracing facility, and if it is zero (the value it has to start with unless changed by the programmer in his program), no trace will appear even if trace control cards are present. By compiling his program with *TRACEALL$ at the beginning, and then setting ENABLT to zero or non-zero at various points in his program, the programmer may selectively trace various subroutines or routines under program control, without the nuisance of punching up many *TRACE$ cards. A program that has been compiled with *TRACE$ or *TRACEALL$ cards may be run without any trace output by leaving ENABLT set to zero. In general, it is a good practice to remove all *TRACE$ and *TRACEALL$ cards from a program and recompile when it is about to become operational, after debugging is done, since the testing of the switch ENABLT uses up a certain number of extra machine cycles even if the switch is zero and no tracing occurs.

5.9.2 Displaying: Verb DISP.

*T DISPC TYPE [LABEL] SOURCE / THEN R$

*T1 DISPI K / THEN R $
second card - 1 to 24 characters after blank in col 1.

*T DISPL TYPE SOURCE [S L] / THEN R $

TYPE - format of output; may be D (decimal), 8 (octal), H (character-stream)

SOURCE - a cell name, list name, or an IBMAP address; any single TEMAC name other than that of a paragraph or master-block.

K - the number of characters (maximum 24) to be displayed

S - starting cell relative to source name; a positive non-zero decimal integer only.

L - length of cell sequence to be displayed; a positive non-zero decimal integer only.

DISPC ("Display cell") displays the contents of single cells, and labels the output with the source name if "LABEL" is requested (see examples below). Its output is one or two lines on the RUNLOG, containing the label if requested and a single five digit decimal number, octal (12 digit) number, or 6 BCD characters. If the cell contained zero, the output will be blank. DISPI ("Display immediate message") displays

from 1 to 24 characters specified on a single overflow card following the header of the DISPI statement; this card must have a blank in col. 1, and the characters for display are counted starting with position 2. It produces one line on the RUNLOG containing the displayed characters exactly as they appear on the DISPI overflow card. DISPL("Display List") displays a sequence of cells as a single line on the RUNLOG in one of three formats. Decimal format provides from 1 to 17 five-digit decimal numbers, each preceded by a plus or minus sign. Octal format provides from 1 to 9 12-digit octal numbers separated by blanks. Character-stream format provides from 1 to 2Ø cells full of BCD characters in a continuous stream. There is always a blank in the first position of the output line, so that program control may be used for listing. If no S and L fields appear in the DISPL statement after the source, the maximum number of cells for the format will be displayed. S may be any number, from 1 to 32767 (the limit of storage size), regardless of TEMAC object boundaries. L may be any number less than or equal to the maximum for the format; if a larger number is specified (e.g., 2Ø for "octal"), the maximum number only will be provided. If sequences of cells greater than the maximum allowed are to be displayed, several DISPL statements should be used, with S values or IBMAP source addresses so chosen as to form the desired sequence. The source in DISPL may not be a one-component operand, but must be a single name.

Examples of DISP Statements:

```
*T,DSC1,DISPC,8,LABEL,BUFR$

*T,DSC2,DISPC,D,LABEL,ONE,VCLS(J)/THEN,STX$

*T,DSC3,DISPC,H,STREM+2$

*T,DSC4,DISPC,H,LABEL,ONE,STREM(3)$

*T1,DSI1,DISPI,21/THEN,STX$
,END,OF,FILE,ON,INPUT.

*T1,DSI2,DISPI,16$
,TAPE,READ,ERROR.

*T,DISPL,D,VCLS,1,3/THEN,STX$

*T,DISPL,D,VCLS$

*T,DISPL,8,LIST,3,3$

*T,DISPL,H,BUFR$
```

5.9.3 Snapshots: SNAP Subroutine.

Snapshots provide an octal memory dump of sections of storage on the RUNLOG file. They are familiar tools for 7094 programmers, and can be useful to others who wish to learn their simple format. The following calling sequence, inserted into the program in IBMAP coding at any point where a SNAP is desired, will produce a dump of the TEMAC reference cells (PARLO, SNTLO, WRDLO, MBKLO, LSTLO), the programmers data objects and "constants" (CONST), and the TEMAC-compiled constants (LITRL) for the average program: (See Part II for an explanation of these names).

```
col. 8   col 16
     TSX SNAP,4
     MON RUNLOG,,2
     PZE PARLO,,LITRL-2ø
```

If the program itself is to be snapped as well (which is not usually necessary), the last line of the calling sequence should be:

```
     PZE NAME,,LITRL-2ø    (where NAME is the first executable
                            instruction, e.g. the one specified
                            after START)
```

The snapshot output is preceded by a console dump, telling contents of key machine registers (M Q, accumulator, index registers, etc).

5.9.4 Dumps: DUMP Call.

Dumps are, like snapshots, a tool for the machine programmer, which may not appeal to those not familiar with their format. If the programmer wishes a dump at the end of the program, just before terminating the run, he uses the following MAP "call":

```
col. 8   16
     CALL DUMP(NAME1,NAME2,ø)
```

This statement provides an octal dump of locations NAME1 through NAME2, with a few extra preceding and following, and a console dump of machine registers. The DUMP call turns over control to IBJOB as soon as the dump is completed, so that a "FINISH" statement is not required. If dumps during the program run are desired, the programmer is advised to use SNAPSHOT (5.9.3), as PDUMP wastes machine time and print time with no advantage over the output of SNAPSHOT.

6. The Object Program.

The TEMAC object program, when punched on cards and supplied with appropriate control cards for the IBJOB monitor and IBSYS system, will be submitted as one deck of a job, to be compiled, assembled and run by 7094 operations.

6.1 The Job and the Deck.

6.1.1 The Job.

A "job" is all of the material submitted by a programmer for a machine run; it has a "job name" distinct from the names of any of its component decks. A job may contain only one program deck, or it may contain many separate decks, each of which may use a different compiler or assembly program within the system (e.g., FORTRAN, COBOL, TEMAC, IBMAP); some decks may not require compilation or assembly ("object decks" as opposed to "source decks", in IBM terminology); there may be data cards to be read in by one or more of the decks, and placed at the end of the entire job. Data tapes ("files") may also be submitted with the job, to be read by some deck or decks within the job at run-time.

6.1.2 The TEMAC Deck; Its Relation to Other Decks in the Job.

The TEMAC deck may contain TEMAC statements alone, or intermixed with IBMAP statements. It may not contain FORTRAN, COBOL, or any other non-IBMAP statements intermixed with TEMAC statements, but it may "call" other decks within the job, compiled or assembled by FORTRAN, COBOL, etc., as external closed subroutines, or it may call subroutines within the IBLIB library (e.g., SNAP, DUMP, AID), or in the COBOL or FORTRAN subroutine libraries. The calling of external subroutines may be done either by using the "calling sequences" for them in IBMAP language, (which may be learned by reference to the appropriate manual or technical memo, or from someone in C443), or by using an IBMAP "call" statement, or forming a "MACRO" in IBMAP (for which see someone in C443 or the IBMAP manual). FORTRAN or IBMAP decks may call TEMAC decks within the same job in a similar way. The deck has a deck name, or "program name", distinct from the job name. Deck names and job names are formed the same way as data object and statement names, and may have from 1 to 6 characters. All the decks of a job are assigned storage locations for a given run in accordance with a "relocatable" or relative system of addressing, and loaded into the machine by a sophisticated "loader" program; there are many advantages to relocatable addressing, and it is a real necessity where multi-deck jobs, in many different languages, are being run as one entity. In general, names within any one deck may not be used to refer to the same objects or locations in any other deck, unless the deck where they are first defined or set up has an "ENTRY" statement in IBMAP for them (e.g., for the name "BLOCK", defined in Deck A, to be used to mean the same storage location in Deck B, Deck A must contain a statement "ENTRY BLOCK", with "ENTRY" in col. 8 and "BLOCK" in col. 16. Similarly, if names in one deck are to be "SNAPPED" in another deck, they must have ENTRY cards.

6.1.3 The Arrangement of Cards for a Job.

The Job contains the following parts:

```
                col.            col.            col.
                 1               8              16

Job             $JOB                           JOBNAME, NUMBER, PROGRAMMER'S
                                                 NAME, PHONE
control
cards           $PAUSE
                $EXECUTE                       IBJOB
                $IBJOB          JOBNAME        MAP
                [$IEDIT                        ALTER]
                [$OEDIT                        PREST]
first           $TEMAC          DECK1          NODECK
deck            *T^DECK1^START^ X$
(TEMAC)         (file cards may come anywhere before "END")
                TEMAC and IBMAP data definitions, alphabets.
                TEMAC and IBMAP operation statements

                                END
second          $IBMAP          DECK2          NODECK   (other options)
deck                 (file cards)
(IBMAP)              IBMAP statements
                                END
                7/8
                Data cards (if any)
end of job      7/8
```

6.1.4 Control Cards.

6.1.4.1 The $JOB Card. This card, the first of the job, contains the job name, job number, programmer's name, phone number, and any other useful information, such as the date or a comment on the run.

6.1.4.2 The $PAUSE Card. It need contain only the word $PAUSE in column 1.

6.1.4.3 The $EXECUTE Card. It need contain only the word $EXECUTE in columns 1-8 and the word IBJOB in columns 16-20.

6.1.4.4 The $IBJOB Card. It contains the word $IBJOB in cols. 1-6 and the job name, as in the $JOB card, starting in col. 8. The word MAP in col. 16 should be included, since it causes a "memory map" to be provided by the system, telling where in storage the relocatable decks were loaded after assembly for execution. This information, while primarily useful to the experienced machine programmer, should be routinely included in the listing of TEMAC programs.

6.1.4.5 $IEDIT and $OEDIT Cards. These are optional, if the programmer wishes to change his deck before recompiling through the "ALTER" facility in the system. If he wishes to use this facility, he must use an $OEDIT card calling for "PREST" (col. 16) in his assembly and compilation. On subsequent recompilations he may then use an $IEDIT card calling for "ALTER" (col. 16). For the instructions for using ALTER the programmer should consult the relevant manual or C443. The PREST option produces a special type of program deck, in condensed form, which may then be corrected by ALTER cards in the next run.

6.1.4.6 The $TEMAC Card. This card is similar in every way to an $IBMAP card, except for the first 6 positions. It may have many option fields, which control the printing of the assembly listing, the production of program decks, etc. These may be left out, as on the other control cards, unless the programmer requires some provision which is different from what the system routinely provides. The $TEMAC card need have only the deck name (1-6 chars in col. 8) and a card count in column 16, which should be a rough upward estimate of the total number of cards in the deck.

6.1.4.7 The END Card. This card marks the end of the program deck; it contains only the word END in col. 8.

6.1.4.8 7/8 Card. This card is the equivalent of an "end-of-file" on the job tape; if there are no data cards for the job, it marks the end of the job. There may be one or more files of data cards, separated by 7/8 cards, and the last one terminated by a 7/8 card. These will be read in by the object program as an input file (file definition FNAME5 below) at run time.

6.1.5 File Cards.

The following forms for file definitions are recommended for those who are unfamiliar with IOCS. For an explanation of details see Part II. No spaces must appear after col. 16 until the end of the card.

```
col      col     col
 1        8      16
FNAME1   FILE    ,A(1),MOUNT,INPUT,BLK=XXXX,BCD,MULTIREEL,{LOW,|HIGH,}{HOLD|PRINT}
FNAME2   FILE    ,B(1),MOUNT,OUTPUT,BLK=XXXX,BCD,MULTIREEL,{LOW,|HIGH,}{HOLD|PRINT}
FNAME3   FILE    ,A(2),MOUNT,INPUT,BLK=XXXX,BCD,MULTIREEL,{LOW,|HIGH,}{HOLD|PRINT}
FNAME4   FILE    ,B(2),MOUNT,OUTPUT,BLK=XXXX,BCD,MULTIREEL,{LOW,|HIGH,}{HOLD|PRINT}
FNAME5   FILE    ,INL,READY,INPUT,BLK=14,MULTIREEL,BCD,NOLIST
RUNLOG   FILE    ,OUL,READY,OUTPUT,BLK=22,BCD,MULTIREEL,NOLIST
```

OUTPUT/INPUT - The programmer may specify this for FNAME1 to FNAME4; the suggested arrangement for these files is appropriate to a program with two input and two output files. FNAME5 (data cards at the end of a job) must be an INPUT FILE, and the RUNLOG must be an OUTPUT FILE.

BLK=XXXX - block size; the number of cells in each physical record on the file. This must be the same as the cell count in any WRITE statement that writes on the file. It is a decimal number with one to four digits.

MULTIREEL - allows for running over onto another tape reel, which is handled automatically by IOCS.

LOW/HIGH - the "density" of the tape; this will be determined by the use to be made of output tapes, or the source of input tapes, but it must be known.

HOLD/PRINT - the tape is kept in the 7094 area to be picked up by the programmer, or else sent to be printed out.

NOLIST - concerns 7094 operator actions for certain files.

(other fields in the cards may be looked up in Part II.

6.2 Compilations, Assemblies, Runs.

6.2.1 The TEMAC Program at Compile and Assembly Time.

At this time, the TEMAC Compiler Program is being executed by the 7094, under control of IBSYS and IBJOB. The TEMAC Compiler is in memory, and is reading in the TEMAC program deck within the job. TEMAC has been called in by IBSYS, because of the $TEMAC control card which IBSYS has found in front of the deck. The TEMAC compiler places its output on a system tape, and returns control to IBSYS after it finds the "END" card for the deck. Then IBSYS calls in the IBMAP assembler and turns over control to it. IBMAP reads the system tape as requested by the programmer. (The "two-phase" action of TEMAC followed by IBMAP is similar to the method of COBOL compilation; it adds only a few seconds to the total processing time). The TEMAC Compiler produces a listing of the TEMAC statements exactly as the programmer submitted them, with various indications of compile errors, which forms part of the RUN LOG file for the job. The IBMAP assembler then produces a listing of the machine instructions, along with all the TEMAC statements, which appear

as "comments," and all the indications of assembly errors; in addition, there is much useful information about the assembly and the job as a whole: for example, a "memory map" to provide the actual locations of the decks in memory at run time, and a "cross reference dictionary" to list every reference to every name in the program. If there were no serious errors during the compilation or assembly, the object program just created will be read in by the loader (IBLDR) and executed, so that "run-time" begins immediately after assembly.

6.2.2 The TEMAC Program at Run Time.

All the decks of the job are in the machine. One of the decks (probably a TEMAC or IBMAP deck) is the "main program" and is being executed; it may call in (i.e., jump in and out of) various other decks in the job which are its external subroutines. Subroutine decks may themselves call in other decks as subroutines. Debugging output from SNAPSHOT, DUMP, and TEMAC debugging statements is being written on the RUN LOG, where it will follow the compiler listing and the assembly listing for the job. The programmer's input and output files are being read and written. When the path of control encounters a DUMP or FINISH statement, the run is terminated, any unclosed files are closed, and control returns to IBJOB. The RUN LOG will be printed and returned to the programmer, along with his tapes or the listings of any files for which he has requested printing. For further instructions regarding debugging and operational procedures for the 7094, see Part II.

6.2.3 The Octal and IBSYS Character Sets.

octal value	print/punch	octal value	print/punch
00	0	40	-
01	1	41	J
02	2	42	K
03	3	43	L
04	4	44	M
05	5	45	N
06	6	46	O
07	7	47	P
10	8	50	Q
11	9	51	R
12	invalid	52	invalid
13	-	53	$
14	- or '	54	*
15	invalid	55	invalid
16	invalid	56	invalid
17	invalid	57	invalid
20	+	60	blank
21	A	61	/
22	B	62	S
23	C	63	T
24	D	64	U
25	E	65	V
26	F	66	W
27	G	67	X
30	H	70	Y
31	I	71	Z
32	invalid	72	invalid
33	.	73	,
34	)	74	(
35	invalid	75	invalid
36	invalid	76	invalid
37	invalid	77	invalid

6.2.4 Compiler Error Messages at Compile Time.

During compilation, the compiler places error messages on the RUNLOG file along with a listing of the TEMAC statements of the object program, as a part of the Source Program Listing. The error message is preceded by six asterisks, and appears immediately following the statement to which it refers. The same error message, both preceded and followed by six asterisks, is placed on the IBMAP Assembly Listing (which follows the Source Program Listing on the RUNLOG print); the error message may appear after a few MAP instructions have been assembled and placed on the print, but usually it follows the TEMAC instruction immediately. The compiler usually forms all the MAP instructions up to the point of error, and frequently leaves space for the missing instructions, so that they may be corrected or entered by means of "ALTER" cards if the programmer wishes to do so. The error messages, while accurate in most cases, do not always exactly describe the condition that prevented further compilation of the statement; this is because errors in variable fields, which cannot be caught when they occur, are not sensed by the compiler until they indirectly displace some required or restricted field a few words further on in the statement. The error is apt to be no more than two words before the point specified in the error message. We hope to improve the accuracy and informativeness of error messages as we gain experience with the compiler; in the meantime, if a programmer is puzzled about an error message, he should consult someone in the TEMAC group of C443. It is also possible that certain conditions, which will cause the compiled coding to be different from the programmer's intentions, may slip through the compiler without being signalled by an error message. Until all such unsignalled errors can be found and dealt with, we suggest that the programmer who understands IBMAP coding check his IBMAP assembly for undefined virtuals, improperly qualified addresses, and assembler error messages if he can find nothing wrong with his TEMAC Source Program Listing. Others should consult someone in C443, or M. D'Imperio. Below is a list of the error messages and an explanation of their probable causes.

6.2.4.1 NOT A TEMAC VERB. The verb of the TEMAC statement was unrecognizable (misspelled, broken, run onto other words).

6.2.4.2 NOT A TEMAC DATA TYPE. One of the data type codes on the verb was wrong, although the verb itself was recognized.

6.2.4.3 NO $ TO END STATEMENT. The dollar sign, which must end the TEMAC statement, was not found (because it was mis-punched, left out, or else the value in the overflow card number after *T was wrong).

6.2.4.4 TOO MANY OBJECT NAMES. There were more than 5ØØ TEMAC data object names in the program, and the compiler symbol table is full. No further definitions will be compiled, but compilation proceeds. Any later program statements referring to objects that were not compiled will cause error 6.2.4.14 (see below).

6.2.4.5 STATEMENT INCOMPLETE. Some crucial field or fields were left off at the end of the statement; it is too short.

6.2.4.6 TOO MANY COMPONENT NAMES. One data object had more than 200 named components, causing an overflow in a temporary table. Compilation of the DEF statement involved ceases, but all following statements are compiled correctly, and the name of the bad data object is still recognized by the compiler. Statements referring to names of components that were not compiled will cause error 6.2.4.14.

6.2.4.7 TOO MANY CARDS. An overflow card number was greater than 17.

6.2.4.8 SOME M-BLOCK INCOMPLETE. A component list required for a general master-block definition earlier in the program has been left out, or has failed to compile. The incomplete block is the master-block definition just before the one currently being compiled (the compiler discovers the condition when it tries to compile the next master-block). The compiler wipes out the left-over information for the missing list and proceeds to compile the new master-block. There will be an error in the reference tables at this point, concerning the bad master-block only; all other data objects are unaffected.

6.2.4.9 SOME PARA INCOMPLETE. A component sentence required by a general paragraph definition has been left out or has failed to compile.

6.2.4.10 COMPONENT NUMBER WRONG. The number of components called for by the header of a DEF statement without component names does not agree with the actual number of components found in the data cards for that statement.

6.2.4.11 ERROR IN CONTROL STRING. A match or edit control string contains an improper character (a non-control character in a position where a control character is expected). This may be caused by a literal character left out, an = sign before an octal coded literal mis-punched or left out, etc.

6.2.4.12 PREPOSITION NOT FOUND. The preposition separating the primary operands of a TEMAC statement was not found; it was mis-spelled, run onto nearby words, or left out, or else an error in a previous word caused it to be missed.

6.2.4.13 EXIT ROUTINGS INCORRECT. The statement does not have the exit routing labels it should have; a conditional statement does not have a YES exit, a read/write statement has no ERR or EOF exits, or an exit label occurs with no exit address following it.

6.2.4.14 ERROR IN OBJECT NAME. A TEMAC data object name referred to in the operand of an operation statement was not recognized as one of those defined by the DEF statements. This may be due to the mis-spelling of the name, failure to include a DEF statement for it, or the failure of its DEF statement to compile. Run-on or left out fields earlier in the statement may cause the compiler to attempt to look up the wrong word as an object name.

6.2.4.15 NO RETURN ADDRESS BOX. The label "BOX" did not appear, or was not found where expected, in an EXIT or RETURN statement.

6.2.4.16 WRONG DATA TYPE ON VERB. A verb has a data type code which, while one of the possible TEMAC types, is not allowed on this verb in that position.

6.2.4.17 NO ALPHABET NAME. The alphabet name was not found in a SEG statement; it was left out, run onto other words, or a previous error caused it to be over-run.

6.2.4.18 ERROR IN WORD XXX. This message is by far the most frequent, and is intended to cover all cases not covered by the specific messages above. If the programmer cannot find anything wrong with the word designated, he should look at the words immediately preceding it. If the cause of error is still a mystery, see someone in C443, or M. D'Imperio.

6.2.5 Debug Error Codes at Run Time.

While the object program is running, every time a TEMAC statement which uses one of the processor subroutines is executed, a value is placed in a system communication cell called "DBUG", which is accessible to the programmer. If no error or special condition occurred during execution of the statement, DBUG contains zero. A non-zero value in DBUG does not always mean an error; it may simply report a condition which may or may not be an error depending upon the action that is being performed in the program (for example, if a result list is "full", there may be no error if the intention was to fill the list). If a trace is requested, the contents of the DBUG cell are printed next to the name of the statement in the trace output. If DBUG was zero, or was not used by the statement, a blank will appear for its value. Statements which use processor subroutines, and create new values in DBUG, are the following: SEG, SEL, COL, OMT, INC, REF, EXT, EDT, CVT, MOVE, REPT, CVN, LOC, COM, FIND). DBUG error codes are numbers between Ø1 and 52; of these, all numbers between Ø1 and 1Ø refer to errors in the source operand; 11 to 2Ø refer to the target operand; 21 to 3Ø refer to the comparand, 31 to 4Ø refer to an auxiliary (MCS, ECS, added or subtracted value, etc.), and 41 to 52 are miscellaneous codes for special conditions. Below is a list of DBUG error codes and their meanings; "x" below may be Ø, 1, 2, 3, or 4, for source, target, comparand, and auxiliary.

x1 - object unobtainable
x2 - component unobtainable
x3 - object flagged with omit flag
x4 - pointer has negative or zero value
x6 - pointer exceeds length of object it refers to
x8 - set length zero or too large for set
x9 - sequence length zero or too large for length of object
(x+1)Ø - object length zero
41 - target object threatens to overflow; is full.
42 - target component threatens to overflow; is full.
43 - result list
44 - result list is unobtainable.
45 - control string is unobtainable.
46 - control string is incorrect.
47 - end of source stream reached.

48 - attempt to convert non-numeric character or stream longer than eleven numbers to cellular form in CVN.
50 - TAB cell negative on entering SEG.
51 - target stream filled; end of target stream reached.
52 - maximum allowable number of characters moved (7200) in one EXT, EDT, or CVT statement.

Study Questions, pages 1 -- 4

1. What data types are possible for TEMAC data object definitions?

2. What is the difference between a data object and an operand?

3. What is the difference between textual data and cellular data?

4. What is the difference between a TEMAC cell-list which happens to contain text and a TEMAC word or sentence?

5. Give examples of things that might be TEMAC paragraphs made up of sentences in some program; give examples of TEMAC words.

6. Give examples of TEMAC master-blocks, cell-lists, cells.

7. When does textual data have to be converted to cellular form?

8. When does cellular data have to be converted to textual form?

9. What purposes can a TEMAC alphabet be used for?

10. Make up some examples of character-categories that might be useful in a program.

11. Does data always have to be moved in a TEMAC program? When can structural operations be used instead of data-moving operations?

12. Think of some examples of situations in a program where data need not be moved, but a new arrangement of data must be temporarily recorded or "remembered".

Study Questions, pages 5 -- 10

1. What are the things every TEMAC statement must have?

2. What would happen if some words of a TEMAC statement were run together: a) in an operation statement ? b) in the data cards of an immediate definition statement?

3. What would happen if more than one blank were to come between words of a TEMAC definition or operation statement?

4. What would happen if a slash came between two separate data words in a TEMAC immediate definition statement?

5. Tell in your own words what the following definitions do: dummy, immediate, padded, fielded.

6. Give examples of situations in a program where an immediate, dummy, padded, and fielded definition would be useful.

7. What is the purpose of a source operand? a target operand?

8. Where are the data type codes for the primary operands in the operation statement?

9. What two special-purpose data types are possible on verbs of data manipulation statements?

10. Given a sentence named "B", with seven component words named "W1, W2, W3, W4, W5, W6, W7", write TEMAC operands to obtain the following data from the sentence; (let the sentence contain the data words "HERE ^IS^ A^SENTENCE^WITH^SEVEN^WORDS". a) The words "HERE, A, SEVEN". b) All the words of B from "IS" on to the end. c) The single word "SENTENCE". d) Four neighboring words starting with "IS". e) All of sentence B. f) If sentence B is the third sentence in a paragraph "PB", give another way of getting all of sentence B.

11. Write "word-based character-stream" operands to obtain the characters "ENCE^WI" from sentence B in two different ways. (The character ^ stands for blank, or space.)

12. If the character-stream "^^ABC^DEFG" is in the cell-list "IBUFR", which is the second of two cell-lists in master-block BUFFS, write "list-based character-stream" operands to obtain the characters "C^DE" in two different ways.

13. If a source word 6 characters long, e.g. the word "FISHES", is to be physically moved into a target word-space only 4 characters long, what would happen (i.e., would there be an error exit from the statement, and what would be moved into the target?) a) if the source data type code is "F" on the verb? b) if neither data type code is "F"?

14. If the 6-character source word "EATING" is to be compared against a 3-character comparand word "EAT", what would happen (i.e., would there be a hit? Which exit would be taken from the data testing statement? How many character-pairs would be tested?) a) if the source data type code is "F"? b) if the comparand data type code is "F"? c) if neither data type code is "F"?

15. Give some examples of impossible or unobtainable operands for sentence B.

Study Questions, pages 11 -- 14.

1. Set up alphabet definition statements for the following alphabets (here and in all following questions, use the character ^ to mean "blank" in your answers; word spacers are perhaps the single most important feature in the TEMAC statement, and should be positively and specifically indicated.) a) An alphabet named "ABET1", having the following categories: even numbers; letters A, B, C; odd numbers; letters X, Y, Z; word-boundary letters comma and blank; sentence-boundary letters period and slash. b) An alphabet named "ALPH2", having categories: vowels; consonants before

M; all the numbers zero to nine; word-boundary characters plus, dash, blank; sentence-boundary characters asterisk, dollar-sign. c) An alphabet named AB2, having categories: all the letters in reverse order; all the numbers in reverse order; word-boundary letters blank and plus.

2. Set up a definition for a dummy paragraph named "PD", with spaces for four sentences, having maximum lengths of 3, 5, 3, and 7 words.

3. Set up definitions for a general paragraph named "PARA" and its component sentences named "SA, SB, SC". SA is a dummy sentence having space for 10 words. SB is a fielded sentence having 3 words all in the base stream "OBUFR" and named "FA, FB, FC"; choose your own starting positions and lengths for the fields. SC is an immediate sentence containing the three words "THREE$_\wedge$WORD$_\wedge$SENTENCE".

4. Set up a definition for a dummy word named "VWORD".

5. Set up a definition for a fielded word named "FIELD", in base stream OBUFR, starting at character 12 and 6 characters long.

6. Set up a definition for an immediate word named "WCONS", containing the letters "YESTERDAY".

Study Questions, pages 14 -- 17

1. Set up a general master-block named "LISTS" containing the following components: "NLS", an immediate numeric list containing decimal 5, 3, 4, 2, 1; "NAMES", an immediate symbolic list containing the symbols (statement names) JUMP, J2, BLOCK, J2A; "PNTRS", a padded list preset to zero, containing cells named I, J, K, L.

2. Set up a dummy master-block named "MATRIX", containing space for 6 lists.

3. Set up an immediate literal list named "CHARS", containing the characters "NOW$_\wedge$IS$_\wedge$THE$_\wedge$TIME*/.THE$_\wedge$LAZY$_\wedge$DOG*."

4. Set up an edit control string called "ECSX" to do the following: transmit one character; insert the character "Ø"; delete two characters; transmit three characters; stop editing.

5. Set up a match control string named "COMCS" for a compare statement, calling for: matching one character position; skipping two positions; character-category 2 in one position; a word-boundary character in one position; stop matching.

6. Set up a match control string named "LCMCS" for a locate statement, calling for: the literal character "Z"; skip one position; a word-boundary character; the literal character "Ø"; character-category 1; stop matching. Is the "stop" character necessary?

7. Set up a dummy cell-list named "VLIST".

8. Set up a general master-block called "IBUFS", containing three input buffer cell-lists padded to blanks, called "IBUF1, IBUF2, IBUF3", each 20 cells long.

9. Set up definitions for the following cells: a) a cell called "CT" containing zero. b) a cell called "BASE" containing the symbol (address or name) of list "PNTRS". c) a cell called "MINUS" containing the value -75. d) a cell called "OCTCN" containing the octal value 777.

10. What is the purpose of a "symbolic" immediate cell-list or cell?

Study Questions, pages 17 -- 23.

1. What would happen if, in example ST1 on p. 19, the target operand were VP1 instead of VP? Which exit would be taken?

2. Give the following information for each of statements ST1, ST2, ST3: a) What is in SLX and/or SN? b) What character does TAB point to after the statement is done? c) Which exit is taken?

3. Given the following stream of characters, read into the cell-list BUFR:

^^THIS/SENTENCE^^^(SHOULD)^HAVE^/SINGLE/BLANKS,),BETWEEN^WORDS.*

Write a SEG statement called "SCAN" which would allow the words to be edited by a later EXT statement in the way the meaning of the sentence suggests. Choose a variable for the target operand from those used as examples of sentence definitions on p. 13 (for this and all subsequent questions of this type, alphabets, exit routings, result cells, etc. may be supplied as you wish, so long as they obey the rules).

4. What value is in result cell "CT" after each of the examples on page 20 has been executed? Which exit is chosen after each? (SEL)

5. Write a SEL statement named "STSL" to select the fifth, second, first, and third words of S2, in that order, and form from them a sentence in variable VS.

6. Write a SEL statement called "RPTS2" which will form a paragraph variable in VP consisting of three repetitions of sentence S2.

7. a) What would the variables created in VP, ONE VP(1), and VM of examples ST9, ST10, and ST11 look like if they were moved to an output area (converted to textual form if necessary) and printed out with a single blank after each component word?

8. Write a COL statement called "MIXR" that would form a sentence variable in VS from the words of S1 in such a way as to say "A^SENTENCE^IS^HERE^" if moved and printed out with single blanks after words.

9. a) What would happen if an attempt were made to process VP after statement ST12 had been executed? b) How many words of sentence S2 will be left for processing after ST13 has been executed? Which words are left?

10. Given cell-list HITS, as defined on p. 17, but now containing the position numbers of all words in S2 having more than 4 characters, (i.e., position numbers 4, 6, 7), and a cell CT containing the total number of these (3), write an OMIT statement named "OMTX" that will cause just those words to be left out of S2 the next time it is processed. Do you have to know the contents of HITS and CT to write the statement?

11. What would happen if an attempt were made to process BP after statement ST15 had been executed?

12. a) What does TAB point to after statement ST17? b) What does TAB point to after statement ST18? c) What will CELL contain after ST19? d) What will CELL contain after ST2Ø? e) What will CT contain after ST21?

13. What would happen if statement ST2Ø said "ONE S1(5)" instead of "ONE S1(4)"? What if it said "ONE S1(J)" and J contained zero?

14. Write a REF statement called "SETUP" to put a value for the beginning of cell-list BUFR into TAB. When might such a REF statement be necessary in a program?

Study Questions, pages 23 -- 26.

1. What characters are moved, and where are they placed, by each of statements ST22, ST23, ST24, ST25, ST26, ST26A, ST26B?

2. Write an EXT statement named "XTO" to move the words of the variable VS, produced as an answer to question 3 for pages 17 to 23, into an output buffer area OBUFR with a blank after each word. Precede the EXT statement with a REF statement called "GETBG" to set TTAB to the beginning of OBUFR, and use STAB and TTAB as result cells for the EXT statement. What will be in OBUFR after the REF and EXT statements have been executed?

3. Write an EXT statement named "XTS2" to move some of the words of S2 to OBUFR in such a way as to create the sentence: "THIS^SENTENCE^IS^OF^WORDS", with a single blank after each component word, and also before the first word.

4. What characters are moved into BUFR by each of statements ST27, ST28? What character positions do they occupy in BUFR? Is there an error?

5. Set up an edit control string called "ECSB" and write an EDT statement called "EDWD" which uses ECSB to correct the spelling of a source word "*MIST*EAKS" which is the third word of a sentence named "INST", and moves the corrected word "MISTAKES" to OBUFR, followed by a comma.

6. What characters are moved by ST29, ST3Ø, ST31; where are they placed?

7. Write a textual convert statement (CVT) called "CVX" to convert and move the number "1ØØ" in the cell-list "LIST" (p. 16) to the fielded word variable "FW" (p. 14), preceded by leading zeroes to fill out the field. What is the difference in the way the number "1ØØ" looks in the cell of the cell-list "LISTS" and in the field "FW" within cell-list BUFR? (For example, what would they look like if they were displayed or SNAPPED?

Study Questions, pages 26 -- 30.

1. What has to be in variable "VM" in order to prevent an error exit from ST32? What kind of statements might have to be executed before ST32 to ensure that VM contains the right thing?

2. If a value had been assigned to VM by the following definitions and operation statement:

```
*T STOR DEFM/STOR1 STOR2$
*T STOR1 DEFL P Ø 2Ø$
*T STOR2 DEFL P Ø 1Ø$
*T GETM COLLM STOR1 + STOR2 AS VM/CT$
```

which and how many cells would be moved by statement ST32? Which cells would the material be moved into? What would CT say after ST32?

3. What values would be moved by statement ST33? Which target cells would receive the material? What would CT say after ST33?

4. Write a MOVE statement called "ADD5" to add 5 to each of I, J, and K from VCLS and place the results in the first three cells of LIST.

5. What values are moved by ST35, ST36, ST37, ST38, ST39? Where are the values placed? What is in CT?

6. Write a REPT statement called RESET to fill the cells of STREM with blanks.

7. What characters are obtained as sources by statements ST4Ø, ST41, ST42? What target cells are the converted values placed in? What is in CT?

8. What would happen if ST4Ø had "STREM 8 4" as its source?

9. Write a CVN statement called "CVNX" to convert the characters "567" in STREM and store the value in the third cell of "HITS".

10. Write an ENTER statement called "INIT" to place zeroes in N2, the second cell of HITS, CT, and SN.

11. What happens when ST46 and ST47 are executed? (Trace the lookups through each step until the final value is obtained and placed in the target).

Study Questions, pages 31 -- 32.

1. Which and how many characters are compared in ST48, ST49? Which exit is chosen? What is in HITS and CT?

2. The following is a table showing some of the results for various forms of the COM statement with "WORD" as a source and each of the words of "SA" as a comparand; various data type code combinations are shown, and several different match conditions. The numbers in the body of the table

are those values of "J", as J increases from 1 through 5, for which a positive exit would be taken from the statement; in other words, those components of "SA" which would be accepted as hits under the given match conditions. The number "5" circled in the row for condition ID represents the single word "BBCD" which alone would be accepted as a match in statement ST48. Study the table and verify for yourself the results for data types WW, WF, and FW (the character-stream data type, H, does not differ from W in its effects on the outcome of matching).

Match Condition	Meaning of condition	data type codes WW(HH,HW,WH)	WF(HF)	FW(FH)
ID	"WORD" is identical to	(5)	4,5	1,2,5
EL	"WORD" is of same length	5	not used	not used
AFT	"WORD" is after	1,2,3	3	3,4 (1)
BEF	"WORD" is before	4	1,2 (2)	none
AFQ	"WORD" is after or equal to	1,2,3,5	3,4,5 (3)	(5) 1,2,3,4,5
BFQ	"WORD" is before or equal	4,5	1,2,(4),4,5	1,2,5

notes:
1 BBCD$\wedge$ is after BBCDE.
2 BBCD is before BBC$\wedge$.
3 remember the blank after words 1 and 2 of SA.
4 BBCD is equal to the first 4 characters of word 4 of SA.
5 The first 3 characters of "WORD" are equal to word 2 of SA.

3. Write COM statements for some other entries in the table (using pointer J to select a comparand word from sentence SA as in ST48.)

Study Questions, pages 33 -- 36.

1. Which, and how many comparand words or patterns are accepted as hits in each of statements ST51, ST52, ST53, ST54?

2. How many characters of each comparand word are compared to the source in ST51 and ST52?

3. The following table shows the results for various forms of the LOC statement, in the same way as the table given for COM in the previous study question section. Compare the two tables and verify the results in this table to your own satisfaction. The numbers in the body of the table below represent the position numbers of words in the comparand sentence SA that would be accepted as hits for each LOC statement. It should be remembered that LOC matches the source against all the words of SA, and not only one, so that, if "ALL" were requested in the statement, the result list HITS would contain all the position numbers in each box of the table after execution of the statement. The circled value represents the result list contents for statement ST52. Data type codes WH and HH are not included, since they have a special meaning (comparison at all single-character offsets) for LOC.

Match Condition	Meaning of condition	data type codes		
		WS(HS)	WF(HF)	FS
ID	"WORD" equals words of SA	5	4,5	1,2,5
EL	"WORD" same length as words of SA	5	not used	not used
AFT	"WORD" after words of SA	1,2,3	3	3,4
BEF	"WORD" before words of SA	4	1,2	none
AFQ	"WORD" after or equal words of SA	1,2,3,5	3,4,5	1,2,3,4,5
BFQ	"WORD" before or equal words of SA	4,5	1,2,4,5	1,2,5

4. Write LOC statements for some other boxes in the table.

5. Which and how many cells pass the tests in ST55, ST56, ST57, ST58, ST59?

6. Write a FIND statement called "ZROS" to find all zero values in VCLS.

7. Write a FIND statement called "NEG" to find all negative values in the three cells TAB, STAB, TTAB.

8. Write a FIND statement called "OVER3" to find all values greater than 3 in LIST.

Study Questions, pages 36 -- 40.

1. Describe the results of executing statements ST6Ø, ST61, ST62, ST63, ST64, ST65.

2. What would happen if statement ST63 said "OPEN REW RUNLOG"?

3. Let TEXTFL be a file with physical records (blocks) each containing 100 cells, and let statement ST6Ø have been executed three times, so that 60 cells will have been processed in all, the first 20 of each of three successive physical blocks. If statement ST68 is then executed, backspacing a record, which block will the next 20 cells be read from the next time ST6Ø is executed? Where was the tape positioned after the third "READ", before the backspace?

4. What would happen if a TEMAC programmer wrote end-of-file, backspaced a record, backspaced a file, or rewound the RUNLOG?

5. Write a "READ" statement called RDTX to read 10 cells from TEXTFL into the last 1Ø cells of BUFR.

6. Write a "WRITE" statement called OUTPT to put out, as one physical record 25 characters long, the contents of BUFR followed by the contents of STREM to the file MACFIL. Since BUFR and STREM are next to each other in storage, what is another, simpler way the WRITE statement could be written?

Study Questions, pages 40 -- 43.

1. Write a MOVAD statement called "JTEST" to decrement J by -5, and go to STX when the result is greater than or equal to -15. How many times would the statement JTEST be executed, if J starts at zero, before the routing of control will change to something other than STX?

2. Write a MOVAD statement called"FIXTB" to subtract 30 from TTAB and place it in TAB; if the result is negative go to STX.

3. Write an IF statement called "KTEST" to go to STX if K is within the range -4 through +4.

4. In statement ST79, where will control go if J happens to contain 3? -1? 1Ø?

5. In ST8Ø, where will control go if J happens to be 2? -1? Ø? 5?

Study Questions, pages 43 - 55.

1. What statements can be the objects of "DO" statements?

2. What is the result of executing statements ST83, ST84?

3. Describe the sequence of events in the example on p. 47 for EXIT and RETURN.

4. Write FILE cards for the following files: a) INFIL, 200 cells in each block, high density, to be held. b) OUTFL, 20 cells in each block, high density, to be printed.

5. What card should be placed at the beginning of the program to cause a TRACE of every traceable statement? What else has to be done to cause trace output to appear?

6. What statements are not traceable (i.e., produce no trace output on RUNLOG)?

7. Write a DISP statement named "DP" to display the decimal contents of the cell N2, labelled with its name. What will the display output on RUNLOG look like?

8. Write a DISP statement named "DL" to display the decimal contents of the first three cells of LIST. What will the display look like?

9. Write a DISP statement "DI" to display the message "BAD RECORD".

Answers to Study Questions, pages 1 -- 4.

1. P, S, W, M, L, C (Paragraph, sentence, word, master-block, cell-list, cell).

2. An operand is a selection of data from a data object to be acted on in a particular operation statement. A data object is a named constant or variable set up by a definition statement. Operands are used to obtain data by referring to data object names; data objects are set up in order to provide data for operands.

3. Textual data is made up of characters, packed into memory in a continuous stream. It may be structured as a character-stream, a word, a sentence, or a paragraph. Cellular data is any material whatsoever that is arranged in individual 7094 machine registers in binary or machine-internal form. Characters may be treated as cellular data if they are to be moved or acted upon a cell-full (six characters) at a time, instead of as a continuous stream or as text words running over machine-register boundaries.

4. The characters within a TEMAC cell-list which happens to contain text (for example, an input or output buffer area), may only be processed in one of two ways: character by character (in a list-based character-stream operand), or 6 characters at a time (cell by cell), in operands of cellular data manipulation statements. In other words, to get at meaningful words or character groups within the cell-list, the programmer must know that they are there, that they are correct, and that they are located in exactly specified character positions in the cell-list. In a TEMAC textual object like a word or sentence, however, a word may be acted upon as an independent unit, regardless of where it is in the machine memory or how long it is, or what comes before or after it.

5. Paragraphs, sentences: a message made up of teletype print lines, (boundary characters space, carriage return). A message made up of English sentences (boundary characters space, period). A dictionary made up of entries. A format or file description or logical or mathematical expression made up of smaller formulae, expressions, specifications. A group of code-word strings or lookup glossaries. A program made up of statements in some programming language. Words: English words, code-groups, elements of formulae or expressions, (e.g., "A-B", "$500.00", "8ALPHA-2NUM"), descriptors or tags ("XXX123", "AAA"), cue words, endings, stems, abbreviations, etc. for lookups ("ing", "ed", "196-" "JAN.").

6. Master-blocks, cell-lists: a group of lists containing numeric weights, additives, factors, sums, etc. A group of input or output buffer areas. A group of lists containing program symbols (names of data objects or statements). A group of "pointer-sets" for indexing in TEMAC set operands. A group of match or edit control strings. A group of count cells. Cells: a single count, sum, difference, etc. A program symbol. A numeric constant for adding or subtracting. A test threshold or limit. A pointer for indexing. The length of a data object. A "tab" cell containing the cell-address and character position of a TEMAC word.

(Answers, pages 1 -- 4, cont'd.)

7. Textual data must be converted to cellular form when it consists of numeric words (words containing numeric characters only) which are to be used as counts, as pointers, in computations, or stored or picked up one to a cell. If a numeric word is to be used only as part of an output print line, or for matching to other textual words, it need not be converted to cellular form. In matching numeric words in textual form, it should be remembered that all non-significant positions must contain zeroes if the words are to be accepted, e.g., "15Ø" will not match "ØØ15Ø"; both number words must be the same length.

8. Cellular data must be converted to textual form when it is to be printed out, or if it is to be matched against something else which is already in textual form. A cell containing a value "2Ø" developed in the machine as a count cannot be matched against a TEMAC word containing the characters "2Ø" in any TEMAC data testing statement. One or the other must be converted so that both have the same form (usually, it is easier to convert the textual one to cellular form).

9. A TEMAC alphabet may be used to establish useful character-categories for matching; to establish a sorting sequence for matching; and to set up boundary categories for segmenting data into words, sentences, paragraphs or for matching.

11. No, data does not always need to be moved. When the results of selection or test are not to be printed or written out immediately, but just need to be temporarily "remembered" until some later test or selection is applied to them, a structural statement may be used to save their descriptors or tags in a TEMAC variable without moving any data.

12. All the words in a sentence ending in "-ed" have just been found; their position numbers are now known. These words may be selected by a structural statement as a sentence variable named "EDWDS", without moving the words themselves. Then a second test might be applied to "EDWDS", for example, finding all those words that ended in "-ed" and also began with "c-". This smaller group of words might be selected to form a new variable "CEDWDS", etc. Whenever no further tests are to be applied, and the desired set of words for output has been arrived at, they may be physically moved into an output buffer area by referring to the appropriate variable name ("EDWDS" or "CEDWDS") in a source operand of a data moving statement. For many purposes, it is sufficient to leave their position numbers in the "pointer-set" variable or result list produced by the data testing statement that found them, and refer to this set or list in the source operand of a data moving statement.

Answers to Study Questions, pages 5 -- 10.

1. *T in cols. 1 and 2; NAME, VERB, $ at end of statement.

2. The statements would not compile correctly. a) a compile error would be noted on the program listing. b) the data words run together would be compiled as one word, and there would probably also be a compile error message.

3. The statement would compile correctly; any number of blanks, slashes, or parentheses may come between the words of an operation statement; any number of blanks between words of data in a definition statement.

4. The words of data separated only by a slash would be treated as one word with a slash in the middle; a compile error would probably be noted as well. Only blank may be used as a word separator in data.

6. Immediate: for setting up a print heading; a list of constant numeric values for thresholds, lengths, index limits; a string of cue words, endings, match control characters for matching. Dummy: for sentences to be segmented as variables in data read in from tape; for sentences to be selected as a result of tests applied to words of other sentences, or collected from scattered words. Padded: for input or output areas preset to blank; for count cells, pointer cells, pointer lists preset to zero or some constant value. Fielded: for describing the format of an output area containing characters; for describing restricted fields within an input area.

7. A source operand tells where data comes from; a target operand tells where source data is to be moved (or temporarily "remembered"); a comparand tells what data is to be matched to the source data.

8. Data type code(s) for primary operand(s) in the operation statement are on the end of the verb.

9. H (character-stream) and F (word-related field).

10. a) ∧SET∧3∧B(1∧3∧6)∧ b) ∧B(2)∧ (length of B need not be known), or ∧SEQ∧6∧B(2)∧ (B can only be 6 words long) c) ∧ONE∧B(4)∧ or ∧W4∧ d) ∧SEQ∧4∧B(2)∧ e) ∧B∧ (note: also possible are ∧SET∧7∧B(1∧2∧3∧4∧5∧6∧7)∧, SEQ∧7∧B(1)∧, ∧B(1)∧).

11. ∧W4∧5∧7∧ and ∧ONE∧B(4)∧5∧7∧

12. ∧IBUFR∧5∧4∧ and ∧ONE∧BUFFS(2)∧5∧4∧

13. a) The first four letters of "FISHES" ("FISH") will be moved into the target; then the normal exit will be taken, since target is full and the target length is the controlling factor when the source data type is "F". b) The first four letters will be moved into the target, but an error exit will be taken when the attempt to move a fifth letter threatens to cause an overflow, and the source has not been completely moved.

14. a) The first 3 character-pairs will be compared and a hit will be found. The positive exit will be taken. b) Six character-pairs will be compared; the first three will match, but the last three almost certainly will not (we do not know for certain what comes after "EAT" in the data area where it is stored); no hit will be found, and the negative exit will be taken. Care must be exercised with these "field" or incompletely matching operands, since sometimes things will "match" by chance which should not have been hits. c) In some statements, no character-pairs will be matched, and a negative exit will be taken as soon as the discrepancy in lengths between the operands is spotted; in other statements, three character-pairs will be matched for the shorter of the two operands, but as soon as a character cannot be obtained from the shorter one to match a character of the longer one, matching will cease and the negative exit will be taken. In either case, no hit will occur.

15. Impossible operands (not allowed, will cause compiler errors):

^ONE^B(Ø)^ ^ONE^B(-1)^ ^SET^Ø^B(1^2^3)^ ^SET^-1^B(1^2^3)^ ^SET^3^B(Ø^-1^5)^
^SEQ^Ø^B(1)^ ^SEQ^-1^B(1)^ ^B(Ø)^ ^B(-1)^

Unobtainable operands (not possible at a given time, or in a given data object; will cause error exits at run time):

ONE^B(9)^ ^SEQ^4^B(6)^ ^SET^3^B(1^9^5)^ ^B(8)^ ^ONE^B(I)^ (where I contains zero, minus 1, number greater than 7) similarly^B(I)^ ^SEQ^N^B(I)^ (where I is as above or N contains zero, a negative number, or too large a number),^SET^N^B(SET)^ (where SET contains zeroes, negative numbers or excessive numbers)

Answers to Study Questions, pages 11 -- 14.

1. a) *T^ABET1^ ALF^1^Ø2468^^2^ABC^^3^13579^^4^XYZ^^W^,=6Ø^^S^./^^ $
   b) *T^ALPH2^ALF^1^AEIOU^^2^BCDFGHJKL^^3^Ø**9^^W^+-=6Ø^^S^*=53^ $
   c) *T^AB2^ALF^1^ZYXWVUTSRQPONMLKJIHGFEDCBA^^2^9876543210^^^W^=6Ø-^ $

2. *T^PD^DEFP^4/3^5^3^7$

3. *T^PARA^DEFP^4/SA^SB^SC$
   *T^SA^DEFS^D^1Ø$
   *T1^SB^DEFS^F/FA^FB^FC$
   ^OBUFR^3^6^^1Ø^3^^14^5^$
   *T1^SC^DEFS^I^3$
   ^THREE^WORD^SENTENCE$

4. *T^VWORD^DEFW^D$

5. *T^FIELD^DEFW^F^OBUFR^12^6$

6. *T1^CONS^DEFW^I$
   ^YESTERDAY^$

Answers to Study Questions, pages 14 -- 17.

1. *T^LISTS^DEFM/NLS^NAMES^PNTRS$
   *T1^NLS^DEFL^I^NUM^5$
   ^5^3^4^2^1$
   *T1^NAMES^DEFL^I^SYM^4$
   ^JUMP^J2^BLOCK^J2A$
   *T^PNTRS^DEFL^P^Ø^I^J^K^L$

2. *T^MATRIX^DEFM^6$

3. *T1^CHARS^DEFL^I^LIT^32$
   ^NOW^IS^THE^TIME*/.THE^LAZY^DOG*.

4. *T1^ECSX^DEFL^I^ECS^9$
   ^TIØDDTTTE

(Answers, pages 14 -- 17, cont'd)

5. *T1^COMCS^DEFL^I^MCS^6$
^M**2WE

6. *T1^LCMCS^DEFL^I^MCS^8$
^LZ*WLØ1E (the stop character is required for LOCATE)

7. *T^VLIST^DEFL^D$

8. *T^IBUFS^DEFM/IBUF1 IBUF2 IBUF3$
*T^IBUF1^DEFL^P^=BLANK^2Ø$
*T^IBUF2^DEFL^P^=BLANK^2Ø$
*T^IBUF3^DEFL^P^=BLANK^2Ø$

9. a) *T^CT^DEFC^Ø$ b) *T^BASE^DEFC^PNTRS$ c) *T^MINUS^DEFC^-75$
d) *T^OCTCN^DEFC^=777$

10. A symbolic immediate list contains "program symbols"; these are not textual objects, characters, or words like those in a sentence or literal immediate list; instead, the symbols are machine addresses for statements or names of data objects having storage addresses in memory. Such a list is useful for decisions in directing the path of control (in the BRANCH statement, for example). Otherwise, symbolic lists and cells are useful primarily to programmers familiar with IBMAP who wish to write addresses directly into TEMAC calling sequences and system cells for gimmicks and shortcuts. Allowable symbols for such purposes, in addition to TEMAC or MAP names, are such things as: J+2; *-3; **; BASE,1; WRDLO+9; ADDRS,4,DECR; ADDRS,4,19.

Answers to Study Questions, pages 17 -- 23.

1. The error exit would be taken, and control would go to STX, because the first sentence of VP1 has space for only five words, while there are six words in the first sentence in BUFR if segmented as specified.

2. a) ST1: SLX contains 6, 2, Ø, Ø, Ø; SN contains 2.
ST2: SN contains 6, SLX unused. ST3: SN is 2, SLX unused.
b) ST1: TAB points to the character (probably a blank) following the dollar sign. ST2: TAB points to the blank after the asterisk. ST3: TAB points to the second period.
c) Normal exit in all cases.

3. *T^SCAN^SEGLS^BUFR^AB^AS^VS/SN^TAB/THEN^STX$

(When the words of VS are moved to an output cell-list by a data moving statement, they will be edited in the specified way; see question 2, pp. 23--26).

4. ST4: CT is 3, normal exit. ST5: CT is 3, normal exit. ST6: CT is 1, normal exit. ST7: CT is 2, normal. ST8: CT is 1, normal.

5. *T^STSL^SELSS^SET^4^S2(5^2^1^3)^AS^VS/CT$

6. *T^RPTS2^SELPP^SET^3^P1(2^2^2)AS^VP/CT/THEN^STX$

(Answers, pages 17 -- 23, cont'd)

7. a) In VP: HERE^IS^A^SENTENCE^HERE^IS^A^SENTENCE^A^SENTENCE^OF^FIVE^WORDS^
In ONE^VP(1): THIS^IS^DISESTABLISHMENTARIANISM^
In VM: 5^2^1^3^ABCDEFGZØ123456789^.,-/()'--$*^5^2^1^3^

b) ST9: CT is 3. ST1Ø: CT is 3. ST11: CT is 3.

8. *T^MIXR^COLWS^W1^+^W4^+^W2^+^W1^AS^VS/CT$

9. a) An error exit would be taken from any structural, data testing, or data moving statement which attempted to refer to VP or to any of its component sentences or words as a source or comparand. b) Four words are left, namely "A^SENTENCE^OF^WORDS" (numbers 3, 4, 5, and 7).

10. *T^OMTX^OMTS^SET^CT^S2(HITS)$
No, they are variables.

11. An error exit would be taken from any statement referring to VP or to one of its sentences as a source or comparand.

12. a) The first character, "H", in the word "HERE" in S1. b) The last letter "S" of word "WORDS" in S2. c) The address of the first cell in BUFR (IBMAP address BUFR or BUFR+Ø). d) The number 8 (length of word "SENTENCE". e) The number 4 (length of list "NUMS".

13. An error exit would result in both cases; however, since no explicit error exit has been provided in ST2Ø, control would proceed to the next statement in the program; the programmer must test to see if anything was placed in CELL (if there was an error CELL will be zero) or if the DBUG cell contains a non-zero error value.

14. *T^SETUP^REFL^BGN^BUFR^AS^TAB$

This statement might be necessary just before a SEGLS or SEGLP statement, or an EXTSL, EXTWL, EDTWL, CVTCL etc. , to set the TAB cell referred to by the statement to the correct initial value.

Answers to Study Questions, pages 23 -- 26.

1. ST22: the five characters "WORDS" are moved to positions 6 through 10 of BUFR; no error. ST23: the six characters ",SENTE" are moved to positions 5 through 10 of BUFR; no error. ST24: the six characters "^ANTI^" are moved to positions 9 through 14 of BUFR, the last character being moved first, etc.; no error. ST25: The characters "SENTENCE^OF^SEVEN^WORDS^" are moved into the first few cells of BUFR; no error. ST26: The eighteen characters "ANTIDISESTABLISHME" are moved into positions 1 through 18 of BUFR, then an error exit is taken to STX because the target is full while the source is not completely moved. If the instructions at STX process and clear BUFR and then return to ST26, the remaining 10 characters "NTARIANISM" will be moved into positions 1 through 10 of BUFR and a normal exit will be taken. This is the "continuous" option. ST26A: BUFR will contain: ^^A^^^THIS^^^^^^^^^^IS . (characters inserted by the instruction are underlined). ST26B: BUFR will contain ^^^^^^^^^^^^ØØØØØ1234. Four leading zeroes were inserted.

(Answers, pages 23 -- 26, cont'd)

2. *T^GETBG^REFL^BGN^OBUFR^AS^TTAB$
*T^XTO^EXTSL^VS^A^=6Ø^TO^OBUFR/STAB^TTAB$

In OBUFR after XTO: "THIS^SENTENCE^SHOULD^HAVE^SINGLE^BLANKS^BETWEEN^WORDS^"

3. *T^XTS2^EXTSL^SET^4^S2(1^4^2^5^7)^BA^=6Ø^TO^OBUFR/STAB^TTAB/THEN^STX$

(BA does not insert two blanks between words, but merely ensures that a blank will come before the first word and after all words.)

4. ST27: The seven characters "SENØ/77" are moved into whatever positions of BUFR were specified by "TTAB"; "77" is an "invalid" character, which cannot be punched or printed but may be manipulated in memory. ST28: The six characters "77/ØFGZ" are moved into positions 2 through 8 of BUFR. The first edit control character of ECS1 applies to the last character of the operand in "STREM".

5. *T1^ECSB^DEFL^I^MCS^12$
^DTTTTDDTTIET
*T^EDWD^EDTWL^ONE^INST(3)^A^,^TO^OBUFR^BY^ECSB/STAB^TTAB$

6. ST29: The five characters "ØØ1ØØ" are moved into positions 6 through 10 of BUFR. ST3Ø: The eight characters "5,2,1,3," are placed in positions 1 through 8 of BUFR. ST31: Characters in BUFR are as follows:

pos 1 3 6 10 15 23
^^3-- ^^^^1---- ^^.^^^^^^2----

7. *T^CVX^CVTCW^ONE^LIST(5)^LZ^TO^FW/CT^TTAB$

In LISTS, if the cell containing "1ØØ" were "SNAPPED" or "DISPLAYED" it would look as follows:

SNAP: ^-ØØØØØØØØØ144^ (octal)
DISPLAY: ^ØØØØØØØØØ144^

In BUFR, the characters for "1ØØ" in the first two cells, if SNAPPED or DISPLAYED, would look as follows:

SNAP: ^-2Ø6Ø6Ø6ØØØØØ^-ØØØ1ØØØØ6Ø6Ø^ (octal)
DISPLAY: ^6Ø6Ø6Ø6ØØØØØ^ØØØ1ØØØØ6Ø6Ø^

(The underlined characters are the BCD numbers for "ØØ1ØØ". It is assumed that the rest of the BUFR area contains blanks or octal "6Ø". The TTAB cell points to that character in BUFR marked with an arrow on completion of the statement CVX. The difference in form before and after conversion, and the reasons why conversion is necessary, should become clearer with study of this example.

Answers to Study Questions, pages 26 -- 3Ø.

1. A master-block structure having component cell-lists suitable for receiving the cells to be moved must be in VM; the lists must have been selected or collected by a structural statement (SEL, COL) to form VM before ST32 is executed. In other words, a value must have been assigned to the "dummy" variable VM in terms of some other data object which may receive material,

before VM can be used as a target operand in the MOVE statement. If ST32 were executed without any value having been assigned to VM by a previous structural statement, an error exit would result, because VM would be "unobtainable" or "null". What has just been said about VM applies also to the dummy list VL in statement ST34.

2. Twenty cells would be moved from BUFR into twenty cells of STOR1. Then five cells would be moved from STREM into the first five cells of STOR2. CT would say "2", for the two cell-lists moved.

3. The values 1, 2, 3, from cells N3, N2, and N4 of NUMS are each decremented by 50 and the results (-49, -48, -47) would be placed in the last three cells of LIST, where they would cover up the values already there.

4. *T/ ADD5/MOVLL/SEQ/3/VCLS(1)/+/5/TO/SEQ/3/LIST(1)/CT$

5. ST35: The values -2, 5, octal 777, and zero are placed in the first four cells of STREM and the four cells of JUMPS, destroying their previous contents (for purposes of this example only). CT is 2, for lists moved. ST36: The values 2, 5, 1 from NUMS are moved into the first three cells of STREM and of JUMPS; CT is 2. ST37: The 20 cells of BUFR are set to "blank"; CT is 20. ST38: The 5 cells of HITS are set to zero; CT is 5. ST39: The value 2 from N2 is moved into the first, third, and fifth cells of LIST; CT is 3.

6. *T/RESET/REPTCL/=BLANK/TO/STREM/CT/THEN/STY$

7. ST40: The four characters "1234" (positions 10 through 13 of STREM) are converted to cellular form and moved to cell "CELL"; CT is 1. ST41: The characters "20" , "64", and "200" from NSENT are converted and moved into the first three cells of NUMS; CT is 3. ST42: The characters "200" are converted and moved to "CELL"; CT is 1.

8. An error exit would be taken from statement ST40, because a non-numeric character (z) was included in the source. The source tab cell contains a value pointing to the non-numeric character.

9. *T/CVNX/CVNHC/STREM/14/3/TO/ONE/HITS(3)/CT/STAB/THEN/STY$

10. *T/INIT/ENTER/0/,/,/,/INTO/N2/ONE/HITS(2)/CT/SN$

11. ST46: The value 1 is obtained by looking up 2 on LA; the value 1 is obtained again by looking up 1 on LB; the value 4 is obtained by looking up 1 on LC; 4 is placed in CELL; control proceeds to STY. ST47: The value 3 is obtained by looking up 3 (in N4) on LC; the value 4 is obtained by looking up 3 in LB (which is ONE M2(2)); the 4 is stored in K, which is ONE VCLS(3); control proceeds to the next statement after ST47.

Answers to Study Questions, pages 31 -- 32.

1. ST48: One character-pair is tested - the first B of "BBCD" in WORD against the single B in the first word of SA. Matching stops then because there are no more characters in the first word of SA to pair with the remaining characters of WORD. The negative exit is chosen, even though the single pair tested was a match, because the data type codes on the verb (WW) require that the entire source word should match the entire comparand word as a whole. HITS and CT contain zero (since no character position actually tested failed to match.) ST49: One character-pair is tested, as in ST48. This time, however, the positive exit is chosen, because the data type codes "FW" on the verb require only that as many characters of the source be tested as are needed to pair with characters of the comparand, and the single pair called for by that requirement was a match. HITS and CT are zero.

3. *T/CX/COMWF/WORD/ID/IN/ONE/SA(J)/HITS/CT/YES/STY$

*T/CY/COMWW/WORD/BFQ/AB/IN/ONE/SA(J)/CT/YES/STY$

*T/CZ/COMWW/WORD/EL/IN/ONE/SA(J)/CT/YES/STY$

Answers to Study Questions, pages 33 -- 36.

1. ST51: All three endings in "ENDS" match portions of the ending of "WORD2"; HITS contains positions "1, 2, 3"; CT contains 3. ST52: Only the fifth word of SA matches "WORD"; HITS contains 5, CT is 1. ST53: No match is found, since the source does not end with numbers (category 2 of alphabet AB); CELL contains zero. ST54: The second pattern, PAT1, is a match, since it calls for two numbers, a "Z", two letters, and a word-boundary; CELL is 2.

2. ST51: seven character-positions are matched for the first ending, four for the second, and two for the last. ST52: no character-pairs are matched for any word of SA except the last, for which four pairs are tested. In LOCATE, if the data type codes are "WS", a word-pair is rejected immediately unless the lengths of the words are the same.

4. *T/LX/LOCFS/ALL/WORD/AFT/AB/IN/SA/HITS/CT/YES/STY$

*T/LY/LOCWF/ALL/WORD/ID/IN/SA/HITS/CT/YES/STY$

*T/LZ/LOCWW/ALL/WORD/EL/IN/SA/HITS/CT/YES/STY/ELSE/STZ$

5. ST55: One cell, N1, is chosen; the position number 1 is placed in "CELL". ST56: Only the first cell of LIST passes the test; HITS contains the number 1; CT is also 1. ST57: The last three cells of NUMS pass the test; HITS contains 2, 3, 4; CT is 3. ST58: The second and third cells of NUMS pass the test; HITS contains 2, 3; CT is 2. ST59: All but the third cell pass the test; HITS contains 1, 2, 4, 5; CT is 4.

6. *T/ZROS/FINDCL/ALL/ZRO/IN/VCLS/HITS/CT/YES/STY$

7. *T/NEG/FINDCL/ALL/MNS/IN/SEQ/3/VCLS(4)/HITS/CT/YES/STY/ELSE/STZ$

8. *T/OVER3/FINDCL/ALL/GRT/3/IN/LIST/HITS/CT/YES/STY/ELSE/STZ$

Answers to Study Questions, pages 36 -- 40.

1. ST60: 20 cells of data are read from the file TEXTFL into the 20 cells of BUFR. ST61: 20 cells of data are written as one physical record onto the file PRNTFL. ST62: The file TEXTFL is made ready for reading and writing; the tape reel is rewound to its beginning, IOCS buffers are made ready and the first data is brought into memory from an input tape to be picked up and moved to the programmer's buffers by a "read" statement later. ST63: The RUNLOG is made ready to be written on by the program; the tape reel is not rewound. ST64: Processing of the file TEXTFL is terminated. If TEXTFL is an output file, an end-of-file is written; the reel is rewound and unloaded from the tape unit in any case. TEXTFL cannot be referred to again in any way by the program after ST64 is executed, and the file may not be re-opened. ST65: Processing of the file MACFIL is terminated; if it is an output file, an end-of-file is written; the tape is rewound, but remains loaded in the columns on the tape unit. If MACFIL were opened again, it could be reread or rewritten from the beginning to the end-of-file.

2. The RUNLOG, system tape "SYSOU1" on unit B1, would be rewound. This would undoubtedly result in the destruction of the stacked output of many other programs already on the tape. The 7094 operators would express their displeasure to the programmer in vehement terms, and he would probably also be unpopular with the other programmers whose runs preceded his that day.

3. The same 20 cells as just read would be read over again from the third block; "BSR" backspaces to the beginning of the 100-cell physical record, not the logical record read by the programmer's READ statement. The third "READ" caused the tape to be positioned at the beginning of the fourth physical block before the backspace record.

4. The same answer given for question 2 apply to question 4. The programmer should never write end-of-file, backspace records or files, or rewind the RUNLOG.

5. *T/RDTX/READ/TEXTFL/TO/BUFR/11/10/EOF/STX/ERR/STY/THEN/STZ$

6. First way: *T/OUTPT/WRITE/MACFIL/FROM/BUFR/1 20/STREM/1/5/THEN/STX$

   Second way: *T/OUTPT/WRITE/MACFIL/FROM/BUFR/1/25/THEN/STX$

Answers to Study Questions, pages 40 -- 43.

1. *T/JTEST/MOVAD/J/-/5/TO/J/GRE/-15/STX$

If J starts at zero, control will go to STX the first three times JTEST is executed (as J becomes -5, -10, -15); when J becomes -20, control proceeds to the next statement after JTEST.

2. *T/FIXTB/MOVAD/TTAB/-/30/TO/TAB/MNS/STX$

3. *T/KTEST/IF/K/RNG/-4/4/STX$

(Answers, pages 40 -- 43, cont'd)

4. If J is 3, control goes to STB; if J is -1, control goes to STA; if J is 10, control goes to STX.

5. If J is 2, control goes to ST2; if J is -1, control will attempt to go to the next to last cell of STREM, which does not contain a jump, and will cause a program error or a machine stop. If J is Ø, control will attempt to go to the last cell of STREM, causing an error or stop. If J is 5, control will attempt to go to the first cell of NUMS, causing a machine stop. When a machine stop occurs during a run it is always an error, and the 7094 operators usually take a dump of memory and note the fact that the program stopped in execution; they should also note the octal location where it stopped.

Answers to Study Questions, pages 43 -- 55.

1. SEG, SEL, COL, OMT, INC, REF, MTY, EXT, EDT, CVT, MOVE, REPT, CVN, LOC, COM, FIND.

2. ST83: The extract action of ST831 is executed; it it was successful, control goes to STY; if there was an error, control goes to STX. ST84: The tests of the LOC statement ST841 are made; if any hit is found, control goes to STX; otherwise control goes to the next statement after the DO statement ST84.

3. Statement "ENDT1" in the main program causes an entry into subroutine "FDDAT" via Box 1. FDDAT contains a test, with two possible outcomes; if the test has a positive exit, control proceeds to statement "FDDT1", which causes a return to the second exit name on statement ENDT1, which is "ENDT2", in the main program. If the test has a negative exit in the subroutine, control proceeds within FDDAT to statement "FDDX", which causes a return to the first exit address on statement ENDT1, which is "S1TX1". Statement S1TX1 continues in the main program, selecting a sentence for further processing. Statement ENDT2, which was reached by a positive exit from subroutine FDDAT, causes an exit via Box 2 to a new subroutine, "WRITR", which selects a sentence variable S2 and exits to a sub-subroutine "LOGR2", via Box 1 (which is now free again). LOGR2 displays the sentence in S2 on the RUNLOG and returns control to subroutine WRITR again. At the end of WRITR, a return is made via Box 2 to the main program, to the single exit address on statement ENDT2, which sends control to a statement "R2" which is not shown in the example.

col.1 4 16

4. a) INFIL,,FILE,,,,,A(1),MOUNT,INPUT,BLK=2ØØ,BCD,MULTIREEL,HIGH,HOLD
   b) OUTFL,,FILE,,,,,B(1),MOUNT,OUTPUT,BLK=2Ø,BCD,MULTIREEL,HIGH,PRINT

5. A *TRACEALL$ card. RUNLOG must have been opened, ENABLT set non-zero.

6. DISP, READ, WRITE, WEF, BSR, BSF, REW, OPEN, CLOSE, GO TO, START, FINISH, DEF, ALF.

7. *T,DP,DISPC,D,LABEL,N2$ Output: { N2 / 2

8. *T,DL,DISPL,D,LIST,1,3$

   Output: ,+ØØØØØ -ØØØØ5 +ØØØ63

(Answers, pages 43 -- 55, cont'd)

9. *T1^DI^DISPI^10$
^BAD^RECORD

Output: ^BAD^RECORD

Alphabetical List of Data Objects in the Text.

Name	Type	Definition
AB	alphabet	*T/AB/ALF/1/A**Z//2/Ø**9//W/=6Ø,/()//S/.*//P/=53//$
BUFR	cell-list	*T/BUFR/DEFL/P/=BLANK/2Ø$ (see also M1)
CELL	cell	*T/CELL/DEFC/Ø$
CMCS	match control string (COM)	*T1/CMCS/DEFL/I/MCS/1Ø$ /M112WW**ME
CT	cell	(see VCLS)
DW	word	*T/DW/DEFW/D$
ECS1	edit control string	*T1/ECS1/DEFL/I/ECS/13$ /TTTDIØR/R=77E
ENDS	sentence	*T1/ENDS/DEFS/I/3$ /IONALLY/ALLY/LY$
FLDS	paragraph	*T/FLDS/DEFP/FS1/FS2$ (see also FS1, FS2)
FS1	sentence	*T1/FS1/DEFS/F/3$ /BUFR/3/7//STREM/12/5//A2/9/6$
FS2	sentence	*T1/FS2/DEFS/F/F1/F2/F3$ /BUFR/15/9//6/5//1/3$
FW	word	*T/FW/DEFW/F/BUFR/5/6$
F1,F2,F3	words	(see FS2)
HITS	cell-list	*T/HITS/DEFL/P/Ø/5$
I	cell	(see VCLS)
IW	word	*T1/IW/DEFW/I$ /ANTIDISESTABLISHMENTARIANISM$
J	cell	(see VCLS)
JUMPS	cell-list	*T1/JUMPS/DEFL/I/SYM/J1/J2/J3/J4$ (see M1) /ST1/ST2/ST3/ST4$
J1, J2, J3, J4	cells	(see JUMPS)
K	cell	(see VCLS)

LA	cell-list	*T1^LA^DEFL^I^NUM^4$ (see also M2) ^2^1^4^3^$
LB	cell-list	*T1^LB^DEFL^I^NUM^4$ (see also M2) ^1^3^4^2^$
LC	cell-list	*T1^LC^DEFL^I^NUM^4$ (see also M2) ^4^2^3^1^$
LIST	cell-list	*T1^LIST^DEFL^I^NUM^5$ ^Ø^-5^=77^=BLANK^1ØØ$
LMCS	match control string (LOC)	*T1^LMCS^DEFL^I^MCS^15$ ^11LAL,**WL=77SE
M1	master-block	*T^M1^DEFM/BUFR^STREM^JUMPS$ (see BUFR, STREM, JUMPS)
M2	master-block	*T^M2^DEFM/LA^LB^LC$ (see LA, LB, LC)
NB	cell	*T^NB^DEFC^=BLANK$
NC	cell	*T^NC^DEFC^-5$
NMW	sentence	*T1^NMW^DEFS^I^2$ ^ABC15Ø^89Z6Ø^^$
NS	cell	*T^NS^DEFC^BUFR$
NSENT	sentence	*T1^NSENT^DEFS^I/NS1^NS2^NS3^NS4$ ^15Ø^2Ø^64^2ØØ^^$
NUMS	cell-list	*T1^NUMS^DEFL^I^NUM/N1^N2^N3^N4$ ^5^2^1^3$
N1, N2, N3, N4	cells	(see NUMS)
PATRN	master-block	*T^PATRN^DEFM/PAT1^PAT2$ (see PAT1, PAT2)
PAT1	match control string (LOC)	*T1^PAT1^DEFL^I^MCS^8$ (see also PATRN) ^222LZ1WE
PAT2	" "	*T1^PAT2^DEFL^I^MCS^8$ " " ^22LZ11WE
P1	paragraph	*T^P1^DEFP/S1^S2^S3$ (see S1, S2, S3)
SA	sentence	*T1^SA^DEFS^I^5$ ^B^BBC^BBB^BBCDE^BBCD^$
SLS	cell	(see VCLS)
SLX	cell-list	*T^SLX^DEFL^P^Ø^5$

SN, STAB	cells	(see VCLS)
STREM	cell-list	*T1^STREM^DEFL^I^LIT^30$ (see also M1) ^ABCDEFGZ0123456789^.,^/()'+-$*
SVAR	sentence	*T^SVAR^DEFS^D/WV1^WV2^WV3$
S1	sentence	*T1^S1^DEFS^I/W1^W2^W3^W4$ (see also P1) ^HERE^IS^A^SENTENCE^$
S2	sentence	*T1^S2^DEFS^I^7$ (see also P1) ^THIS^IS^A^SENTENCE^OF^SEVEN^WORDS$
S3	sentence	*T1^S3^DEFS^I^5$ (see also P1) ^A^SENTENCE^^WITH^FIVE^^^WORDS^$
TAB,TTAB	cells	(see VCLS)
VCLS	cell-list	*T^VCLS^DEFL^P^0/I^J^K^TAB^STAB^TTAB^CT^SLS^SN$
VL	cell-list	*T^VL^DEFL^D$
VM	master-block	*T^VM^DEFM^3$
VP	paragraph	*T^VP^DEFP^3^7$
VP1	paragraph	*T^VP1^DEFP^3/5^4^3$
VS	sentence	*T^VS^DEFS^D^10$
WORD	word	*T1^WORD^DEFW^I$ ^BBCD^$
WORD2	word	*T1^WORD2^DEFW^I$ ^INTENTIONALLY$
WV1,WV2, WV3	words	(see SVAR)
W1,W2, W3,W4	words	(see S1)

List of Operation Statements in the Text, Alphabetical by Verb.

```
BRAN    *T^ST79^BRANI^ON^J^LSE^1/4/7/8^TO^STA/STB/STC/STD/ELSE^STX$
        *T^ST8Ø^BRANL^ON^J^TO^JUMPS$
        *T^ST8ØA^BRANL^ON^I^+^J^TO^JUMPS$
        *T^ST8ØB^BRANL^ON^,^+^2^TO^JUMPS$
        *T^ST8ØC^BRANI^ON^CELL^-^=77^EQU^=76/=75/=74^TO^STA/STB/STC$

BSF     *T^ST69^BSF^2^TEXTFL$

BSR     *T^ST68^BSR^TEXTFL/EOF^STX$

CLOSE   *T^ST64^CLOSE^REWUNL^TEXTFL$
        *T^ST65^CLOSE^REWEOF^MACFIL/THEN^STX$

COL     *T^ST9^COLSP^S1^+^S1^+^ONE^P1(3)^AS^VP/CT/THEN^STX$
        *T^ST1Ø^COLWS^ONE^S2(L)^+^W2^+^IW^AS^ONE^VP(1)/CT$
        *T^ST11^COLLM^NUMS^+^ONE^M1(2)^+^NUMS^AS^VM/CT$

COM     *T^ST48^COMWW^WORD^ID^IN^ONE^SA(J)/HITS^CT/YES^STY^ELSE^STZ$
        *T^ST49^COMFW^WORD^IT^1^IN^ONE^SA(J)/HITS^CT/YES^STY^ELSE^STZ$

CVN     *T^ST4Ø^CVNHC^STREM^1Ø^4^TO^CELL/CT^STAB/THEN^STY$
        *T^ST41^CVNSL^NSENT(2)^TO^NUMS/CT^STAB$
        *T^ST42^CVNWC^NS4^TO^CELL/CT^STAB$

CVT     *T^ST29^CVTCW^ONE^LIST(N1)^LZ^TO^F2/CT^TTAB/ERR^STX^THEN^STY$
        *T^ST3Ø^CVTLH^NUMS^A^,^TO^BUFR^1^12/CT^TTAB/THEN^STY$
        *T^ST31^CVTLS^SEQ^N4^NUMS(2)^LB^TO^FS2/CT^TTAB/THEN^STX$

DISP    *T^DSC1^DISPC^8^LABEL^BUFR$
        *T^DSC2^DISPC^D^LABEL^ONE^VCLS(J)/THEN^STX$
        *T^DSC3^DISPC^H^STREM+2$
        *T^DSC4^DISPC^H^LABEL^ONE^STREM(3)$
        *T1^DSI1^DISPI^21/THEN^STX$
        ^END^OF^FILE^ON^INPUT.
        *T1^DSI2^DISPI^16$
        ^TAPE^READ^ERROR.
        *T^DSL1^DISPL^D^VCLS^1^3/THEN^STX$
        *T^DSL2^DISPL^D^VCLS$
        *T^DSL3^DISPL^8^LIST^3^3$
        *T^DSL4^DISPL^H^BUFR$

DO      *T^ST83^DO^ST831/ERR^STX^THEN^STY$
        *T^ST84^DO^ST841/YES^STY$
           col 8      col 16
DUMP    ^^^^^^^CALL^^^^DUMP(NAME1,NAME2,Ø)

EDT     *T^ST27^EDTWL^W4^A^=6Ø^TO^BUFR^BY^ECS1/STAB^TTAB/THEN^STY$
        *T^ST28^EDTHH^STREM^1^8^RIGHT^TO^BUFR^1^8^BY^ECS1/STAB^TTAB$
```

```
ENTER    *T^ST43^ENTER^Ø/,/,,^INTO^I/J/K/THEN^STY$
         *T^ST44^ENTER^1/Ø/-5^INTO^J/ONE^NUMS(3)/CELL$
         *T^ST45A^ENTER^Ø/,/,/1/,/,^INTO^A/B/C/D/E/F$
         *T^ST45B^ENTER^ONE^NUMS(I)/ONE LIST(,)^INTO^ONE^HITS(,)/ONE^VCLS(,)$

EXT      *T^ST22^EXTWW^ONE^S2(7)^TO^F2/STAB^TTAB/THEN^STY$
         *T^ST23^EXTFW^W4^B',^TO^FW/STAB^TTAB$
         *T^ST24^EXTHH^IW^1^4^BA^=6Ø^RIGHT^TO^BUFR^5^1Ø/STAB^TTAB/ERR^STX$
         *T^ST25^EXTSL^S2(4)^A^=6Ø^TO^BUFR/STAB^TTAB$
         *T^ST26^EXTWH^IW^C^TO^BUFR^1^18/STAB^TTAB/ERR^STX^THEN^STY$
         *T^ST26A^EXTSS^SEQ^3^S2(1)^LB^TO^FS2/STAB^TTAB$
         *T^ST26B^EXTHW^STREM^9^5^LZ^TO^ONE^FS2(1)/STAB^TTAB$

EXIT     *T^ENDT1^EXIT^TO^FDDAT^ON^BOX^1/THEN^S1TX1^ENDT2$
         *T^ENDT2^EXIT^TO^WRITR^ON^BOX^2/THEN^R2$
         *T^WRTR1^EXIT^TO^LOGR2^ON^BOX^1$

FIND     *T^ST55^FINDCL^GST^IN^NUMS/CELL$
         *T^ST56^FINDCL^ALL^ZRO^IN^LIST/HITS^CT/YES^STY^ELSE^STZ$
         *T^ST57^FINDCL^ALL^RNG^1^3^IN^NUMS/HITS^CT/YES^STY$
         *T^ST58^FINDCL^ALL^LES^N4^IN^NUMS/HITS^CT/YES^STY$
         *T^ST59^FINDCL^ALL^NEQ^=77^IN^LIST/HITS^CT/YES^STY$

FINISH   *T^ST82^FINISH$

GOTO     *T^ST81^GOTO^STX$

IF       *T^ST76^IF^I^ZRO/STX$
         *T^ST77^IF^J^-^2^MNS/STX^ELSE^STY$
         *T^ST78^IF^I^-^J^GRT^K/STX$
         *T^ST78A^IF^,^.D^6^ZRO/STX$
         *T^ST78B^IF^ONE^VCLS(CELL)^.^ONE NUMS(,)^+^2^EQU^ONE^HITS(,)/STX$

INC      *T^ST14A^INCS^SET^3^S2(1^2^6)$

LKP      *T^ST46^LKP^ONE^LB(4)^ON^LA/LB/LC^TO^CELL/THEN^STX$
         *T^ST47^LKP^N4^ON^LC/LB^TO^ONE^VCLS(3)$
         *T^ST47A^LKP^N4^-^2^ON^LC/LB^TO^CELL$

LOC      *T^ST51^LOCFS^ALL^WORD2^ID^RIGHT^IN^ENDS/HITS^CT/YES^STY^ELSE^STX$
         *T^ST52^LOCWS^ALL^WORD^ID^IN^SA/HITS^CT/YES^STY$
         *T^ST53^LOCWM^ONE^NMW(2)^EC^AB^RIGHT^IN^PATRN/CELL/YES^STY$
         *T^ST54^LOCWM^ONE^NMW(2)^EC^AB^IN^PATRN/CELL/YES^STY$

MOVAD    *T^ST73^MOVAD^I^+^J^TO^K/THEN^STX$
         *T^ST74^MOVAD^J^-^1^TO^ONE^VCLS(1)/RNG^-2^3^STX$
         *T^ST75^MOVAD^ONE^VCLS(3)^+^5^TO^I/GRE^1Ø^STX^ELSE^STY$
         *T^ST75A^MOVAD^-J^.D^6^+^K^-^ONE^HITS(3)^TO^I$
         *T^ST75B^MOVAD^ONE^HITS(I)^.^5^+^ONE^NUMS(,)^TO^ONE^VCLS(,)$

MOVE     *T^ST32^MOVEMM^SEQ^2^M1(1)^TO^VM/CT/ERR^STX^THEN^STY$
         *T^ST33^MOVELL^SET^3^NUMS(3^2^4)^-^5Ø^TO^LIST(N4)/CT/THEN^STY$
         *T^ST34^MOVELL^ONE^M1(3)^TO^VL/CT$
```

```
MTY       *T^ST15^MTYP^VP/ERR^STX^THEN^STY$
          *T^ST16^MTYS^S1/ERR^STX$

OMT       *T^ST12^OMTP^VP/THEN^STX$
          *T^ST13^OMTS^SET^3^S2(1^2^6)$
          *T^ST14^OMTW^ONE^S3(1)/ERR^STX$

OPEN      *T^ST62^OPEN^REW^TEXTFL^/^THEN^STX$
          *T^ST63^OPEN^NOREW^RUNLOG$

READ      *T^ST6Ø^READ^TEXTFL^TO^BUFR^1^2Ø/EOF^STX^ERR^STY^THEN^STZ$
          *T^ST6ØA^READ^INFIL^TO^BUFR^1^14/EOF^ST6ØX^ERR^ST6ØY^THEN^STZ$
          *T^ST6ØB^READ^TEXTFL^TO^BUFR^1^1Ø/STREM^1^5/HITS^1^5/EOF^STX^ERR^STY$

REF       *T^ST17^REFP^BGN^P1^AS^TAB/THEN^STY$
          *T^ST18^REFS^END^S2^AS^TAB/ERR^STX$
          *T^ST19^REFM^BGN^M1^AS^CELL$
          *T^ST2Ø^REFW^LTH^ONE^S1(4)^AS^CELL$
          *T^ST21^REFL^LTH^NUMS^AS^CT/THEN^STY$

REPT      *T^ST35^REPTLM^-2^5^=777^Ø^TO^M1(2)/CT/THEN^STY$
          *T^ST36^REPTLM^SET^3^NUMS(2^1^3)^TO^SEQ^2^M1(2)/CT$
          *T^ST37^REPTCL^=BLANK^TO^BUFR/CT$
          *T^ST38^REPTCL^Ø^TO^HITS/CT$
          *T^ST39^REPTCL^N2^TO^SET^3^LIST(1^3^5)/CT/THEN^STY$

RETURN    *T^FDDX^RETURN^TO^BOX^1/THEN^EXIT^1$
          *T^FDDT1^RETURN^TO^BOX^1/THEN^EXIT^2$
          *T^WRTR6^RETURN^TO^BOX2$
          *T^LOGR3^RETURN^TO^BOX^1$

REW       *T^ST66^REW^TEXTFL/THEN^STX$

SEG       *T^ST1^SEGLP^BUFR^AB^AS^VP/SLX^SN^TAB/ERR^STX^THEN^STY$
          *T^ST2^SEGLS^BUFR^AB^AS^VS/SN^TAB/THEN^STY$
          *T^ST3^SEGHS^BUFR^38^2Ø^AB^AS^VS/SN^TAB$

SNAP          TSX SNAP,4
              MON RUNLOG,,2
              PZE PARLO,,LITRL-2Ø

START     *T^TXEDT^START^BGIN$

TRACE     *TRACE$
          *TRACEALL$

WEF       *T^ST67^WEF^PRNTFL/EOT^STX^THEN^STY$

WRITE     *T^ST61^WRITE^PRNTFL^FROM^BUFR^1^2Ø/THEN^STY$
          *T^ST61A^WRITE^PRNTFL^FROM^LIST^1^5/BUFR^11^1Ø/HITS^1^5$
```