An experimental compiler-compiler system

James Robert Van Doren

Iowa State University

Follow this and additional works at: http://lib.dr.iastate.edu/rtd

Part of the Computer Sciences Commons

Recommended Citation
http://lib.dr.iastate.edu/rtd/4516
VAN DOREN, James Robert, 1939-
AN EXPERIMENTAL COMPILER-COMPILER SYSTEM.
Iowa State University, Ph.D., 1971
Computer Science

University Microfilms, A XEROX Company, Ann Arbor, Michigan
An experimental compiler-compiler system

by

James Robert Van Doren

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of
The Requirements for the Degree of
DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Of Science and Technology
Ames, Iowa

1971
PLEASE NOTE:

Some Pages have indistinct print. Filmed as received.

UNIVERSITY MICROFILMS
TABLE OF CONTENTS

I. INTRODUCTION 1

II. REVIEW OF THE LITERATURE 4

III. THE METAX METALANGUAGES 14
   A. Developmental Rationale 14
   B. Bootstrapping 16
   C. The METAX9 Metalanguage 18
      1. Compiler definition commands 18
      2. Elementary syntactical commands 20
      3. Metasyntactic elements 26
      4. Semantic commands 27
      5. Output operators 35
      6. Error processing 37
   D. The METAX9 Pseudo-Machine 41
      1. Primitive operations 41
      2. Control stack 46
      3. Symbol table structure 47
      4. Addressing structure 51
      5. Block list 51
      6. General comments 52
   E. METAX8 and METAX9 Pseudo-Machine Differences 52
   F. Chronological Development 54

IV. THE METAX SYSTEM ORGANIZATION 57
   A. Control Program 57
   B. Control Record Analyzer 59
   C. Communications Region 60
   D. Main Storage Usage 62

V. PLEX: THE LANGUAGE AND ITS TRANSLATOR 64
   A. The PLEX Language 64
iiii

B. The PLEX Pseudo-Machine
   71
C. The PLEX Translator
   87

VI. CONCLUSIONS AND SUGGESTED FURTHER WORK
   95

VII. BIBLIOGRAPHY
   99

VIII. ACKNOWLEDGEMENTS
   104

IX. APPENDIX A
   105
X. APPENDIX B
   113
XI. APPENDIX C
    220
XII. APPENDIX D
     317
XIII. APPENDIX E
      325
I. INTRODUCTION

This dissertation presents the investigation of a number of factors in the development of an experimental compiler-compiler system capable of accommodating the implementation of reasonably sophisticated high level programming languages such as ALGOL (41,42) and PL/I (29) as well as implementation of compiler writing languages or metalanguages. The primary results include a compiler writing language with which a translator for a language similar to PL/I was implemented and a metalanguage version in which compiler writing languages may be written. Perhaps a minor result is the overall system organization. However, certain aspects of this organization, particularly the job control scheme, materially aided in the development of the principal results.

We refer frequently to the METAX (META experimental) system due to the influence of the META series of translator writing systems (43,44,45,52,54) and we also refer to the compiler-compiler system because compilers for compiling a compiler written in a metalanguage are included. It should be noted that "metalanguage" and "compiler writing language" are used almost synonymously throughout this dissertation although it is certainly true that one may metalinguistically describe a language without necessarily implying anything about a computer program. Additionally, "semantic" may be
used in a manner meaning postsyntactic, that is, not syntactic.

With reference to the functions of the various object languages and their interpreters, "primitive" and "pseudo-machine instruction" are also used synonymously.

In Chapter II we review the literature with emphasis on translator writing systems having the most influence on the one presented here.

Chapter III contains a brief discussion of a number of practical matters having an influence on the course of development. The culmination of this development is the METAX9 metalanguage with which the PLEX translator was written, PLEX being the PL/I-like language mentioned. A review of METAX9 and its object code interpreter is presented.

The overall system organization is presented in Chapter IV including descriptions of the supporting assembler language programs and the control record analyzer, the processor for which was written in one version of the metalanguage.

Chapter V comprises a discussion of a PL/I-like language, PLEX, the object language interpreter and the implementation of its compiler which illustrates most of the features of METAX9. Run-time storage administration represents an important aspect of the implementation and is considered extensively.

In Chapter VI we review the results, present some conclusions and suggest some areas for future investigation.
It should be noted that the appendices comprise an essential segment of this dissertation as the syntax and semantics for PLEX and METAX9 are displayed. Additionally the object code interpreters are presented which may be used to resolve fine points and to give more detail about certain aspects of the implementations. Three PLEX programs are included, two of which at least partially demonstrate the viability of PLEX, the third being an error diagnostic example.

One of the PLEX programs, program ESYLST, was used to prepare the assembler language listings and symbol definition indices found in Appendices B, C and D.

Finally, some hardware and software characteristics of the host computer system, an H-1200, are given in Appendix E for the benefit of readers not familiar with that system.
II. REVIEW OF THE LITERATURE

This chapter begins with a review of the META series of translator writing systems as the compiler-compiler system reported in this dissertation owes much to this particular series.

Schorre (53) and Metcalfe (39) first reported syntax directed compiling schemes which contained the basic ideas leading to the META series. Metcalfe (39) presented a translation machine with primitives strikingly similar to the object code primitives of Schorre's META II system (54) and also recognized the potential application of his translation machine in translating directly from a higher order metalanguage to syntax code using a metagrammar.

META II (54) is the first published work on the META series itself. A basic characteristic of the META II system is a top-down syntax analysis without backup directed by an encoding language (translator writing language) similar to the Backus Naur Form (BNF), BNF having been originally used in defining the syntax of ALGOL 60 (41,42). Furthermore, the META II language (or metalanguage) enabled the specification of symbolic code and label generation to be output subsequent to successful recognition of specified syntactical constructs.

With this scheme Schorre (54) constructed translators
for two languages he called VALGOL I and VALGOL II based on restricted subsets of ALGOL.

Although this system has many deficiencies such as the lack of a symbol table, no error recovery and the inability to handle anything but a deterministic syntax analysis it presents rather interesting bootstrapping capabilities of interest here.

BNF as it stands has a limitation as a metalanguage in that it is not possible to describe the syntax of BNF in BNF due to metasymbol conflicts. The syntax specifications used in META II resolve this problem by making certain changes in metasymbol usage and thus it is possible to describe the META II language or other metalanguages with META II.

Specifically, terminal symbols in META II are indicated with enclosing delimiters (") and nonterminal symbols are indicated with no enclosing delimiters whatsoever as opposed to the use of the symbols "<" and ">" to delimit nonterminal symbols and the lack of any enclosing syntax for terminal symbols in BNF. Moreover, an iteration operator, "$", is used in place of left recursion. The use of parentheses as meta symbols for denoting metaexpressions is also included in META II.

Thus the representation of a subscript list may be written as

\[
\text{SUBLIST:= EXP \$ ( "," EXP )};
\]
in META II as opposed to

\[ \text{<SUBLIST> := <SUBLIST> , <EXP> | <EXP>;} \]

in BNF.

The fact that specific symbols in the language being described are enclosed in quotes permits description of metasyntactic symbols if that is desired. Thus it is possible to denote the alternation symbol "|" as a syntax specification which cannot be accomplished in BNF because such a symbol in unquoted form would represent a metasymbol and not a syntax specification of the language being described.

It is thus possible to represent the syntax of META II in META II as follows:

```
.SYNTAX PROGRAM

PROGRAM := "SYNTAX" .ID $ ST ".END";

ST := .ID ":=" EX1 ";";

EX1 := EX2 $ ( "|" EX2 );

EX2 := ( EX3 | OUTPUT ) $ (EX3 | OUTPUT );

EX3 := .ID | .STRING | "ID" | "NUMBER" | "STRING"
   | "(" EX1 ")" | "EMPTY" | "$" EX3 ;

OUTPUT := ".OUT" "{ "$ OUT1 "}" | ".LABEL" "{ OUT1 "}";

OUT1 := "+1" | "+2" | "*" | .STRING ;

.END .
```

Certain additional liberties with BNF may be evident in that .ID and .STRING are assumed to be terminal symbols which
will in fact be references to corresponding pseudo-machine interpreter segments for recognition purposes.

Whitney (56) presents other types of modifications to BNF specifically aimed at the syntax of declarations which represents a degree of context sensitivity that BNF cannot represent.

The full translator of META II in META II is not given here. Suffice it to say that Theys (55) presents an extensive and detailed review of META II as well as other members of the META series.

The importance to us here is that given an initial translating facility similar to META II, a scheme for developing successively more powerful translator writing languages is available.

Although not part of the META series Wilkes (57) has published a list processing language which may be used in a self-compiling compiler and may be of further interest in terms of bootstrapping methods.

META-3, reported by Schneider and Johnson (52) was developed for the purpose of generating symbolic code to be assembled on the IBM 7090 as opposed to pseudo-code to be interpretively executed.

META5, reported by Oppenheim and Haggerty (45), was developed at System Development Corporation and has been used primarily for source-to-source language translations and for
data conversion purposes. Presser (46) reports an implementation of META5 at UCLA as well as some of the history of development of META5 and the META series in general.

Schaefer (51), also of System Development Corporation, presents a data base conversion language which was developed from a new META version, META6.

One of the most significant members of the META series is the META PI system developed by O'Neil (43,44) at RCA within the framework of an interactive system. META PI can be more properly classified as a compiler-compiler system as it facilitates a wide class of postsyntactic processing which removes many of the restrictions of the earlier META series, generates actual machine code and has been used to implement interactive FORTRAN and BASIC compilers. Furthermore, extensibility is a significant feature of META PI.

O'Neil's dissertation (44) may be used as a reference manual to that system.

Book et al. (1) present a metacompiler system, CWIC/360, which appears to have extensive multiple pass capability. The system is comprised of three special purpose languages for syntax specifications, object code generation and operating system interfacing.

CWIC/360 appears to be a significant follow on to the earlier members of the META series but unfortunately detailed information does not seem to be available from System De-
velopment Corporation which has begun pursuit of profit making goals.

Additionally a META-type language translator imbedded in an interactive educational environment has been developed by Branstadt (2) for use in studying formal language theory.

The META translator writing systems which have had the most influence on the system reported here are META II (54) and META PI (43,44).

To be certain, there are many other other translator writing systems of significant importance. Not all of these can be covered here but Feldman and Gries (16) and Presser (46) report on many of the significant developments.

Irons (30) developed one of the earliest automatically constructed translators, perhaps founding the notion of syntax directed compiling.

Another early and still very important system is the compiler-compiler system reported by Brooker and Morris (3,4), Brooker et al. (5) and Rosen (50). Perhaps the main emphasis is on semantics in a top-down syntax analysis.

An interesting division of syntax classes is made in which a phrase definition is concerned with syntax specifications while a format class may specify semantic routines to be invoked if the requisite syntax is matched.

Additionally there are provisions for optional syntax and iteration in lieu of left recursion which are found in a
different form in the META series.

We add that the elimination of recursion is dealt with more extensively by Carr and Weiland (8) who describe a scheme for reforming the syntax of Chomsky (10,11) type 3 grammars (regular grammars).

Feldman (15) describes a metalanguage designed specifically for encoding semantic operations. The framework within which it is presented assumes the existence of a semantic loader and a syntax loader for translating the respective semantic and syntax specifications into internal tables to be used by a compiler kernel. His approach also contains facilities for declaring certain compile time data structures.

Although Feldman's semantic language has been called FSL, the Formal Semantic Language, it is becoming known as the Feldman Semantic Language due to the much more formal and theoretical method outlined by Lucas and Walk (36) in using the Vienna Definition Language to formally describe PL/I. Although this definition is being used primarily for internal control of the development of PL/I within IBM it seems destined to a significant role in language definition.

Reynold's COGENT system (48), which has been used largely for problems in symbolic mathematics, contains an interesting approach to syntactical recognition in that alternatives are processed in parallel in a modified top-down
analysis. COGENT is another example of a system written in its own language.

Two systems with a degree of similarity to the META series are McClure's TMG (37) and the GARGOYLE system by Garwick (19).

TMG is again a top-down approach allowing embedded semantic rules and backup. Freiburghouse (18) reports that the MULTICS PL/I compiler was first written in EPL (Early PL/I) which was produced using TMG.

GARGOYLE also is a top-down syntax directed processor. Its uniqueness is perhaps in the scheme used which requires a five entry tabular representation for all syntactic and semantic statements.

Ritland (49) describes an interesting implementation of SOL, a simulation language presented formally by Knuth and McNeely (35). In constructing the translator a BNF-like metalanguage with embedded references to postsyntactic routines was used as input to a processor which then built a set of tables similar to those described by Cheatham and Sattley (9) for a syntax directed compiling scheme. Perhaps the most interesting part of Ritland's work, from the point of view of the metalanguage bootstrapping scheme outlined in Chapter III, is that the metalanguage used for describing SOL was itself described by a set of hand compiled tables which were then used with the same syntax directed compiling scheme to
process the description of SOL to produce the requisite tables. Thus a hand compiled version of a metalanguage was used to create a compiler for another language.

Evans (14) describes an ALGOL implementation using a metalanguage outlined by Floyd (17) which has become known as the Floyd or Floyd-Evans Production Language and forms the basis of many bottom-up translators (16). Although this approach has not had the success of others in terms of automatically constructed recognizers DeRemer (12) presents a scheme for generating bottom-up parsers of the Floyd-Evans type for languages whose syntax can be described in BNF.

Of more recent development is the XPL system of McKeeman et al. (38). XPL is a PL/I-like language which forms the basis of a compiler writing system whose components are written in XPL, including the compiler for XPL which leads to an interesting bootstrapping history. McKeeman et al. (38) also discuss a parser for LR(k) grammars of Knuth (34), the most general grammar for which it has been shown that efficient recognizers can be mechanically built.

Insofar as the PLEX language described in Chapter V has a number of similarities to ALGOL, previous publications on ALGOL translation methods are of importance to us here particularly because of the block structure of the source languages for PLEX and ALGOL.

Higman (23) describes many of the problems introduced by
ALGOL structure and suggests solutions in a multiple pass scheme. Randell and Russell (47) present a one pass ALGOL translator in which they give a solution for identifier resolution in a block structure with a certain symbol table processing scheme which is also hinted at by Gries (20). Although this method as it stands is not suitable for PLEX or PL/I in one pass, a revised scheme which is suitable is described in Chapter V.

Irons and Feurzig (31) give an interpretive solution to the problems caused by jumps out of a block in which dynamic storage has been allocated.

Concerning the PLEX implementation, the schemes outlined by Dijkstra (13), Naur (40), Gries (20), Gries et al. (21) and Randell and Russell (47) for dynamic storage and stacking mechanisms have had an influence.

One of the better references this observer has found on compiler writing in general is the notes of David Gries (20). A full range of subjects from grammars and recognizers to semantic routines, symbol tables, run-time storage administration and general hints to the compiler writer are covered.
III. THE METAX METALANGUAGES

A. Developmental Rationale

The principal goal in mind was to develop a compiler writing system with which to produce a one pass translator for a PL/I-like language. To this end a number of practical matters had to be considered which are discussed here because of the impact on the methods used.

Firstly the system was to be developed on an H-1200 computing system available to the author through his employer, Drake University, during the time of development. Although this system had adequate secondary storage, a relatively modest main storage (64K six bit characters) was available and a suitable high level language for system development was lacking. Perhaps FORTRAN or COBOL could have been used to some degree but the main storage limitations seemed to preclude serious efforts with those languages.

Furthermore, the system was planned as a new effort, utilizing the ideas of many others to be certain, but not necessarily as a modification of an existing operational system. Finally, it was considered that even with the use of very sophisticated compiler writing aids, suitable restrictions would have to be placed on the goal language in order to bring the matter within the range of feasibility.
Faced with the above factors, bootstrapping a workable system with which to further develop itself was of paramount importance. Thus it was that ease of implementation and the bootstrapping capabilities of the META series of translating systems (43, 44, 45, 52, 54) provided a convenient starting point. The success of O'Neil's META-PI system (43, 44) in demonstrating the approach could be extended to include considerable postsyntactic processing was a significant factor.

Furthermore, the implementation of the MULTICS PL/I compiler (18) with a top-down recursive approach suggested the viability of that approach which is a basic characteristic of the META series.

After initial consideration of generating actual machine code, it was decided to generate code for appropriately designed pseudo-machines to be interpretively executed to reduce the level of detail required. It was also thought that sufficiently ideal pseudo-machines would imply smaller main storage requirements although we have no direct evidence to back this up.

Moreover, the goal of implementing a PL/I-like language, albeit with the target machine being a pseudo-machine, suggested the necessity of a degree of postsyntactic processing which implied significant duplication of effort if symbolic code were to be generated which would then have to be assembled.
In view of the goal in mind and the degree of development anticipated at the outset, an intermediate goal of implementing a translator for a relatively simple language such as BASIC (33) was established in order to gain experience and experimental insight. Although this translator is not presented here, its development comprised a valuable step in reaching the main results from the author's point of view.

The translator implementations which are presented have not necessarily been carried forward to logical conclusions in all cases but rather to the point of demonstrating certain capabilities.

B. Bootstrapping

The initial version of the compiler writing language or metalanguage was based to a degree on META II (54) except that symbol table facilities and revised code generation and internal label facilities were incorporated at the outset. This version, called METAX0, was first manually translated to a symbolic form of the code for the pseudo-machine or translation machine and then manually assembled into absolute octal form and punched into card form. The interpreter for the pseudo-code was then written, a simple card loader prepared and an elementary control program written to appropriately control the loading and execution sequence.
The hand compiled version of METAX0, with patches for initial hand translation errors, comprised 491 six bit characters of pseudo-code exclusive of the control stack and symbol table used by the interpreter. In the debugging phase of the initial step METAX0 written in METAX0 was used as source input and the output of this used to compile itself again to determine if the results were the same. The machine compiled version of METAX0 resulted in 512 characters of code, the difference being attributable to certain redundant instructions being left out of the original hand compiled version.

The resulting compiler, or translator, was then used to develop a revised version with additional capabilities, this chain of events continuing through a number of steps which are delineated in the section on chronological development.

We next present the latest version of the metalanguage which was used in implementing the PLEX translator.
C. The METAX9 Metalanguage

METAX9 was developed for the specific purpose of implementing the PLEX translator but not necessarily for the purpose of being able to compile itself and is an extensive revision of the previous metalanguage version, METAX8, in which it is written. In perusing the following discussion it is pertinent to keep in mind that the metalanguage presented contains facilities for describing the source language syntax and the postsyntactic processing to be performed which is in contradistinction to the compiler-compiler scheme outlined by Feldman (15) in which the syntactic and semantic languages are separated. In addition there are certain declarative elements which do not comprise active program constructs but are used to define compile time variables and symbolic equivalents of identifiers which have an effect at metacompile time.

The examples given are taken from the PLEX translator in Appendix B.

1. Compiler definition commands

There are four types of compiler definition commands which may be used.

The definition command specified by the key word .PROG and followed by a procedure identifier must occur at the beginning of a program and nowhere else. It specifies the
first procedure to be executed which corresponds to the goal symbol in a top-down syntax analysis.

Declarations are preceded by the key word .DECLARE followed by a sequence of identifiers with initial values in string or octal form, the length of storage assigned being an implicit function of the initial value. No type codes are attached and these metavariables or compile time variables may be used for various arithmetic or symbolic reasons which are amply illustrated in the PLEX translator. All such declarations must precede any symbolic equivalence or procedure definition commands.

Example: .DECLARE DYNAMP "1008", DYNAMB "0000";

An identifier followed by the key word .OEQU or .IEQU followed by an octal number of an even number of digits or a valid decimal integer, respectively, and optionally followed by a type code comprises the symbolic equivalence capability. Note that there is a distinct difference in function between .DECLARE and .OEQU or .IEQU similar to the distinction between data definition and equivalence pseudo-operation codes in an assembler language.

With .OEQU definitions one can symbolically reference operation codes, either those to be generated by the specified translator or those to be executed directly in the specified translator, as well as fixed addresses particularly
in the communications region.

Example: PUSHLB .OEOU 46 .TYPEO;

Having put this facility in METAX1, the first version up from METAX0, the author found this to be a particularly useful feature.

Recursive procedure statements comprise the last compiler definition category. The left hand side label is a unique identifier naming the procedure and is followed by the character pair ":=" and then by a sequence of semantic and/or syntactic commands which comprise the body of the procedure, a semicolon serving as an end delimiter. A procedure may call itself directly or indirectly but left recursion is not permitted. Left recursion is determined by the first syntactic command regardless of any preceding semantic commands.

Example:

    SUBPART:="(" SUBLIST ")" .OUT(INDXA,*) / .EMPTY;

2. **Elementary syntactical commands**

The elementary syntactical commands comprise tests on the input string in a left to right manner except for certain tests which may be more properly classified as semantic checking commands but are included here due to conditional jumping and error code generation. For tests on the input string leading blanks will be deleted prior to any test being
made. Since these commands are active program constructs they may occur only in the body of a procedure definition, that is, on the right hand side of a procedure statement. The execution of any syntactic command causes an internal true-false indicator to be set true if the test is satisfied or false if it is not.

Contained within the parentheses following the syntactic commands below are the symbolic representations of the pseudo-machine operation codes which may possibly result in the object version of a procedure utilizing the specified command.

"XYZ" (TEST):

This represents a test on the input string for the terminal character string contained within the quote marks. If the string is found the input pointer is advanced. In the object code representation of this test the terminal string is a literal operand immediately following the operation code.

PQR (CLM):

This represents a call upon the procedure named PQR which must be defined as the left hand side of a procedure statement, forward references being permitted. It is expected that the called procedure will cause the internal true-false indicator to be set although it is entirely possible, and sometimes desirable, to have a procedure consisting en-
tirely of nonsyntactical commands. Parameter passing may only be implicit through the use of metavariables or internally generated labels in the control stack.

This command is the BNF counterpart to a reference to a nonterminal syntax category. If it comprises the prefix of a syntactical alternative and if the result is false then the object code representation jumps to the next alternative or returns to the calling procedure. If it comprises a test after the prefix of an alternative has been recognized, then the result will be some form of error action discussed below if a false indicator results.

.LATCH(...) (LATCH):

The .LATCH command represents a departure from conventional syntax representation as it has a controlled effect on backup. The argument is a procedure to be invoked with a backup latch set in case a potential error is encountered in the invoked procedure. The discussion below on error processing itemizes pertinent details about its effect. Suffice it to say that it is useful in resolving ambiguities in an otherwise deterministic syntax analysis.

.EMPTY (SET):

This is the null true test corresponding to the BNF null rule. No tests of any kind are performed and the input pointer is not changed. It may be used as the last of a se-
quence of syntactical alternatives which may be optionally true or any place where one desires to set the true-false indicator.

.ONUM (ONUM):

The input string is tested for a valid octal number which must consist of an even number of digits, two such digits conveniently representing the contents of the six data bits of a storage character on the host machine. If an octal number is found it is converted to binary form in the SYMBOL field of the communications region and the input pointer is advanced.

.INUM (INUM):

This test is similar to .ONUM except that the test is for a valid decimal integer not followed by a decimal point, the test for an exponent not following having been inadvertently left out although easily correctable. A valid result is converted to a 24 bit two's complement integer in SYMBOL, an out of range number resulting in a warning message from within the interpreter rather than the translator.

.PNUM (PNUM):

This test is similar to .INUM except the test is for a valid floating point number. If such is found it is converted to an eight character 48 bit floating binary number.
according to the requirements of the host computer (28).

**.STRING (STRTST):**

The input string is tested for a string; that is to say, the test is for a quote mark followed by one or more characters the last of which must be a quote mark. A single character string of one quote mark is represented by a pair of quote marks. If a valid string is found it is moved to SYMBOL with the surrounding quote marks removed and the input pointer is advanced.

**.ID (ID):**

The test represented by this command is for a valid identifier, that is, a letter followed by an arbitrary sequence of letters or digits. If the test is passed the identifier is moved to SYMBOL and the input pointer is advanced. For input processed by METAX9 object programs the first eight characters of an identifier are saved and for input processed by METAX8 object programs the first six characters are used.

**.TSTTBA (TSTTBA):**

A test is made against the symbol table entry addressed by the last search of the symbol table. Two arguments are required, the first being an octal number pointing to the left most position of the symbol table field within the current entry to be tested (zero origin pointer) and the second
being a compile time variable against which the symbol table field is to be tested. The test is a raw binary comparison suitable for testing character fields and unsigned binary integers, the length of the test being controlled by the second argument. If equality results the true-false flag is set true, otherwise false. It should be noted that it would be an easy matter to extend this test for order relationships if desired.

Example: .TSTTBA(07,PARMCNT)

.TSTTBL (TSTTBL):

The test represented by this command is similar to .TSTTBA except that the second argument is a literal character string or octal number.

Example: .TSTTBL(00,11)

Both .TSTTBL AND .TSTTBA permit a rather wide latitude in testing symbol table entries in that any part of an entry may be examined. One could, for example, easily determine default data types for undeclared identifiers from an entry name by testing the leading character, particularly if these operators were extended to include relational testing. FORTRAN-like IMPLICIT data typing could also be accommodated by using compile time data fields. Providing a mask for bit testing may also be a desirable addition.

.TEST (COMP):
A relational test between two metavariables is indicated by the .TEST command, the relational test being specified by the relational operator separating the two identifiers. Correct relationships can be determined between nonnegative binary integers or character fields but not between signed binary integers.

Example: .TEST(DYNAHB>DYNAHP)

.STKCHK (CHKSYM):

The top of the control stack is compared with the contents of SYMBOL.

3. **Metasyntactic elements**

O'Neil (44) points out that "the metasyntactic elements define the relationship of the ... syntactic elements to each other and also describe the sequencing of control through the syntactic elements."

The "$" element is an iteration operator used in lieu of recursion, particularly left recursion, which is not permitted. For example, in describing the syntax of a subscript list we may write

\[
\text{SUBLIST} := \text{EXP} \, \, \, \, \, _{\$} \, \, \, \, \, (\, \, \, \, \, \text{","} \, \, \, \, \, \text{EXP} \, \, \, \, \, )
\]

as opposed to

\[
\text{SUBLIST} := \text{SUBLIST} \, \, \, \, \, \text{","} \, \, \, \, \, \text{EXP} \, \, \, \, \, / \, \, \, \, \, \text{EXP} \, \, \, \, ;
\]

The above example also partially illustrates the use of parentheses metasyntactically; that is to say, the iteration
operator applies to both syntactic elements enclosed in parentheses. Such metasyntactic expressions may be nested to any desired level with parentheses.

Another example illustrates the use of factoring in describing the syntax of an END statement in PL/I.

ENDING := "END" (.ID/.EMPTY);

The slash is used as an alternation symbol and corresponds to "|" in BNF.

4. **Semantic commands**

`...` (LB1, LB2, EVAL, OUTSYM, RESTORE, OUT):

Using this command directs, according to the output operators enclosed, that object code is to be suffixed to the output code area. The smallest unit of code which may be specified for output is one character or two octal digits. The output operators are covered below in the corresponding section.

Example: .OUT(DYNAM,*1)

`.DO(...)` (Complete instruction set):

The code specified by the enclosed output operators specifies that such code is to be included directly in the compiler being generated. There is a distinct difference in function between `.DO` and `.OUT`. If one discusses the difference in terms of the action the compiler takes when processing these commands then `.OUT` causes the generation of code
that will generate the specified code and .DO specifies directly the code to be generated perhaps in a manner more akin to an assembler language. The output operators which may be used with .DO are a subset of the operators usable with .OUT and are outlined in the section on output operators.

With .DO it is possible to specify code sequences which may not be generated automatically by translation of other commands. For example, the object code of either META9 or METAX8 may contain binary addition or binary multiplication operation codes but the source languages provide no means other than .DO for specifying these operations. Thus .DO(A,ONE,LEVNO) and .DO(M,LEVNO,DYNAMP) specify these respective operations.

.SAV(...) or .SAV (MARK,SAVE):

.SAV(...), which is in effect a combination of .MARK, .OUT and .SAV commands, causes the code specified by the enclosed output operators to be saved in a variable length code stack. .SAV causes the code output since the last .MARK command to be saved in the code stack. If a code marker is not at the top of the control stack then a null operation results. This facility provides a convenient means for reordering code, numerous examples existing in the PLEX translator in Appendix B. The only limit on the amount of code which may be saved or the number of items which may be
in the code stack is the amount of storage space available in the stack area.

**.MARK (MARK):**

.MARK causes the current output pointer to be pushed onto the control stack and appropriately marked to identify the stack element as a code marker. It is intended for use with a subsequent .SAV command.

**.NEWLAB (PUSHLB):**

A new internal label is created and pushed onto the control stack and appropriately marked as a label if this command is issued.

**.DEFLAB(...) (LB1,LB2,ENTLOC):**

The operator "*1", "*2" or "*" may occur as an argument of this command, either of the first two specifying a search of the control stack for the first or second internally generated label respectively and to put this label into SYMBOL. If "*" is used then the desired symbol is already in SYMBOL. Once the proper symbol has been placed in SYMBOL then a full search of the symbol table is made with the respective symbol as a search argument and the current value of the compilation program counter is entered into the address portion and marked relocatable. If the symbol whose address is being defined is not in the table it will be entered. If the address
has been previously defined the new value overlays the old one.

`.STACK(...) (MOVE,STKSYM):

One may specify as arguments to the stacking command any number of metavariables to be pushed onto the control stack marked as symbols, the variables being stacked in the order of appearance. An eight character limit must be observed on the length of items stacked. This command is convenient for saving information in a recursive environment. A side effect is that the contents of the last variable will be in SYMBOL upon completion. `.STACK` is equivalent to a sequence of `.SET` and `.STKSYM` commands.

`.UNSTACK(...) (POP,MOVE):

Of course this command is intended to be used in conjunction with the `.STACK` command, the identifier list being in reverse order from the order of the elements on the stack. Thus `.UNSTACK(ID3,ID2,ID1)` will cause the first, second and third items in the stack, counting from the top, to be placed in ID1, ID2, and ID3 respectively. The stack is of course reduced by the requisite number of elements.

Some caution must be exercised, however, in determining that the top of the stack has the right elements in it. If an attempt is made to `.UNSTACK` a code marker or internally generated label anomalous behavior may occur due to the behavior of the POP primitive. The implementation of this
command, which may be found in the METAX9 translator in Appendix A is a relatively clear and uncluttered example of code reordering.

.SET(...) (MOVE, MOVI):

This command is in effect an assignment command allowing the assignment of a metavariable or literal value to a metavariable. Anomalous behavior may result if the receiving field is shorter than the sending field.

Example: .SET(DYNAMP=DYNAMB)

.BLKENT (BLKENT):

The .BLKENT semantic action command causes a new entry in the internal block list to be constructed. Subsequent block searches of the symbol table will cause new symbols to be entered into the symbol table linked to this block entry. This command as well as the .BLKEXT command are intended to be tied very closely to the block structure of a source language such as PLEX.

.BLKEXT(...) or .BLKEXT (BLKEXT):

The specific purposes of this command are to restore the block list pointers for the surrounding block and, if requested, to perform certain functions with respect to any unresolved symbols remaining in the terminating block. Upon detection of such a symbol the immediately surrounding block
is searched. If it is found to be resolved then certain values are moved to the entry in the terminating block. If the symbol is not found then it is entered into the surrounding block and the chain field is set in the terminating entry to point to the entry in the surrounding block and a parameter is set to mark it is a resolution link for the terminal RESOLVE primitive to use in replacing a pseudo-address in the object code with the actual address after it is defined. A resolution chain may be formed which may extend outward over a number of enclosing blocks. It is this command, or perhaps more specifically its interpretation, which permits proper resolution of label references in a block structure.

A single one character argument is expected for this command, a "1" preventing unresolved symbol chaining and a "0" requesting it, the latter being the default value if not specified.

.CAT(...) (CAT):

This is a concatenation command causing the output code specified by the enclosed output operators to be suffixed to the top entry of the variable length code stack. This is effected by marking the top of the control stack with the current output pointer, restoring the top of the code stack to the output area, executing the specified output operators and then saving all the code back to the code mark.
An example of its use is in the implementation of the DO CASE construction in the PLEX translator.

Concatenation on the left, prefixing, may also be performed but a specific command has not been provided, the implementation of .UNSTACK in the METAX9 translator being an example.

.ENTERL(...) (ENTL):

Both this and the following command are used for entering values into the symbol table. In this case two arguments are expected, the first being an octal number specifying the left most position of the symbol table entry (zero origin pointer) and the second argument being a literal value to be entered. The table entry affected is the result of the last search operation.

Example: .ENTERL(00,01)

.ENTERA(...) (ENTA):

This command is identical to .ENTERL except that the second argument is a metavariable.

Example: .ENTERA(07,DIMCNT)

.SEARCH or .SEARCH(...) (SEARCH):

An explicit search of the symbol table is directed with the entry in SYMBOL being used as a search argument. The search may be a block search or a full table search depending
on the single character literal argument, "1", or "0" respectively, the latter being the default if not specified. In a block search only the block specified by the last .BLKENT command is searched while in a full search the block list is consulted to search enclosing blocks if the symbol is not found in the inner block. In either type of search if the symbol is not found it is entered into the table, in the outer block for a full search and in the most recent inner block for a block search.

Searching of parallel or inactive contained blocks is not permitted and thus multiple identifier use in a block structure is permitted. An ALGOL-like tree structure for blocks and procedures is maintained in the block list which controls the order of searching of the symbol table.

.SEARCH, as well as other commands causing a search of the symbol table, result in the true-false code being set depending on whether the specified symbol is already in the table or whether it must be entered.

.STKSYM (STKSYM):

The symbol contained in SYMBOL is pushed onto the control stack and so marked.

.SCAN(...) (SCAN):

The purpose of this semantic command is to cause a scan of the input string for the enclosed character string, this
being the only semantic command affecting the input pointer. It may be used, for example, in scanning for the terminating "*/" delimiter of a comment or scanning for the end of a statement in error recovery.

Example: .SCAN("*/")

.RETURN (R):
This has the same effect as .DO(R) and may be used to force a return to the invoking procedure prior to a normal return.

5. **Output Operators**

In general one or more output operators may be specified as arguments with the .OUT, .SAV, .CAT and .DO commands. Output operators with the .ERR command are optional.

*1 (LB1,EVAL):
This operator causes the first label in the control stack to be extracted, regardless of position relative to the other elements in the stack, the symbol table to be searched for the label and a four character address from the table to be suffixed to the output code. In the event that the label has not been defined a pseudo-address is extracted.

*2 (LB2,EVAL):
This is identical in function to *1 except that the second label in the control stack is extracted.
**(...) or ** (EVAL):

The current symbol in SYMBOL is used as a symbol table search argument with the value being extracted dictated by two octal parameters which specify the number of characters to be extracted and the left most position within the entry. If no arguments are given a default assumption of (05,00) is made. As with the *1 and *2 operators the resulting value from the search is suffixed to the output code.

Example: **(04,01)

* (OUTSYM):

The contents of SYMBOL are suffixed directly to the output code string, this being a convenient operator for inserting literal values in the code string.

# (RESTORE):

Use of this output operator causes the top of the variable length code stack to be popped and the contents suffixed to the output code string. If the code stack is empty a null operation results.

Identifier, octal number, string (OUT):

Using any one of these as output operators causes a literal value, or symbol table value in the case of an identifier, to be generated as the operand of an OUT operation code. Execution of such code then causes the literal operand
to be appended to the output code. It is an identifier as an output operator in conjunction with the .OEQU definition command which allows symbolic reference to operation codes but literally to have the octal equivalent generated.

These three operators are the only ones which may be used with the .DO or .ERR commands and cause literal inclusion of the specified code in the output code string.

It must be further noted that identifier usage as an output operator is dependent on the operation of the METAX8 pseudo-machine with punctuation marks delimiting symbol table values. Since the PLEX translator is written in METAX9 and METAX9 is written in METAX8, the object representation of METAX9 processes the PLEX translator on the METAX8 pseudo-machine. Thus the definition commands for the operation codes in the PLEX translator are effected with punctuation marked value entries.

To be certain some revisions in the source program for METAX9 would have to be made if the METAX9 translator were to be rewritten in METAX9.

6. **Error processing**

`.CANCEL (CANCEL)`:

If a latched call was made to the procedure containing this command then its effect is to turn off the backup latch, thus preventing backup in the case of a subsequent error. If
a latched call was not made then its effect is that of a null operation.

**.ERRLATCH (MOVI):**

On the surface .ERRLATCH does nothing more than set a certain character in the communications region but it has an important effect on error processing. Whenever a procedure call is made the error latch is stacked with the return address on the control stack and then reset. It is restored upon returning from a procedure. If during syntactical processing an apparent fatal error occurs and if the backup latch is not set then the error message will only be issued if the error latch is set and then processing continues. Otherwise, the error message is left in the output buffer and a return to the calling procedure is forced. Thus a source language facility is available, in conjunction with the .ERR command discussed below, for controlling error recovery at whichever procedure level desired.

Error handling code may only be specified and will only be generated for syntactical tests occurring after the first for a given alternative; that is to say, error action may only be specified after the prefix of the alternative has been found and after any ensuing syntactical tests. If the prefix is not found then a jump to the next alternative or a return to the invoking procedure will occur. Procedure
ALTERN of METAX9 clearly represents the conditions under which error handling code will be generated.

The error command, .ERR, allows specification of an error message and a sequence of operation codes to be executed after that message if certain conditions are met. The message itself occurs as a literal operand of the .ERR command with the first character of the message having an effect on the action taken. If the first character is a "W" then the message is interpreted as a warning message and it is printed with a preceding line marking the position of the input pointer at the point of error detection.

If the message is not a warning message then it will be printed and processing will continue only if the .ERRLATCH command has been issued and only if a previous fatal message, marked by a leading "F", is not pending in the message buffer. With the error latch set and a previous fatal message pending then the current error message is discarded and the pending message is printed. If the error latch is not set then a previous pending message will remain in pending status or the current message will be set as the pending message if there is no other. Note that if a message is not a warning, not marked fatal and the error latch is not set the message will be lost.

Pending fatal messages in the manner discussed provide a method for avoiding multiple error messages.
In terms of the object code generated for error handling, the BM primitive is generated for the .ERR command with the message occurring as a literal operand. If no command is given then BEF is generated which results in a default fatal message if an error has in fact occurred. The printing of this default message is subject to the same conditions as outlined above for error messages specified by the .ERR command.

With either BM or BEF backup will occur if the backup latch is on and no further error processing will occur. The backup activity consists of restoring the input pointer to the position at the time of the latched call which set the backup latch and erasing any generated code from the code string generated since that call as well as forcing a return to the invoking procedure. Other actions which may have taken place are not undone, however.

The error processing scheme presented provides a wide degree of error control at the source language level. No attempts have been made to incorporate error correction methods such as those presented by Hedrick (22).

META PI (43,44) has an .ERR command but the error recovery scheme in an interactive environment for line oriented languages such as FORTRAN is somewhat less demanding than for a block structured language such as PLEX in a batch processing environment where it is considered desirable to be able
to continue compilation but not to lose knowledge of certain program structure already gained. For example, if one detects a fatal error in the middle of a DO group or block and the error recovery scheme exits to a procedure that takes certain standard action and then continues processing by calling for recognition of certain program segments, say statements, but does not make use of the fact that the head of a DO group or block has been processed, then an END statement may cause interesting problems.

In a top-down recursive environment essential information about the structure of the program segment already processed may be contained in the control stack by virtue of the sequence of procedure calls and the respective return addresses which represent the syntactical path followed in reaching the point of the error. It is thus desirable to allow reasonably graceful returns to invoking procedures until a reasonably intelligent recovery attempt can be made without losing essential information of the type described.

D. The METAX9 Pseudo-Machine

1. **Primitive operations**

There are 47 operation codes or primitives available on the METAX9 pseudo-machine. Only selected primitives are reviewed here due to the pertinent discussions with the cor-
responding source language elements and also due to the extensive comments in the assembler language listing of the corresponding interpreter in Appendix B. In some cases the symbolic representation of a primitive differs between the interpreter and the corresponding translator, the name in the interpreter being given in parentheses in that case. The index at the end of the interpreter listing may be used for consulting the interpreter.

**B ABC (BRANCH):**

Jump unconditionally to ABC.

**BT ABC (BRANCHT):**

Jump conditionally to ABC depending on a true setting of the true-false code.

**BF ABC (BRANCHF):**

Jump conditionally depending on a false setting of the true-false code.

**R (RETURN):**

Return to the calling procedure by searching down the stack for the first return address popping the stack the appropriate number of elements. The error latch and instruction counter are restored and the backup latch is reset.

**POP:**
Pop the control stack by one element restoring SYMBOL if the element is marked as a symbol.

**SWAP:**
Exchange the top two elements on the control stack.

**SETP:**
Set the true-false code false.

**MOVE ABC,DEF:**
The contents of ABC are moved to DEF. The transmitting field is delimited on the right with an item mark, the item mark punctuation of the receiving being identical upon completion of the move.

**MOVI ABC, "$ACTIVE " (MOVLIT):**
Move the literal operand to ABC in a manner similar to MOVE.

**A FOUR, DYNAREM:**
Perform a storage to storage binary add with the data fields matched up on their right boundaries and the result placed in the second field. No boundary alignment or data field size considerations are required.

**M FOUR, DYNAMP:**
Perform a storage to storage binary multiply similar to A. If either operand exceeds 24 bits it will be truncated on
the left. The result will also be truncated to a maximum of 24 bits.

EXIT (EXITI):

Set the completion code to fatal and exit to the calling program, the METAX control program.

RESOLVE (RESOLV):

Call the EXIT primitive if the completion code is fatal; otherwise, print certain compilation statistics, scan the object code for unresolved addresses, consulting the appropriate symbol table entry for the address resolution and print messages for any addresses remaining undefined.

If a postlisting has been requested then exit to that routine after performing the above functions. In any case RESOLVE represents a terminal primitive, control returning to the METAX control program upon completion.

It should be recalled that an unresolved address in the object code contains a pseudo-address which is a pointer to the corresponding symbol table entry for RESOLVE to use. Additionally a chain of entries may be consulted because of the block structure permitted in the symbol table and because the translation is essentially one pass.

RESOLVE does represent a "small" second pass but only through the resident object code. Its function could have been assumed by maintaining linked lists of references to
undefined addresses which then could be used to fill in the appropriate spots in the object code upon definition. Randell and Russell (47) describe such a scheme for a one pass ALGOL compiler.

The postlisting mechanism does not necessarily properly represent the block structure of source programs. Variables with similar characteristics in parallel blocks or procedures may have the same dynamic storage address representation. The postlisting scheme outputs the first symbolic name found in the symbol table with the requisite address, and type code for five character addresses. An example occurs in the pseudo-symbolic code for the DOGRP block of program TEST in Appendix C where the variable J is represented by A of the previous parallel block.

It is also possible that statement labels may not appear in the postlisting if an internally generated label refers to the same address first in the symbol table; that is to say, only one label is given even though there may be several. Several examples occur in the postlisting of TEST because of jumps around format code which lead to the beginning of a block.

The relative address, in decimal, is given on the left hand side of each instruction. Execution time diagnostics given by the FLEX pseudo-machine also refer to the relative decimal address for ease of debugging.
2. Control stack

A review of the workings of the control stack is presented here as this structure plays an important role in the execution of programs on the METAX9 pseudo-machine. Several references above related to source language commands and object operation codes have alluded to some of the characteristics of the control stack.

Firstly there are four kinds of elements which may be pushed onto the control stack. An element may be a return address and associated error latch, a symbol, an internally generated label or an output code mark. Each type of entry is appropriately marked.

A stack element consists of a single character type code plus eight characters for information for a total of nine characters. Thus symbols which are pushed onto the stack must not exceed eight characters in length. Entries are always made at the top of the stack but in the case of the LB1 and LB2 primitives retrieval may not be from the top nor do such labels necessarily bear any fixed relationship to return addresses. A label may be extracted and placed into SYMBOL which may be below any number of return addresses.

It is perhaps worth mentioning that this facility is distinctly separate from internal label mechanisms of META II and META PI. With the scheme used here a label may be created on one procedure level and used or defined on another
lower level. Perhaps this provides a useful degree of source language control over the creation, use and value definitions of labels while retaining source language expressive power.

Stacking and popping symbols has perhaps been adequately covered elsewhere but it is worth recalling that during the execution of the POP primitive the top of the stack will be placed into SYMBOL if and only if it has a symbol type code.

The output code marker is pushed onto the stack by a MARK primitive but this will only be used by a SAVE primitive and then only if the marker is on top of the stack.

During the execution of the R primitive the stack is popped until a return address is found, all labels, symbols and code markers above the address being discarded.

The main storage area for the control stack is delimited by the contents of two fields in the communications region, this being discussed in more detail in the next chapter. However, the amount of space available may vary during the operation of the METAX9 pseudo-machine because space for the variable length code stack is taken out of the top of this area, extending downward while the control stack itself extends upward.

3. Symbol table structure

As with the control stack the main storage area available for the symbol table is a function of information in the
communications region.

The presentation here is based on the assumption that the PLEX translator is being executed although it is possible to vary the function of this table due to the latitude permitted in testing, inserting and extracting items of information for a particular entry.

Each entry comprises 20 characters of information divided into eight fields which may be treated individually, collectively or character by character if so desired. Counting from zero the left and right character positions of each field are included in parentheses after the name of each field.

**DTYPE (0-0):**

The data type field is the left most field and consists of one six bit character. Chapter V outlines the codes used in implementing the PLEX compiler.

**LEVEL (1-1):**

This six bit field specifies the dynamic storage level of the address, there being a limit of 63 dynamic storage levels. A zero level denotes a static address.

**ADDR (2-4):**

An eighteen bit field is used to represent addresses relative to a dynamic storage block except in the case of a
static level meaning the address is relative to zero. Addresses entered by the ENTLOC primitive will always be marked relocatable with a word mark in the left most character. An execution of the EVAL primitive encompassing the address field will cause the word mark to be output. A table search for a nonexistent symbol will cause its entry with the address field being set with a pseudo-address which points to the table entry itself. Thus the insertion of such a pseudo-address into the output code stream provides a means of detection and resolution as described under the RESOLVE primitive.

LENGTH (5-6):

The execution time length of a particular data field is represented by this field, there being an implementation defined limit of 4095 characters for the maximum length. There may be more than 4095 characters allocated to an array, the limit applying to individual elements. This field is of most importance with respect to character strings.

DIMCNT (7-7):

The number of dimensions of an array or the number of parameters for a procedure may be recorded in this field. The six bit limit on the field implies an implementation defined limit of 63 dimensions and 63 parameters.

CHAIN (8-10):
This field points to the next entry for the current block while it remains active. A zero chain field implies the last entry for a particular block. Once a block becomes inactive, that is, after execution of the BLKEXT primitive, the chain field may serve the function outlined under ETYPE.

Because of nested blocks or procedures the chain field is essential for linking potentially fragmented entries for a particular block. An example of this occurs in program TEST in Appendix C. The program consists primarily of a series of parallel blocks contained within the main procedure with the name of each block (preceding statement label) belonging to the table entries for the surrounding procedure. Since the symbol table entries for the contained blocks are necessarily completed prior to the block of entries containing the block labels, these labels may not occur in a contiguous fashion.

ETYPE (11-11):

This field serves only one function and that is to serve as a link marker for unresolved label resolution. This means that the chain field points to an entry for a surrounding block which may contain a resolved address or possibly a link to another block.

NAME (12-19):

This is an eight character field containing the identifier or symbol table search argument used in referencing the
table. If the identifier is shorter than eight characters then it is padded on the right with blanks.

4. **Addressing structure**

The addressing structure suggested by the symbol table is for the PLEX pseudo-machine while the addressing structure for the METAX9 pseudo-machine is a simple 18 bit address permitting a one-to-one mapping between host machine addresses and METAX9 pseudo-machine addresses. These addresses may, however, be relocatable prior to loading for execution.

5. **Block list**

A number of previous references have been made to the block list which is maintained for proper referencing of symbol table entries for a particular source language block or procedure. This block list is maintained in the upper end of the area assigned for symbol table storage, expanding down from the top while the symbol table proper expands from the bottom up.

Each entry in the block list comprises a single character surrounding block number and an address of the first entry in the symbol table for the particular block. The block number represents the relative position within the block list of the entry for the immediately surrounding block, this being used for full table searches and for proper restoration to the surrounding block upon execution of the
BLKEXT primitive. The block list represents a tree structure that is searched from bottom-to-top or leaf-to-root during a full table search. The scope of identifiers is thus properly preserved.

6. General comments

The general execution structure of the interpreter for the pseudo-machine is relatively simple. After initialization of the block list, registers and the I/O buffers instruction fetching commences. The operation code is extracted, an address of the proper interpretation routine is computed and then that routine is invoked, addressing and operand extraction being the responsibility of the individual routine.

If an interpretation error does not occur or a terminal primitive is not executed then a return is made to the fetching routine. Register usage and fine detail about certain operations may be obtained from the annotated interpreter listing (MTXINT04) in Appendix B.

E. METAX8 and METAX9 Pseudo-Machine Differences

The METAX9 pseudo-machine is a major revision of the METAX8 pseudo-machine in that seven primitives of the earlier version were dropped and eight new ones were added. For the most part these changes reflect changes in the manner of er-
ror handling and symbol table processing.

The symbol table for METAX8 is a straightforward linear table with a ten character entry consisting of a six character name, a single character type code and a one to three character value field. The value field normally contains one character operation codes or three character addresses. A principal difference in operation occurs with the EVAL primitive. On the METAX8 machine the result of EVAL is a one to three character value depending on punctuation marks in the address field while with the METAX9 machine EVAL requires position and length parameters as described. Thus equivalence definitions on METAX8 with two digit octal numbers result in one character entries and one character evaluations automatically.

We have mentioned earlier that the METAX9 language and corresponding pseudo-machine were constructed for the purpose of implementing the PLEX translator and not necessarily for the purpose of being able to translate its own language. Although this is probably possible the postlisting mechanism of the METAX9 interpreter would have to be completely revised or possibility separated into separate programs which could be called depending on the type of code generated if that option were to be retained.

With the METAX9 interpreter, which is not presented here, the code generated for various pseudo-machines (BASIC,
METAX8, METAX9) and the symbol table structure is sufficiently simple that it is possible to give a reasonably meaningful postlisting without the interpreter having any knowledge of the code being compiled. It is dependent, however, on the type codes given for operation codes which may be generated (.TYPEG and .TYPEB) by the compiled code versus operation codes that may be executed (.TYPEB and .TYPEO), this being required because of the use of literal operands in code generation.

The .TYPE# code is used to prevent an inadvertent match between an absolute address in the communications region and a relocatable address in the object code.

The postlisting of the PLEX translator given in Appendix B was generated by the METAX8 pseudo-machine and is quite different compared to the postlisting given in Appendix C for TEST. On the former the operands for a particular operation may not occur on the same line as the symbolic operation code. If there are multiple operands then they will occur on successive lines.

F. Chronological Development

To give a bit of chronological perspective to the development of the metalanguage a brief review of each stage is presented here.
As has been mentioned METAX0 was the initial bootstrapping version. METAX1 was originally written in METAX0 and added the equivalence definition facility as well as commenting capability.

METAX2 added the ability to set communications region fields specifically for being able to specify a name under which a translated program could be stored in the METAX library. METAX3 added more error code generation although not with the backtracking and/or error latching mechanisms described for METAX9. METAX4 was basically a minor revision with certain syntactical changes made plus the inclusion of the .ERR command for issuing error messages.

The system control record analyzer (MTXCRA) was written in METAX4 and a number of other system components were then changed to take advantage of its capability. With this addition the whole operation became decidedly more automatic. MTXCRA is discussed in Chapter IV.

METAX5 contained some code generation revisions which eliminated some of the redundant code generated by earlier versions. This version was then used to produce a syntax analyzer for BASIC.

METAX6 added the ability to specify a type code to be used for postlisting purposes as well as the .LATCH and .CANCEL commands.

With METAX7 table testing and code reordering facilities
were incorporated and the first versions of the BASIC trans- 
lator were then written.

The ability to declare compile time variables was in- 
cluded in the METAX8 version which was then used for 
implementing the final version of the BASIC translator. Of 
course METAX9 was implemented using this version.

In all cases except the last a version was first 
debugged using the previous version and then revised if ne- 
cessary or desirable and retested by translating its own 
translator.

Of course revisions and additions were made to the cor- 
responding interpreters along the way.
IV. THE METAX SYSTEM ORGANIZATION

A. Control Program

The operation of the METAX system is generally under the control of MTXMCP03, the listing for which is contained in Appendix D. This program as well as other supporting programs referenced but not listed are written in the assembler language of the host computing system (27).

The initial program loaded is MTXMCP02 which contains the resident I/O routines for card input and printer output. Upon initialization little more is performed than opening the requisite files and calling the supervisor to load and execute MTXMCP03 which overlays the part of MTXMCP02 no longer required.

MTXMCP03 then processes an input record which determines the names of the control record analyzer and a METAX translator both of which are to remain resident in main storage. MTXLDR, a supporting assembler program, is then loaded into the transient program area and called to load the two resident programs from the METAX library. Once these functions are complete the main control processing loop commences.

The first activity of the main loop is to call the appropriate interpreter into the transient program area to interpretively execute the control record analyzer which is
usually MTXCRA. The details of its functions are covered in the next section but basically it causes certain address, name and option parameters to be set in the communications region.

Upon return from MTXCRA the name of the METAX program to be executed next, as well as its interpreter name, has been set in the communications region. If it matches the resident METAX program the interpreter is loaded into the transient program area and executed. Otherwise, MTXLDH is called to load the requested METAX program and then the specified interpreter segment is called.

Upon return from interpretive execution of the requested METAX program a number of parameters in the communications region may be tested to determine the next activity. If the completion code is fatal the current job is flushed and the next iteration of the main loop begins. Otherwise, a request for updating the METAX program library is honored, utilizing the assembler program MTXSTR.

Then if the GO option is requested the object program from the just completed program, if in fact a translation was performed, is moved and relocated to the execution area and the specified interpreter is called into the transient program area and executed. Upon return the next iteration of the main control loop commences.

An internal memory clearing routine is executed at sev-
eral places in the main loop to clear certain segments of main storage.

Appendix E contains some information about the manner in which the system supervisor is called to perform program loading from its residence file. The requisite Honeywell publication (24) should be consulted for further detail.

B. Control Record Analyzer

MTXCRA, which is normally loaded as the resident control record analyzer, is a METAX program originally written in the METAX4 version although it may be translated by any of the later versions up through METAX8.

Strictly speaking it is not a translator in that no object code is generated. It processes control records and sets specified and default values in the communications region.

Specifically a control record may specify the METAX program and corresponding interpreter to be executed as well as the main storage area to be utilized for pushdown stacks, symbol table space (dynamic storage space in the case of PLEX object programs), object program execution and code generation. Furthermore, parameters for postlisting, METAX library updating and a "go" option may be set in the case a translation is to be performed.
In all cases default parameters will be set if none is specified.

The program listings for METAX9, PLXCPL and the three PLEX programs in Appendices A, B and C, respectively, are preceded by examples of control record usage.

MTXCRA itself is not listed. Suffice it to say that it consists of elementary syntactical tests and usage of the equivalent of the .SET semantic command.

C. Communications Region

In the discussion below of the several fields of the communications region each field description is preceded by the field name and inclusive storage locations (in decimal) in parentheses. For those fields which are set by MTXCRA an asterisk is also recorded.

**INSTECT (205-209);**

The instruction count listed at the end of METAX program executions is accumulated here.

**GENFLD (213-215) *:**

The beginning location for output code generation is utilized by the respective interpreters.

**LODFLD (217-219) *:**

This field specifies the beginning location for loading
object programs for execution and is used by MTXLDR and the
control program for loading purposes.

**STCKP1 (221-223)* and STCKP2 (225-227)*:**

These two fields delimit the boundaries of the pushdown
stack area.

**SYMP1 (229-231)* and SYMP2 (233-235)*:**

The beginning and ending locations for symbol table
space (dynamic storage space) are kept in these two fields
and utilized by the respective interpreters.

**CMPLCD (236):**

Translators are expected to set the completion code to
record the status of a translation.

**DSKLOD (237)*:**

If a library update (STORE=YES) subsequent to the next
translation is requested it is recorded here.

**EXCPPG (238)*:**

A request for execution (GO=YES) of the object program
from the resulting translation is set in this field.

**PSTLST (239)*:**

A postlisting request (POSTLIST=YES), honored by the re-
spective interpreter, may be recorded in this field.

**SYMBOL (243-282):**
The SYMBOL field referred to in Chapter III and used by all the translators and their interpreters resides in the specified locations.

PRGNME (283-290)*:

The name of a METAX program to be transmitted to or retrieved from the METAX library may be used by MTXLDR and MTXSTR.

INTNME (291-298)*:

The name of the interpreter corresponding to the specified METAX program to be executed is utilized by the control program.

A translator is expected to set the name of the program being translated and the name of an interpreter in the proper communications fields for library updating and the GO option, respectively.

D. Main Storage Usage

Recall that a 56K memory segment is utilized by the METAX system. The memory segments listed below are given with inclusive storage locations given in decimal. An asterisk means the respective area is under control of fields set by MTXCRM in which case the locations given are default. A control record may, however, alter the sequence prescribed.
Host machine index registers.  (1-60)
System communications region.  (61-189)
METAX communications region.  (200-399)
Resident I/O routines.  (400-3399)
Control program.  (3400-4500)
Pushdown stack region.  *(5000-5999)
Symbol table or dynamic storage region.  *(6000-9999)
METAX program execution region.  *(10,000-32,767)
Code generation region.  *(32,768-40,959)
Resident METAX program region  (40,960-45,055)
Transient program region.  (45,056-57,343)

The interpreter for PLEX object programs, in addition to residing in the transient program region during execution, utilizes part of that space for character string working storage.
V. PLEX: THE LANGUAGE AND ITS IMPLEMENTATION

PLEX (Programming Language Experimental) is based to a large degree on PL/I (29) and ALGOL (41,42), the purpose in its implementation being to demonstrate the capability of the METAX9 compiler-compiler in implementing a one pass compiler for a reasonably sophisticated language. A separate description of the syntax of PLEX is not given as the similarity of the syntactical aspects of METAX9 to BNF should suffice. The reader should consult Appendix B to determine precise syntactic information.

A. The PLEX Language

Five data types may be declared for identifiers including FLOAT, FIXED, CHARACTER, LABEL and LOGICAL, the scope of the identifiers being determined by the block and/or procedure structure of the program. Binary precision of arithmetic variables is (35) and (23,0) for the FLOAT and FIXED attributes respectively, these not being adjustable by declaration but being implementation defined characteristics. All variables have a storage class attribute similar to the PL/I AUTOMATIC attribute or the standard ALGOL assumption. The LABEL attribute may be used only for label variables and not for resolving references to label constants as in XPL
(38).

Character string variables by an undeclarable assumption have a PL/I-like VARYING attribute with a maximum length value required in the declaration of the string. Arrays may be declared with an implementation defined limit of 63 dimensions but with default or explicit lower bounds and explicit upper bounds all of which may be integer constants or variables but with no expressions, the amount of storage allocated for an array depending upon run-time evaluation of the array bounds.

Any of the five data types may occur in assignment statements with the restrictions that certain cases of indirect label assignment are not permitted and that data conversion is permitted directly only between the two arithmetic data types although GET STRING and PUT STRING may be used to accomplish certain conversions indirectly. Multiple left parts are permitted in assignment statements.

The standard arithmetic operations with the exception of exponentiation and the standard arithmetic built-in functions are provided, the latter having been left out to reduce the size of the required object interpreter. It is perhaps worth mentioning, however, that exponentiation and the standard BASIC functions were included in the BASIC implementation.

Character string assignments may include the use of SUBSTR either as a pseudo-variable in a left part or as a
built-in function in an expression. The concatenation operation is denoted by a pair of slashes (//) due to character set limitations of the host computer. Proper truncation on the right occurs with character string assignment statements as required.

Label assignment may include assignment of label variables or label constants, the latter being determined by the context in which they occur. However, assignment of label variables and/or constants not known within the scope of the immediately enclosing procedure is denied.

Logical assignment statements may include the use of the logical constants .T. and .F. as well as the logical operators .AND., .OR. and .NOT., the latter being used because suitable characters were not available on the print drum of the host computer. Relational operators are, however, suitably represented.

With the exception of label variables conditional assignments may be specified. Additionally function references to declared function procedures may be specified with the exception that functions with a LABEL attribute are not permitted.

Conditional statements (if statements) may include relational tests on character string expressions or arithmetic expressions. The ELSE option may be used but in any case the statement following the THEN may not be a conditional
statement in order to avoid the dangling ELSE ambiguity of ALGOL 60. It has been pointed out (38), however, that this does not comprise a strong restriction in that any statement may be embedded within a DO;...END; construct to make it basic, that is, not conditional. Should there be character string relational tests between strings of unequal length the shorter of the two strings will be padded on the right with blanks before testing takes place.

DO groups are represented by several types all of which occur in PL/I except for the DO CASE construct which may be found in XPL (38) and bastard ALGOL (20). This latter construct allows much greater selectivity than conditional statements as any one of a number of statements may be selected by the CASE expression. Iterative DO constructs may contain expressions in the iteration specification and also a negative step may be specified but only one iteration specification is permitted although the PL/I WHILE option is available. The DO WHILE construct is also included in the DO group category. The noniterative DO construct provides additional logical control in that a sequence of statements may be logically treated as a single statement by enclosure with the DO;...END; syntax.

The object of a GO TO statement may be a label constant or label variable. If it is a label constant then it must occur within the currently active procedure although jumps
across block boundaries are permitted so long as scope considerations are correct. If the object is a label variable the contents must satisfy the above label constant requirements except that if the label variable is a dummy argument in the formal parameter list of the currently active procedure then a RETURN is effected to the address specified. The detection of this situation is at compile time rather than at execution time.

With respect to input and output there are two kinds of each, unit record and character string, as exemplified by PUT EDIT and PUT STRING (...) EDIT for output and GET EDIT and GET STRING (...) EDIT for input, the unit record devices being a line printer and standard 80 column card reader. An I/O list and format list are required with output lists admitting expressions to be evaluated before being output. The reader is advised, however, that a function evaluation specified in an output expression may cause anomalous behavior if that function itself attempts an output operation.

Format lists may contain both data format items and control format items although somewhat limited in scope compared to PL/I, the intent being to provide only enough format capability to effectively demonstrate the rest of the language and its translator. No explicit decimal point specifications and no repetition factors are permitted. The E format item represents a generalized floating point format while the L
format item represents a logical format. Control format items include X, COL, PAGE and SKIP, the latter two causing execution time errors if used in the wrong context.

Procedures must be declared in a block head in an ALGOL-like fashion although this is a restriction which could probably be removed. Procedures need not be declared prior to their use in the case of function usage due to the RETURNS attribute being required to establish the data attribute of the function value. Additional usage of the ENTRY attribute, which must be used in declaring a formal procedure parameter, could permit removal of the restriction mentioned above. Except for the main program procedure which is automatically entered at the beginning of execution of a PLEX object program all procedures have an undeclarable recursive attribute in an ALGOL-like fashion. A BEGIN;...END; block is not required if local declarations or multiple statements occur in the procedure body but the attributes of the formal parameters must be declared in a single DECLARE statement prior to the declaration of any local variables.

A procedure may be invoked with a CALL statement if a RETURNS attribute has not been established or otherwise as a function reference. LABEL functions, however, are not permitted because of restrictions on label assignments delineated below. A return from a procedure may be effected by a RETURN statement, by the flow of control reaching the
procedure END statement or by a jump to a formal parameter as specified by a GO TO as outlined above. However, a direct jump to a label outside the procedure is not permitted, the restrictions on label assignment having been established to prevent this from happening indirectly via a label variable. As a further explanation the denial of a direct jump is based on dynamic storage and pushdown stack considerations.

Invoking a procedure whose name has been passed as an actual parameter requires special considerations by the compiler and run-time storage administration scheme because of the scope of variables required to be available at the time such a procedure is invoked. This subject is covered more thoroughly below under interpreter and translator considerations. Other than actual procedure parameters the actual parameters are established in a call by reference manner with constants and expression values being referenced in dynamic storage. Call by name with associated thunks is not provided except perhaps to the degree that procedure parameters are similar to call by name.

It should be noted that there are certain implementation restrictions on actual parameters which are not necessarily restrictions on the methods used but rather are limitations of the effort expended. For example, a determination has to be made at compile time whether a particular actual parameter is a simple reference requiring only an address to be passed
or an expression requiring evaluation with the result being put in dynamic or temporary storage and the address of that being passed. To further complicate the matter a function procedure reference presents a potential ambiguity between classification as a procedure parameter or as an expression requiring evaluation. Further discussion about the exact limitations and methods for their removal are discussed below. Suffice it to say, as has been previously mentioned, that the implementation has not been carried to its logical conclusion in all cases but rather to the point of demonstrating certain capabilities.

Comments are specified in the usual PL/I-like fashion with an opening "/*" and a closing "*/".

A more complete discussion of the PLEX language is outside the scope of this dissertation but it is hoped that the discussion below of the object interpreter and the translator, the sample PLEX programs in Appendix C and the PLEX translator (PLXCPL) listing in Appendix B will provide adequate additional detail.

B. The PLEX Pseudo-Machine

The discussion which follows is divided into two basic parts with run-time storage administration and related primitives being considered first and then followed by a descrip-
tion of the pseudo-machine interpreter.

Run-time storage administration is concerned with the proper management of available dynamic storage. With respect to the METAX system the space available to PLEX object programs for dynamic storage is delimited by the same communications area fields as used for specifying the symbol table space at compilation time. The scheme outlined here owes much to previous publications on ALGOL translators (13, 21, 40, 47), being rather similar, but not identical, to one outlined in detail by Gries (20). One consequence of the method is that an exit from a block requires no special action for releasing dynamic storage, thus implying that a jump outside the block (but within the containing procedure) requires no special handling. A negative consequence is that explicit source program controlled allocation and release of dynamic storage in a random manner is not possible. Thus a fuller implemention of PL/I dynamic storage facilities would require a very different scheme.

Basically the method is a stack allocation scheme which represents the nested block and procedure structure of a PLEX program, the dynamic storage stack being completely separate from the pushdown stack used for expression evaluation and procedure calls and returns. At the beginning of each dynamic storage area is a vector of addresses, called the active display, containing the base address of each active dynamic
storage area, one for each procedure data area required to be accessible because of scope considerations. The beginning entry (entry zero) is used as a pointer to the top of the dynamic storage area for the procedure activation to which the display belongs. The dynamic storage for the main procedure is considered to be on level one with successively nested procedures on higher levels. Level zero is considered to be static or absolutely addressable storage and is used to reference the program itself.

A separate display is not established upon block entry, only one per procedure being required. Suffice it to say that upon procedure entry, embodied by the ENTPRO primitive, a new display is established permitting addressing of globally active data areas as well as for the current procedure. Primitive DYNAM is used to specify the exact amount of fixed storage required, this comprising the least upper bound of storage required for parameter addresses, simple variables, temporaries, dynamic dope vectors and the procedure display itself for the execution of any block within the procedure, storage for arrays being allocated separately as discussed below. Parallel blocks share storage in this scheme but it is possible to allocate more storage than is required for a particular execution if, for example, a block which uses the uppermost locations is not executed. It is the responsibility of the compiler to determine the required fixed
storage and compile it into the DYNAM instruction.

It is pertinent to note that there is an implementation defined limit of 63 levels of active dynamic storage. This does not mean that a procedure may not recursively call itself to a depth of more than 63 calls but rather that procedures may not be lexicographically nested to a level greater than 63.

Upon block entry the STKTOP primitive is issued which identifies the address of the dynamic stack top variable for the immediately enclosing structure (block or procedure) as well as a new address in dynamic storage for a stack top variable for the current block, storage for this variable having been accounted for by the DYNAM primitive. Then individual allocations of dynamic storage for arrays by the ALLOC primitive reference this location for updating purposes and thus array allocations within a block belong only to the block and not to any surrounding structure. Dynamic storage allocation for a particular array may be variable from execution to execution due to potentially variable bounds. Additionally the ALLOC primitive references a static dope vector constructed at compile time and computes a new dynamic dope vector which is then used to reference the array at execution time. The dynamic dope vector has a size computable at compile time; therefore, dynamic storage for it is accounted for by the DYNAM primitive. Arrays allocated at the beginning of a pro-
procedure not within a separate BEGIN;...END; block reference
the procedure stack top variable at the beginning of the pro-
cedure display.

Upon block exit no provision is necessary to release any
storage. If a new block is entered storage in the fixed area
will be reused and arrays allocated based upon the stack top
value for the surrounding structure. All knowledge of stor-
age for the block terminated is lost which is precisely what
is desired.

Upon executing the RETURN primitive a certain register,
namely index register 14 of the host machine, is restored
with a value pointing to the beginning of the display for the
invoking procedure and thus dynamic storage addressing is
restored.

We now return to the discussion of procedure entry and
preparatory considerations with the dynamic storage scheme in
mind. The invocation of a procedure requires certain infor-
mation to be available to establish the display, the formal-
actual parameter correspondence and the global display ad-
dress which may not be the same as the display address of the
calling procedure in the event a call is made via a procedure
parameter for which the scope of variables may not be the
same. An example of this latter situation is contained in
the sample PLEX program TEST in Appendix C. Further informa-
tion is contained in the section on the PLEX compiler below.
In addition to the information for procedure invocation
the return address and active display of the invoking proce­
dure must be available to properly establish a normal return. To accomplish all of this the return address and the two
display addresses are pushed onto the pushdown stack followed
by a flag marker, which is established by the FLAG primitive,
followed by the parameter addresses and then the dynamic
storage stack top value at the point of call which in effect
is the address of the new display. A procedure is entered by
a normal jump but the first instruction of a procedure is
ENTPRO which stores the parameter addresses immediately be­
yond the display area of the procedure, copies the required
numbered of active storage addresses from the global display
and initializes the stack top variable for the current proce­
dure.

It should be noted that ENTPRO requires a one character
 literal parameter establishing the lexicographical level and
hence the number of display entries for the procedure.
DYNAM, which requires a 24 bit literal operand, is normally
executed next which brings the dynamic storage stack top val­
ue up to date and renders that storage ready for use by the
rest of the procedure.

For further discussion of the execution time storage
scheme Gries (20) should be consulted.

In presenting the discussion above primitives FLAG,
ENTPRO, DYNAM, ALLOC, RETURN and STKTOP have been reviewed and are mentioned below only to add information not given above.

The PLEX pseudo-machine interpreter (PLXINT00) generally functions in a manner somewhat similar to the METAX9 interpreter with one major difference being that address computations for operands potentially in dynamic storage are computed by a subroutine named ADCOMP prior to execution of the interpreter segment for an individual primitive. ADCOMP is also called by ALLOC to evaluate addresses of array bounds. One similarity between the two interpreters is the use of literal operands.

The address vector for the primitive interpreter segments contains an additional character with indicators for address computation requirements which the fetching code examines to determine the necessity of calling ADCOMP. Of course the primitives comprise a rather different set of functions in that the METAX9 pseudo-machine was conceived as a rather special purpose translation machine for a block structured source language like PLEX while the PLEX pseudo-machine is intended for interpretively executing object programs of a more general scope. It is true, as has been mentioned elsewhere, that the pseudo-machine was conceived to make the translation process somewhat simpler than one would encounter for most actual machines in use today. We add
parenthetically, however, that designing actual machines with ease of translation in mind has advantages that appear worthy of serious consideration as in the Burroughs B6500 (6,7) class of machines.

In any case a pushdown stack organization with arithmetic and logical operations appearing in a postfix-like manner comprises one major aspect of the organization. This stack also facilitates procedure invocation and return as outlined above. The pushdown stack area is delimited by the same communications area fields as used for the control stack of the translation pseudo-machines. Again a nine character entry is used with a one character type code and a maximum of 48 information bits. The stack may contain arithmetic operands and addresses, including the maximum string length in the event of a string address, but in no case may character strings be pushed onto the stack - only their addresses. In one case, after completion of the EMTPRO primitive, a return address and display address are packed together in one entry to facilitate a return from a function procedure.

The addressing computation is based on an addressing structure in which the first character represents both the storage reference type and the data type of the operand. These types may be conveniently separated by interpreting the six bit character as a pair of octal digits.

Storage reference type:
0x conventional static or dynamic reference.

1x Direct procedure reference.

2x Indirect procedure reference requiring two levels of indirect addressing to establish the actual address.

6x Indirect parameter reference requiring one level of indirect addressing to establish the actual address.

7x Literal operand following.

Data type:

- x0 Undefined.
- x1 Binary integer.
- x2 Floating point.
- x3 Logical.
- x4 Character string.
- x5 Label variable.
- x6 Label constant.
- x7 Universal data type.

Of course not all combinations are valid. The ADCOMP routine separates the storage reference type and data type into two separate fields for further internal use. If the
storage reference type is literal the address of the next location results. If the storage level, indicated by the first character following a nonliteral type code, is zero (static) then the result is the three character absolute address following. With a dynamic storage level indicated the current display is consulted for the base address and then the 18 bit address following the level code is used as a displacement. If any indirect addressing is indicated it is then performed. A second address may then be computed which may be a dynamic storage reference only - primitives ALLOC and STKTOP being the only primitives in this class. The two other primitives requiring address computations are LD (load to stack) and LDA (load address to stack).

Static storage addresses are used with some of the other primitives but a separate address computation is not required and not performed. Furthermore these addresses are four characters having no type codes as opposed to the addressing described above.

We next commence a discussion of the individual primitives not already covered.

LDA:

The single address computed by the ADCOMP routine is pushed onto the stack along with the data type code as the stack item type. In the event the data type denotes a char-
character string the maximum length is extracted from the instruction (immediately following the address) unless it is a literal string in which case the length is computed by scanning for a delimiting item mark on the right.

LD:

Push the operand at the address computed by ADCOMP onto the stack including the data type code.

STO:

Store the item at the top of the stack at the address specified next to the top of the stack. Special handling occurs if character string or substring assignment is made. Firstly the top stack element is the address of the string to be transmitted and secondly a truncated assignment may have to be made. Furthermore substrings may not have delimiting punctuation and therefore require special consideration. The reader is referred to the annotated listing in Appendix C for further detail. Data conversion for arithmetic operands may occur in order to meet the requirements of the receiving field. At the end of this operation the top two items are popped off the stack.

SST (save and store):

The same function as STO is performed except that the top of the stack is retained but the item next to the top is
discarded upon completion.

**JUMPA:**

Replace the instruction counter with the address on top of the stack; pop the stack one element.

**JUMP:**

Replace the instruction counter with the four character static address following the operation code.

**JUMPT:**

Conditionally replace the instruction counter based on a true logical item at the top of the stack; pop the stack.

**JUMPF:**

This is similar to JUMPF except the condition on the stack must be false.

**STCKC:**

A relational test is performed between the top two items of the stack based on the literal test code immediately following the operation code. Both operands are expected to be arithmetic and are converted to floating point if necessary in order to use the floating hardware on the host machine. The result is a logical value pushed onto the stack.

**COMPC:**

This is similar to STCKC except that the test is between
two character strings. Certain provisions for substrings and moving and padding on the right with blanks for a short field may be made. Character string working storage is used for substrings and right end padding.

**SWAP:**

Exchange the top two items on the stack.

**POPUP:**

Pop one element off the stack.

**ADD, MULT, SUB, DIV, NEG:**

These five arithmetic operations comprise rather standard postfix arithmetic, the reader being referred to Appendix C for further detail.

**FMT:**

A four character static address and a single character literal parameter following the operation code are used to establish the address of the format code, an input or output indicator and a string or unit record indicator. In the case of string I/O the address and length are extracted from the stack.

**GET:**

A stream input function is performed according to the current data format code. Preceding control format items are
executed prior to any data transmission. The address of the receiving field is at the top of the stack.

EDIT:

An output editing function is performed according to the current data format code. Preceding control format items are executed prior to any data transmission. The item being output, or its address in the case of a string, is at the top of the stack.

PUT:

In the case of printer output the standard macro-instruction for the host system is issued and the print buffer is cleared. With string output a right end item mark is established to delimit the resulting character string.

At this point it is perhaps pertinent to mention that well over 6K of the 12K characters of storage assigned to the interpreter and string working storage are dedicated to format routines, albeit of the "quick and dirty" variety, conversion routines and other supporting code but not including the resident unit record I/O routines. The intent here is to cast the size of the interpreter, exclusive of I/O, into proper perspective.

OR, AND, NOT:

Like the arithmetic instructions the logical instruc-
tions comprise rather standard operations not further
delineated here.

INDXR,INDXA:
The two indexing primitives expect the requisite number
of arithmetic indexing values on the stack with the right
most index on top. A single character literal value follow­
ing the operation code specifies the number of indices. The
index values are examined for conversion to binary integer
form and then the dynamic dope vector, whose address is just
below the left most index value, is consulted to compute the
address of the array element. In the case of INDXA all index
values are popped off the stack and the dope vector address
is replaced by the computed array element address. The only
difference with INDXR is that the value at the computed ad­
dress is loaded to the stack. In constructing the dynamic
dope vector from the static dope vector the ALLOC primitive
separates the vector elements into a constant part, an ele­
ment size part and a series of multipliers, possibly null in
the case of a single subscript, which then results in a
rather simple computation for the indexing primitives.

CAT:

This primitive is a string concatenation operator which
suffixes the string whose address is at the top of the stack
to the string whose address is next to the top of the stack,
the result being placed in string working storage. The two addresses on the stack are replaced by the result address. Again special handling is required for substrings which have no delimiting punctuation.

**SUBSTR:**

The address and length of the specified substring are pushed onto the stack with a special type code being set to mark it as a substring. This primitive is generated for both pseudo-variable and built-in function usage in the source language.

**STOP:**

This represents a terminal operation which results in an instruction count message and an exit to the METAX control program.

In examining the interpreter it is pertinent to observe that character string variables are not implemented with dope vectors but rather punctuation mark delimiters are used to indicate the size of a string although the maximum size is maintained in the compiled code. This represents a significant dependence on the host machine structure for the implementation of variable length character strings.

Decidedly more detail about the pseudo-machine may be gleaned from the annotated program listing in Appendix C.
Additionally the postlisting of the compiled PLEX program TEST in Appendix C should provide additional insight into the manner in which the pseudo-machine is intended to run.

C. The PLEX Translator

The PLEX translator is written in METAX9 as a series of recursive procedures. Extensive comments are included in the source listing given in Appendix B for procedures making fairly elaborate usage of postsyntactic commands and perhaps serve as a guide to their use.

The discussion which follows next comprises additional comments about some, but not all, of the procedures in a sequential fashion including additional information about certain restrictions mentioned above. A careful preliminary or parallel perusal of PLXCPL, the PLEX translator, may enhance the meaning of what follows.

Procedure PROGRAM effects certain initialization and permits leading comments to be processed prior to calling BHDBDY (block head and block body). The .DO(RESOLVE) command at the end of the procedure is normally the last command executed. BLKBDY (block body), which is called by BHDBDY, specifies that an arbitrary number of statements, possibly none, satisfies the syntactical requirements for that procedure.
The STHENT procedure permits an arbitrary number of comments and then labels to precede either a conditional statement or a basic statement. The call on ENDTST represents an example of "looking ahead" for an END statement and then backing up so that it may be processed as the end of a procedure, DO group or block. Although it isn't necessary to back over the END delimiter only to allow another procedure to test for it, it does perhaps make the procedures representing DO group, block or procedure syntax somewhat more readable.

The basic statement procedure BSCSTM contains latched calls to the DOGROUP and BLOCK procedures due to the possibility of valid identifiers containing "DO" or "BEGIN" as identifier prefixes.

Procedure BLOCK contains an example of stacking compile time variables for potentially recursive calls to BLOCK as well as STKTOP code compilation at the beginning of a block. The labels for dynamic storage stack top variables are created in a manner apart from the usual internal label generation partly to make the postlistings somewhat more readable and also for other debugging reasons.

BHDBDY contains a semantic check of fundamental importance in compiling dynamic storage administration code. Variable DYNAMB represents the discernable dynamic storage requirements, excluding arrays, for the block just processed
and including any enclosing active blocks, while DYNAMP represents the least requirements for the previously processed program segment belonging to the immediately enclosing procedure. Thus if DYNAMB exceeds DYNAMP a new upper bound must be established for the current procedure requirements. It is pertinent to recall at this point that dynamic storage levels are based on procedures rather than blocks. Of further interest should be the restoration of DYNAMB in BLOCK and PROCDEF after the return from BHDBDY.

PROCDEF, which processes procedure declarations in a block head, represents one of the larger procedures in the translator, the annotated listing providing considerable detail about its functions. Specifically it should be clear that declaration of attributes for formal parameters (dummy arguments) is required to be separated from the declaration of any local identifiers.

CALLPRT contains testing for a procedure call to a globally known procedure versus a call to an indirectly known procedure via a formal parameter, the code being generated for the cases being distinctly different in order to establish properly the global display address. The postlisting for the global display test in program TEST perhaps demonstrates this more graphically.

Procedure PARM and the two procedures called by it partially represent certain implementation restrictions to which
we alluded earlier. In effect PARMID should "look ahead" beyond an identifier, and possibly a subscript list in the case of an array identifier, to determine whether a terminating comma or right parenthesis is present or whether an expression is present. Code could be generated on the assumption that an expression is not present and erased by a forced backup (.DO(SETF,BEP) with a latched call in effect) followed by an alternative call to PARMEXP. Additionally, procedure PROCHK needs a simple extension to test for function identifiers.

One change in the backup mechanism which may aid any revision would be to delete automatic cancellation of the backup latch in the RETURN primitive and also to cause backup to the procedure in which the latched call occurs, rather than returning to the immediately preceding procedure. An additional alternative one may pursue is to establish a separate primitive function in the compiler interpreter to process the parameters in an ad hoc manner, or perhaps to perform a classification function which will direct the selection of a proper alternative. The FORTRAN PI (43,44) compiler written in META PI contains a call on a special subroutine for processing subroutine parameters as well as other cases in which special classification routines are used. These schemes may not be esthetically pleasing from a formal syntactical point of view but they may be rather effective.
As PLXCLPL is presently written, an actual parameter such as \((I-1)\) must be enclosed in parentheses as shown. An example may be found in the recursive factorial computation in program TEST.

Procedures GPROC and IPROC are concerned with the proper establishment of a procedure address and active display (global display) address for a procedure name as an actual parameter as explained in the annotated listing.

DECLARE through IDSEM represent procedures containing the syntax and semantics of declarations. It should be observed that attribute factoring is limited to one level and that identifiers, and hence dynamic storage addresses, are entered into the symbol table in reverse order due to the stacking of identifiers until the attributes are established. Additionally array dimensions may not be factored. It appears to this observer that multiple levels of attribute factoring represent a rather perverse problem for the type of language used in writing the PLEX translator although XPL (38) has the same limitation even though its compiler is written in XPL.

The CASE procedure contains an interesting example of the use of the \(.CAT\) command in constructing the list of branch instructions, one of which is selected at execution time and thus causes the proper statement to be executed.

Procedure WHLPT contains an example of internal label
usage where the label is established and defined elsewhere.

ITERPRT may be called by either LOOP or ERPLIST for the purpose of compiling loop iteration code for DO groups and I/O list iteration respectively. Procedure IOCHK called by ITERPRT accommodates the differences in requirements between the two cases. ITERPRT also contains the only example of code optimization in PLXCPL. In the case that an iteration limit or increment is not a simple primary, that is, evaluation is required, the expression code is placed prior to the main loop code with code for storing in dynamic storage for later reference. A considerable amount of code reordering may occur in generating the proper code.

The basic scheme used in ITERPRT was first developed during the experimental development of the BASIC translator when the author was involved with various alternatives for implementing loops. It is perhaps a tribute to the metalanguage approach that several alternatives could be explored without undue time constraints.

We then pass on to a long sequence of procedures which are fairly straightforward and on which only selective comments are given.

Under SVARBLE (string variable) it should be noted that the character string length (**(02,05)) is generated as part of the LDA instruction.

Procedure LTERM (label term) also requires special con-
sideration in terms of the code generated for a label identifier not known within the immediately enclosing block at the point of generation. In this case \( .OUT(LD,76,**(04,01)) \) is issued, being equivalent to \( .OUT(LDA,06,**(04,01)) \), which identifies the address as a literal address constant. Recalling the label resolution function of BLKEXT, primitive RESOLVE, in establishing the proper address from the pseudo-address, will also convert the type code to 05 (label variable) in the event the identifier happens to represent a label variable in a surrounding block. This is perhaps a special case of the skeletal operations used in the one pass ALGOL compiler described by Randell and Russell (47).

Certainly the translator contains errors of commission or omission as extensive testing on it has not been undertaken. The speed of the translator is relatively good, processing input at essentially card reader speed, 400 cards per minute, except perhaps in cases of multiple statements per card. It is true, however, that certain improvements in speed could be made primarily because an initial identifier in a statement may under go considerable rescanning and use as a symbol table search argument in classifying a statement, particularly an assignment statement. Revising the scheme to incorporate one initial identifier match and symbol table search for positioning purposes would surely be an improvement. Alternatively a substantially different syntax such as pre-
sented for XPL (38) may represent an improvement although this implies data conversions not provided here.

One additional PL/I facility which was seriously contemplated but not implemented is that of data structures. It appears possible that adding additional options as arguments of the .SEARCH command would facilitate symbol table entries and searches for structured operands. The entry and testing primitives will easily accommodate manipulation of three contiguous 20 character entries should they be required for processing a particular structure component as the positioning parameter in the respective commands has an upper limit of 63. It remains to be shown, however, that such structures can be accommodated within the basic framework provided.

In any case it is hoped that PLXCPL effectively represents the capability of METAX and that the sample PLEX programs in Appendix C demonstrate the efficacy of the whole matter.
VI. CONCLUSIONS AND SUGGESTED FURTHER WORK

It has been shown that a suitable compiler writing language, as an extension and revision of other such languages, has been developed with which a translator for a rather complex language such as PLEX can be readily written. In particular error recovery methods, internal label manipulation methods, block structured symbol table processing schemes and code reordering techniques as well as other semantic processing facilities have been incorporated into a basic top-down syntax analysis by a postsynthetic command structure which permits a significant amount of processing to be expressed in a single pass translator.

Certainly, then, a fundamental aspect of the METAX experimental compiler-compiler system is the METAX9 compiler writing language with which the PLEX translator was written. A major factor to be considered, however, is that in designing and developing a translator writing system of reasonable generality and efficacy it is rather difficult to conceive and implement a single compiler writing language which anticipates all of the features which may be required or considered useful for implementing a translator for a particular language. An alternative to be considered is to have available a basic compiler writing language with which one can implement a new compiler writing language satisfying the
requirements at hand. That is to say, it is suggested that a fixed language for compiler writing need not be a necessary goal of a translator writing system. It has certainly been the author's experience that this alternative approach is a viable one.

Furthermore, the above approach permits a wider range of implementation techniques to be considered for a particular situation. It may also be that changes or alterations to the metalanguage may be simpler to make than using an awkward approach in an existing language. The type of system presented here readily accommodates such matters.

It has also been shown that a relatively sophisticated compiler-compiler system can be implemented on a fairly modest computer system. For the METAX system described the largest demands on main storage resources have occurred while compiling the TEST program during which approximately 16k characters of the 56K available remained unused. Certain revisions could be made to reduce memory requirements but it does seem unlikely that a one pass translator for PLEX which could compile a program of the size of TEST could be implemented in much less than 32K characters of memory on the host machine.

As a result of the experience with matters covered in this dissertation there are a number of additions, extensions and/or revisions which come to mind as being worthy of
In view of the similarity of PLEX to PL/I further investigation into the implementation of additional PL/I-like features within the basic framework presented suggests itself.

Even though it has been shown that a great deal of processing can be effected in one pass, one pass translators present serious obstacles if object code optimization is a goal although it has been shown elsewhere (38,44) that some local optimization can be performed in one pass. Thus one may wish to consider using a scheme to generate a suitable form of intermediate code as input to another pass. We see no inherent limitations of the general approach presented here in developing such a system which would of course require changes in the code generation structure as presented. The CWIC/360 system (1) appears to offer a capability in this area but, as we have mentioned earlier, detailed information on that system is not freely available.

Contemplating some additional features which may be of use in writing translators, particularly if code for most conventional machines is to be generated, it firstly appears that the ability to declare directly the specifications for certain data structures such as last-in-first-out and first-in-first-out queues of both the fixed and variable length variety as well as symbol table structures may be of considerable utility. Of course a metalanguage command structure
would also be needed for data transmission from one structure to another. A limited form of this suggestion may be found in Feldman's semantic language (15).

Moreover, the input scanning and lexical analysis of the METAX translators is of a rather ad hoc nature. Recognition of certain syntactical entities which appear as terminal categories in the metalanguage depends rather heavily on the corresponding interpreter. Incorporating a scheme such as the AED RWORD scheme (20,32) for automatic generation of lexical analyzers would certainly provide a degree of generality and flexibility now only available by rewriting an interpreter appropriately.

The implementation of MTXCRA, although relatively elementary, provides a hint that substantially more could be accomplished with automatically generated processors for control languages, this possibility as well as the compilation of tables for table driven operating systems having been recognized and suggested by others (1,16).

Finally, extending the postsyntactic command structure to permit direct specification of error correction methods, such as those presented by Hedrick (22), may be of significant import.
VII. BIBLIOGRAPHY


VIII. ACKNOWLEDGEMENTS

The author owes a large debt of gratitude to Dr. Clair Maple for the guidance given not only during the preparation of this dissertation but also throughout the author's pursuit of graduate studies.

A special note of appreciation is given to the author's wife, Sharon, for her unswerving support and patience during the last four years.
IX. APPENDIX A
// CONTROL RECORD.
FUNCTION MEX8.
INTERPRETER=MTXINT03.
STORE=YES.
GO=NO.
//
**PROGRAM-ID: METAX9.**
**AUTHOR: J. R. VAN DOREN.**
**SOURCE LANGUAGE: METAX.**
**OBJECT LANGUAGE: METAX PSEUDO-MACHINE CODE.**
**OBJECT INTERPRETER: MXINT03.**

**PURPOSE:**

**METAX9 IS A REVISION OF THE METAX METALANGUAGE AND ASSOCIATED**
**COMPILER-COMPILER AND IS INTENDED FOR USE IN IMPLEMENTING THE PLEX**
**COMPILER.**

---

**THE DEFINITION OF THE OCTAL EQUIVALENT OF SYMBOLIC OPERATION CODES**
**FOLLOWS. FIRST THE CODES SPECIFIC TO METAX8 OBJECT PROGRAMS ARE**
**GIVEN FOLLOWED BY THE CODES COMMON TO METAX8 AND METAX9 AND THEN**
**THOSE CODES SPECIFIC TO METAX9. THE CODES FOR METAX8 ARE REQUIRED**
**ONLY FOR POSTLISTING PURPOSES AND FOR SYMBOLIC OPERANDS OF THE**
**".DO" CONSTRUCT. THE TYPE CODES ARE REQUIRED FOR POSTLISTING ONLY.**

**OPERATION CODES SPECIFIC TO METAX8 OBJECT PROGRAMS.**

BEM .OEQU 16 .TYPEO; MOVSYM .OEQU 27 .TYPEO; DELETE .OEQU 32 .TYPEO; TYPST .OEQU 34 .TYPEO; ENNTYP .OEQU 35 .TYPEO; ENTR .OEQU 72 .TYPEO; DECNUM .OEQU 75 .TYPEO;
/* OPERATION CODES COMMON TO METAX7 AND METAX9 OBJECT PROGRAMS. */

MOVE .OEQU 12 .TYPEB; A .OEQU 13 .TYPEB; M .OEQU 14 .TYPEB;
EXIT .OEQU 17 .TYPEB; RESOLVE .OEQU 20 .TYPEB; B .OEQU 21 .TYPEB;
BT .OEQU 22 .TYPEB; BF .OEQU 23 .TYPEB; DM .OEQU 24 .TYPEB;
SET .OEQU 25 .TYPEB; SETF .OEQU 15 .TYPEB; SCAN .OEQU 26 .TYPEB;
MOVI .OEQU 30 .TYPEB; SEARCH .OEQU 33 .TYPEB; LATCH .OEQU 36 .TYPEB;
CANCEL .OEQU 37 .TYPEB; CLM .OEQU 42 .TYPEB; R .OEQU 43 .TYPEB;
PUSHLB .OEQU 46 .TYPEB; POP .OEQU 47 .TYPEB; LB1 .OEQU 50 .TYPEB;
LB2 .OEQU 51 .TYPEB; OUT .OEQU 62 .TYPEB; OUTSYM .OEQU 63 .TYPEB;
TEST .OEQU 64 .TYPEB; ID .OEQU 65 .TYPEB; ONUM .OEQU 66 .TYPEB;
STRST .OEQU 67 .TYPEB; EVAL .OEQU 70 .TYPEB; ENTOC .OEQU 71 .TYPEB;
INUM .OEQU 73 .TYPEB; FNUM .OEQU 74 .TYPEB; STKSYM .OEQU 52 .TYPEB;
CHKSYM .OEQU 53 .TYPEB; SWAP .OEQU 54 .TYPEB; MARK .OEQU 40 .TYPEB;
SAVCL .OEQU 41 .TYPEB; RESTORE .OEQU 45 .TYPEB; ERASE .OEQU 31 .TYPEB;

/* OPERATION CODES SPECIFIC TO METAX9 OBJECT PROGRAMS. */

BFL .OEQU 16 .TYPEG; COMP .OEQU 32 .TYPEG; TSTTBL .OEQU 34 .TYPEG;
TSTTBA .OEQU 35 .TYPEG; ENTA .OEQU 32 .TYPEG; ENTL .OEQU 56 .TYPEG;
BLKENT .OEQU 75 .TYPEG; BLKEXT .OEQU 76 .TYPEG;

/* THE DEFINITION OF COMMUNICATIONS AREA FIELD LOCATIONS FOLLOWS. */
/* OBSERVE THE NONRELOCATABLE TYPE CODE USED. */

ELATCH .OEQU 000362 .TYPEN; SYMBOL .OEQU 000363 .TYPEN;
CMPLCD .OEQU 000354 .TYPEN; PRGNME .OEQU 000433 .TYPEN;

MTX009 := .EMPTY PRGHDR .ERR("W: INVALID OR MISSING PROGRAM NAME",SET) PRGNDY .DO(RESOLVE);

/* THE PROGRAM NAME AND DECLARATIONS COMPRISE THE PROGRAM HEADER */

PRGHDR := "PROG" .ID ";
.DO(MOVSYM .PRGNME)
.OUT(B**) $( DECLPT / COMMENT ) ;
/* DECLARATIVE STATEMENTS SPECIFY THE INITIAL VALUE AND SIZE OF VARIABLES */
/* BY THE SPECIFIED STRING, OCTAL OR DECIMAL INTEGER VALUE */

DECLPT :="·DECLARE" $(·ID ·DEFLAB(*)
  $(·STRING /
    ·OUT(*) ("," / ";" ·RETURN) )
  $(·ONUM)) ;

/* THE PROGRAM BODY COMPRISRES RECURSIVE PROCEDURE STATEMENTS, SYMBOL */
/* EQUATE STATEMENTS AND COMMENTS. OBSERVE THAT EQUATE STATEMENTS */
/* PROVIDE A PARAMETERLESS MACRO FACILITY. */

PRGIDY :=$ ST "·END" ·ERR("F: UNRECOGNIZABLE STATEMENT") ;
ST :=·ID ("::=") $DO(ENTLOC) MTXEXP ·OUT(R) / $DO(SEARCH)
  ("·GEQU" ·ONUM / "·IEQU" ·INUM) $DO(ENTER)
  ("·TYPEO" $DO(ENTTYP,"O")
   / "·TYPEG" $DO(ENTTYP,"G")
   / "·TYPEB" $DO(ENTTYP,"B")
   / "·TYPEN" $DO(ENTTYP,"N")
   / "·EMPTY") ;"$
  ·ERR("W: EXPECTED ;") ·SET
  / COMMENT ;
  COMMENT:="/" ·SCAN("/")

/* THE MTXEXP PROCEDURE CONTAINS THE SYNTAX AND SEMANTICS FOR THE RIGHT */
/* HAND SIDE EXPRESSION OF A RECURSIVE PROCEDURE STATEMENT. OBSERVE THE */
/* SPECIFIED SEMANTIC ACTIONS ASSOCIATED WITH INTERNAL LABELS AND */
/* REDUNDANT CODE ERASURE. */

MTXEXP :=·NEWLAB ·NEWLAB ALTERN
  $( "/" ·DEFLAB(*1) $DO(POP)
    ·NEWLAB ALTERN )
  $DO(MVO,SYMBOL,"4",ERASE)
  ·DEFLAB(*1) ·DEFLAB(*2) ) ;

/* THE ALTERN PROCEDURE CONTAINS THE SYNTAX AND SEMANTICS FOR THE PART */
/* OF A METAX9 EXPRESSION CONTAINED WITHIN A SYNTACTICAL ALTERNATIVE. */
THE METALANGUAGE CONSTRUCTS CONTAINED THEREIN ARE CLASSIFIED AS ELEMENTARY SYNTAX OR SEMANTIC ACTION.

```
ALTERN :=# SEMACT ( ELMSTX \OUT(BF,*1) / .EMPTY )
    $( SEMACT / ELMSTX ERRACT ) \OUT(E,*2) ;
```

THE ELEMENTARY SYNTACTICAL CONSTRUCTS ARE OUTLINED BELOW. OBSERVE THAT "STKCHK", "TSTTBA", "TSTTBL" AND "TEST" COMPRISE SEMANTIC CHECKING RATHER THAN PHRASE STRUCTURE SYNTAX. THESE CONSTRUCTS ARE INCLUDED HERE DUE TO CONDITIONAL JUMPING AND ERROR CODE GENERATION.

```
ELMSTX :=.ID \OUT(CL.*1) / .STRING \OUT(TET.*1)
    / "ID" \OUT(ID) / "STRING" \OUT(STRTST)
    / "ONUM" \OUT(ONUM) / "INUM" \OUT(INUM) / "EMPTY" \OUT(SET)
```

OBSERVE THAT THE NEXT ALTERNATIVE PERMITS FACTORING OF METALANGUAGE EXPRESSIONS.

```
"(" MTXEXP ")" .ERR("W: EXPECTED ")".;SET
```

THE NEXT ALTERNATIVE PERMITS ITERATIVE EXPRESSIONS.

```
"$" .NEWLAB .DEFLAB(*1) ELMSTX \OUT(BT,*1) \OUT(SET)
    / "FNUM" \OUT(FNUM) / "STKCHK" \OUT(CHKSYM)
    / "LATCH(" .ID \OUT(LATCH,**) ")"
    / "TSTTBA(" .OUT(TSTTBA) .ONUM \OUT(*) ");" .ID \OUT(**) ")"
    / "TSTTBL(" .OUT(TSTTBL) .ONUM \OUT(*) ");" ( .ONUM \OUT(*)
    / "STRING \OUT(SET)")")
    / "TEST(" .OUT(COMP) .ID \OUT(**) TESTOP .ID \OUT(**,*) ")";
```

TESTOP :=="=" .SAV(02) / "=" .SAV(06) / "=" .SAV(04)
    / ">=">" .SAV(03) / ">=">" .SAV(01) / ">=" .SAV(05) ;

"COMMENT" IS INCLUDED AS A SEMANTIC ACTION ALTERNATIVE ONLY FOR THE CONVENIENCE OF PERMITTING FREE INSERTION OF COMMENTS.

```
SEMACT :=OUTPUT / COMMENT / SEMI ;
```
/* THE OUTPUT PROCEDURE IS CONCERNED WITH THE SPECIFICATION OF CODE */
/* GENERATION, EITHER DIRECTLY IN THE COMPILER BEING GENERATED OR */
/* FOR THE CODE THE COMPILER ITSELF IS TO GENERATE. */

OUTPUT := "OUT(" OUT $ ( "", OUT1 ) ")"
    / "DO(" OUTDO $ ( "", OUTDO ) ")" ;

OUTDO := ID OUT(**) / ONUM OUT(*) / STRING OUT(*) ;

OUT1 := "*1" OUT(LH1,EVAL,04,01) / "*2" OUT(LB2,EVAL,04,01)
    / "*" OUT(EVAL) ( "(" ONUM OUT(*)
        / "", ONUM OUT(*) / EMPTY OUT(00) ) ")"
    / EMPTY OUT(05,00) )
    / "*" OUT(OUTSYM) / "*" OUT(RESTORE)
    / ID OUT(OUT,**) / ONUM OUT(OUT,*)
    / STRING OUT(OUT,*) ;

SEMI := "NEWLAB" OUT(PUSHLB)
    / "DECLAB" ( "*1" OUT(LB1) / "*2" OUT(LB2) / "*" ) OUT(ENTLOC)
        OUT(ENTL,00,06) )"
    / "STACK(" STKID $ ( "", STKID ) ")"
    / "UNSTACK(" MARK UNSTKID SAV $ ( "", MARK UNSTKID
        OUT(#) SAV ) ")" OUT(#) )
    / "SET(" ID "=" SAV(**) ( ID OUT(MOVE,**,#)
        / ( STRING / ONUM ) OUT(MOVI,**,*) ) ")"
    / "BLKEXT" ( "(" STRING OUT(BLKEXT,**) ")"
        / EMPTY OUT(BLKEXT,"0")
    )
    / "MARK" OUT(MARK)
    / ( "SAV(" OUT(MARK) OUT1 $ ( "", OUT1 ) ")" / "SAV" ) OUT(SAVE)
    / "CAT(" OUT(MARK,RESTORE) OUT1 $ ( "", OUT1 ) OUT(SAVE) ")"
    / "BLKENT" OUT(BLKENT)
    / "ENTERL" OUT(ENTL) ONUM OUT(*) )"
        ( STRING / ONUM ) OUT(*) ")"
    / "ENTERA" OUT(ENTA) ONUM OUT(*) / ID OUT(**) )"
    / "SEARCH" ( "(" STRING OUT(SEARCH,**) ")"
        / EMPTY OUT(SEARCH,"0")
    )
/* STKSYM OUT(STKSYM) */
/* SCAN(" STRING OUT(SCAN,"") */
/* ERKLATCH OUT(MOV,ELATCH,"1") */
/* RETURN OUT(R) / CANCEL OUT(CANCEL) */

STKID := ID OUT(MOVE,**,SYMBOL,STKSYM) ;

UNSTKID := ID OUT(POP,MOVE,SYMBOL,**) ;

/* THE ERROR ACTION PROCEDURE GENERATES CODE FOR ERROR MESSAGES, */
/* DEFAULT AND SPECIFIED ERROR ACTION. */
/* "COMMENT" IS INCLUDED ONLY FOR THE CONVENIENCE OF COMMENT PLACEMENT. */

ERRACT := "ERR(" NEWLAB OUT(BT,1) STRING OUT(BM,**)
$ ( ":, OUTDO ) ")" DEFLAB(*1)
/ EMPTY OUT(BEF) ;

/* THE FOLLOWING PROCEDURE COMPRIS A REQUIRED DEFAULT ERROR PROCEDURE */
/* FOR PROGRAMS WRITTEN IN THE METAX8 LANGUAGE. THIS IS NOT REQUIRED */
/* FOR METAX9 PROGRAMS. */

ERROR := DO(MOVI,CMPLCD,"F") SCAN(";") PRGBDY DO(EXIT) ;

END}

***COMPILED PROGRAM SIZE = 2,653; METAX INSTRUCTION COUNT = 33,888***

***SYMSTAB SEARCH COUNT = 895; SYMTAB COMPARE COUNT = 143,068***

***SYMBOL TABLE ENTRY COUNT = 346***
X. APPENDIX B
FUNCTION MTX009.
INTERPRETER=MTXINT03.
GO=NO.
START SYMTAB AT 5096, END SYMTAB AT 15096.
EXECUTE AT 15097.
POSTLIST=YES.
STORE=YES.
**IDENTIFICATION:**

PROGRAM-ID: PLXCPL

AUTHOR: J. R. VAN DOREN

SOURCE LANGUAGE: METAX9

OBJECT LANGUAGE: METAX9 PSEUDO-MACHINE CODE

OBJECT INTERPRETER: MTXINT04

**PURPOSE:**

PLXCPL IS THE COMPILER FOR THE PLEX LANGUAGE.

*DECLARE VARIABLE NAMES AND INITIAL VALUES REQUIRED BY THE COMPILER.*

*NOTE THAT INITIAL VALUES ARE IN OCTAL OR CHARACTER STRING FORM.*

*THE LENGTH OF A VARIABLE BEING IMPLICIT IN ITS INITIAL VALUE.*

*VARIABLES USED FOR COMPILING DYNAMIC STORAGE ADMINISTRATION CODE.*

*VARIABLES USED FOR SYNTACTIC ANALYSIS*
/* VARIABLES USED FOR ATTRIBUTE PROCESSING */
\*DECLARE TYPE "", LENGTH 0000, DIMCNT 00, FUNCT 0000;

/* VARIABLES USED FOR PROCEDURE PROCESSING (DECLARATIONS AND CALLS) */
\*DECLARE ARGCNT 00, ADECNT 00, PARMCNT 00, OCTEN 10, OCTL60 60,
OCTL20 20;

/* VARIABLES USED FOR LABEL PROCESSING (PRIMARILY FOR PROCEDURE EXITS) */
\*DECLARE PEXITF 77, PEXITT 00;

/* VARIABLES USED FOR TESTING I/O ITERATION LOOPS */
\*DECLARE IOITER 77, IOSW 00;

/* OTHER VARIABLES AND VALUES FOR GENERAL USE */
\*DECLARE ONE 01, FOUR "0004", EIGHT "0008", BLNK8 "", ZERO 00;

/* THE DEFINITION OF THE OCTAL EQUIVALENT OF SYMBOLIC OPERATION CODES */
/* FOLLOWS. FIRST THE CODES FOR METAX9 OBJECT PROGRAMS ARE GIVEN */
/* AND THEN THE CODES FOR PLEX OBJECT PROGRAMS FOLLOW. THE METAX9 */
/* CODES ARE REQUIRED FOR POSTLISTING AND FOR SYMBOLIC OPERANDS OF */
/* THE "*.DO" AND "*.ERR" CONSTRUCTS. THE TYPE CODES ARE REQUIRED FOR */
/* POSTLISTING ONLY. */

/* OPERATION CODES FOR METAX9 OBJECT PROGRAMS. */

MOVE  \*OEQU 12 \*TYPEO; A \*OEQU 13 \*TYPEO; M \*OEQU 14 \*TYPEO;
BEF   \*OEQU 16 \*TYPEO; EXIT \*OEQU 17 \*TYPEO; RESOLVE \*OEQU 20 \*TYPEO;
R     \*OEQU 21 \*TYPEO; BT \*OEQU 22 \*TYPEO; BF \*OEQU 23 \*TYPEO;
BM    \*OEQU 24 \*TYPEO; SET \*OEQU 25 \*TYPEO; SETF \*OEQU 15 \*TYPEO;
SCAN  \*OEQU 26 \*TYPEO; MOVI \*OEQU 30 \*TYPEO; SEARCH \*OEQU 33 \*TYPEO;
COMP  \*OEQU 32 \*TYPEO; TSTTBL \*OEQU 34 \*TYPEO; TSTTBA \*OEQU 35 \*TYPEO;
LATCH \*OEQU 36 \*TYPEO; CANCEL \*OEQU 37 \*TYPEO; CLM \*OEQU 42 \*TYPEO;
R .OEQU 43 .TYPEO; PUSHLB .OEQU 46 .TYPEO; POP .OEQU 47 .TYPEO;
LBI .OEQU 50 .TYPEO; LB2 .OEQU 51 .TYPEO; ENTA .OEQU 55 .TYPEO;
ENTL .OEQU 56 .TYPEO; OUT .OEQU 62 .TYPEO; OUTSYM .OEQU 63 .TYPEO;
TEST .OEQU 64 .TYPEO; ID .OEQU 65 .TYPEO; ONUM .OEQU 66 .TYPEO;
STRST .OEQU 67 .TYPEO; EVAL .OEQU 70 .TYPEO; ENTOC .OEQU 71 .TYPEO;
INUM .OEQU 73 .TYPEO; LB2 .OEQU 74 .TYPEO; BLKENT .OEQU 75 .TYPEO;
BLKEXT .OEQU 76 .TYPEO; STKSYM .OEQU 52 .TYPEO; CHKSYM .OEQU 53 .TYPEO;
SWAP .OEQU 54 .TYPEO; MAPK .OEQU 40 .TYPEO; SAVE .OEQU 41 .TYPEO;
RESTORF .OEQU 45 .TYPEO; ERASE .OEQU 31 .TYPEO;

/* OPERATION CODES FOR PLEX OBJECT PROGRAMS */

DYNAM .OEQU 10 .TYPEG; STKTOP .OEQU 11 .TYPEG; ALLOC .OEQU 12 .TYPEG;
LDA .OEQU 20 .TYPEG; LD .OEQU 21 .TYPEG; STC .OEQU 22 .TYPEG;
JUMPF .OEQU 32 .TYPEG; STCKC .OEQU 33 .TYPEG; COMPC .OEQU 34 .TYPEG;
ADD .OEQU 40 .TYPEG; MULT .OEQU 41 .TYPEG; SUB .OEQU 42 .TYPEG;
DIV .OEQU 43 .TYPEG; NEG .OEQU 44 .TYPEG; FMT .OEQU 50 .TYPEG;
GET .OEQU 51 .TYPEG; PUT .OEQU 52 .TYPEG; EDIT .OEQU 53 .TYPEG;
OR .OEQU 60 .TYPEG; AND .OEQU 61 .TYPEG; NOT .OEQU 62 .TYPEG;
INDXR .OEQU 66 .TYPEG; INDXA .OEQU 67 .TYPEG; CAT .OEQU 70 .TYPEG;
SUBSTR .OEQU 71 .TYPEG; JUMPA .OEQU 27 .TYPEG; STOP .OEQU 76 .TYPEG;
ENTPRO .OEQU 25 .TYPEG; RETURN .OEQU 26 .TYPEG; FLAG .OEQU 24 .TYPEG;
SWP .OEQU 35 .TYPEG; POPUP .OEQU 36 .TYPEG;

/* THE DEFINITION OF COMMUNICATIONS AREA FIELD LOCATIONS follows. */
/* OBSERVE THE NONRELOCATABLE TYPE CODE USED. */

SYMBOL .OEQU 000363 .TYPEN; PRGNME .OEQU 000433 .TYPEN;
INTNME .OEQU 000443 .TYPEN; SYMPTWO .OEQU 000365 .TYPEN;
ELATCH .OEQU 000362 .TYPEN;

/* COMMENCE THE RECURSIVE PROCEDURES COMPRISING THE COMPILER PROPER. */
PLXCPL := .EMPTY .ERRLATCH PROGRAM .ERR("F: COMPILER ABORT - BAD PROGRAM",EXIT);
PROGRAM := .ERRLATCH .NEWLAB .ID "": .STKSYM /* SAVE NAME FOR END CHECK */
•SET(PRGNME=Ssymbol)  /* SET PROG NAME FOR STORE OPTION /
•SET(INTNME="PLXINT00")  /* SET INTERPRETER NAME FOR "GO" OPT. */
•SET(SYMBOL="$ACTIVE")  /* SYMBOL AND ADDRESS VALUE OF INDEX */
•SEARCH *ENTERL(00,0100000065)  /*REGISTER POINTING TO ACTIVE */
•SET(DYNAM=DYNAMP)  /* DYNAMIC STORAGE AT RUN TIME */
•SET(SYMCL=SYMCL)  /* INITIALIZE BLOCK DYNAMIC STORAGE */
**PROCEDURE" "MAIN" .BLKENT";"  /* REQUIRED SYNTAX; START BLOCK LIST */
•OUT(DYNAM,*)  /* PROCEDURE DYNAMIC STORAGE CODE */
•SET(SYMBOL=STACKTP)  /* TYPE AND INIT STACKTOP LOCATION */
SEARCH *ENTERL(00,0501000000)  /* LABEL CONSTANT TYPE */
$ COMMENT  /* ADMIT HEADER COMMENTS */
BHDBDY  /* PROCESS BLOCK HEAD AND BODY */
"END" *OUT(STOP)  /* POSITION SYMTAB POINTER */
•DEFLAB(*)  /* MAXIMUM SIZE OF LEVEL 1 STORAGE */
•ENTERA(01,DYNAMP)  /* FOR THIS PROCEDURE */

( .ID .STKCHK .ERR("W: POSSIBLE PROC CLOSING ERROR",SET)
/ *EMPTY ) .BLKEXT("1") .DO(RESOLVE) ;

BLKEDY := S ( .SET(STMLAB=BLMKB) STMENT ) ;

STMENT ::= ERRLATCH $ COMMENT $ LATCH(LABEL)  /* PROCESS COMMENTS AND LABELS */
( ENDTST .DO(SETF,R)  /* DO NOT MISTAKE "END" AS AN ID */
/ CONDST / BSCSTM ) .ERR("F: BAD STATEMENT",SET,SCAN,; ");

BSCSTM ::= ( LATCH(DOGRUP) / LATCH(BLOCK) / UNCOND / .EMPTY ) "; ";

COMMENT ::= "/"*/" .SCAN("*/") ;

LABEL ::= .SET(STMLAB=Ssymbol)  /* SAVE LABEL */
•SEARCH("1") .DO(ENTLOC)  /* ENTER LABEL AND VALUE */
•ENTERL(00,06)  /* LABEL CONSTANT TYPE */

ENDTST ::= LATCH(ENDSTM) .DO(SETF) / .EMPTY ;

ENDSTM:"END" .DO(SETF,BEF) / .EMPTY ;
```plaintext

BLOCK := "BEGIN" ";" *BLKENT *ERRLATCH /* SET BLOCK LIST AND ERROR LATCH */
  *STACK(STMLAB,DYNAMP,STACKTP) /* STACK VARIABLES TO PREPARE FOR */
  *OUT(STKTOP,**),DO(A,ONLLNG,STACKTP) /* POTENTIAL RECURSIVE CALL OF "BLOCK"*/
  *DO(A,ONEELLNG,STACKTP) /* CREATE NEW STACK TOP LABEL */
  *SET(SYMBOL=STACKTP) /* SET FOR SYMTAB PROCESSING */
  *SEARCH("1") ENTERA(01,DYNAMB) /* SYMTAB ENTRY */
  *ENTERL(00,05)/* NEW STACK TOP ADDRESS AS 2ND OPER */
  *DO(A,FPUR,STACKTP) /* INCREMENT DYNAMIC STORAGE COUNTER */
  BHDERY /* PROCESS BLOCK HEAD AND BODY */
  *UNSTACK(DYNAMB,STACKTP) /* RESTORE PREVIOUS STACK TOP SYMBOL */
  "END" *BLKEXT /* RESTORE SYMTAB TO PREVIOUS LEVEL */
  *ID *STKCHK *ERR("W: POSSIBLE BLOCK CLOSING ERROR",SET)
  /* END */

BHDERY := BHKD BLDKEDY /* DECLARATIONS AND BLOCK BODY */
  *TEST(DYNAMB > DYNAMP) /* DETERMINE LEVEL OF STORAGE REQUIRED */
  *SET(DYNAMP=DYNAMB) /* SET NEW LEVEL */
  /* EMPTY */) /* OLD LEVEL O.K. */

BHKD := $DECLAR /* PROCESS IDENT AND PROC DECLARATIONS*/
  *NEWLAB *OUT(JUMP,**1) $ *LATCH(PROCDEF) *DEFLAB(*1) $

PROCDEF := *ID "" "PROCEDURE" *CANCEL /* LABEL FOR STORAGE COUNT */
  *SEARCH("1") *DO(ENTLOC) /* LOCATION AND PROCEDURE TYPE */
  ( *TSTTBL(00,10) *DO(BM,"W: DUP PROC DCL",SET)
    / *TSTTBL(00,00) *ENTERL(00,10) / *DO(SET) )
  *BLKENT *ERRLATCH
  *STKSYM /* SAVE PROCEDURE ID FOR ENDCHECK */
  *STACK(DYNAMP,DYNAMB,LEVNO,STACKTP) /* SAVE PREVIOUS PROCEDURE INFO */
  *DO(A,ONE,LEVNO) /* SET UP */
  *SET(DYNAMP=ONELEV) /* DYNAMIC STORAGE */
```
DO(M,LEVNO,DYNAMP) /* COUNTERS */
SET(DYNAMB=DYNAMP) /* FOR THIS PROCEDURE */

DO(A,ONELNG,STACKTP) /* NEW STACKTOP SYMBOL FOR THIS PROC */
SET(SYMBOL=STACKTP)
SEARCH("1") ENTERA(01,DYNAMP) /* LOCATION IN FIRST WORD OF */
ENTERL(00,05) /* THE DISPLAY FOR THIS PROC */

SET(SYMBOL=FOUR) /* COMPUTE SIZE OF DISPLAY FOR */
/* DYNAMIC STORAGE COUNT */

DO(M,LEVNO,SYMBOL,A,FOUR,SYMBOL,A,SYMBOL,DYNAMB)
OUT(ENTPRO) /* PROCEDURE ENTRY */
SET(SYMBOL=LEVNO) OUT(*) /* DYNAMIC STORAGE LEVEL NUMBER */
OUT(DYNAM,1) /* CODE TO ALLOCATE DYNAMIC STORAGE */
SET(ARGCNT=ZERO) ("(" ARGID "(" ARGID ")") "/ EMPTY ");" /* DUMMY ARGUMENTS */
SET(SYMBOL=STMLAB) /* ENTER ARGUMENT COUNT */
SEARCH ENTERA(07,ARGCNT)
SET(ARGCNT=ZERO) ( TARGCNT /* TEST FOR ARGUMENT COUNT */
/* ATTRIBUTES FOR DUMMY ARGUMENTS */
(="DECLARE" ARGDEC "+=" ARGDEC ");" /* EMPTY */
TEST(ARGCNT=ADECNT) ERR("W: INCORRECT ARG DCL COUNT",SET)
/ EMPTY ) /* BYPASS IF NO ARGS */
BHDEDY /* PROCESS THE REST OF THIS PROCEDURE */
"END" OUT(RETURN)
DEFLAB(*1) /* POSITION SYMTAB */
ENTERA(01,DYNAMP) /* MAXSIZE OF FIXED DYNAMIC STORAGE */
/* FOR THIS PROCEDURE */
BLKEXT("1") /* PREVENT UNRESOLVED LABEL LINKAGE */
/* OUTSIDE THIS PROCEDURE */
UNSTACK(DYNAMP,DYNAMB,LEVNO,STACKTP) /* RESTORE FOR PREV PROC */
ID ERR("W: EXPECTED CLOSING PROC NAME",SET)
STKCHK ERR("W: POSS PROC CLOSING ERR",SET) ";" ;

TARGCNT:=.TEST(ARGCNT=ZERO) DO(SETF=) / EMPTY:
ARGID := ID SEARCH("1") ENTERA(01, DYNAMB) /* ACTUAL PARAMETER ADDRESSES */
  ENTERL(00, 60) DO(A, ONE, ARGCNT) /* IN DYNAMIC STORAGE MARK */
  DO(A, FOUR, DYNAMB, SFT) /* DUMMY ARGS INDIRECT REF */

ARGDEC := ID SEARCH("1") EPRLATCH TSTTBL(00, 60)
  ERR("F: NON-EXISTENT ARG") ARGARY ARGATR DO(A, ONF, ADECNT) ;

ARGARY := "(" * SET(DIMCNT=ZERO)
  'SET(DIMCNT=ZERO) 'DO(A, ONE, DIMCNT) $ ( "* " * "'DO(A, ONE, DIMCNT))")"
  ENTERA(07, DIMCNT)
  / 'EMPTY ;

ARGATR :=ATTRIBT ENTERA(05, LENGTH) DO(A, OCTL60, TYPE)
  ENTERA(00, TYPE)
  / "ENTRY" ( "RETURNS(" ATTRIBT ")" DO(A, OCTL20, TYPE) ENTERA(00, TYPF)
  / 'EMPTY ENTERL(00, 20) ) ;

CLLSTMT := ID TEST(SYMBOL=CALLCON) ID CANCEL SEARCH CALLPRT;

CALLPRT := SAV(LDA, **, JUMPA) 'EMPTY /* SAVE CODE TO JUMP TO PROC */
  SET(SQLAB=SYMBOL) /* SAVE PROC NAME FOR CHECKING */
  NMLAB OUT(LDA, 06, 01)
  OUT(LD, 0100000065) /* CODE TO LOAD RETURN ADDRESS */
  SET(SQLAB=SYMBOL) SEARCH
    ( ( * TSTTBL(00, 10)
      / * TSTTBL(00, 11)
      / * TSTTBL(00, 12)
      / * TSTTBL(00, 13)
      / * TSTTBL(00, 14) )
    OUT(LD, 0100000065) /* CURRENT DISPLAY ADDR IS GLOBAL */
    / ( * TSTTBL(00, 20)
      / * TSTTBL(00, 21)
      / * TSTTBL(00, 22)
      / * TSTTBL(00, 23)
      / * TSTTBL(00, 24) )
  SAV(LD, 01, **(04, 01)) /* CODE TO LOAD ADDR OF PROC ADDR */
*NEWLAB *CUT(LDA,01,*1)  /* CODE TO LOAD ADDR FOR STOPING
   COMPUTED GLOBAL DISPLAY ADDR */
*OUT(*)
*CUT(LD,71,"0004",ADD)
*CUT(STO)
*CUT(LD,01) *DEFLAB(*1)
*ENTERL(00,01)
*OUT("0000")
*DO(POP)
*DO(SET)
*ERR("F: INVALID PROC CALL")

CLLPRM
*SET("SYMBOL=STACKTP")
*OUT(LD,**)
*OUT(#)
*DEFLAB(*1) ;

/* CODE TO LOAD ADDR FOR STOPING
   COMPUTED GLOBAL DISPLAY ADDR */
/* COMPUTE ADDR OF GLOBAL DISPLAY ADDR */
/* CODE TO STORE ADDR OF GLOBAL DISPLAY ADDR IN NEXT INSTR */
/* CODE TO LOAD GLOBAL DISPLAY ADDR */
/* POP *1 LABEL */
/* SET TF CODE FOR COMPILER TESTING */

CLLPRM := *OUT(FLAG)
  *STACK(PARMCNT) *SET(PARMCNT=ZERO)
  *STACK(STMLAB) /* SAVE PROC NAME */
  ( PARMPRT / *EMPTY ) /* PROC NAME OFF STACK FOR SEARCHING */
  *SEARCH("0")
  ( (.TSTTBL(00,11) / .TSTTBL(00,12) / .TSTTBL(00,13) / .TSTTBL(00,10))
   .TSTTBA(07,PARMCNT)
   *ERR("W: INCORRECT PARM CNT",SET)
   / *EMPTY ) ) UNSTACK(PARMCNT);

PARMPRT := "(" " PARM $ ( ""," PARM")")" ;

PARM := ( PARMID / PARMEXP ) *DO(A,ONE,PARMCNT) ;

PARMID := LATCH(ALFTPT) / LATCH(BLFTPT) / LATCH(SLFTPT) / LATCH(LLFTPT)
         / LATCH(PROCHK) ;

PROCHK := *ID ( "(" *DO(SETF,BEF) / *EMPTY )
SEARCH("0")
( TSTTBL(00,10) CANCEL GPROC /* CHECK FOR GLOBAL PROC PARM */
/ TSTTBL(00,20) CANCEL IPROC ) ; /* CHECK FOR INDIRECT PROC PARM */

PARMEXP:= NEWLAB .DO(LB1) .SEARCH("1") ENTERA(01,DYNAMB)
ENTERL(00,07) .DO(A,EIGHT,DYNAMB) OUT(LDA,**,LDA,**)
( EXP / SSE / LTERM / #PRMRY )
OUT(STO) ;

/* SAVE ADDRESS OF PROCEDURE AND ACTIVE DYNAMIC STORAGE AREA AT THE */
/* POINT OF CALL FOR PROCEDURE NAME AS FORMAL PARAMETER. THE ADDRESS */
/* PASSED IN THE RUN TIME STACK IS THE ADDRESS IN DYNAMIC STORAGE */
/* OF THE PROCEDURE ADDRESS FOLLOWED BY THE ACTIVE DISPLAY ADDRESS. */

GPRCC := SAV(LDA,01,**(04,01)) /* CODE TO LOAD PROC ADDRESS */
NEWLAB .DO(LB1) .SEARCH("1")
ENTERL(00,01) ENTERA(01,DYNAMB)
OUT(LDA,**,LDA,**,STO) /* CODE TO PUT PROC ADDR IN DYNAMIC */
/* STORAGE AND LEAVE ADDRESS OF THAT */
/* ADDRESS ON THE STACK AS PARM ADDR */
.DO(A,FOUR,DYNAMB) /* UPDATE DYNAMIC STORAGE REQUIREMENT */
NEWLAB .DO(LB1) .SEARCH("1") /* LABEL FOR GLOBAL DISPLAY ADDRESS */
ENTERA(01,DYNAMB) ENTERL(00,01)
OUT(LDA,**,LD,0100000065,STO) /* CODE TO PUT CURRENT DISPLAY ADDR */
/* IN DYNAMIC STORAGE AS GLOBAL DISPLAY */
/* AT POINT OF CALL */
.DO(A,FOUR,DYNAMB) ;

IPROC := SAV(LD,01,**(04,01)) /* CODE TO LOAD ADDR OF PREVIOUSLY */
/* PASSED PROC ADDRESS */
.SAV(LD,61,**(04,01)) /* CODE TO LOAD PREVIOUSLY PASSED ADDR*/
NEWLAB .DO(LB1) .SEARCH("1")
ENTERL(00,01) ENTERA(01,DYNAMB)
OUT(LDA,**,LDA,**,STO) /* SIMILAR TO GPROC */
.DO(A,FOUR,DYNAMB)
NEWLAB .DO(LB1) .SEARCH("1") /* LABEL FOR GLOBAL DISPLAY */
ENTERA(01,DYNAMB) ENTERL(00,01)
•OUT(LDA,**)
  •DO(A,F0UR,DYNAMD)
  •NEWLAB
  •OUT(LDA,O1,*,1)

•OUT(#)
•OUT(LD,71,"0004",ADD)
•OUT(STU)
•OUT(LD,01)•DEFLAB(*1)
•ENTERL(00,01)
•OUT("0000")
•OUT(STU);

/* CODE TO LOAD ADDR FOR STORING
  GLOBAL DISPLAY */

/* CODE TO LOAD ADDR FOR STORING
  COMPUTED ADDR OF PASSFD GLOBAL
  DISPLAY ADDRESS */

/* RESTORE ADDR OF PROC ADDR CODE */

/* CODE TO LOAD ADDR OF GLOBAL DISPLAY ADDR */

/* CODE TO STORE IN NEXT INSTR */

/* CODE TO LOAD GLOBAL DISPLAY ADDR */

/* CODE TO STORE IN CURR DYNAMIC STOR */

DECLAR :="DECLARE" •ERRLATCH DECL1 •ERR("F",SCAN,";",SET)
$ ("," DECL1 •ERR("F",SCAN,";",SET) ) "":";

DECL1 :=IDELMNT / IDGROUP;

IDGF0UP:="(" LISTPT ;

LISTPT :=•ID •STKS1M:
  (ARRYPT
   •STACK(DIMC1T)
   IDLIST
   •UNSTACK(SYM SAV•DIMC1T)
   ARRYSEM
   /IDLIST
   •DO(POP).SEARCH("1")
   IDSEM
   •DO(A,LENGTH,DYNAMD) ) ; /* BUMP DYNAMIC STORAGE COUNTER */

IDLIST :="•" LISTPT / ")" ABOUT ; /* NOTE MUTUAL RECURSION WITH LISTPT */

ATTRIBT:=("CHARACTER(" / "CHAR()".INUM ")")
SET(LENGTH=SYMPTWO) /* TWELVE BIT (TWO CHARACTER) LENGTH */
SET(TYPE= 04) /* CHARACTER TYPE CODE */
/ "RETURNS( "ATTRIB")"
DO(A+OCTEN+TYPE)
SET(LENGTH=0000)
/ "FIXED" *SET(LENGTH=0004) /* FOUR CHAR LENGTH FOR BINARY INT */
SET(TYPE=01) /* BINARY INTEGER TYPE CODE */
/ "FLOAT" *SET(LENGTH=0010) /* EIGHT CHAR LENGTH */
SET(TYPE=02) /* FLOATING PT TYPE CODE */
/ "LOGICAL" *SET(LENGTH=0001) /* ONE CHAR LENGTH */
SET(TYPE=03) /* LOGICAL TYPE CODE */
/ "LABEL" *SET(LENGTH=0004) /* FOUR CHARACTER LABEL */
SET(TYPE=05); /* LABEL TYPE CODE */

ARRYPT := "(" *SET(DIMCNT=00) /* INITIALIZE DIMENSION COUNT */
BDPRLST ")" ;

BDPRLST := MARK BNDPAIR SAV /* SAVE BOUND PAIR CODE */
("*" MARK OUT(#) BNDPAIR SAV); /* Catenate BOUND PAIR CODE */

PNDPAIR := DO(A+ONE+DIMCNT) /* ACCUMULATE DIMENSION COUNT */
INTBND ("": OUT(#) INTBND OUT(#)
/ EMPTY OUT(7100000001,#) ) ; /* DEFAULT LOW BOUND ONE */

INTBND := ID SEARCH TSTTBL(00 01)
*ERR("W: INVALID ARRAY BOUND")
*SAV(**)
/ INUM SAV(71**); ;

IDE. MNK := ID STKSYM /* SIMILAR TO "LISTPT" BUT NO */
( ARRYPT ATTRIBT /* RECURSION WITH "IDLIST" */
UNSTACK(SYMSAV)
ARRYSEM
/ ATTRIBT DO(POP) SEARCH("1")
ICSEM DO(A+LENGTH+DYNAMB) ) ;

ARRYSEM := SET(SYMBOL=STACKTP)
OUT(ALLOC,**)
* OP CODE AND STACK TOP ADDRESS        */
* FOLLOWED BY SDV                        */
SET(SYMBOL=BLNK8)
* CLEAR SYMBOL TO BLANKS                 */
SET(SYMBOL=SYMSAV) SEARCH("1")          /* POSITION SYMTAB POINTER               */
ENTERA(07,DIMCNT)
* ENTER DIMENSION COUNT                  */
IDSLM
* ENTER TYPE, LENGTH AND ADDRESS         */
OUT(**)
* DYNAMIC DOPE VECTOR ADDRESS            */
OUT(**(01,07))
* DIMCNT ALSO TO SDV                     */
OUT(**(02,05))
* LENGTH TO SDV                         */
OUT(\)
* FINISH SDV WITH BOUND PAIR CODE         */
DO(A,D0PFIX,DYNAMB)
* ADD FIXED DOPE VECTOR SIZE            */
SET(SYMBOL=FOUR)
* ADJUST FOR MULTIPLIER STORAGE          */
DO(M,DIMCNT,SYMBOL)
* ADJUST FOR MULTIPLIER STORAGE          */
DO(A,SYMBOL,DYNAMB)
* ADJUST FOR MULTIPLIER STORAGE          */
DO(SET);
* ADJUST FOR MULTIPLIER STORAGE          */

IDSLM := ENTERA(00,TYPE)
* ENTER TYPE CODE                      */
ENTERA(05,LENGTH)
* ENTER LENGTH                        */
( .TEST(LENGTH=FUNCT) .RETURN
/ .EMPTY .ENTERA(01,DYNAMB)); /* ENTER LEVEL AND DISPLACEMENT       */

DOGROUP := ID .TEST(SYMBOL=DOSYM) .ERRLATCH
( ";;" TAIL / CASE / DOWHILE / LOOP )
.ERR("F: INVALID DO GROUP SYNTAX";CLM,TAIL);

TAIL := BLKBDY ENDING .ERR("w: INVALID DO GROUP END";SCAN;"","SET);

CASE := "CASE" .NEWLAB .ERRLATCH /* TRANSFER VECTOR LABEL         */
* NEWLAB                             /* LABEL FOR TV ADDRESS CONSTANT  */
EXP .ERR("F: BAD CASE EXPRESSION";SCAN;"","CLM,TAIL,R) ";"
OUT(LD,7100000005,MULT) /* CODE TO MULT BY TV ELEMENT SIZE  */
DO(LB1) .SEARCH("1") .ENTERL(00,01) /* MARK ACON AS AN INTEGER */
OUT(LD,**,ADD) /* CODE TO ADD ADDRESS CONSTANT */
OUT(JUMPA)
* NEWLAB                             /* CASE GROUP EXIT LABEL              */
* NEWLAB .DEFLAB(*1) /* STATEMENT LABEL AND VALUE */
STMENT /* COMPILE STATEMENT CODE */
DO WHILE := "WHILE" NEWLAB DEFLAB(*) NEWLAB WHLPT ";" TAIL
   OUT(JUMP*2) DEFLAB(*)
ENDING DEFLAB(*) ;

WHILE := "WHILE" WHLPT ";" ;

WHLPT :=
   "(" BOOLEXP • ERR("F: LOGICAL EXPRESSION ERROR", SCAN", ";", R)
   OUT(JUMPF*1) ")" ;

LOOP := ID UNDTST
   NEWLAB
   OUT(LDA,**)
   STKSYM
   ITRPRT
   OUT(JUMP*,1)
   (WHILE / ":" ) • ERR("W: EXPECTED ";", SET)
TAIL
   DO(Pop)
   OUT(LDA,**,LD,**)
   OUT(*)
   DEFLAB(*) ;
ITERPT:="=" ERR(LATCH EXP
  ERR("F: INVALID INITIAL INDEX") OUT(SST)
"TO"
  NEWLAB .DEFLAB(*1)
  MARK
(LATCH(LOCPRM)
/  NEWLAB .DEFLAB(*1)
  ENTERA(01*DYNAMB)
  ENTERL(00*07)
  DO(A*EIGHT*DYNAMB)
  OUT(LDA) .DO(LBI) OUT(**)
EXP .OUT(STO)
  MARK
  DEFLAB(*2)
  OUT(LD) .DO(LBI) OUT(**)
  DO(SwAP,POP)
) ERR("F: INVALID INDEX LIMIT")

"BY" .OUT(STCKC)
(="" .OUT(02) IOCHK .SAV(SUB) /* AND STACK PROPER OP CODE FOR */
/ .EMPTY .OUT(04) IOCHK .SAV(ADD) ) /* INCREMENTING LOOP INDEX */
  MARK
(LATCH(LGCPRM)
/ .DO(FOP) .SAV
  NEWLAB .DEFLAB(*1)
  ENTERA(01*DYNAMB)
  ENTERL(00*07)
  DO(A*EIGHT*DYNAMB)
  OUT(LDA) .DO(LBI) OUT(**)
EXP .OUT(STO)
  DEFLAB(*2)
  OUT(*)
  MARK
  OUT(LD) .DO(LBI) OUT(**)
  DO(SwAP,POP)
) ERR("F: INVALID INCREMENT")
OUT(#) /* RESTORE INCREMENT UP CODE */ / EMPTY OUT(STCKC+04) /* DEFAULT TEST */ ICCHK MARK
OUT(LD+71,*ADD) /* AND INCREMENT CODE */
OUT(SST::JUMP,*1) /* FINAL LOOP END CODE */ SAV ; /* SAVE ALL LOOP END CODE */
/* RESTORE ILIST CODE IF I/O ITERATION */

IOCHK := .TEST(IOSw=IOITER) .OUT(JUMPT,*2) .OUT(#) .DO(SET) / .EMPTY ;

LOPRM := (.ID .OUT(LD**))
.INUM .OUT(LD+71,**)
.FNUM .OUT(LD+72,**)
( ( [+/*] / ["-*/] / ["/* ]/ ["*/] / ["*/] )
 .DO(SetF.BEF) /* FORCE BACKUP IF NO SIMPLE PRIMARY */
 / .EMPTY ) ;

UNDST := .SEARCH .TSTTB(00,00)
 .DO(BM,"W: UNDECLARED IDENTIFIER") / .EMPTY ;

DUPTST := .SEARCH("1") .EMPTY .TSTTB(00,00) .ERR("W: MULTIPLE DECLARATION") ;

ENDING := "END" (.ID / .EMPTY ) ;

BOOLEXP := (.LATCh(IFCLSE) .NEWLAB .OUT(JUMP,*1)
.BTERM .NEWLAB .OUT(JUMP,*1)
.ELSE" .DEFLAB(*2) BOOLEXP
 .DEFLAB(*1) )
 / BTERM ;

BTERM := BFACTOR $ ( "OR" BFACTOR .OUT(OR) ) ;

BFACTOR := BSCNDRY $ ( "AND" BSCNDRY .OUT(AND) ) ;

BSCNDRY := BPRMRY/ "NOT" BPRMRY .OUT(NOT) ;
BPRMRY := BVALUE / LATCH(BVARBLE) / LATCH(BFNCT) / RELATN /
(" BOOLEXP ") . ERP("W: EXPECTED ") . SET) ;

RELATN := SAE RELOP SAE . OUT(STCKC*#) /
SSE RELOP SSE . OUT(COMPC*,#) ;

BFNCT := ID SEARCH("O") ( TSTTBL(00,13) / TSTTBL(00,23) ) . CANCEL CALLPRT ;

BVALUE := "T" OUT(LD,73,"T") / "F" . OUT(LD,73,"F") ;

BVARBLE := ID SEARCH("O") ( TSTTBL(00,03) / TSTTBL(00,63) ) . CANCEL
("( . OUT(LDA,**) SUBLIST ")" . OUT(INDXR*)
/ . EMPTY . OUT(LD,**) ) ;

RELOP := "=" . SAV(01) / "<=" . SAV(02) / "<" . SAV(02)
/ ">=" . SAV(04) / "#" . SAV(06) ;

SAE := ( TERM / "-" TERM . OUT(NEG) / "*" TERM )
& ( "+" TERM . OUT(ADD) / "-" TERM . OUT(SUB) ) ;

TERM := PRIMRY & ( "*" PRIMRY . OUT(MULT) / "/" PRIMRY . OUT(DIV) ) ;

PRIMRY := "(" EXP ")" / CONST / LATCH(VARBLE) / LATCH(AFUNCT) ;

AFUNCT := ID SEARCH("O") ( TSTTBL(00,11) / TSTTBL(00,12)
/ TSTTBL(00,21) / TSTTBL(00,22) ) . CANCEL CALLPRT ;

CONST := INUM . OUT(LD,71,**) / FNUM . OUT(LD,72,**) ;

VARBLE := ID SEARCH("O") ( TSTTBL(00,01) / TSTTBL(00,02)
/ TSTTBL(00,61) / TSTTBL(00,62) ) . CANCEL
("( . OUT(LDA,**) SUBLIST ")" . OUT(INDXR*) / . EMPTY . OUT(LD,**) ) ;

EXP := LATCH(IFCLSE) . NEWLAB . OUT(JUMPF*1) . SAE
"ELSE" . NEWLAB . OUT(JUMP*1)
```
*DEFLAB(*2) EXP*DEFLAB(*1)
/ SAE ;

SEXF := LATCH(IFCLSE) *NEWLAB* OUT(JUMPF*1) SSE
"ELSE" *NEWLAB* OUT(JUMPF*1)
*DEFLAB(*2) EXP*DEFLAB(*1)
/ SSE ;

SSE := STEM $ ( "//" STEM *OUT(CAT) ) ;

STEM := SBSTRNG / STRING *OUT(LDA,74,* ) / LATCH(SVARBLE) / LATCH(SFUNCT) ;

SFUNCT := ID SEARCH("0") ( TSTTBL(00,14) / TSTTBL(00,24) ) .CANCEL CALLPRT ;

SBSTRNG :="SUBSTR(" SEXP "," EXP
(""," EXP / EMPTY *OUT(LDA,71,"0000"))")" .OUT(SUBSTR) ;

SVARBLE := ID SEARCH("0") ( TSTTBL(00,04) / TSTTBL(00,64) ) .CANCEL
*OUT(LDA,**,**(02,05)) ( "(" SUBLIST ")" .OUT(INDXA,*)
/ EMPTY ) ;

SUBLIST := DO(MOVE,ONE,DIMCNT) /* INITIALIZE SUBSCRIPT COUNTER */
STKSYM /* SAVE IDENTIFIER */
EXP /* COMPILe SUBSCRIPT EXPRESSION */
$ ( "," EXP) DO(A,ONE,DIMCNT)
DO(POP) SEARCH /* RESTORE IDENTIFIER AND SYMTAB POINT*/
TSTTBA(07,DIMCNT) .ERR("W: INCORRECT SUBSCRIPT COUNT")
DO(MOVE,DIMCNT,SYMBOL) ; /* SUBSCRIPT COUNTER FOR CODE GEN */

LTERM := ID SEARCH("1")
( LBLCON / LBLVAR
/ ( "(" *OUT(LDA,05,**(04,01)) EXP $ ( ""," EXP")" .OUT(INDXR,**)
/ EMPTY *OUT(LDA,76,**(04,01)) ) ) ) ;

LBLCON := TSTTBL(00,06) .OUT(LDA,**);
```
LBLVAR := ( .TSTTLB(00,65) .SET(PEXITT=PEXITF) / .SET(PEXITT=00) .TSTTLB(00,05))
        ( "(" .OUT(LDA,***) .SUBLIST "")" .OUT(INDXR,*)
            / .EMPTY .OUT(LD,***) ) ;

CONDST := .LATCH(IFCSE) .NEWLAB .OUT(JUMPF,***)
          .SCSTM ( "ELSE" .NEWLAB .OUT(JUMPF,***)
            .DEFLAB(*2) STMENT .DEFLAE(***)
                / .EMPTY .DEFLAB(***) ) ;
IFCSE := .ID .TEST(SYMBOL=IFSYM) .CANCEL BOOLEXP "THEN" ;
IDENT := .ID .SEARCH("0") .TSTTLB(00,00) .DO(DM:"W: UNDECLARED VARIABLE") ;
UNCOND := .LATCH(GOTOS) / INOUT / .LATCH(CLLISTMT) / .LATCH(RTNSTMT)
            / STPSTMT / ASSGNST / IDENT ;
STPSTMT := "STOP" .OUT(STOP) ;

RTNSTMT := "RETURN" ( "(" ( EXP / SSE / BPRMRY ) ")" .OUT(SWP)
            / .EMPTY ) .OUT(RETURN) ;

GOTOS := "GO" "TO" .CANCEL .MARK LTERM
          ( .TEST(PEXITT=PEXITF) .SAV .OUT(POPUP,***) .OUT(RETURN)
            / .EMPTY .OUT(JUMPA) ) ;

ASSGNST := AASSGN / BASSGN / SASSGN / LASSGN ;

AASSGN := .LATCH(ALFTPT) .SAV(STO)
            $ ( ";" ALFTPT .MARK .OUT(STST,#) .SAV )
            "=" EXP .OUT(#) ;

BASSGN := .LATCH(BLFTPT) .SAV(STO)
            $ ( ";" BLFTPT .MARK .OUT(STST,#) .SAV )
            "=" BOOLEXP .OUT(#) ;

SASSGN := SLFTPT .SAV(STO)
            $ ( ";" SLFTPT .MARK .OUT(STST,#) .SAV )
            "=" SEXP .OUT(#) ;
LASSIGN := LATCH(LLFTPT) • SAV(STO)
          $ ( "" LLFTPT • MARK • OUT(SST*) • SAV )
          "=" LTFRM • TEST(P~XIT=ZERO) • ERR("F: INDIRECT LABEL ASSIGNMENT")
          • OUT()

ALFTPT := ID • SEARCH("O")
          ( • TSTTBL(00,01) / • TSTTBL(00,02)
            / • TSTTBL(00,61) / • TSTTBL(00,62) ) • CANCEL
          • OUT(LDA,**) SUBPART

BLFTPT := ID • SEARCH("O")
          ( • TSTTBL(00,03) / • TSTTBL(00,63) ) • CANCEL
          • OUT(LDA,**) SUBPART

LLFTPT := ID • SEARCH("O")
          ( • TSTTBL(00,05) / • TSTTBL(00,65) ) • CANCEL
          • OUT(LDA,**) SUBPART

SLFTPT := LATCH(SVARBIBLE) / LSUBSTR

LSUBSTR := SUBLIST(" SVARIBLE ") EXP
          ( "" EXP / • EMPTY • OUT(LD,71,"0000") ) ""
          • OUT(SUBSTR)

SUBPART := "(" SUBLIST ")" • OUT(INDXA**) / • EMPTY

INOUT := LATCH(INPUT) / • LATCH(OUTPUT)

INPUT := "GET" ( "" STRING" • SAV(70) "(" SLFTPT ")"
          / • EMPTY • SAV(71) )
          "EDIT" • CANCEL • NEWLAB • OUT(FMT,**1)
          "(" ILIST ")" • NEWLAB • OUT(JUMP,**1)
          • DEFLAB(**2) FMTLIST • DEFLAB(*1)

ILIST := IELMNT $ ( "", IELMNT ) ;
IELMN'T := IRPLIST
   / (.LATCH(SLFTPT) / .LATCH(ALFTPT) / .LATCH(BLFTPT) ) .OUT(GET) ;
IRPLIST:="(" .MARK ILMN'T .SAVE ERPLIST ;
ERPLIST:="DO" .ID UNDTST .NEWLAB /* "ERPLIST" IS SIMILAR TO "LOOP" */
   .OUT(LDA,**) .STKSYM
   .SET(IOSW=10ITER) /* SET SWITCH FOR ITERPRT TO TEST */
ITERPRT
   .SET(IOSW=00) /* RESTORE IOSW */
   .DO(POP) .OUT(LDA,**,LD,**)
   .OUT(#)
   .DEFLAB(*1) "") ;
OUTPUT := "PUT" ( "STRING" .SAVE(00) "(" SLFTPT ")"
   / .EMPTY .SAVE(01) )
   "EDIT" .CANCEL .NEWLAB .OUT(FMT,*1,*1)
   "(" OLIST ")" .OUT(PUT) .NEWLAB .OUT(JUMP,*1)
   .DEFLAB(*2) FMTLIST .DEFLAB(*1) ;
OLIST := OELMN'T $ ( "*" OELMN'T ) ;
OELMN'T := ORPLIST
   / ( EXP / SSE / .LATCH(BVARBLE) ) .OUT(EDIT) ;
ORPLIST:="(" .MARK OLIST .SAVE ERPLIST ;
FMTLIST:="(" .OUT(77) FMTITEM $ ( "*" FMTITEM ) ")" .OUT(77) ;
FMTITEM:= CTRLFMT / DATAFMT ;
CTRLFMT:="X" .OUT(00) SPEC
   / "PAGE" .OUT(01)
   / "SKIP" .OUT(02) ( SPEC / .EMPTY .OUT( "0001") )
   / "COL" .OUT(03) SPEC ;
DATAFMT:="A" .OUT(10) ( SPEC / .EMPTY .OUT( "0000") )
/ "E" OUT(11) SPEC
/ "I" OUT(12) SPEC / "L" OUT(13) SPEC ;

SPEC := "(" INUM OUT(*) ")" ;

*ENC

****COMPILED PROGRAM SIZE = 7,362; METAX INSTRUCTION COUNT = 128,483****

****SYMTAB SEARCH COUNT = 2,308; SYMTAB COMPARE COUNT = 857,907****

****SYMBOL TABLE ENTRY COUNT = 846****
B PLXCPL

CLM PROGRA

BT $002

BM "F: COMPILER ABORT"

EXIT

MOVI ELATCH "1"

PUSHLB ID

BF $005

TEST ".:"

BEF

STKSYM

MOVE SYMBOL

MOVI PRGNME

MOVI INTNME

"PLXINTO" "$ACTIVE"

SEARCH "0"

ENTL "0"

IOCHK

MOVE DYNAMP

DYNAMB

TEST "PROCEDURE"

BEF

TEST "MAIN"

BEF

OUT DYNAM

LB1

EVAL "4"

"1"
<table>
<thead>
<tr>
<th>MOVE</th>
<th>STACKT</th>
<th>SYMBOL</th>
<th>MOVE</th>
<th>STACKT</th>
<th>SYMBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEARCH</td>
<td>&quot;O&quot;</td>
<td></td>
<td>CLM</td>
<td>$015</td>
<td>BT</td>
</tr>
<tr>
<td>ENTL</td>
<td>&quot;0&quot;</td>
<td>&quot;51000&quot;</td>
<td>SET</td>
<td>$006</td>
<td>BT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BEF</td>
<td></td>
<td>BLKBDY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CLM</td>
<td>$006</td>
<td>BEF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$012</td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>$006</td>
<td>CLM</td>
<td>&quot;1&quot;</td>
<td>$018</td>
<td>CLM</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td></td>
<td>COMMENT</td>
<td></td>
<td>SET</td>
<td>$017</td>
<td>LATCH</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BEF</td>
<td></td>
<td>LABEL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TEST</td>
<td>&quot;END&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OUT</td>
<td>STOP</td>
<td></td>
</tr>
<tr>
<td>LB1</td>
<td>ENLOC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENL</td>
<td>&quot;O&quot;</td>
<td>&quot;6&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;1&quot;</td>
<td>DYNAMAP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENTA</td>
<td>$008</td>
<td>$009</td>
<td>$021</td>
<td>$022</td>
<td>$020</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CLM</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>BF</td>
<td>BM</td>
<td></td>
<td>CONDST</td>
<td>$020</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$020</td>
<td>$020</td>
</tr>
<tr>
<td></td>
<td>$009</td>
<td>$007</td>
<td>$017</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SET</td>
<td>BF</td>
<td>LATCH</td>
<td>BSCSTM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$007</td>
<td>$007</td>
<td></td>
<td></td>
<td>$017</td>
</tr>
<tr>
<td></td>
<td>BEF</td>
<td>$008</td>
<td>$028</td>
<td>LATCH</td>
<td>BLOCK</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BF</td>
<td>$028</td>
<td></td>
</tr>
<tr>
<td>$005</td>
<td>BLKBDY</td>
<td>MOVE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>BLNKH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>STMLAB</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
$029  CLM  UNCOND  BF  $030  BF  $039
  BF  $027  BF
$030  SET  BF  $027  BF
$027  BF  $026  BF  $030
  TEST  "1"
$026  R  "/*"
  TEST  BF  $033  SCAN  DOSYM
$033  R  ID  $035  BF  $036  TEST  SYMBOL
  "1"  MOVE  STMLAB
  MOVE  DYNAMB  "0"
$035  R  "0"  "6"
$036  ENDSTM  LATCH  ENDSTM  $037  SETF  $036  B
$037  SET  BF  $036  B
$036  R  TEST  "END"  BF  $040  SETF  $039  BEF
  BF  $039  BEF
$040  SET  CLM  BHDDBY
BF "0"
ENTL "0"
B $059
SET BEF
BLK\ LNT
MOVI ELATCH "1"
MOVE SYMBOL STMLAB
STKSYM MOVE DYNAMP SYMBOL
STKSYM MOVE DYNAMB SYMBOL
STKSYM MOVE LEVNO SYMBOL
STKSYM MOVE STACKT SYMBOL
STKSYM A ONE LEVNO
MOVE ONELEV DYNAMP
M LEVNO DYNAMP
MOVE DYNAMP DYNAMB
A ONELN4G STACKT
MOVE STACKT SYMBOL
SEARCH "1"
ENTA "1"
ENTL DYNAMB
"0"
"5"
MOVE FOUR SYMBOL
M LEVNO SYMBOL
A FOUR SYMBOL
A SYMBOL DYNAMB
OUT ENTPRO MOVE LEVNO SYMBOL
OUTSYM OUT DYNAM
LBI EVAL "4"
"1"
MOVE ZERO ARGCNT
TEST "(" BF $064
CLM ARGID BEF
TEST "," BF $065
CLM ARGID BEF
TEST ")" BT $067
SET BEF
TEST BEF
BM "w: POSS
SET BEF
$080
TEST "/;
BEF
R
$058
TARGETC
COMP ARGCNT
ZERO "2"
BF SETF
R $082
"21*
DYNAMB
$087 R ARGARY TEST "(" BF $090
MOVE ZERO DIMCNT "*
ST $087

ARGID ID
BF $085
SEARCH "1"
ENTA "1"
ENTL "0"
"<"
A ONE ARGCNT
A FOUR DYNAMB
$093 BT SET
BEF TEST ")

ARGDEC ID
BF $087
SEARCH "1"
MOVI ELATCH "1"
TSTTBL "0"
"<"
BT $088
BM "F: NON-E
$080
CLM ARGARY
BEF
CLM ARGATR
BEF
A ONE ADECNT
BEF

$090 SET
BEF
$091 TEST ","
BEF
TEST "*
BEF
A ONE DIMCNT
BEF

ARGATR CLM ATTRIB
BF $096
ENTA "5"

LENGTH
A OCTL60
ENTL "0" "6"

$104 R
CLLPRM OUT FLAG PARMCN SYMBOL
MOVE ZERO PARMCN SYMBOL

SYM
MOVE STMLAB SYMBOL
STKSYM CLM PARMPR
BF $124
B $123

$124 SET
BF $123
$123 BF $122
POP SEARCH "0"
TSTTB6L "0"
BF $129
B $128

$129 TSTTB6L "0"
BF $130
B $128
$130 TSTTB6L "0"
BF $131
B $128
$131 TSTTB6L "0"
BF $128

$128 BF $127

TSTTB6A "7"
PARMCN
BT $133
BM "W: INCORRECT PAR
SET B $126
BF $126
BEF
POP
MOVE SYMBOL PARMCN

STKSYM CLM PARMPR
BF $133
BM "W: INCORRECT PAR
SET B $126
BEF $126
BEF
POP
MOVE SYMBOL PARMCN

$122 R
PARMPR TEST "("
BF $136
CLM PARM
BEF

$137 TEST ",,"
BF $139
CLM PARM
BEF

$139 BT $137
SET
BEF
TEST ")".
BEF

$136 R
PARM CLM PARMID
BF $143
B $142

$143 CLM PARMID
BF $142
BEF $141
A ONE
PARMCN

$141 R
PARMID LATCH ALFTPT
BF $146
B $145
LATCH BLFTPT
BF $147
B $145
LATCH SLFTPT
BF $148
B $145
LATCH LLFTPT
BF $149
B $145
LATCH PROCHK
BF $145
R $152
PROCHK ID
BF $152
TEST "(" BF $154
SETF BF $153
SET BF $153
BEF $154
SEARCH "0" TSTTbL "0" "8"
BF $157
CANCEL CLM GPROC
BEF $156
B $156
TSTTbL "0" "++"
BF $156
CANCEL CLM IPROC

BEF $156
BEF $152
R $152
PARMEX PUSHLB
LB1 SEARCH "1"
ENTA "1"
DYNAMB
ENTL "0"
"7"
A EIGHT
DYNAMB
OUT LDA
EVAL "5"
"0"
OUT LDA
EVAL "5"
"0"
CLM EXP
BF $162
E $161
CLM SSE
BF $163
B $161
CLM LTERM
BF $164
B $161
CLM BPRMRY
BF $161
BF $160
OUT STO
R $160
GPROC MARK
OUT LDA
OUT "1"
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTL</td>
<td>DYNAMB</td>
</tr>
<tr>
<td>ENTL</td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td>ENTL</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>OUT</td>
<td>LDA</td>
</tr>
<tr>
<td>EVAL</td>
<td>&quot;5&quot;</td>
</tr>
<tr>
<td>A</td>
<td>FOUR</td>
</tr>
<tr>
<td>DYNAMB</td>
<td></td>
</tr>
<tr>
<td>PUSHLB</td>
<td>LDA</td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>LBI</td>
<td>EVAL</td>
</tr>
<tr>
<td>&quot;4&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;1&quot;</td>
<td></td>
</tr>
<tr>
<td>RESTOR</td>
<td>OUT</td>
</tr>
<tr>
<td>LD</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;Z&quot;</td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;0004&quot;</td>
</tr>
<tr>
<td>OUT</td>
<td>ADD</td>
</tr>
<tr>
<td>OUT</td>
<td>STO</td>
</tr>
<tr>
<td>OUT</td>
<td>LD</td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>LBI</td>
<td>ENTLLOC</td>
</tr>
<tr>
<td>ENTL</td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td>ENTL</td>
<td>&quot;6&quot;</td>
</tr>
<tr>
<td>ENTL</td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td>ENTL</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;0000&quot;</td>
</tr>
<tr>
<td>OUT</td>
<td>STO</td>
</tr>
<tr>
<td>DECLAR</td>
<td>TEST</td>
</tr>
<tr>
<td>&quot;DECLARE&quot;</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$171</td>
</tr>
<tr>
<td>MOV1</td>
<td>ELATCH</td>
</tr>
<tr>
<td>$168</td>
<td>R</td>
</tr>
<tr>
<td>$172</td>
<td>BT</td>
</tr>
<tr>
<td>$175</td>
<td>BT</td>
</tr>
<tr>
<td>$172</td>
<td>BT</td>
</tr>
<tr>
<td>$175</td>
<td>BT</td>
</tr>
<tr>
<td>$171</td>
<td>R</td>
</tr>
<tr>
<td>$175</td>
<td>BT</td>
</tr>
<tr>
<td>$172</td>
<td>BT</td>
</tr>
<tr>
<td>$178</td>
<td>CLM</td>
</tr>
<tr>
<td>$177</td>
<td>R</td>
</tr>
<tr>
<td>$177</td>
<td>R</td>
</tr>
<tr>
<td>$181</td>
<td>R</td>
</tr>
<tr>
<td>$183</td>
<td>BF</td>
</tr>
<tr>
<td>$185</td>
<td>BF</td>
</tr>
<tr>
<td>MOVE</td>
<td>DIMCNT</td>
</tr>
<tr>
<td>SYMBOL</td>
<td></td>
</tr>
</tbody>
</table>
STKSYM
CLM IDLIST
BEF
POP
MOVE SYMBOL
DIMCNT
POP
MOVE SYMBOL SYMSAV
CLM ARRAYSE
BEF B $184
$185 CLM IDLIST
BF $184
POP
SEARCH "1"
CLM IDSEM
BEF A LENGTH
DYNAMB
$184 BEF
$183 R
IDLIST TEST ","
BF $188
CLM LISTPT
BEF B $187
$188 TEST ")"
BF $187
CLM ATTRIB
BEF
$187 R
ATTRIB TEST "CHARACTER"
BF $193
B $192
$193 TEST "CHAR(" 
BF $192
$192 BF $191
INUM
BEF
TEST ")
MOVE SYMPTW
LENGTH
MOVI TYPE
"4"
B $190
TEST "RETURNS"
BF $195
CLM ATTRIB
BEF
TEST ")"
A OCTEN
TYPE
MOVI LENGTH
"00"
B $190
TEST "FIXED"
BF $196
MOVI LENGTH
LEVNO
MOVI TYPE
"1"
B $190
TEST "FLOAT"
BF $197
MOVI LENGTH
LEVNO
MOVI TYPE
"2"
B $190
TEST "LOGICAL"
BF $198
<table>
<thead>
<tr>
<th>COMP</th>
<th>SYMBOLE</th>
<th>DOSYM</th>
<th>&quot;2&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEF</td>
<td>MOVI</td>
<td>ELATCH</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>BF</td>
<td>$231</td>
<td>CLM</td>
<td>TAIL</td>
</tr>
<tr>
<td>BEF</td>
<td>$230</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$231</td>
<td>CLM</td>
<td>CASE</td>
<td></td>
</tr>
<tr>
<td>$232</td>
<td>CLM</td>
<td>DOWHIL</td>
<td></td>
</tr>
<tr>
<td>$233</td>
<td>CLM</td>
<td>LOOP</td>
<td></td>
</tr>
<tr>
<td>$230</td>
<td>BT</td>
<td>$229</td>
<td></td>
</tr>
<tr>
<td>BM</td>
<td>&quot;F: INVALID&quot;</td>
<td>CLM</td>
<td>TAIL</td>
</tr>
<tr>
<td>$229</td>
<td>R</td>
<td>BLKBDY</td>
<td></td>
</tr>
<tr>
<td>TAIL</td>
<td>CLM</td>
<td>ENDING</td>
<td></td>
</tr>
<tr>
<td>$237</td>
<td>P</td>
<td>&quot;CASE&quot;</td>
<td></td>
</tr>
<tr>
<td>CASE</td>
<td>TEST</td>
<td>&quot;CASE&quot;</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$240</td>
<td>PUSHLB</td>
<td></td>
</tr>
<tr>
<td>PUSHLB</td>
<td>MOVI</td>
<td>ELATCH</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>$237</td>
<td>P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td>&quot;$241&quot;</td>
<td>TEST</td>
<td>&quot;;&quot;</td>
</tr>
<tr>
<td>BF</td>
<td>$231</td>
<td>CLM</td>
<td>TAIL</td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$230</td>
<td>CLM</td>
<td>CASE</td>
<td></td>
</tr>
<tr>
<td>$232</td>
<td>CLM</td>
<td>DOWHIL</td>
<td></td>
</tr>
<tr>
<td>$233</td>
<td>CLM</td>
<td>LOOP</td>
<td></td>
</tr>
<tr>
<td>$230</td>
<td>BT</td>
<td>$229</td>
<td></td>
</tr>
<tr>
<td>BM</td>
<td>&quot;F: INVALID&quot;</td>
<td>CLM</td>
<td>TAIL</td>
</tr>
<tr>
<td>$229</td>
<td>R</td>
<td>BLKBDY</td>
<td></td>
</tr>
<tr>
<td>TAIL</td>
<td>CLM</td>
<td>ENDING</td>
<td></td>
</tr>
<tr>
<td>$237</td>
<td>P</td>
<td>&quot;CASE&quot;</td>
<td></td>
</tr>
<tr>
<td>CASE</td>
<td>TEST</td>
<td>&quot;CASE&quot;</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$240</td>
<td>PUSHLB</td>
<td></td>
</tr>
<tr>
<td>PUSHLB</td>
<td>MOVI</td>
<td>ELATCH</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>$237</td>
<td>P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td>&quot;$241&quot;</td>
<td>TEST</td>
<td>&quot;;&quot;</td>
</tr>
<tr>
<td>BF</td>
<td>$231</td>
<td>CLM</td>
<td>TAIL</td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$230</td>
<td>CLM</td>
<td>CASE</td>
<td></td>
</tr>
<tr>
<td>$232</td>
<td>CLM</td>
<td>DOWHIL</td>
<td></td>
</tr>
<tr>
<td>$233</td>
<td>CLM</td>
<td>LOOP</td>
<td></td>
</tr>
<tr>
<td>$230</td>
<td>BT</td>
<td>$229</td>
<td></td>
</tr>
<tr>
<td>BM</td>
<td>&quot;F: INVALID&quot;</td>
<td>CLM</td>
<td>TAIL</td>
</tr>
<tr>
<td>$229</td>
<td>R</td>
<td>BLKBDY</td>
<td></td>
</tr>
<tr>
<td>TAIL</td>
<td>CLM</td>
<td>ENDING</td>
<td></td>
</tr>
<tr>
<td>$237</td>
<td>P</td>
<td>&quot;CASE&quot;</td>
<td></td>
</tr>
<tr>
<td>CASE</td>
<td>TEST</td>
<td>&quot;CASE&quot;</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$240</td>
<td>PUSHLB</td>
<td></td>
</tr>
<tr>
<td>PUSHLB</td>
<td>MOVI</td>
<td>ELATCH</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>$237</td>
<td>P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td>&quot;$241&quot;</td>
<td>TEST</td>
<td>&quot;;&quot;</td>
</tr>
<tr>
<td>BF</td>
<td>$231</td>
<td>CLM</td>
<td>TAIL</td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$230</td>
<td>CLM</td>
<td>CASE</td>
<td></td>
</tr>
<tr>
<td>$232</td>
<td>CLM</td>
<td>DOWHIL</td>
<td></td>
</tr>
<tr>
<td>$233</td>
<td>CLM</td>
<td>LOOP</td>
<td></td>
</tr>
<tr>
<td>$230</td>
<td>BT</td>
<td>$229</td>
<td></td>
</tr>
<tr>
<td>BM</td>
<td>&quot;F: INVALID&quot;</td>
<td>CLM</td>
<td>TAIL</td>
</tr>
<tr>
<td>$229</td>
<td>R</td>
<td>BLKBDY</td>
<td></td>
</tr>
<tr>
<td>TAIL</td>
<td>CLM</td>
<td>ENDING</td>
<td></td>
</tr>
<tr>
<td>$237</td>
<td>P</td>
<td>&quot;CASE&quot;</td>
<td></td>
</tr>
<tr>
<td>CASE</td>
<td>TEST</td>
<td>&quot;CASE&quot;</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$240</td>
<td>PUSHLB</td>
<td></td>
</tr>
<tr>
<td>PUSHLB</td>
<td>MOVI</td>
<td>ELATCH</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>$237</td>
<td>P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td>&quot;$241&quot;</td>
<td>TEST</td>
<td>&quot;;&quot;</td>
</tr>
<tr>
<td>BF</td>
<td>$231</td>
<td>CLM</td>
<td>TAIL</td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$230</td>
<td>CLM</td>
<td>CASE</td>
<td></td>
</tr>
<tr>
<td>$232</td>
<td>CLM</td>
<td>DOWHIL</td>
<td></td>
</tr>
<tr>
<td>$233</td>
<td>CLM</td>
<td>LOOP</td>
<td></td>
</tr>
<tr>
<td>$230</td>
<td>BT</td>
<td>$229</td>
<td></td>
</tr>
<tr>
<td>BM</td>
<td>&quot;F: INVALID&quot;</td>
<td>CLM</td>
<td>TAIL</td>
</tr>
<tr>
<td>$229</td>
<td>R</td>
<td>BLKBDY</td>
<td></td>
</tr>
<tr>
<td>TAIL</td>
<td>CLM</td>
<td>ENDING</td>
<td></td>
</tr>
<tr>
<td>$237</td>
<td>P</td>
<td>&quot;CASE&quot;</td>
<td></td>
</tr>
<tr>
<td>CASE</td>
<td>TEST</td>
<td>&quot;CASE&quot;</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$240</td>
<td>PUSHLB</td>
<td></td>
</tr>
<tr>
<td>PUSHLB</td>
<td>MOVI</td>
<td>ELATCH</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>$237</td>
<td>P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td>&quot;$241&quot;</td>
<td>TEST</td>
<td>&quot;;&quot;</td>
</tr>
<tr>
<td>BF</td>
<td>$231</td>
<td>CLM</td>
<td>TAIL</td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$230</td>
<td>CLM</td>
<td>CASE</td>
<td></td>
</tr>
<tr>
<td>$232</td>
<td>CLM</td>
<td>DOWHIL</td>
<td></td>
</tr>
<tr>
<td>$233</td>
<td>CLM</td>
<td>LOOP</td>
<td></td>
</tr>
<tr>
<td>$230</td>
<td>BT</td>
<td>$229</td>
<td></td>
</tr>
<tr>
<td>BM</td>
<td>&quot;F: INVALID&quot;</td>
<td>CLM</td>
<td>TAIL</td>
</tr>
<tr>
<td>$229</td>
<td>R</td>
<td>BLKBDY</td>
<td></td>
</tr>
<tr>
<td>TAIL</td>
<td>CLM</td>
<td>ENDING</td>
<td></td>
</tr>
<tr>
<td>$237</td>
<td>P</td>
<td>&quot;CASE&quot;</td>
<td></td>
</tr>
<tr>
<td>CASE</td>
<td>TEST</td>
<td>&quot;CASE&quot;</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$240</td>
<td>PUSHLB</td>
<td></td>
</tr>
<tr>
<td>PUSHLB</td>
<td>MOVI</td>
<td>ELATCH</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>$237</td>
<td>P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td>&quot;$241&quot;</td>
<td>TEST</td>
<td>&quot;;&quot;</td>
</tr>
<tr>
<td>BF</td>
<td>$231</td>
<td>CLM</td>
<td>TAIL</td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$230</td>
<td>CLM</td>
<td>CASE</td>
<td></td>
</tr>
<tr>
<td>$232</td>
<td>CLM</td>
<td>DOWHIL</td>
<td></td>
</tr>
<tr>
<td>$233</td>
<td>CLM</td>
<td>LOOP</td>
<td></td>
</tr>
<tr>
<td>$230</td>
<td>BT</td>
<td>$229</td>
<td></td>
</tr>
<tr>
<td>BM</td>
<td>&quot;F: INVALID&quot;</td>
<td>CLM</td>
<td>TAIL</td>
</tr>
<tr>
<td>$229</td>
<td>R</td>
<td>BLKBDY</td>
<td></td>
</tr>
<tr>
<td>TAIL</td>
<td>CLM</td>
<td>ENDING</td>
<td></td>
</tr>
<tr>
<td>$237</td>
<td>P</td>
<td>&quot;CASE&quot;</td>
<td></td>
</tr>
<tr>
<td>CASE</td>
<td>TEST</td>
<td>&quot;CASE&quot;</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$240</td>
<td>PUSHLB</td>
<td></td>
</tr>
<tr>
<td>PUSHLB</td>
<td>MOVI</td>
<td>ELATCH</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>$237</td>
<td>P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td>&quot;$241&quot;</td>
<td>TEST</td>
<td>&quot;;&quot;</td>
</tr>
<tr>
<td>BF</td>
<td>$231</td>
<td>CLM</td>
<td>TAIL</td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$230</td>
<td>CLM</td>
<td>CASE</td>
<td></td>
</tr>
<tr>
<td>$232</td>
<td>CLM</td>
<td>DOWHIL</td>
<td></td>
</tr>
<tr>
<td>$233</td>
<td>CLM</td>
<td>LOOP</td>
<td></td>
</tr>
<tr>
<td>$230</td>
<td>BT</td>
<td>$229</td>
<td></td>
</tr>
<tr>
<td>BM</td>
<td>&quot;F: INVALID&quot;</td>
<td>CLM</td>
<td>TAIL</td>
</tr>
<tr>
<td>$229</td>
<td>R</td>
<td>BLKBDY</td>
<td></td>
</tr>
<tr>
<td>TAIL</td>
<td>CLM</td>
<td>ENDING</td>
<td></td>
</tr>
<tr>
<td>$237</td>
<td>P</td>
<td>&quot;CASE&quot;</td>
<td></td>
</tr>
<tr>
<td>CASE</td>
<td>TEST</td>
<td>&quot;CASE&quot;</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$240</td>
<td>PUSHLB</td>
<td></td>
</tr>
<tr>
<td>PUSHLB</td>
<td>MOVI</td>
<td>ELATCH</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>$237</td>
<td>P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td>&quot;$241&quot;</td>
<td>TEST</td>
<td>&quot;;&quot;</td>
</tr>
<tr>
<td>BF</td>
<td>$231</td>
<td>CLM</td>
<td>TAIL</td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$230</td>
<td>CLM</td>
<td>CASE</td>
<td></td>
</tr>
<tr>
<td>$232</td>
<td>CLM</td>
<td>DOWHIL</td>
<td></td>
</tr>
<tr>
<td>$233</td>
<td>CLM</td>
<td>LOOP</td>
<td></td>
</tr>
<tr>
<td>$230</td>
<td>BT</td>
<td>$229</td>
<td></td>
</tr>
<tr>
<td>BM</td>
<td>&quot;F: INVALID&quot;</td>
<td>CLM</td>
<td>TAIL</td>
</tr>
<tr>
<td>$229</td>
<td>R</td>
<td>BLKBDY</td>
<td></td>
</tr>
<tr>
<td>TAIL</td>
<td>CLM</td>
<td>ENDING</td>
<td></td>
</tr>
<tr>
<td>$237</td>
<td>P</td>
<td>&quot;CASE&quot;</td>
<td></td>
</tr>
<tr>
<td>CASE</td>
<td>TEST</td>
<td>&quot;CASE&quot;</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$240</td>
<td>PUSHLB</td>
<td></td>
</tr>
<tr>
<td>PUSHLB</td>
<td>MOVI</td>
<td>ELATCH</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>$237</td>
<td>P</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LBl
EVAL
"4"
II2^ II

$242

SAVE
pop
PUShLB
LBl
ENTLOC
ENTL

CLM
bF
OUT
LB2
EVAL

SWAP
POP
L62
EVAL
"4"
"1"
"0"
"6"
STMENT
S244
JUMP

"4"
"1"
" MARK
RESTOR
OUT
LBl
EVAL

1244

LB2
ENTLOC
ENTL
"0"
"6"
RESTOR
CLM
BEF
LBl
ENTLOC
ENTL

JUMP

"4"
"1"
SAVE
POP
BT
SET
BEF
POP
OUT
OUT
LB2
ENTLOC
ENTL

ENTL

"0"
"6"
"0"
"1 "

$242

ENDING

"0"
"6"
5.240
R
DOwHIL TEST
BF
PUSMLB
LBl
ENTLOC
ENTL

"WHILE"
$246

"0"
"6"
LD
"6"

PUShLB
CLM
BEF
TEST
BEF

WHLPT
"i"


ENTL "0" LB1 EVAL "5"
"6" "0"

$253 R TEST "-" CLM EXP
BF $259 BF $261
MOVI ELATCH OUT STO
"1" MARK
CLM EXP LB2
BT $260 ENTLOCLTL
BM "F: INVALID ENTL

$260 OUT SST "0"
TEST "TO"
BEF OUT LD
PUSHLB LB1 LBL1 EVAL
ENTL "5"
ENTL "0"

MARK $261 STO $264
LATCH LOOPRM BT $266
BF $262 TEST "BY"
BM "F: INVALID INDEX $266
G $264 TEST "-"

$262 PUSHLB TEST "$266
LB1 $267 LBL1
ENTLCE TEST "$268
ENTL "0"

"6"

ENTA "1"
ENTL "0"
"7"

A EIGHT $268 SET $267
DYNAMB BF $267
OUT LDA OUT "$4"
CLM IOCHK
BEF
MARK
OUT ADD
SAVE BEF MARK
LATCH LOOPRM BF $271 B $270 $270 BT $273
POP SAVE PUSHLB LB1 ENTL
ENTL "0"
"6"
"1"
ENTL "0"
"7"
A EIGHT DYNAMB OUT "$265 BEF"
OUT "$265 STCKC"
OUT "$0001"
OUT "$270"
OUT "$276"
EVAL LB1
"5"
"0"
"4"
"1"
CLM EXP BF $270 OUT ST0 OUT "$270"
OUT ST0
SAVE LB2 $259 R
ENTLOC I0CHK COMP I05w IOITER "$270"
"6"
"0"
BF $276
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT</td>
<td>JUMPT</td>
<td>BF</td>
<td>$289</td>
</tr>
<tr>
<td>LB2</td>
<td></td>
<td></td>
<td>$286</td>
</tr>
<tr>
<td>EVAL</td>
<td>&quot;4&quot;</td>
<td>BF</td>
<td>$289</td>
</tr>
<tr>
<td></td>
<td>&quot;1&quot;</td>
<td>B</td>
<td>$286</td>
</tr>
<tr>
<td>RESTOR</td>
<td></td>
<td>TEST</td>
<td>&quot;/&quot;</td>
</tr>
<tr>
<td>SET</td>
<td></td>
<td>BF</td>
<td>$290</td>
</tr>
<tr>
<td>B</td>
<td>$275</td>
<td></td>
<td>$286</td>
</tr>
<tr>
<td>SET</td>
<td>BF</td>
<td>$286</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$275</td>
<td></td>
<td>$285</td>
</tr>
<tr>
<td>R</td>
<td>$275</td>
<td>SET</td>
<td></td>
</tr>
<tr>
<td>LOOPRM</td>
<td>ID</td>
<td></td>
<td>$284</td>
</tr>
<tr>
<td>BF</td>
<td>$281</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>LD</td>
<td>BF</td>
<td>$284</td>
</tr>
<tr>
<td>EVAL</td>
<td>&quot;5&quot;</td>
<td>BEF</td>
<td>$285</td>
</tr>
<tr>
<td></td>
<td>&quot;0&quot;</td>
<td></td>
<td>$286</td>
</tr>
<tr>
<td>B</td>
<td>$280</td>
<td></td>
<td>$286</td>
</tr>
<tr>
<td>INUM</td>
<td>BF</td>
<td>$282</td>
<td>SET</td>
</tr>
<tr>
<td>OUT</td>
<td>LD</td>
<td>BF</td>
<td>$283</td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;2&quot;</td>
<td>BM</td>
<td>&quot;W: UNDECLARED IDE</td>
</tr>
<tr>
<td>OUTSYM</td>
<td>E</td>
<td>BF</td>
<td>$284</td>
</tr>
<tr>
<td>$281</td>
<td>FNUM</td>
<td>BF</td>
<td>$285</td>
</tr>
<tr>
<td>OUT</td>
<td>LD</td>
<td>BF</td>
<td>$286</td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;e&quot;</td>
<td>TSTTBL</td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td>OUTSYM</td>
<td>BF</td>
<td>$287</td>
<td>DUPTST</td>
</tr>
<tr>
<td>$282</td>
<td></td>
<td>BF</td>
<td>$288</td>
</tr>
<tr>
<td>TEST</td>
<td>&quot;+&quot;</td>
<td>BM</td>
<td>&quot;W: MULTIPLE DECLA</td>
</tr>
<tr>
<td>B</td>
<td>$286</td>
<td>$289</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>TEST</td>
<td>&quot;-&quot;</td>
<td>RT</td>
<td>$290</td>
</tr>
<tr>
<td>B</td>
<td>$286</td>
<td>SET</td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td>&quot;(&quot;</td>
<td>BF</td>
<td>$287</td>
</tr>
<tr>
<td>B</td>
<td>$288</td>
<td>TSTTBL</td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td>END</td>
<td>BF</td>
<td>$289</td>
<td>END</td>
</tr>
<tr>
<td>ID</td>
<td>BF</td>
<td>$300</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$302</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$323</td>
<td>LATCH</td>
<td>BVARBL</td>
<td>BF</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
<td>--------</td>
<td>----</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>$324</td>
<td>LATCH</td>
<td>BFNCT</td>
<td>BF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>$325</td>
<td>CLM</td>
<td>RELATN</td>
<td>BF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>$326</td>
<td>TEST</td>
<td>&quot;(&quot;</td>
<td>BF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CLM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BEF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TEST</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SET</td>
</tr>
<tr>
<td></td>
<td>RELATN</td>
<td>SAE</td>
<td>BF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CLM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CLM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BEF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OUT</td>
</tr>
<tr>
<td></td>
<td>RESTOR</td>
<td></td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CLM</td>
</tr>
<tr>
<td>$330</td>
<td>BF</td>
<td>$329</td>
<td>CLM</td>
</tr>
<tr>
<td></td>
<td>CLM</td>
<td>SSE</td>
<td>BEF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BEF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OUT</td>
</tr>
<tr>
<td>$329</td>
<td>BFNCT</td>
<td>ID</td>
<td>BF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CANCEL
TEST "(" OUT "2"
BF $346 SAVE B $343
OUT LDA $351 TEST "\geq"
EVAL BF $352
"5" MARK OUT "5"
"0"
CLM SUBL15 SAVE B $348
BEF TEST ")" $352 TEST "\rangle"
BEF BF $353
OUT INDXR MARK OUT "4"
OUTSYM SAVE B $348
$346 SET BF $345 $353 TEST "\neq"
$345 OUT LD BF $348
EVAL MARK OUT "6"
"5"
"0"
$345 SAVE B $350
$341 R R $348 CLM TERM
RELOP TEST ";=" $351
BF $349 BF $357
MARK B $358
OUT "1"
SAVE EF $359
B $348 CLM TERM
$349 TEST ";\leq"
BF $350 BEF
MARK OUT NEG B $357
SAVE $359 TEST ";>"
B $348 CLM TERM
$350 TEST ";<"
BF $357 BEF
MARK B $356
$351 TEST ";\neq"
BF $363
CLM  TERM
BEF  OUT  ADD
  $362 TEST  "-"
BF  $362 CLM  TERM
BEF  OUT  SUB
  $362 BT  $361
SET  BEF
TERM
  $366 R
CLM  PRIMRY
BEF  $366
TEST  "*"
BF  $369 CLM  PRIMRY
BEF  OUT  MULT
  $368 B  $368
TEST  "/"
BF  $368 CLM  PRIMRY
BEF  OUT  DIV
  $367 BT  $367
SET  BEF
  $366 R
PRIMRY
TEST  "("  $372 CLM  CONST
BF  $373 BEF
B  $371
$373 LATCH  VARIABLE
BF  $374 BEF
B  $371
$374 LATCH  AFUNCT
BF  $371
$371 R
AFUNCT
ID
BF  $377
SEARCH  "0"
TSTTBL  "0"
$379
BF  $378
B  $379
$379 TSTTBL  "0"
BF  $380
B  $378
$380 TSTTBL  "0"
BF  $381
E  $378
$381 TSTTBL  "0"
BF  $378
BEF  $378
CANCEL
CLM  CALLPR
BEF
$377 R
CONST
INUM
BF  $384
OUT  LD
OUT  "Z"
OUTSYM
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Address</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTLOC</td>
<td></td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>ENTL</td>
<td></td>
<td>$400</td>
<td>$399</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>$396</td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td></td>
<td>$399</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td></td>
<td>SSE</td>
<td></td>
</tr>
<tr>
<td>SSE</td>
<td></td>
<td>STERM</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td></td>
<td>$403</td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td></td>
<td>//</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>$406</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>CAT</td>
<td></td>
</tr>
<tr>
<td>SFUNCT</td>
<td></td>
<td>$404</td>
<td></td>
</tr>
<tr>
<td>STERN</td>
<td></td>
<td>$407</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>$408</td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td></td>
<td>ELSE</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>$403</td>
<td></td>
</tr>
<tr>
<td>SFUNCT</td>
<td></td>
<td>$402</td>
<td></td>
</tr>
<tr>
<td>LB1</td>
<td></td>
<td>JUMPF</td>
<td></td>
</tr>
<tr>
<td>EVAL</td>
<td></td>
<td>$407</td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td></td>
<td>SSE</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>S$BSTRN</td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td></td>
<td>STRTST</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td></td>
<td>$409</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td></td>
<td>LDA</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td></td>
<td>&quot;(&quot;</td>
<td></td>
</tr>
<tr>
<td>OUTSYM</td>
<td></td>
<td>$407</td>
<td></td>
</tr>
<tr>
<td>LB2</td>
<td></td>
<td>$409</td>
<td></td>
</tr>
<tr>
<td>ENTLOC</td>
<td></td>
<td>LATCH</td>
<td></td>
</tr>
<tr>
<td>ENTL</td>
<td></td>
<td>SFUNCT</td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td></td>
<td>$410</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>S$BSTRN</td>
<td></td>
</tr>
<tr>
<td>LB1</td>
<td></td>
<td>SEARCH</td>
<td></td>
</tr>
<tr>
<td>SFUNCT</td>
<td></td>
<td>$407</td>
<td></td>
</tr>
<tr>
<td>LB1</td>
<td></td>
<td>&quot;0&quot;</td>
<td></td>
</tr>
<tr>
<td>ENTLOC</td>
<td></td>
<td>TSTTBL</td>
<td></td>
</tr>
<tr>
<td>ENTL</td>
<td></td>
<td>&quot;0&quot;</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>&quot;$&quot;</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td></td>
<td>$413</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>&quot;$&quot;</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td></td>
<td>$415</td>
<td></td>
</tr>
</tbody>
</table>
$415 \quad B \quad $414
TSTTBL \quad "0"
 BF \quad $425
 $424
TSTTBL \quad "0"
 $425
BF \quad $424

$414 \quad BEF
CANCEL
CLM \quad CALLPR
BEF

$413 \quad R
STRTN
TEST \quad "SUBSTR("$418
BF \quad $419
CLM \quad SEXP
BEF
TEST \quad ","
BEF
CLM \quad EXP
BEF
TEST \quad ","
BEF
CLM \quad EXP
BEF
BEF

$420 \quad SET
BF \quad $419
OUT \quad LD
OUT \quad "Z"
OUT \quad "0000"

$419 \quad BEF
TEST \quad ")"
BEF
OUT \quad SUBSTR

$418 \quad R
STRTN
ID
BF \quad $423
SEARCH \quad "0"
TSTTBL \quad "0"

$414 \quad BEF
CANCEL
CLM \quad CALLPR
BEF
OUT \quad LDA
EVAL
BEF
EVAL \quad "5"
BEF
EVAL \quad "0"
BEF
EVAL \quad "5"
BEF
SUBLIS
BEF
SUBLIS

$419 \quad BEF
TEST \quad ")"
STKSYM
CLM \quad EXP
BEF
SUBLIS
BEF
SUBLIS

$418 \quad R
STRTN
ID
BF \quad $423
SEARCH \quad "0"
TSTTBL \quad "4"

$420 \quad SET
BF \quad $427
BEF
BEF
OUT \quad INDXA
OUTSYM
BEF
SUBLIS
BEF
MOVE
ONE
DIMCNT

$419 \quad BEF
TEST \quad ")"
STKSYM
CLM \quad EXP
BEF
SUBLIS
BEF
SUBLIS

$418 \quad R
STRTN
ID
BF \quad $432
SEARCH \quad "0"
TSTTBL \quad "0"

$420 \quad SET
BF \quad $428
BEF
BEF
OUT \quad INDXA
OUTSYM
BEF
SUBLIS
BEF
MOVE
ONE
DIMCNT

$419 \quad BEF
TEST \quad ")"
STKSYM
CLM \quad EXP
BEF
SUBLIS
BEF
SUBLIS

$418 \quad R
STRTN
ID
BF \quad $434
SEARCH \quad "0"
TSTTBL \quad "0"

$420 \quad SET
BF \quad $427
BEF
BEF
OUT \quad INDXA
OUTSYM
BEF
SUBLIS
BEF
MOVE
ONE
DIMCNT

$419 \quad BEF
TEST \quad ")"
STKSYM
CLM \quad EXP
BEF
SUBLIS
BEF
SUBLIS

$418 \quad R
STRTN
ID
BF \quad $432
SEARCH \quad "0"
TSTTBL \quad "0"
<table>
<thead>
<tr>
<th>$434</th>
<th>BT</th>
<th>$432</th>
<th>TEST</th>
<th>&quot;()&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET</td>
<td>BEF</td>
<td>OUT</td>
<td>INDXR</td>
<td></td>
</tr>
<tr>
<td>POP</td>
<td>SEARCH</td>
<td>OUTSYM</td>
<td>B</td>
<td>$442</td>
</tr>
<tr>
<td>&quot;0&quot;</td>
<td>$443</td>
<td>SET</td>
<td>BF</td>
<td>$442</td>
</tr>
<tr>
<td>&quot;7&quot;</td>
<td>DIMCNT</td>
<td>OUT</td>
<td>LD</td>
<td>&quot;()&quot;</td>
</tr>
<tr>
<td>BT</td>
<td>$435</td>
<td>OUT</td>
<td>EVAL</td>
<td>&quot;4&quot;</td>
</tr>
<tr>
<td>BM</td>
<td>&quot;() INCOR</td>
<td>&quot;1&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$435</td>
<td>MOVE</td>
<td>DIMCNT</td>
<td>$442</td>
<td>BF</td>
</tr>
<tr>
<td>SYMBOL</td>
<td>$438</td>
<td>BEF</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>$431</td>
<td>R</td>
<td>LTERM</td>
<td>$437</td>
<td>R</td>
</tr>
<tr>
<td>ID</td>
<td>BF</td>
<td>$437</td>
<td>LBLCON</td>
<td>TSTTBAL</td>
</tr>
<tr>
<td>SEARCH</td>
<td>&quot;1&quot;</td>
<td>BF</td>
<td>$449</td>
<td>OUT</td>
</tr>
<tr>
<td>CLM</td>
<td>LBLCON</td>
<td>BF</td>
<td>$449</td>
<td>EVAL</td>
</tr>
<tr>
<td>BF</td>
<td>$438</td>
<td>CLM</td>
<td>$440</td>
<td>&quot;5&quot;</td>
</tr>
<tr>
<td>B</td>
<td>$438</td>
<td>BF</td>
<td>$440</td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td>$439</td>
<td>CLM</td>
<td>$440</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>LBLVAR</td>
<td>BF</td>
<td>$440</td>
<td>LBLVAR</td>
<td>TSTTBAL</td>
</tr>
<tr>
<td>BF</td>
<td>$443</td>
<td>BF</td>
<td>$442</td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td>OUT</td>
<td>LDA</td>
<td>MOVE</td>
<td>PEXITF</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;5&quot;</td>
<td>PEXITT</td>
<td>B</td>
<td>$452</td>
</tr>
<tr>
<td>EVAL</td>
<td>&quot;4&quot;</td>
<td>MOVI</td>
<td>PEXITT</td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td>&quot;1&quot;</td>
<td>$453</td>
<td>BEF</td>
<td>&quot;5&quot;</td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td>EXP</td>
<td>$453</td>
<td>BF</td>
<td>$451</td>
</tr>
<tr>
<td>BEF</td>
<td>TSTTBAL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$444</td>
<td>TEST</td>
<td>&quot;()&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$446</td>
<td>BF</td>
<td>$452</td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td>EXP</td>
<td>BF</td>
<td>$451</td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td>$452</td>
<td>TEST</td>
<td>&quot;()&quot;</td>
<td></td>
</tr>
<tr>
<td>$446</td>
<td>BT</td>
<td>$444</td>
<td>BF</td>
<td>$456</td>
</tr>
<tr>
<td>SET</td>
<td>$444</td>
<td>OUT</td>
<td>LDA</td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td>$444</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
EVAL  "5"
      "0"
CLM  SUBLIS
BEF  
TEST  
B6F  
OUT  INDXR
OUTSYM  
B  $455
$456  SET
BF  $455
OUT  LD
EVAL  "5"
      "0"

$455  B6F
$451  R
CONDST  LATCH  IFCLSE
BF  $459
PUSHLB  JUMPF
L61  
EVAL  "4"
      "1"
CLM  BSCSTM
BEF  
TEST  "ELSE"
B6F  $461
PUSHLB  
OUT  JUMP
L61  
EVAL  "4"
      "1"

ENTLOC  ENTL  "0"
CLM  "6"
BEF  
L61  
ENTLOC  ENTL  "0"

SET  B  $460
BF  $460
L61  
ENTLOC  ENTL  "0"

$460  B6F
$459  R
IFCLSE  ID
BF  $464
COMP  SYMBOL
IFSYM  "2"

SYMBOL  IFSYM  "THEN"
BOOLEX  "THEN"

IDENT  ID
BF  $466
SEARCH  "0"
TSTTBL  "0"
<table>
<thead>
<tr>
<th>Page</th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>BF</td>
<td>$492</td>
<td>SAVL</td>
<td>&quot;$&quot;,</td>
</tr>
<tr>
<td>B</td>
<td>$491</td>
<td>BF</td>
<td>$505</td>
</tr>
<tr>
<td>CLM</td>
<td>BASSGN</td>
<td>CLM</td>
<td>BLFTPT</td>
</tr>
<tr>
<td>BF</td>
<td>$493</td>
<td>BEF</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>$491</td>
<td></td>
<td>SST</td>
</tr>
<tr>
<td>CLM</td>
<td>SASSGN</td>
<td>MARK</td>
<td>OUT</td>
</tr>
<tr>
<td>BF</td>
<td>$494</td>
<td>RESTOR</td>
<td>SASSGN</td>
</tr>
<tr>
<td>B</td>
<td>$491</td>
<td>SAVE</td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td>LASGN</td>
<td>BEF</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$491</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>$491</td>
<td>BT</td>
<td>$503</td>
</tr>
<tr>
<td>AASSGN</td>
<td>LATCH</td>
<td>SET</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$497</td>
<td>BEF</td>
<td>&quot;=&quot;</td>
</tr>
<tr>
<td>MARK</td>
<td>ALFTPT</td>
<td></td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>OUT</td>
<td>STO</td>
<td>BEF</td>
<td></td>
</tr>
<tr>
<td>SAVE</td>
<td></td>
<td></td>
<td>RESTOR</td>
</tr>
<tr>
<td>TEST</td>
<td>&quot;,&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$500</td>
<td>SASSGN</td>
<td>SLFTPT</td>
</tr>
<tr>
<td>CLM</td>
<td>ALFTPT</td>
<td>BF</td>
<td>$507</td>
</tr>
<tr>
<td>BEF</td>
<td></td>
<td>MARK</td>
<td>OUT</td>
</tr>
<tr>
<td>MARK</td>
<td>OUT</td>
<td>SAVE</td>
<td></td>
</tr>
<tr>
<td>STO</td>
<td>SST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESTOR</td>
<td>SAV</td>
<td>RESTOR</td>
<td>SAVE</td>
</tr>
<tr>
<td>TEST</td>
<td>&quot;,&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$498</td>
<td>B</td>
<td>$510</td>
</tr>
<tr>
<td>SET</td>
<td></td>
<td>CLM</td>
<td>SLFTPT</td>
</tr>
<tr>
<td>BEF</td>
<td>&quot;,=&quot;</td>
<td></td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>TEST</td>
<td></td>
<td>BEF</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>EXP</td>
<td>CLM</td>
<td>SST</td>
</tr>
<tr>
<td>CLM</td>
<td>EXP</td>
<td>RESTOR</td>
<td>SAVE</td>
</tr>
<tr>
<td>BEF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESTOR</td>
<td>SAV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BT</td>
<td>$500</td>
<td>SET</td>
<td></td>
</tr>
<tr>
<td>$497</td>
<td>RASSGN</td>
<td>BEF</td>
<td>&quot;=&quot;</td>
</tr>
<tr>
<td>BF</td>
<td>$502</td>
<td></td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>MARK</td>
<td>BLFTPT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>STO</td>
<td>BEF</td>
<td></td>
</tr>
<tr>
<td>LATCH</td>
<td></td>
<td>CLM</td>
<td>SEXP</td>
</tr>
</tbody>
</table>
$507  LASSGN
    BEF  $507
    RESTOR
    R

$512  BF  $512
    LATCH  LLFTPT
    BF  $512
    MARK
    OUT  STO
    SAVE

$513  BF  $513
    TEST  ","
    BF  $513
    CLM  LLFTPT
    BEF
    MARK
    OUT  SST
    RESTOR
    SAVE

$515  BF
    BT  $515
    SET
    BEF
    TEST  "="
    BEF
    CLM  LTERM
    BEF
    COMP  PEXIT
    ZERO
    "2"
    BT  $516
    BM  "F: INDIR"

$516  BF
    RESTOR
    ID

$512  BF  $512
    ALFTPT  ID
    BF  $512
    SEARCH  "0"
    TSTTBL  "0"

$520  BF  $520
    TSTTBL  "0"
    "2"

$521  BF  $521
    LF  $521

$519  BF  $519
    BEF
    CANCEL
    OUT  LDA
    EVAL  "5"
    "0"

$519  BF  $519
    BEF
    CLM  SUBPAR

$518  BF
    BT  $518
    ID

$518  BF
    BT  $518
    ID

$525  BF  $525
    SUBPAR
    SEARCH  "0"
    TSTTBL  "0"
    "3"

$527  BF  $527
    BEF
    TSTTBL  "0"
    "T"
    "0"

$526  BF  $526
    BEF
    CANCEL
    OUT  LDA
    EVAL  "5"
    "0"

$519  BF  $519
    BEF
    CLM  SUBPAR
SAVE TEST "(" BEF CLM SLFTPT BEF TEST ")" BEF SET $550 BF MARK OUT $550 SAVE BEF TEST "EDIT" BEF CANCEL PUSHLB OUT FMT LB1 EVAL "4" "1" RESTOR TEST "(" BEF CLM ILIST BEF TEST ")" BEF PUSHLB OUT JUMP LB1 EVAL "4" "1" LB2
BF  $566
MARK
CLM  ILIST
BEF
SAVL
CLM  ERPLIS
BEF  R
TEST  "DO"
BF  $568
ID
BEF
CLM  UNDTST
BEF
PUSHLB
OUT  LDA
EVAL  "5"
"0"
STKSYM
MOVE  IO1TER
IOSW
CLM  ITERPR
BEF
MOV1  IOSW
"0"
POP
OUT  LDA
EVAL  "5"
"0"
RESTOR
LB1
TEST "(" BF $580
BEF b $579
CLM OLIST $580 CLM EXP
BEF BF $583
TEST ")" B $582
BEF BF $583
OUT PUT BF $584
PUSHLB B $582
OUT JUMP $584 LATCH BVARBL
LB1 BF $579
EVAL OUT EDIT
"4" $579 R
"1" ORPLIS TEST "(
LB2 BF $587
ENTLOC MARK $578
ENTL CLM OLIST
"0" BEF $579
"6" CLM FMTLIS
CLM SAVE
BEF $587 R
LB1 FMTLIS TEST "(
ENTLOC BF $589
ENTL OUT "c"
"0" $570 R
"6"
OELMNT CLM FMTITE
BEF $575 $590 TEST ","
$576 TEST "a" BF $592
BEF CLM FMTITE
$578 BT $576
SET BEF $590
$575 R
OELMNT CLM ORPLIS
<table>
<thead>
<tr>
<th>$589</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMTITE CLM</td>
<td>CTRLFM</td>
</tr>
<tr>
<td>BF</td>
<td>$594</td>
</tr>
<tr>
<td>B</td>
<td>$593</td>
</tr>
<tr>
<td>BF</td>
<td>$593</td>
</tr>
<tr>
<td>$594</td>
<td>CLM</td>
</tr>
<tr>
<td>BF</td>
<td>$593</td>
</tr>
<tr>
<td>$593</td>
<td>R</td>
</tr>
<tr>
<td>CTRLFM TEST</td>
<td>&quot;X&quot;</td>
</tr>
<tr>
<td>BF</td>
<td>$597</td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td>CLM</td>
<td>SPEC</td>
</tr>
<tr>
<td>BEF</td>
<td>B</td>
</tr>
<tr>
<td>$597</td>
<td>TEST</td>
</tr>
<tr>
<td>BF</td>
<td>$598</td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>B</td>
<td>$596</td>
</tr>
<tr>
<td>$598</td>
<td>TEST</td>
</tr>
<tr>
<td>BF</td>
<td>$599</td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;2&quot;</td>
</tr>
<tr>
<td>CLM</td>
<td>SPEC</td>
</tr>
<tr>
<td>BEF</td>
<td>B</td>
</tr>
<tr>
<td>$601</td>
<td>SET</td>
</tr>
<tr>
<td>BF</td>
<td>$600</td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;0001&quot;</td>
</tr>
<tr>
<td>BEF</td>
<td>B</td>
</tr>
<tr>
<td>$600</td>
<td>BEF</td>
</tr>
<tr>
<td>INUM</td>
<td></td>
</tr>
<tr>
<td>OUTSYM</td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td>&quot;(&quot;</td>
</tr>
<tr>
<td>BF</td>
<td>$613</td>
</tr>
<tr>
<td>$599</td>
<td>TEST</td>
</tr>
<tr>
<td>BF</td>
<td>$596</td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;3&quot;</td>
</tr>
<tr>
<td>CLM</td>
<td>SPEC</td>
</tr>
<tr>
<td>BEF</td>
<td>B</td>
</tr>
<tr>
<td>$596</td>
<td>R</td>
</tr>
<tr>
<td>DATAFM TEST</td>
<td>&quot;A&quot;</td>
</tr>
<tr>
<td>BF</td>
<td>$605</td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;B&quot;</td>
</tr>
</tbody>
</table>
IDENTIFICATION:

PROGRAM-ID: MTXINT04.

AUTHOR: J. R. VAN DOREN.

SOURCE LANGUAGE: EASYCODER.

SOURCE COMPUTER: H-1200.

OBJECT COMPUTER: H-1200.

PURPOSE:

MTXINT04 INTERPRETIVELY EXECUTES OBJECT PROGRAMS PRODUCED BY THE METAX9 COMPILER-COMPILER.

ASM0DE4 ASSEMBLE IN FOUR CHAR ADDRESSING MODE

ORG 45056 EXECUTION LOCATION

OCTAL ADDRESS DEFINITIONS OF PERTINENT SYMBOLS IN THE RESIDENT INPUT/OUTPUT ROUTINE.

COMMUNICATION AREA FIELD LOCATION DEFINITIONS
ELATCH EQU 242  BACKUP ERROR LATCH
SYMBOL EQU 243  CURRENT SYMBOL VALUE FIELD
CMPLCD EQU 236  COMPLETION CODE FIELD
PSLST EQU 239  POST LISTING OPTION FIELD
INSTCT EQU 209  INSTRUCTION COUNT FIELD
GENLOC EQU 215  BEGINNING CODE GENERATION POINTER
LODLOC EQU 219  BEGINNING LOCATION POINTER FOR INTERP.
SYMSTR EQU 231  POINTER TO BEGINNING OF SYMBOL TABLE
SYMEND EQU 235  UPPER BOUND OF SYMBOL TABLE AREA
STKSTR EQU 223  POINTER TO START OF CONTROL STACK AREA
STKEND EQU 227  UPPER LIMIT OF CONTROL STACK AREA

IR1 EQU 4  INSTRUCTION COUNTER FOR PROGRAM BEING INTERPRETED
IR2 EQU 8  SYSTEM PUSHDOWN STACK POINTER
IR3 EQU 12  PROGRAM COUNTER FOR PROGRAM BEING COMPILED
IR4 EQU 16  WORK REGISTER
IR5 EQU 20  POINTER TO NEXT OUTPUT CODE LOCATION
IR6 EQU 24  POINTER TO NEXT CHARACTER IN INPUT STRING
IR7 EQU 28  USED BY INSTRUCTION FETCH
IR8 EQU 32  WORK REGISTER
IR9 EQU 36  WORK REGISTER
IR10 EQU 40  WORK REGISTER
IR11 EQU 44  WORK REGISTER
IR12 EQU 48  WORK REGISTER
IR13 EQU 52  WORK REGISTER
IR14 EQU 56  WORK REGISTER
IR15 EQU 60  POINTER TO SYMTAB ENTRY FOUND BY LAST SEARCH
TF DCF :F:

BEGIN PROGRAM INITIALIZATION
START EQU *

CAM 60 SET FOUR CHAR ADDRESSING FOR EXECUTE
SW STKEND-2 WORD MARK FOR MOVING AND TESTING
MCw STKSTR,IR2 INITIALIZE STACK POINTER
SW IR2-2 SHORTEN ARITHMETIC
SI IR2 ITEM MARK FOR RIGHT MOVE
MCw LODLOC,IR1 INITIALIZE INSTRUCTION COUNTER
SW IR1-2 SHORTEN INDEX ARITHMETIC
SI IR1 ACCOMMODATE RIGHT MOVE
BS IR3 ZERO PROGRAM COUNTER
SW IR3-2 SHORTEN ARITHMETIC
SI IR3 ACCOMMODATE RIGHT MOVE
SI IR15 ACCOMMODATE RIGHT MOVE
MCw GENLOC,IR5 INITIALIZE CODE GENERATION LOCATION
SW IR7-1 SHORTEN FETCH ARITHMETIC
MCw SYMSTR,NEWSYM INITIALIZE NEXT LOCATION IN SYMBOL TABLE
BS =184,SYMEND INITIALIZE
MCw SYMEND,IR14
LCA =1C77+1+X14 BLOCK
LCA NEWSYM+4+X14 LIST
MCw SYMEND+CRBLKT INIT CURRENT BLOCK LIST POINTER
MCw NEWSYM+IR15 INIT SYMTAB POINTER
MCw SYMEND,SVSYME SAVE INITIAL SYMTAB END AS START OF BLOCK LIST
MCw ::OUTPUT+132 CLEAR PRINT
MCw OUTPUT+132 BUFFER
MCw =1C21 CARRIAGE CONTROL
SI INPUT+80 RESTORE LOST ITEM MARK ON INPUT BUFFER
BS :SKIP TO TOP OF PAGE AND INIT INPUT BUFFER
BS :SKIP PRINT+57,
BS :GET READ,
BS :PUT PRINT,INPUT-1,
MCw +INPUT+IR6
B FIRST
B FIRST
0112C* INSTRUCTION FETCHING *
0114C* *
0115C**************************************************************************
01160 FETCH3 BA =IR3,IR1 INSTRUCTION COUNT
01170 FETCH BA =IR1,INSTCT CLEAR SECOND CHAR
01180 FIRST BS IR7-1 INSERT OP CODE
01190 MRSO 0*X1,IR7 BUMP SEQUENCE COUNTER
01200 SAR IR1 MULTIPLY
01210 BA IR7 BY 4
01220 BA IR7
01230 MCw TVEC+3*X7,IR14
01240 B 0*X14
01250 TVEC EQU *
01260 REP 10
01270 DSA ERROR
01280 DSA MOVE
01290 DSA ADD
01300 DSA MULT
01310 DSA SETF
01320 DSA BEF
01330 DSA EXITI
01340 DSA RESOLV
01350 DSA BRANCH
01360 DSA BRANCHF
01370 DSA BRANCHF
01380 DSA BM
01390 DSA SET
01400 DSA SCANI
01410 DSA ERROR
01420 DSA MOVLT
01430 DSA ERASE
01440 DSA COMP
01450 DSA SRCHP
01460 DSA TSTTBL
01470 DSA TSTTHA
01480 DSA LATCH
01490 DSA CANCEL
01500 DSA MARK
01510 DSA SAVE
01520 DSA CLM
01530 DSA RETURN
01540 DSA ERROR
01550 DSA RESTOR
01560 DSA PUSHLB
01570 DSA POP
01580 DSA LB1
01590 DSA LB2
01600 DSA SIKSYM
01610 DSA CHKSYM
01620 DSA Swap
01630 DSA ENTA
01640 DSA ENTL
01650 REP 3
01660 DSA ERROR
01670 DSA OUT
01680 DSA OUTSYM
01690 DSA TEST
01700 CSA ID
01710 DSA ONUM
01720 DSA STRTST
01730 DSA EVAL
01740 DSA ENTLQX
01750 DSA ERROR
01760 DSA INUM
01770 DSA ENUM
01780 DSA BLKENT
01790 DSA BLKEXT
01800 DSA ERROR

0181D***************************************************************************
01820* *
01830* BRANCH, BRANCH TRUE, AND BRANCH FALSE PRIMITIVES *
01840* *
01850* ****************************************************************************
01860  BRANCH MRID 0+X1,2
01870     B   FETCH
01880  BRNCHT BCE FETCH3,TF,F
01890     B   BRANCH
01900  BRNCHF BCE BRANCH,TF,F
01910     B   FETCH3
01920  ********************************************************************
01930  *
01940  *     THE SET AND SETF PRIMITIVES SET THE TRU-FALSE INDICATOR.
01950  *
01960  ********************************************************************
01970  SETF MCW :F,TF
01980     B   FETCH
01990  SET  MCW :T,TF
02000  E   FETCH
02010  ********************************************************************
02020  *
02030  *     THE CLM PRIMITVE (CALL META PROCEDURE) STACKS THE RETURN ADDRESS
02040  *     AND ERROR LATCH CODES, RESETS THE ERROR LATCH AND SETS THE
02050  *     INSTRUCTION COUNTER TO THE BEGINNING OF THE CALLED PROCEDURE.
02060  *
02070  ********************************************************************
02080  CLM  EQU  *
02090  MRID 0+X1,IR1-2
02100  SAR  4+X2     SAVE RETURN
02110  MCW  :00:1+x2
02120  MRSU ELATCH,5+x2     STACK ERROR LATCH WITH RETURN ADDRESS
02130  MCW  :F:ELATCH     RESET ERROR LATCH FOR CALLED PROCEDURE
02140  BA  =189,IR2     BUMP STACK POINTER
02150  B   STKOVF     CHECK FOR POSSIBLE OVERFLOW
02160  ********************************************************************
02170  *
02180  *     THE RETURN PROCEDURE POPS THE CONTROL STACK UNTIL A RETURN ADDRESS
02190  *     IS FOUND WHICH IS SENT TO THE INSTRUCTION COUNTER. THE ERROR LATCH
02200  *     PREVIOUSLY STACKED WITH THE RETURN ADDRESS IS RESTORED.
02210  *
02220  ********************************************************************
02230  RETURN BS =1B9,IR2  SEARCH AND POP
02240  BCE DORET+0*X2,00 UNTIL RETURN
02250  B RETURN ADDRESS IS FOUND
02260  DORET MCW 4*X2,IR1 RETURN ADDRESS TO LOCATION COUNTER
02270  SI IR1 RESTORE ITEM MARK
02280  MRSU 5*X2,ELATCH RESTORE ERROR LATCH UPON RETURN
02290  B CANCEL RETURN CANCELS ANY BACKUP LATCH

***************************************************************************

* PUSHL6 GENERATES A NEW INTERNAL LABEL AND PUSHES IT ON THE *
* CONTROL STACK. *

***************************************************************************

02310* PUSHLB A :1:LABEL
02320* MRIDI LABEL-4*0+X2
02330* BA =1B9,IR2
02340* E FETCH
02350* DCw :6:
02360* RLABEL DCw :000: ITEM MARK RIGHT

***************************************************************************

02380* POP POPS THE CONTROL STACK RESTORING THE VALUE TO SYMBOL IF THE *
02390* TOP OF THE STACK IS MARKED AS A SYMBOL.

***************************************************************************

02410* POPSYM MCW 8*X2,SYMBOL+7 STACK SYMBOL TO SYMBOL AREA
02420* B FETCH

***************************************************************************

02440* LB1 SEARCHES THE CONTROL STACK TO FIND THE FIRST LABEL SYMBOL *
02450* WHICH IS THEN MOVED TO SYMBOL. THE STACK IS NOT AFFECTED.

***************************************************************************

02480* POP BS =1B9,IR2
02490* BCE POPSYM,0*X2,S BRANCH IF STACK TOP IS SYMBOL
02500* B FETCH
02510* POPSYM MCW 8*X2,SYMBOL+7 STACK SYMBOL TO SYMBOL AREA
02520* B FETCH

***************************************************************************

02550* LB1 LCA IR2,IR14
**Fetch**

Test for Alpha Char

* TRUE-FALSE INDICATOR AND MOVING THE IDENTIFIER TO SYMBOL IF FOUND.
* ID TESTS THE INPUT STRING FOR A VALID IDENTIFIER SETTING THE
  NEXT SYMBOL TO CLEAR SYMBOL AREA

```
00040 00116 00160 00150 00140 00130 00120 00110 00100 00090 00080 00070 00060 00050 00040 00030
```

**Scan for Item Mark**

```
03100 03100 03100 03100 03100 03100 03100 03100 03100 03100 03100 03100 03100 03100 03100
```

* FOLLOWING THE OP CODE SETTING THE TRUE-FALSE INDICATOR APPROPRIATELY
* TEST EXAMINES THE INPUT STRING FOR THE SPECIFIED LITERAL STRING

```
03000 03000 03000 03000 03000 03000 03000 03000 03000 03000 03000 03000 03000 03000 03000
```
ONUM TESTS THE INPUT STRING FOR A VALID OCTAL NUMBER SETTING THE TRUE-FALSE INDICATOR AND CONVERTING THE NUMBER TO BINARY IN SYMBOL IF TRUE.

ONUM B NEXT

HS SYMBOL+32 CLEAR TO ZEROES

MCW :T:,TF

LCA =3B0,IR13

BS IR14

ONMTST MRSD 0+X6,IR14
**04080** CI 0-1+X13
**04090** MCW :T:;TF
**04100** B FETCH

**04110**

04120* THE EVAL PRIMITIVE CAUSES A SEARCH OF THE SYMBOL TABLE AND
04130* THEN OUTPUTING OF A SYMBOL TABLE VALUE TO THE CODE STREAM
04140* ACCORDING TO TWO ONE CHARACTER LITERAL PARAMETERS:
04150* 1) THE LENGTH OF THE FIELD
04160* 2) THE RELATIVE POSITION WITHIN THE TABLE ENTRY

**04190**

04200 EVAL MCW :0:+SRCHTP SET SEARCH MODE
04210 B SEARCH GO SEARCH
04220 BS IR13
04230 MRSD 0+X1+IR13 INSERT LENGTH
04240 SAR IR1
04250 BS IR14
04260 MRSD 0+X1+IR14 INSERT POSITION
04270 SAR IR1
04280 BA IR15+IR14 FIND ABSOLUTE POSITION
04290 BA IR14+IR13 FIND FIELD END
04300 SI 0-1+X13
04310 MRILR 0+X14+0+X5 OUTPUT CODE
04320 SBR IR5 NEXT OUTPUT LOCATION
04330 MRIN 0+X14+0+X3 PROGRAM COUNTER
04340 SBR IR3
04350 CI 0-1+X13
04360 B FETCH

**04370**

04380* THE ENLOC PRIMITIVE CAUSES A FULL SEARCH OF THE SYMBOL TABLE
04390* AND THEN CAUSES THE PROGRAM COUNTER TO BE ENTERED AS A VALUE
04400* FOR THE ADDRESS OF THE SPECIFIED SYMBOL. LEVEL 0 (STATIC)
04410* IS ASSIGNED FOR THE STORAGE LEVEL.

**04440**
SEARCH MODE

MOVE IN ADDRESS
MARK RELOCATABLE
STATIC STORAGE INDICATOR

04450 ENTLOC MCw :0:SRCHTP
04460 B SEARCH
04470 MRID IR3-2,ADDP-2+X15
04480 Sw ADDR-2+X15
04490 MCw :0:LEVEL+X15
04500 B FETCH

***************************************************************************
* *
* ENTL AND ENTA ARE PRIMITIVES FOR INTERING LITERAL AND ADDRESSED *
* VALUES, RESPECTIVELY, INTO THE SYMBOL TABLE. INDEX REGISTER 15 *
* MUST POINT TO THE PROPER SYMBOL TABLE ENTRY PRIOR TO EXECUTION. *
* A SIX BIT LITERAL NUMBER FOLLOWS EACH OP CODE SPECIFYING THE *
* RELATIVE POSITION WITHIN THE TABLE ENTRY TO BE ALTERED. *
* *
***************************************************************************

* 04510 ENTA EQU *
* 04520 ENTL EQU *
* 04530 BS IR14 COMPUTE LEFTMOST
* 04540 MRSU 0+X1,IR14 ADDRESS
* 04550 SAR IR1 OF
* 04560 BA IR15,IR14 RECEIVING FIELD
* 04570 CW 0+X14 CLEAR POSSIBLE WORD MARK
* 04580 BCE ENTLIT 0-2+X1,56 TEST FOR LITERAL ENTRY
* 04590 MRID 0+X1,IR13-2 ADDRESS TO IR13
* 04600 SAR IR1 UPDATE INSTRUCTION COUNTER
* 04610 CW 0+X13 AVOID INADVERTENT RELOCATION MARKER
* 04620 MRIDw 0+X13,0+X14 ENTER, CLEAR ANY WORD MARKS
* 04630 B FETCH
* 04640 ENTLIT MRIDw 0+X1,0+X14 ENTER LITERAL
* 04650 SAR IR1 UPDATE INSTRUCTION COUNTER
* 04660 B FETCH

***************************************************************************
* *
* SEARCH IS A SUBROUTINE FOR SEARCHING A BLOCK STRUCTURED SYMBOL *
* TABLE. NOTE THE SEARCH TYPE PARAMETER (SRCHTP) WHICH MAY *
* BE USED TO CONTROL THE SEARCH MODE. SEARCH MAY BE CALLED *
* BY PRIMITIVES EVAL, ENTLOC, SRCHP, OR BLKEXT *
* *
***************************************************************************
<table>
<thead>
<tr>
<th><strong>SYMBOL TABLE ENTRY COUNT</strong></th>
<th><strong>SPACE FOR SEARCH MODE TYPE CODE</strong></th>
<th><strong>POINTER TO NEXT AVAILABLE SLOT IN SYMBOL TABLE END</strong></th>
<th><strong>FIELD WHICH EFFECTIVELY POINTS TO THE START OF THE BLOCK LIST FOR NAMES</strong></th>
<th><strong>FOR SYMBOL TABLE STRUCTURE</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SAVE RETURN TO CALLER.</strong></td>
<td><strong>TALLY SEARCH COUNT.</strong></td>
<td><strong>SET STARTING POINT FOR CURRENT BLOCK.</strong></td>
<td><strong>TALLY COMPARISON COUNT.</strong></td>
<td><strong>NEXT SYMBOL TABLE ENTRY.</strong></td>
</tr>
<tr>
<td><strong>TEST END OF BLOCK.</strong></td>
<td><strong>TEST SINGLE BLOCK SEARCH IF FULL SEARCH GET NEW BLOCK.</strong></td>
<td><strong>COUNT THE NUMBER OF ENTRIES.</strong></td>
<td><strong>CHECK FOR TABLE OVERFLOW.</strong></td>
<td><strong>FILL IN CHAIN ADDRESS FOR LAST ENTRY.</strong></td>
</tr>
</tbody>
</table>

**SYMBOL TABLE ENTRY COUNT**

- **SYMCT** DCW = 3B0
- **SCHR** DCW = 1
- **SYME** DCW = 4

**SPACE FOR SEARCH MODE TYPE CODE**

- **SCHR** = 1
- **SCHR** = 1
- **SCHR** = 1
- **SCHR** = 1

**POINTER TO NEXT AVAILABLE SLOT IN SYMBOL TABLE END**

- **SCHR** = 2
- **SCHR** = 2
- **SCHR** = 2
- **SCHR** = 2

**FIELD WHICH EFFECTIVELY POINTS TO THE START OF THE BLOCK LIST FOR NAMES**

- **SCHR** = 3
- **SCHR** = 3
- **SCHR** = 3
- **SCHR** = 3

**FOR SYMBOL TABLE STRUCTURE**

- **SCHR** = 4
- **SCHR** = 4
- **SCHR** = 4
- **SCHR** = 4

**SAVE RETURN TO CALLER.**

- **SCHR** = 5
- **SCHR** = 5
- **SCHR** = 5
- **SCHR** = 5

**TALLY SEARCH COUNT.**

- **SCHR** = 6
- **SCHR** = 6
- **SCHR** = 6
- **SCHR** = 6

**SET STARTING POINT FOR CURRENT BLOCK.**

- **SCHR** = 7
- **SCHR** = 7
- **SCHR** = 7
- **SCHR** = 7

**TALLY COMPARISON COUNT.**

- **SCHR** = 8
- **SCHR** = 8
- **SCHR** = 8
- **SCHR** = 8

**NEXT SYMBOL TABLE ENTRY.**

- **SCHR** = 9
- **SCHR** = 9
- **SCHR** = 9
- **SCHR** = 9

**TEST END OF BLOCK.**

- **SCHR** = A
- **SCHR** = A
- **SCHR** = A
- **SCHR** = A

**TEST SINGLE BLOCK SEARCH IF FULL SEARCH GET NEW BLOCK.**

- **SCHR** = B
- **SCHR** = B
- **SCHR** = B
- **SCHR** = B

**COUNT THE NUMBER OF ENTRIES.**

- **SCHR** = C
- **SCHR** = C
- **SCHR** = C
- **SCHR** = C

**CHECK FOR TABLE OVERFLOW.**

- **SCHR** = D
- **SCHR** = D
- **SCHR** = D
- **SCHR** = D

**FILL IN CHAIN ADDRESS FOR LAST ENTRY.**

- **SCHR** = E
- **SCHR** = E
- **SCHR** = E
- **SCHR** = E
<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>05560</td>
<td>BS IR14</td>
<td>COMPUTE LEFTMOST ADDRESS OF</td>
</tr>
<tr>
<td>05570</td>
<td>MRSD 0+X1,IR14</td>
<td>TABLE FIELD</td>
</tr>
<tr>
<td>05580</td>
<td>SAR IR1</td>
<td>TO BE TESTED</td>
</tr>
<tr>
<td>05590</td>
<td>BA IR15,IR14</td>
<td>TEST FOR LITERAL TEST</td>
</tr>
<tr>
<td>05600</td>
<td>BCE TSLIT+0-2+X1,34</td>
<td>ADDRESS TO IR13</td>
</tr>
<tr>
<td>05610</td>
<td>MRID 0+X1,IR13-2</td>
<td>UPDATE INSTRUCTION COUNTER</td>
</tr>
<tr>
<td>05620</td>
<td>SAR IR1</td>
<td>WORD MARK TO STOP COMPARE</td>
</tr>
<tr>
<td>05630</td>
<td>Sw 0+X13</td>
<td>POSITION TO RIGHT END</td>
</tr>
<tr>
<td>05640</td>
<td>MRIN 0+X13,0+X14</td>
<td>SET</td>
</tr>
<tr>
<td>05650</td>
<td>SAR IR13</td>
<td>INDEX REGISTERS</td>
</tr>
<tr>
<td>05660</td>
<td>SBR IR14</td>
<td>SET</td>
</tr>
<tr>
<td>05670</td>
<td>C 0-1+X14,0-1+X13</td>
<td>INCREASE END</td>
</tr>
<tr>
<td>05680</td>
<td>BE SET</td>
<td>POSITION TO RIGHT END</td>
</tr>
<tr>
<td>05690</td>
<td>B SETF</td>
<td></td>
</tr>
<tr>
<td>05700</td>
<td>TSLIT Sw 0+X1</td>
<td>SET</td>
</tr>
<tr>
<td>05710</td>
<td>MRIN 0+X1,0+X14</td>
<td>INDEX REGISTERS</td>
</tr>
<tr>
<td>05720</td>
<td>SAR IR1</td>
<td>SET</td>
</tr>
<tr>
<td>05730</td>
<td>SBR IR14</td>
<td></td>
</tr>
<tr>
<td>05740</td>
<td>C 0-1+X14,0-1+X1</td>
<td></td>
</tr>
<tr>
<td>05750</td>
<td>BE SET</td>
<td></td>
</tr>
<tr>
<td>05760</td>
<td>B SETF</td>
<td></td>
</tr>
<tr>
<td>05770</td>
<td></td>
<td><strong>THE COMPARE PRIMITIVE COMPARES THE 2ND ADDRESSED OPERAND TO</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>THE FIRST OF THE SIX BIT CHARACTER FOLLOWING THE OPERAND ADDRESSES</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>IS USED AS THE VARIANT OF THE CONDITIONAL BRANCH INSTRUCTION</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>FOLLOWING THE COMPAREISON</strong></td>
</tr>
<tr>
<td>05840</td>
<td></td>
<td><strong>FIRST ADDRESS TO IR13</strong></td>
</tr>
<tr>
<td>05850</td>
<td>COMP EQU</td>
<td><strong>SECOND ADDRESS TO IR14</strong></td>
</tr>
<tr>
<td>05860</td>
<td>MRID 0+X1,IR13-2</td>
<td><strong>INSERT CONDITIONAL BRANCH CODE</strong></td>
</tr>
<tr>
<td>05870</td>
<td>SAR IR1</td>
<td><strong>WORD MARK TO STOP COMPARE</strong></td>
</tr>
<tr>
<td>05880</td>
<td>MRID 0+X1,IR14-2</td>
<td>****</td>
</tr>
<tr>
<td>05890</td>
<td>SAR IR1</td>
<td>****</td>
</tr>
<tr>
<td>05900</td>
<td>SST 0+X1,COMPT,07</td>
<td>****</td>
</tr>
<tr>
<td>05910</td>
<td>RA =IR1,IR1</td>
<td>****</td>
</tr>
<tr>
<td>05920</td>
<td>Sw 0+X14</td>
<td>****</td>
</tr>
</tbody>
</table>
* SCAN FOR ITEM MARK

06290 MRIN 0+XI*0+X6
06280 SW 0+XI
06270 SCAN B NEXT

06260 SCAN A

06250 * FOLLOWING THE OP CODE.

06240 * SCAN THE INPUT STRING FOR THE SPECIFIED LITERAL STRING

06230 * SCAN THE INSTRUCTION NEXT INSTRUCTION

06220 B FETCH

06210 F: BCUP TURN OFF BACKUP SWITCH

06200 CANCEL MCW: F:BCUP

06190 * CANCEL TURNS OFF ANY BACK UP LATCH.

06180 * CANCEL TURNS OFF ANY BACK UP LATCH.

06170 * 

06160 * 

06150 * 

06140 * 

06130 BCUP DCW = 3

06120 SAVIN DCW = 3

06110 SAVOUT DCW = 3

06100 BCUP B:CM = 3

06100 CALL THE SPECIFIED ROUTINE

06100 SET BACKUP INDICATOR

06090 SAVE 0BCUP POINTER

06090 SAVE INPU POINTERS

06090 AVOID CARD BOUNDARY PROBLEMS

06080 LATCH B NEXT

06070 LATCH B NEXT

06060 LATCH B NEXT

06050 LATCH B NEXT

06040 LATCH B NEXT

06030 LATCH B NEXT

06020 LATCH B NEXT

06010 LATCH B NEXT

06000 LATCH B NEXT

* A RETURN TO THE CALLING PROCEDURE.

* EFFECT IS THAT AN APPARENT ERROR IN THE CALLED PROCEDURE CAUSES

* LATCH SETS POINTERS AND INDICATORS AND THEN INVOKES CLM. THE

* INDICATOR

B SET

SET TF

COMP B CT SET*40

SET TF

06990 MRIN 0+XI*0+X6

06980 3A13*0+X1

06970 0+XI

06960 0+XI

06950 0+XI

06940 0+XI

06930 0+XI
06300          SAR   IR1
06310          SBR   IR6
06320          SCNCMP C  0=1+X6,0=1+X1
06330          EE    SCANT
06340          BI    NEXT,0+X6
06350          BA    =1BI,IR6
06360          B     SCNCMP
06370          SCANT  MCW :T:,TF
06380          B     FETCH
06390***************

06400*          MOVLIT MOVES THE LITERAL CHARACTER STRING FOLLOWING THE ADDRESS
06410*          (WHICH FOLLOWS THE OP CODE) TO THE ADDRESSED LOCATION.
06420*          *
06430*          *
06440***************************************************************************

06450          MOVLIT MRI1: 0+X1,IR13-2  ADDRESS OF RECEIVING CHAR FIFLD
06460          SAR   IR14
06470          MRI2 0+X14,0+X13  MOVE LITERAL DATA
06480          SAR   IR1
06490          SW    0+X14
06500          B     FETCH
06510***************************************************************************

06520*          INUM CALLS INM FOR AN ATTEMPTED RECOGNITION OF AN INTEGER NUMBER.
06530*          *
06540*          *
06550***************************************************************************

06560          INUM  B  INM
06570          B     FETCH
06580***************************************************************************

06590*          SUBROUTINE INM TESTS THE INPUT STRING FOR AN INTEGER NUMBER SETTING
06600*          TRUE-FALSE CODE AND CONVERTING THE NUMBER TO BINARY IN SYMBOL
06610*          IF TRUE.
06620*          *
06630*          *
06640***************************************************************************

06650          INM  SBR  INMRTN+4
06660          B     NEXT
0667C MCW IR6,IR10            SAVE INPUT POINTER
0668C MCW :F:,TF              TEST FOR MINUS SIGN
0669C LCA =380,IR13           TEST FOR PLUS SIGN
0670C BS IR14                MUST BE POSITIVE
0671C BCE STISGN,0*X6,-        TEST FOR POSSIBLE FLTNG PT NM
0672C BCE STISGN,0*X6,+        TEST FOR INTEGER DIGIT
0673C MCW :+ :ISGN            HAVE WE FOUND AN INTEGER
0674C B INMTST               YES, GO CONVERT TO BINARY
0675C ISGN DCW =1              SET TRUE-FALSE INDICATOR
0676C STISGN MRSD 0*X6,ISGN   SAVE SIGN
0677C SAR IR6                UPDATE INPUT POINTER
0678C INMTST MRSD 0*X6,IR14   TEST FOR POSSIBLE FLTNG PT NM
0679C BCE NOINT,0*X6,14       TEST FOR INTEGER DIGIT
0680C BIO MVNUM,IDTAB+X14     HAVE WE FOUND AN INTEGER
0681C BCE NOINT,TF,F          YES, GO CONVERT TO BINARY
0682C B CONVRT                SET TRUE-FALSE INDICATOR
0683C NOINT MCW :F:,TF        HAVE WE FOUND AN INTEGER
0684C MCW IR10,IR6            YES, GO CONVERT TO BINARY
0685C B INMRTN               RESTORE INPUT POINTER
0686C MVNUM MRSDI 0*X6,SYMB0L+X13 RETURN
0687C SAR IR6                SAVE SIGN
0688C DA =181,IR13            UPDATE INPUT POINTER
0689C BCE INMERR,IR13,10      TEST TOO MANY DIGITS
0690C MCW :T:,TF              WE HAVE PART OF AN INTEGER
0691C B INMTST               WE HAVE PART OF AN INTEGER
0692C CONVRT BS CVRFLD        GO LOOK FOR MORE
0693C MCW SYMBOL-1*X13,CVBFLD MOVE IN DECIMAL INTEGER
0694C SST ISGN,CVBFLD,60      MOVE IN DECIMAL INTEGER
0695C DTB CVBFLD,00           SET SIGN IN CONVERSION FIELD
0696C TAM CVBFLD,00           STORE IT
0697C C CVBFLD=6*2C7777       TEST NUMBER TOO LARGE FOR 24 BITS
0698C BE INMOK
0699C C CVBFLD=6*2C0000       TEST NUMBER TOO LARGE FOR 24 BITS
07000 BL INMERR
07010 INMOK EQU *            SAVE 24 BITS
07020 MCW CVBFLD-2*SYMB0L+3   SAVE 24 BITS
07030 SI SYMB0L+3             SAVE 24 BITS
ERASE ERASES THE SPECIFIED NUMBER OF CHARACTERS FROM THE CODE STRING.

ERASE BS
MRSLD SYMBOL,IR13 MOVE ERASE COUNT TO INDEX REG
BS IR13,IR5 ADJUST PROGRAM COUNTER
MCW IR5,IR13 AND OUTPUT POINTERS
ERSST BCE FETCH,IR13,00 TEST ERASE LOOP FINISHED
MRSLR IR13-1,0+X14 ERASE A CHARACTER
SBR IR14 NEXT CHARACTER TO ERASE
BS =1B1,IR13 DECREMENT LOOP COUNT
ERSST

BEF AND BM COMPRISE THE ERROR MESSAGING PRIMITIVES. OBSERVE:
THE SPECIAL ACTION IF THE BACK UP OR ERROR LATCHES ARE SET. NOTE:
ALSO THE DIFFERENCE BETWEEN A WARNING MESSAGE AND A FATAL MESSAGE:
WITH PRIMITIVE BM. NOTE THAT BACKING UP OVER A CARD BOUNDARY IS:
NOT PROVIDED.

BEF EQU *
BCE FETCH,TF,T IF TRUE CONTINUE
BCE DEFMES,BCKUP,F IF NO BACKUP BYPASS BACKUP MECHANISM
MCW SAVIN,IR6 RESTORE INPUT POINTER
BS SAVOUT,IR5 COMPUTE ERASE
MCW IR5,IR13 COUNT
BS IR5,IR3 ADJUST PROGRAM COUNTER
MCW SAVOUT,IR5 RESTORE OUTPUT POINTER
07410  MCw  IR5,IR14  SET UP RETURN FROM CODE ERASURE.
07420  MCw  +ERSRTN,ERTST+4  ERASE CODE
07430  B  ERTST  RESTORE INSTRUCTION IN ERASE ROUTINE
07440  ERSRTN MCw  +FETCH,ERTST+4  BACKUP CANCELS AND RETURNS
07450  B  RETURN  TEST PREVIOUS PENDING MESSAGE
07460  BCE  ERSRTN,OUTPUT+20,F  NO BACKUP SO CONTINUE WITH ERROR MESSAGE
07470  B  ERMPPR  DEFAULT MESSAGE
07480  MCw  $9AF: SYNTAX,OUTPUT+28  FINISH TESTING AND MESSAGE
07490  B  ERRHD  NO BACKUP SO CONTINUE WITH ERROR MESSAGE
07500  ERMES  B  ERMPPR  MOVE IN MESSAGE
07510  MRID  0+X1,OUTPUT+20  SCAN BY ERROR MESSAGE
07520  SAR  IR1  SET FATAL COMPLETION CODE
07530  ERRRTN EQU  *  IF NO LATCH RETURN
07540  MCw  :****ERROR****:,OUTPUT+18  ELSE PRINT AND CONTINUE
07550  :PUT PRINT,OUTPUT,  CLEAR
07560  MCw  ::OUTPUT+132  RESTORE LOST ITEM MARK ON INPUT BUFFER
07570  MCw  OUTPUT+132  IF FATAL CONTINUE TESTING
07580  MCw  =1C21,OUTPUT  SCAN BY ERROR MESSAGE
07590  SI  INPUT+80  MOVE IN MESSAGE
07600  B  FETCH  CHAINED MOVE
07610  H M EQU  *  TEST FOR BACKUP ACTION
07620  BCE  BCF,BCKUP,T  SET UP RETURN FROM CODE ERASURE.
07630  BCE  FERR,0+X1,F  ERASE CODE
07640  B  ERRHD  ERASE CODE
07650  ERMPPR  0+X1,0  SET FATAL COMPLETION CODE
07660  SAR  IR1  IF NO LATCH RETURN
07670  B  ERRRTN  ELSE PRINT AND CONTINUE
07680  FERR BCE  ERMPPR,OUTPUT+20,F  ELSE PRINT AND CONTINUE
07690  B  ERMPPR  IF FATAL CONTINUE TESTING
07700  MRID  0+X1,OUTPUT+20  SET FATAL COMPLETION CODE
07710  SAR  IR1  IF NO LATCH RETURN
07720  ERRHD MCw  :F:,CMPLCD  ELSE PRINT AND CONTINUE
07730  ERMPPR  BCE  ERASE CODE
07740  B  ERRRTN  SET FATAL COMPLETION CODE
07750  BCE  ERMPPR  IF NO LATCH RETURN
07760  SBR  PRRTN+4  ELSE PRINT AND CONTINUE
07770  MCw  ::OUTPUT+132  CLEAR PRINT LINE
07770  MCw  OUTPUT+132  CHAINED MOVE
07780 SI INPUT+80 RESTORE LOST ITEM MARK ON INPUT BUFFER
07790 MCw IR6+IR14 COMPUTE ERROR
07800 BS +INPUT+IR14 LOCATION
07810 MCw ::OUTPUT+1+X14 MARK IT
07820 MCw :1::OUTPUT CARRIAGE CONTROL
07830L :PUT PRINT::OUTPUT,
07840 MCw ::OUTPUT+1+X14 CLEAR ERROR MARK
07850 PRPRTN B *
07860 EQUIT ERRFLG
07870 OVFLW EQU *
07880L :PUT PRINT::OVFMES,
07890 ERRFLG MCw :::CMPLCD FATAL ERROR FLAG
07900 E Exit
07910 OVFMESDCw ::ISYMBOL TABLE OVERFLOW, JOB ABORTED:
07920 L DCw =1C45 RECORD MARK
07930**********************************************************************
07940* ERROR IS EXECUTED IF AN ATTEMPT IS MADE TO INTERPRET AN INVALID *
07950* OP CODE. THE JOB IS ABORTED. *
07960* *
07970* *
07980**********************************************************************
07990 EQUIT *
08000L :PUT PRINT::OPCDMS,
08010 H
08020 B Exit
08030 OPCDMSDCw ::INVALID OP CODE, JOB ABORTED:
08040 L DCw =1C45 RECORD MARK
08050**********************************************************************
08060* THE ENUM PRIMITIVE EXAMINES THE INPUT STRING FOR A FLOATING POINT *
08070* NUMBER SETTING THE TRUE-FALSE INDICATOR AND CONVERTING THE NUMBER *
08080* TO BINARY IN SYMBOL IF TRUE.
08090* *
08100* *
08110**********************************************************************
08120 ENUM MCw +DHOLD+IR12 SET POINTER TO DECIMAL HOLD FIELD
08130 BS SCALE CLEAR SCALE EXPONENT FIELD
08140 B NEXT
CALL DECIMAL NUMBER RECOGNIZER
IF DNUM.TURE THEN GO SAVE INPUT
AVOID POSSIBLE LOGICAL CONSTANT
DETERMINE FRACTION FOLLOWING
IF NONE THEN FETCH NEXT INSTRUCTION
CLEAR HOLD AREA
MOVE DECIMAL CHARACTERS
SAVE POINT FOR MOVING FRACTION
DETERMINE FRACTION FOLLOWING
IF NONE MOVE DECIMAL FIELD FOR CONVERSION
IF DNUM THEN GO SAVE INPUT
IF DUM THEN GO SAVE INPUT
CLEAR SYMBOL AREA
TEST IF FIELD TOO LONG
WARNING MESSAGE
LOSE; OUTPUT+107
SET UP DECIMAL FIELD
FOR DECIMAL TO BINARY CONVERSION
LOOK FOR EXponent
TEST FOR EXPONENT
NO EXPONENT, GO CONVERT
TEST MINUS SIGN
TEST PLUS SIGN
SET ADDITION OP CODE
08520 B PLUSOP NO SIGN, TREAT AS PLUS
08530 SETSM MCw =1C35,ESGNOP SET SUBTRACTION OP CODE
08540 UPDATE BA =1B1,IR6 UPDATE INPUT POINTER
08550 PLUSOP BI NEXT,0+X6 IF END OF RECORD GO GET MORE
08560 B INM IF INCREASE THEN SIGNAL ERROR
08570 BCE BEF,TF,F ADJUST SCALE FACTOR EXPONENT
08580 ESGNOP BA SYMBOL+3,SCALE BINARY FORM OF EXPONENT
08590
08600 B FCNVRT GO CONVERT
08610 SETSP MCw =1C34,ESGNOP SET ADDITION OP CODE
08620 B UPDATE
08630 DNUM SBR DNSMTN+4 SAVE RETURN ADDRESS
08640 BS IR14 CLEAR INDEX
08650 BS IR13 AND COUNTER
08660 MCw :F;TF INITIAL TF SWITCH
08670 DNSMT MRSd 0+X6,IR14 INPUT CHARACTER TO INDEX REGISTER
08680 BIO MVDEC,1DTAB+X14 TEST FOR NUMERIC CHARACTER
08690 BCE (DNSMTN+1),TF,F IF NO NUMERICS RETURN
08700 SI SYMBOL-1+X13 MARK RIGHT END OF NUMERIC FIELD
08710 DNSMTN B * RETURN TO CALLER
08720 MVDEC MRSdI 0+X6,SMBOL+X13 MOVE NUMERIC CHAR TO SYMBOL FIELD
08730 SAR IR6 UPDATE INPUT POINTER
08740 BI NEXT,0+X6 TEST END OF RECORD
08750 BA =1B1,IR13 UPDATE CHARACTER COUNT
08760 MCw :T;TF SET TF FLAG TRUE
08770 B DNSMT LOOK FOR MORE
08780 FCNVRT DTB DHOLD+10,00 CONVERT DECIMAL FIELD TO BINARY IN FRO
08790 AAA 70 NORMALIZE IT
08800 MCw :P;EXPNSGN SET EXPONENT SIGN FLAG
08810 BBE SETN,SCALE-2+40 TEST EXPONENT SIGN FLAG FOR SCALPE FACTOR
08820 E TSTSCL
08830 SETN BS IR14 CONVERT NEGATIVE
08840 BS SCALE,IR14 SCALE EXPONENT
08850 MCw IR14,SCALE TO POSITIVE
08860 MCw :M;EXPNSGN SET EXPONENT SIGN FLAG
08870 TSTSCL C SCALE,3B600 TEST SCALE EXPONENT FOR VALID RANGE
08880 BH SCLOK
CE89C  B  ERMPRP  SET UP
0890C  MCW :EXPONENT OUT OF RANGE:, OUTPUT+25 ERROR MESSAGE
0891C  MCW :F:; CMPLCD  FATAL COMPLETION CODE
0892C  B  ERRPRT  PRINT IT
0893C  SCLOK C  SCALE:=380  IF EXPONENT ZERO CONVERSION FINISHED
0894C  BE FCVEND
0895C  BS IR14  CLEAR INDEX TO ZERO
0896C  SST SCALE, IR14, 17  INSERT LOW 4 BITS FOR INDEXING
0897C  EIM =4BB, IR14  LEFT 3 BITS TO INDEX CVTTAB
0898C  TMA CVTTAB+X14, 01  CONVERSION FACTOR TO FR1
0899C  ES FHOLD+7  CLEAR EIGHT CHAR FLOATING POINT FIELD
0900C  MCW SCALE, FHOLD+5  3 CHAR FIELD TO 8 CHAR FLTNG PT FIELD
0901C  TMA FHOLD+7, 03  LOAD IT TO FR3
0902C  BMS 31, 04  SHIFT RIGHT 4 BITS
0903C  TAM FHOLD+7, 30  STORE IT
0904C  BS IR14  CLEAR
0905C  SST FHOLD+5, IP14, 17  INSERT LOW 4 BITS TO INDEX
0906C  BIM =4BB, IR14  LEFT 3 BITS TO INDEX CVTTAB
0907C  TMA CTAB16+X14, 02  CONVERSION FACTOR TO FR2
0908C  FINTM MAA 21  COMPUTE INTERMEDIATE FACTOR
0909C  TLA 02  SAVE LOW ORDER FOR DOUBLE PRECISION
0910C  BMS 31, 04  NEXT 4 BITS
0911C  TAM FHOLD+7, 30  STORE IT
0912C  BS IR14  CLEAR
0913C  SST FHOLD+5, IP14, 03  MAX 10 BITS FOR SCALE EXPONENT
0914C  BIM =4BB, IR14  SHIFT FOR INDEX
0915C  TMA CTB256+X14, 03  CONVERSION FACTOR TO FR3
0916C  FLOW MAA 32  LOW ORDER FACTOR
0917C  MAA 31  HIGH ORDER FACTOR
0918C  TLA 03  SAVE LOW ORDER
0919C  AAA 32  ACCUMULATE LOW ORDER FACTORS
0920C  BCE FDIV, EXP, SGN, M  TEXT EXPONENT SIGN
0921C  MAA 01  MULTIPLY BY HIGH ORDER SCALE FACTOR
0922C  TLA 03  SAVE LOW ORDER
0923C  MAA 02  LOW ORDER SCALE FACTOR
0924C  AAA 32  ACCUMULATE LOW ORDER FACTORS
0925C  AAA 21  ADD TO UNROUNDED RESULT
ROUND
IT
PUT IT IN FRO
CONVERSION DONE
ADJUST DIVIDEND BY ACCUMULATED LOW ORDER
ROUND
IT
DIVIDE CONVERTED NUM BY SCALE FACTOR
SAVE REMAINDER
DIVIDE
AND
ROUND QUOTIENT
STORE CONVERTED NUMBER
SYMBOL FIELD FOR OUTPUT
SET TF FLAG
NEXT INSTRUCTION
SCALE EXPONENT FIELD
DECIMAL CHAR HOLD FIELD
FLOATING POINT HOLD FIELD
CONVERSION TABLE
09630 DCW F1E15
09640 DCW F1E0
09650 DCW F1E16
09660 DCW F1E32
09670 DCW F1E48
09680 DCW F1E64
09690 DCW F1E80
09700 DCW F1E96
09710 DCW F1E112
09720 DCW F1E128
09730 DCW F1E144
09740 DCW F1E160
09750 DCW F1E176
09760 DCW F1E192
09770 DCW F1E208
09780 DCW F1E224
09790 DCW F1E240
09800 DCW F1E256
09810 DCW F1E512
09820 ***************************************************************************
09830 ***************************************************************************
09840 THE MARK PRIMITIVE PUSHES THE ADDRESS OF THE NEXT OUTPUT STRING
09850 LOCATION ON THE CONTROL STACK FOR LATER USE BY THE SAVE PRIMITIVE.
09860***************************************************************************
09870***************************************************************************
09880***************************************************************************
09890 MARK BA =1B9,IR2 BUMP STACK POINTER
09900 MCw IR5,0-5+X2 SAVE OUTPUT CODE ADDRESS
09910 MCw :M:,0-9+X2 MARK STACK ELEMENT TYPE
09920 B STKOVF CHECK STACK OVERFLOW
09930***************************************************************************
09940***************************************************************************
09950 SAVE PUSHES THE CODE GENERATED SINCE THE LAST MARK OPERATION
09960 ON THE VARIABLE LENGTH CODE STACK AND RESETS THE OUTPUT LOCATION
09970 BACK TO THE MARKED LOCATION.
09980***************************************************************************
09990***************************************************************************
THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE
CONTROL STACK.

STKSYM MCW SYMBOL+7,8+X2 MOVE SYMBOL TO STACK
MCW :S:+0+X2 MARK STACK ELEMENT AS SYMBOL
BA =1B9,IR2 BUMP STACK POINTER
B STKOVF CHECK FOR STACK OVERFLOW

STKSYM MCW SYMBOL+7,8+X2 MOVE SYMBOL TO STACK
MCW :S:+0+X2 MARK STACK ELEMENT AS SYMBOL
BA =1B9,IR2 BUMP STACK POINTER
B STKOVF CHECK FOR STACK OVERFLOW
THE RESTOR PRIMITIVE RESTORES THE TOP OF THE VARIABLE LENGTH CODE STACK TO THE OUTPUT STRING.

RESTOR LCA STKEND,IR14
BCE DORES,1+X14,77 TREAT AS NO-OP IF NULL
B FETCH STK
DORES MCw 4+X14,STKEND RESTORE PREVIOUS STK END POINTER
MCw 4+X14,IR13 ALSO TO IR13 FOR LOOP TEST
BA =1B5,IR14 POINT TO CODE TO MOVE
RESTST C IR13,IR14 TEST MOVE NOTE POSSIBLE
BH FETCH COMPLETION NULL RESTORE
MRSDR 0+X14,0+X5 MOVE CODE
SAR IR14 AND ADJUST
SBR IR5 POINTERS
BA =1B1,IR3
RESTST

THE CHKSYM PRIMITIVE TESTS THE EIGHT CHARACTER FIELD ON TOP OF THE CONTROL STACK AGAINST SYMBOL SETTING THE TRUE-FALSE CODE.

CHKSYM BS =1B9,IR2 ADJUST STACK POINTER
C 8+X2,SYMBOL+7 TEST AGAINST SYMBOL VALUE
BE SET IF EQUAL SET TRUE
B SETF ELSE SET FALSE

SWAP SWAPS THE TOP TWO ELEMENTS ON THE CONTROL STACK.

SWAP MCw 0-1+X2,SWPTMP MOVE TOP TO TEMPORARY
SI 0-18+X2 ITEM MARK FOR NEXT MOVE
MLIDI 0-10+X2,0-1+X2 SWAP
MCw SWPTMP,0-10+X2
B RESTST
THE ADD AND MULT PRIMITIVES COMPREHEND THE BINARY ARITHMETIC
CAPABILITIES (TWO ADDRESS) OF THE META9 PSEUDO-MACHINE.

```
10810 ADD   B  GTOPRA  GO GET OPERAND ADDRESSES
10820   SW  0+X11  WORD MARK TO STOP ADDITION
10830   SW  0+X10
10840  MRIN  0+X10,0  FIND RIGHT POSITION
10850   SAR  IR10  SET INDEX
10860  MRIN  0+X11,0
10870   SAR  IR11  REGISTERS
10880   BA  0-1+X10,0-1+X11  BINARY ADD
10890   B  FETCH
10900 MULT   B  GTOPRA  GO GET OPERAND ADDRESSES
10910   SW  0+X10  WORD MARK TO STOP MOVE
10920  MRIN  0+X10,0  FIND
10930   SAR  IR10  RIGHT END
10940   BS  MFLD1  CLEAR RECEIVING FIELD
10950   MCW  0-1+X10,MFLD1  MOVE MULTIPLIER
10960   SW  0+X11  WORD MARK TO STOP MOVE
10970  MRIN  0+X11,0  FIND
10980   SAR  IR11  RIGHT END
10990   BS  MFLD2  CLEAR RECEIVING FIELD
11000   MCW  0-1+X11,MFLD2  MOVE MULTIPLICAND
11010   BIM  MFLD1,MFLD2  BINARY MULTIPLY REQUIRES 24 BITS
11020   MCW  MFLD2,0-1+X11  PUT BACK RESULT
11030   B  FETCH
11040  MFLD1 DCW  =4BO
11050  MFLD2 DCW  =4BO
11060  GTOPRA SBR  GTRTN
11070  MRID  0+X1,IR10-2  FIRST OPERAND ADDRESS
11080   SAR  IR1  UPDATE LOCATION COUNTER
11090  MRID  0+X1,IR11-2  SECOND OPERAND ADDRESS
11100   SAR  IR1  UPDATE LOCATION COUNTER
```
THE MOVE PRIMITIVE MOVES THE FIRST ADDRESSED FIELD TO THE SECOND.

MOVE B GTOPRA GET ADDRESSES IN IRIO AND IRll
MRIDI O+X10, O+X11 MOVE DATA AND TERMINATING ITEM MARK
B FETCH

THE BLOCK ENTRY PRIMITIVE CONSTRUCTS A NEW TABLE ENTRY IN THE
BLOCK LIST AND MAINTAINS BLOCK COUNTERS AND POINTERS.

BLKCNT DCW :0: BLOCK COUNTER
PRVBLK DCw =1C00 PREVIOUS BLOCK NUMBER
CRBLKT DCW =3 INITIAL SYMTAB SEARCH ENTRY
BLKENT BA =1B1, BLKCNT BLOCK COUNT
BS =1B4, SYMEND REDUCE SYMBOL TABLE END LOCATION

TO ACCOMODATE NEW BLOCK LIST ENTRY
MCW SYMEND, IR13 SET SURROUNDING BLOCK NUMBER
LCA PRVBLK, 1+X13 SET UP FOR NEXT BLOCK ENTRY
MCW BLKCNT, PRVBLK
BA =1B20, NEWSYM SPACE FOR DUMMY ENTRY
LCA NEWSYM, 4+X13 STORE IN BLOCK LIST
MCW SYMEND, CRBLKT SET FOR CURRENT SYMTAB SEARCH
B FETCH

THE BLOCK EXIT PRIMITIVE RESTORES CRBLKT POINTER AND PRVBLK NUMBER FOR THE SURROUNDING BLOCK. THE SYMBOL TABLE ENTRIES FOR THE TERMINATING BLOCK ARE SCANNED FOR UNRESOLVED SYMBOLS. UNRESOLVED ENTRIES ARE ADDED TO THE SURROUNDING BLOCK IF NOT FOUND IN THAT PORTION OF THE TABLE. APPROPRIATE LINKING PARAMETERS ARE SET FOR THE RESOLVE PRIMITIVE TO USE. Thus DIABOLICAL LABEL REFERENCES IN A BLOCK STRUCTURE ARE RESOLVABLE.
BLKSAV DCw =3
BLKPRM DCw =1
BLKEXT MCw CRBLKT,BLKSAV  SAVE FOR UNRESOLVED SEARCH
MRSD 0*X1,BLKPRM  BLOCK EXIT PARAMETER
SAR IR1
BS IR13  CLEAR
MCw CRBLKT,IR14  INSERT PREVIOUS BLOCK NUMBER
MCw 1*X14,IR13  NEW SURROUNDING BLOCK NUMBER
MCw 1*X14,PRVBLK
BIW =4B4,IR13  COMPUTE
MCw SVSYME,IR14  BLOCK LIST
BS IR13,IR14  LOCATION FOR SURROUNDING BLOCK
MCw IR14,CRBLKT  BLOCK LIST POINTER
MCw BLKSAV,IR12  TERMINATING BLOCK LIST POINTER
MCw 4*X12,IR12  SYMBOL TABLE POINTER
BCE FETCH,BLKPRM,01  TEST NO LABEL LINK UP
CHKUND C CHAIN+X12,:000:  CHECK FOR END OF
BE FETCH  BLOCK TABLE ENTRIES
MCw CHAIN+X12,IR12  NOTE 1ST TIME JUMP OVER DUMMY ENTRY
UDCHK BCE CHKPRV,DTYPE+X12,00  CHECK FOR UNDEFINED SYMBOLS
B CHKUND  LOOK FOR MORE
CHKPRV MCw NAME+X12,SYMBOL+7  SET NAME TO USE SEARCH SUBROUTINE
MCw :1:,SRCHTP  BLOCK ONLY SEARCH MODE
B SEARCH
ECE ADDSYM,TF,F  IF FALSE SYMBOL ADDED TO SURROUNDING
BCE PRVUN,0*X15,00  IF FOUND BUT STILL UNDEFINED SET MARKERS
SI 0*X15
MLIDw DIMCNT+X15,12  FOUND, SET VALUES FOR RESOLVE
CI 0*X15
B CHKUND
BCE ADDSYM  EQU *  IF FOUND BUT STILL UNDEFINED SET MARKERS
PRVUN Sw IR15-2  SAVE CURRENT ENTRY POINTER
MCw IR12,IR9
MCw CHAIN+X12,IR12  SAVE NEXT ENTRY POINTER
RESOLVE IS A TERMINAL PRIMITIVE WHICH RESOLVES FORWARD REFERENCES AND DETECTS ANY UNDEFINED ADDRESSES. THE OBJECT TEXT IS SCANNED FOR WORD MARKS TO FIND RELOCATABLE ADDRESSES. THE LEFTMOST BIT OF THE ADDRESS MARKS UNRESOLVED ADDRESSES.

RESOLVE BCE EXIT,CMPLCD,F EXIT IF FATAL COMPILATION TO THIS POINT
BS CVBFLD CLEAR
MCw IR3,CVFBLD-2 MOVE PROGRAM SIZE
TMA CVBFLD,00 LOAD TO FRO
BTD CVBFLD,00 CONVERT TO DECIMAL AND STORE
LCA EwORD,PSIZE EDIT CONTROL WORD
MCE CVBFLD,PSIZE MOVE AND EDIT
CW PSIZE-8 CLEAR WORD MARK
BS CVBFLD CLEAR
MCw INSTCT,CVFBLD-2 MOVE INSTRUCTION COUNT
TMA CVBFLD,00 LOAD TO FRO
BTD CVBFLD,00 CONVERT TO DECIMAL AND STORE
LCA EwORD,ICOUNT EDIT CONTROL WORD
MCE CVBFLD,ICOUNT MOVE AND EDIT
CW ICOUNT-8 CLEAR WORD MARK
:PUT PRINT,EXITMS, PRINT IT
BS CVBFLD
MCw SRCHCT,CVFBLD-2 MOVE SEARCH COUNT
TMA CVBFLD,00
BTD CVBFLD,00
LCA EwORD,SCOUNT
MCE CVBFLD,SCOUNT
BS CVBFLD
<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>12220</td>
<td>MCW</td>
<td>CMPCNT,CVBFLD-2 MOVE COMPARISON COUNT</td>
</tr>
<tr>
<td>12230</td>
<td>TMA</td>
<td>CVBFLD,00</td>
</tr>
<tr>
<td>12240</td>
<td>BTD</td>
<td>CVBFLD,00</td>
</tr>
<tr>
<td>12250</td>
<td>LCA</td>
<td>EWORDT,TCOUNT</td>
</tr>
<tr>
<td>12260</td>
<td>MCE</td>
<td>CVBFLD,TCOUNT</td>
</tr>
<tr>
<td>12270</td>
<td>PUT</td>
<td>PRINT,TABMES,</td>
</tr>
<tr>
<td>12280</td>
<td>BS</td>
<td>CVBFLD</td>
</tr>
<tr>
<td>12290</td>
<td>MCW</td>
<td>SYMCTNY,CVBFLD-2 MOVE TABLE ENTRY COUNT</td>
</tr>
<tr>
<td>12300</td>
<td>TMA</td>
<td>CVBFLD,00</td>
</tr>
<tr>
<td>12310</td>
<td>BTD</td>
<td>CVBFLD,00</td>
</tr>
<tr>
<td>12320</td>
<td>LCA</td>
<td>EWORDE,ECOUNT</td>
</tr>
<tr>
<td>12330</td>
<td>MCF</td>
<td>CVBFLD,ECOUNT</td>
</tr>
<tr>
<td>12340</td>
<td>PUT</td>
<td>PRINT,TABCNT,</td>
</tr>
<tr>
<td>12350</td>
<td>LCA</td>
<td>GENLOC,IR15 START OF COMPILED CODE</td>
</tr>
<tr>
<td>12360</td>
<td>SCAN</td>
<td>MRRN 0+X15,0 SCAN FOR WORD MARK</td>
</tr>
<tr>
<td>12370</td>
<td>SAR</td>
<td>IR15 SAVE NEXT POSITION</td>
</tr>
<tr>
<td>12380</td>
<td>C</td>
<td>IR15,IR5 DETERMINE</td>
</tr>
<tr>
<td>12390</td>
<td>BL</td>
<td>PLIST COMPLETION</td>
</tr>
<tr>
<td>12400</td>
<td>BBE</td>
<td>GETADD+0-1+X15,40 TEST UNRESOLVED ADDRESS</td>
</tr>
<tr>
<td>12410</td>
<td>B</td>
<td>SCAN</td>
</tr>
<tr>
<td>12420</td>
<td>GETADD</td>
<td>MRIDI 0-1+X15,IR14-2 UNRESOLVED ADDRESS IS POINTER TO SYMBOL</td>
</tr>
<tr>
<td>12430</td>
<td></td>
<td>GETADD+ETYPE+X14,77</td>
</tr>
<tr>
<td>12440</td>
<td>HA</td>
<td>=1C40,IR14-2 REMOVE UNRESOLVED MARKER</td>
</tr>
<tr>
<td>12450</td>
<td>C</td>
<td>SYMSTR,IR14 TEST FOR</td>
</tr>
<tr>
<td>12460</td>
<td>EL</td>
<td>STERR VALID</td>
</tr>
<tr>
<td>12470</td>
<td>C</td>
<td>SYMEND,IR14 SYMBOL</td>
</tr>
<tr>
<td>12480</td>
<td>BH</td>
<td>STERR ADDRESS</td>
</tr>
<tr>
<td>12490</td>
<td>CHNTST</td>
<td>BCE CHNADD+ETYPE+X14,77</td>
</tr>
<tr>
<td>12500</td>
<td>BBE</td>
<td>NDEFN+ADDR-2+X14,40 IF STILL UNDEFINED PRINT ERROR</td>
</tr>
<tr>
<td>12510</td>
<td>B</td>
<td>ADDRSL</td>
</tr>
<tr>
<td>12520</td>
<td>CHNADD</td>
<td>MCW CHAIN+X14,IR14 CHAIN TO SURROUNDING BLOCK</td>
</tr>
<tr>
<td>12530</td>
<td>B</td>
<td>CHNTST GO TEST FOR ADDITIONAL CHAINING</td>
</tr>
<tr>
<td>12540</td>
<td>ADDRSL</td>
<td>SI ADDR+X14</td>
</tr>
<tr>
<td>12550</td>
<td>MRIDR</td>
<td>LEVEL+X14,0-2+X15 SET LEVEL AND DISPLACEMENT</td>
</tr>
<tr>
<td>12560</td>
<td>BCE</td>
<td>L8VTST+0-3+X15,76 TEST POSSIBLE LABEL VARIABLIF - LABEL</td>
</tr>
<tr>
<td>12570</td>
<td></td>
<td>BCE CHNADD+ETYPE+X14,77</td>
</tr>
<tr>
<td>12580</td>
<td>B</td>
<td>SCAN</td>
</tr>
</tbody>
</table>
12590 LBVTST BCE LBVDTYPE+X14+05 TEST LABEL VARIABLE IN SYMTAB
12600 B SCAN ELSE CONTINUE
12610 LBV BNP SCAN+0-4+X15 MAKE SURE OP CODEP PRECEDES ADDR
12620 MRSD :5+0-3+X15 CHANGE DATA TYPE TO LABEL VARIABLE
12630 B SCAN AND CONTINUE
12640 STERR EQU *
12650L :PUT PRINT,CMPLM5,
12660 H
12670 B EXIT
12680 CMPLMSDCW :ACOMpiler ERROR DISCOVERED DURING RESOLVE:
12690L DCw =1C45
12700 NTDFN MCw NAME+X14,PSYM
12710L :PUT PRINT,NDFMES,
12720 MCw :F:,CMPLCD SET COMPLETION CODE
12730 B SCAN
12740 NDFMESDCW :1UNDEFINED SYMBOL :
12750 PSYM DCw :
12760L DCw =1C45
12770 TABCNTDCw :A ****SYMBOL TABLE ENTRY COUNT =:
12780 ECOUNT DC =9
12790 DC :****:
12800L DCw =1C45
12810 EXITMSDCw :2 ****COMPILED PROGRAM SIZE =:
12820 PSIZE DC =9
12830 DC :: METAX INSTRUCTION COUNT =:
12840 ICOUNT DC =9
12850 DC :****:
12860L DCw =1C45
12870 SRCHCT DCw =4B0
12880 CMPCNT DCw =4B0
12890 TABMESDCw :B ****SYMTAB SEARCH COUNT =:
12900 SCOUNT DC =9
12910 DC :: SYMTAB COMPARE COUNT =:
12920 TCOUNT DC =9
12930 DC :****:
12940L DCw =1C45
12950 EWORD DCw : , *0 :
EXIT is a terminal point in the program, clearing certain punctuation before exiting.

EXIT EQU *
CI IR1
CI IR2
CI IR3
CI IR15
CW IR7-1
B (164)

NEXT is a subroutine which scans the input string for the next non-blank character reading new record(s) if required. If an end of file is sensed a message is printed and the program exits.

NEXT SBR NxTRTN+4
ENDTST BI GETCRD*0+x6
BLKTST BCE NBLNK*0+x6,15
NXTRTN B *
NBLNK BA =181,IR6
ENDTST
GETCRD EQU *
:GET READ,
MCW =1C21,INPUT-1 CARRIAGE CONTROL
:PUT PRINT,INPUT-1,
MCW +INPUT,IR6
C INPUT+3,:1EOF: END OF FILE TEST
BNE BLKTST
:PUT PRINT,EOFMES,
ERRFLG
EOFMES DCW ;UNEXPECTED END OF FILE, JOB ABORTED:
DCw =1C45 RECORD MARK
PLIST IS EXECUTED IF A POST LISTING IS REQUESTED.

EXIT IF POST LIST NOT REQUESTED

PROGRAM COUNTER

START OF GENERATED CODE

CLEAR

PRINT LINE

CARRIAGE CONTROL

RESTORE LOST ITEM MARK ON INPUT BUFFER

CLEAR CONVERSION FIELD

LOAD TO FRO

CONVERT TO DECIMAL

REMOVE SIGN BITS

DETERMINE SYMBOLIC LABEL, IF ANY

JUMP OVER LITERAL DESIGNATOR

SET UP

LITERAL OPERAND

FOR PRINTING

KILL FIRST TIME BRANCH
13700  BA  =1B1,IR13  BUMP PROGRAM COUNT
13710  BA  =1B1,IR15  AND CODE POINTER
13720  BCE  FMTCD=0-1+X15,77  DETERMINE FORMAT CODE
13730  BCE  ALLOC,0-1+X15,12  TEST ALLOC OP CODE
13740  BCE  LDA,0-1+X15,20  TEST LOAD ADDRESS OP CODE
13750  BS  IR12  CLEAR
13760  MRSID  0-1+X15,IR12  INSERT OP CODE
13770  BCE  IR12  MULT BY TABLE ENTRY SIZE
13780  MCW  OPTAB+4+X12,OUTPUT+21  OP CODE TO PRINT
13790  BCE  PLSTPR,OPTAB+5+X12,00  TEST FOR NO OPERANDS
13800  BBE  TLITRL,OPTAB+5+X12,60  TEST POSSIBLE LITERAL
13810  LITERL  EQU  LITRL
13820  BCE  LITERL,OPTAB+5+X12,01  TEST SINGLE CHARACTER LITERAL
13830  BCE  ADDR4,OPTAB+5+X12,04  TEST FOUR CHAR ADDRESS
13840  BCE  TWOOP,OPTAB+5+X12,10  TEST TWO OPERANDS
13850  B  LITERL
13860  TWOOP  B  ADDFV
13870  MCW  IR14,IR11
13880  B  ADDFV
13890  MCW  NAME+X11,OUTPUT+30
13900  MCW  ::::OUTPUT+31
13910  MCW  NAME+X14,OUTPUT+39
13920  B  PLSTPR
13930  ALLOC  MCW  ::ALLOC::OUTPUT+21  SET OP CODE
13940  B  ADDFV  GET ADDRESS OF FIRST SYMBOL
13950  MCW  IR14,IR11  SAVE IT
13960  B  ADDFV  SECOND SYMBOL
13970  MCW  NAME+X11,OUTPUT+30  FIRST SYMBOL TO PRINT
13980  MCW  ::::OUTPUT+31
13990  MCW  NAME+X14,OUTPUT+39  SECOND SYMBOL TO PRINT
14000L  :PUT  PRINT  PRINT,DPVMES,  DOPE VECTOR MESSAGE
14010  MCW  ::::OUTPUT+32  CLEAR
14020  MCW  OUTPUT+132
14030  MCW  =12C1,OUTPUT  CARRIAGE CONTROL
14040L  :PUT  PRINT,DPVMES,  DOPE VECTOR MESSAGE
14050  BS  IR12  CLEAR
14060  MRSID  0+X15,IR12  INSERT DIM COUNT
BIM = 4B10, IR12  
MULT BY SIZE OF BOUND PAIR CODE

BA = 1B3, IR12  
SIZE OF LENGTH AND DIM COUNT FIELDS

BA  IR12, IR13  
BUMP PROG COUNTER

BA  IR15, IR12  
END OF DOPE VECTOR

SW  0-1+X12  
MARK FOR MOVE

MCW  ":;OUTPUT+15  
END OF DOPE VECTOR

MRRD  0+X15, OUTPUT+16  
MOVE IT

SAR  IR15
SBR  IR11

CW  0-1+X12
MCW  ":;OUTPUT+11

B  PLSTPR  
GO PRINT

DCW  A ***DOPE VECTOR CODE***:

LDA  :LDA  ;OUTPUT+21

SST  0+X15, LITCHR, 70  
LEFT THREE BITS

BCE  LITOPR, LITCHR, 70  
TEST FOR LITERAL

SST  0+X15, OTYPE, 07  
SAVE TYPE

B  ADDFV  
GET OPERAND SYMBOL

BCE  LNGCDE, OTYPE, 04  
TEST FOR STRING TYPE

B  PLSTPR  
GO PRINT

LNGCDE  ":;OUTPUT+32

MRSD  0+X15, OUTPUT+33  
MOVE IN

EXM  
LENGTH CODE

SAR  IR15
MCW  ":;OUTPUT+35

BA  = 1B2, IR13
MCW  ":;OUTPUT+34

OUTPUT

L  DCW  = 1C45

LDA  
OUTPUT

SST  0+X15, LITCHR, 70  
LEFT THREE BITS

BCE  LITOPR, LITCHR, 70  
TEST FOR LITERAL

SST  0+X15, OTYPE, 07  
SAVE TYPE

B  ADDFV  
GET OPERAND SYMBOL

BCE  LNGCDE, OTYPE, 04  
TEST FOR STRING TYPE

B  PLSTPR  
GO PRINT

LNGCDE  ":;OUTPUT+32

MRSD  0+X15, OUTPUT+33  
MOVE IN

EXM  
LENGTH CODE

SAR  IR15
MCW  ":;OUTPUT+35

BA  = 1B2, IR13
MCW  ":;OUTPUT+34

OUTPUT
14440 B PLSTPR
14450 ADDR5 B ADDDFV
14460 B PLSTPR
14470 ADDFR SBR AFRRTN+4
14480 MRID 0+X15,ADD4CN-3
14490 SAR IR15
14500 BA =1B4,IR13
14510 MCw SYMSTR,IR14
14520 A4COMP C 4+X14,ADD4CN
14530 BE ADFRFD
14540 BA =1B20,IR14
14550 C IR14,NEWSYM
14560 BEH A4COMP
14570 MISSAD MCw :*****:,OUTPUT+35
14580 B LITOPR
14590 ADFRFD MCw NAME+X14,OUTPUT+30
14600 AFRRTN B *
14610 ADD4CN DCw =4
14620 ADDFV SBR AFRVTN+4
14630 MRSD 0+X15,ADD5CN-4
14640 EXM
14650 EXM
14660 EXM
14670 EXM
14680 SAR IR15
14690 BA =1B5,IR13
14700 MCw SYMSTR,IR14
14710 A5COMP SI 4+X14
14720 MRID 0+X14,AD5CN1-4
14730 C ADD5CN,AD5CN1
14740 BE ADFVFD
14750 BA =1B20,IR14
14760 C IR14,NEWSYM
14770 BEH A5COMP
14780 BS =1B5,IR15
14790 BS =1B5,IR13
14800 B MISSAD
14810 ADFVFD MCW NAME+X14,OUTPUT+30
14820 AFVR TN B *
14830 ADD5CN Dcw =5
14840 ADD5CN B Dcw =5
14850 SYMADD B OPCODE FIRST TIME ONLY
14860 MCW SYMSTR,IR14 TABLE START
14870 RELOCT BNP NOSYM,2*X14 IGNORE NON-RELOCATABLE SYMBOLS
14880 C 4*X14,IR13 TEST
14890 BE SYMFND EQUALITY
14900 NOSYM BA =1B20,IR14 NEXT
14920 =C SYMEND,IR14 TEST
14930 BH OPCODE TABLE END
14940 B RELOCT TEST NEXT
14950 SYMFND MCW NAME+X14,OUTPUT+15
14960 B OPCODE
14970 L PEXIT MCW :END PROGRAM: OUTPUT+32
14970 L :PUT PRINT,
14980 H EXIT
14990 T LITRL SST 0+X15,LITCHR,70 LEFT THREE BITS OF TYPE CHAR
15000 BCE LITOPR,LITCHR,70 TEST FOR LITERAL
15010 B ADDR5 MUST BE FIVE CHAR ADDR
15020 LITCHR DCW :0;
15030 FMTCDE EQU *
15040 L :PUT PRINT,FMTMES, MESSAGE
15050 MCW :"::OUTPUT+15 BUILD
15060 MCW +OUTPUT+16,IR12 FORMAT
15070 NFMTCH M RSD 0+X15,0+X12 LITERAL
15080 SAR IR15
15090 SBR IR12
15100 BA =1B1,IR13
15110 BCE FMTDNE,0+X15,77
15120 B NFMTCH
15130 FMTDNE MCW :"::0+X12 FINISH
15140 BA =1B1,IR13 BUMP
15150 BA =1B1,IR15 COUNTERS
15160 B PLSTPR
15170 FMTMES D CW :A ***FORMAT CODE***:
OP CODE OPERAND TABLE FOR POSTLISTING

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>LDA 5</td>
</tr>
<tr>
<td>11</td>
<td>LD 5</td>
</tr>
<tr>
<td>12</td>
<td>STQ 0</td>
</tr>
<tr>
<td>20</td>
<td>SST 0</td>
</tr>
<tr>
<td>21</td>
<td>ENTPR1</td>
</tr>
<tr>
<td>22</td>
<td>RETRNO</td>
</tr>
<tr>
<td>23</td>
<td>JUMPA0</td>
</tr>
<tr>
<td>25</td>
<td>JUMPT4</td>
</tr>
<tr>
<td>26</td>
<td>JUMPF4</td>
</tr>
<tr>
<td>27</td>
<td>STCKC1</td>
</tr>
<tr>
<td>28</td>
<td>COMPC1</td>
</tr>
<tr>
<td>29</td>
<td>SWAP 0</td>
</tr>
<tr>
<td>30</td>
<td>POPUPO</td>
</tr>
<tr>
<td>31</td>
<td>ADD 0</td>
</tr>
<tr>
<td>32</td>
<td>MULT 0</td>
</tr>
<tr>
<td>33</td>
<td>SUB 0</td>
</tr>
<tr>
<td>34</td>
<td>DIV 0</td>
</tr>
<tr>
<td>35</td>
<td>NEG 0</td>
</tr>
<tr>
<td>36</td>
<td>ERRORO</td>
</tr>
<tr>
<td>40</td>
<td>ERRORO</td>
</tr>
<tr>
<td>41</td>
<td>ERRORO</td>
</tr>
<tr>
<td>42</td>
<td>ERRORO</td>
</tr>
<tr>
<td>43</td>
<td>ERRORO</td>
</tr>
<tr>
<td>44</td>
<td>ERRORO</td>
</tr>
<tr>
<td>50</td>
<td>ERRORO</td>
</tr>
<tr>
<td>51</td>
<td>ERRORO</td>
</tr>
<tr>
<td>52</td>
<td>ERRORO</td>
</tr>
<tr>
<td>53</td>
<td>ERRORO</td>
</tr>
</tbody>
</table>

**Notes:**
- OP CODE 10: LDA 5
- OP CODE 11: LD 5
- OP CODE 12: STQ 0
- OP CODE 20: SST 0
- OP CODE 21: ENTPR1
- OP CODE 22: RETRNO
- OP CODE 23: JUMPA0
- OP CODE 25: JUMPT4
- OP CODE 26: JUMPF4
- OP CODE 27: STCKC1
- OP CODE 28: COMPC1
- OP CODE 29: SWAP 0
- OP CODE 30: POPUPO
- OP CODE 31: ADD 0
- OP CODE 32: MULT 0
- OP CODE 33: SUB 0
- OP CODE 34: DIV 0
- OP CODE 35: NEG 0
- OP CODE 36: ERRORO
- OP CODE 40: ERRORO
- OP CODE 41: ERRORO
- OP CODE 42: ERRORO
- OP CODE 43: ERRORO
- OP CODE 44: ERRORO
- OP CODE 50: ERRORO
- OP CODE 51: ERRORO
- OP CODE 52: ERRORO
- OP CODE 53: ERRORO
SYMBOL DEFINITION - CARD REFERENCE INDEX

A4COMP 14520;  A5COMP 14710;  AD5CN1 14840;  ADD4CN 14610;  ADD5CN 14830;
ADD 10810;  ADDFR 14470;  ADDFV 14620;  ADDR4 14360;  ADDR5 14450;
ADDR 04950;  ADDRSL 12540;  ADDSYM 11810;  ADFRFD 14590;  ADFVFD 14810;
AFRRTN 14600;  AFRVRTN 14820;  ALLOC 13930;  BCKUP 06130;  BEF 07330;
BLKCNT 11260;  BLKEXT 11290;  BLKPRM 11510;  BLKSAV 11500;
BLKTST 13180;  BM 07610;  BRANCH 01860;  BRNCHF 01900;  BRNCHT 01880;
CANCEL 06190;  CHAIN 04920;  CHKPRV 11710;  CHKSYM 10600;  CHKUND 11660;
CHNADD 12520;  CHNTST 12490;  CMPCNT 12880;  CMPLCD 00420;
CMPLMS 12680;  COMP 05850;  COMPT 05970;  CONUM 03760;  CONVERT 06920;
CRLKST 11280;  CTAB16 09640;  CTB256 09800;  CVBFLD 07050;  CVTTAB 09480;
DECMVE 08340;  DEFMES 07460;  DHOLOD 09450;  DIMCNT 04930;  DMVE 08200;
DNMRTN 08710;  DNMTST 08670;  DNUM 08630;  DOENT 05130;  DORES 10440;
DORRTN 02260;  DOSAVE 10020;  DPVMES 14190;  DTYPE 04970;  ECOUNT 12780;
ELATCH 00400;  ENDTST 13170;  ENTA 04600;  ENTL 04610;  ENLLIT 04730;
ENTLOC 04450;  ENUM 08120;  EOFMES 13310;  ERASE 07140;  ERMES 07500;
TAEMES 12890; TCOUNT 12920; TEST 03080; TESTLB 02600; TESTT 03170;
TF 00710; TLITRL 14990; TSTAN 03340; TSTLB1 02730; TSTLB2 02760;
TSTLIT 05700; TSTSLC 08870; TSTTBA 05540; TSTTEL 05550; TVEC 01250;
TWOOP 13860; UUCHK 11690; UPDATE 08540;

***INSTRUCTION COUNT = 310,489***
XI. APPENDIX C
// CONTROL RECORD.
FUNCTION PLXCPL. INTERPRETER=MTXINT04.
START SYMTAB AT 8000, END SYMTAB AT 20000.
STACK START AT 5000, END STACK AT 7990.
EXECUTE AT 20001.
POSTLIST=YES.

PROCEDURE MAIN:

IDENTIFICATION:

PROGRAM-ID: TEST.

AUTHOR: J. R. VAN DOREN.

SOURCE LANGUAGE: PLEX.

OBJECT LANGUAGE: PLEX PSEUDO-MACHINE CODE.

OBJECT INTERPRETER: PLXINT.

PURPOSE:

TEST DEMONSTRATES MOST OF THE FEATURES OF THE PLEX LANGUAGE.
/*
  INPUT / OUTPUT (INCLUDING STRING I/O)
  */

IOBLK:BEGIN;
DECLARE (A,B,C,M(5)) FIXED, (X,Y,Z) FLOAT, (T) CHAR(20);
PUT EDIT ("BEGIN I/O BLOCK") (SKIP(3),A);
GET EDIT (S) (A(15));
GET EDIT (A,B,C) (COL(1),I(5),I(5),I(5));
PUT STRING (T) EDIT (A,B,C) (I(5));
PUT EDIT (S,T,A,B,C) (SKIP,A,COL(20),A,COL(40),I(5),I(5),I(5));
GET EDIT (S,T) (SKIP,A(20),A(20));
GET STRING (5) EDIT (X,Y) (E(10));
GET STRING (T) EDIT (Z) (E(10));
PUT EDIT (X,Y,Z,S,T) (SKIP(2),T(20),E(20),E(20),SKIP,A(20),A(20));
DO A=1 TO 5;
  M(A)=A;
END;
PUT EDIT((M(A) DO A=1 TO 5)) (I(5));
PUT EDIT ("EXIT I/O BLOCK") (SKIP,A);
END IOBLK;

/*
  DO GROUPS
  */

DOGRP:BEGIN; DECLARE (I,J,K,M(-2:10,10)) FIXED;
PUT EDIT ("BEGIN DO GROUPS") (SKIP(3),A);
DO I=0 TO 4;
  DO CASE 4-I;
    PUT EDIT ("CASE 0") (A);
    PUT EDIT ("CASE 1") (A);
    PUT EDIT ("CASE 2") (A);
    PUT EDIT ("CASE 3") (A);
    PUT EDIT ("CASE 4") (A);
  END CASE;
END;
DO I=10 TO -2 BY -1;

M(I,5)=0;
DO J=(3*2) - 5 TO 10 WHILE(J<5);
  M(I+J)=J;
END;
PUT EDIT ("I = ",M(I+1)," M(I+1) = ",M(I+1)," M(I+5) = ",M(I+5)) (A+I(5));
END;
PUT EDIT ("EXIT DO GROUPS") (SKIP,A);
END DOGRP;

/*
ARITHMETIC
*/

ARITH:BEGIN; DECLARE (X,Y,Z) FLOAT; (A,B,C) FIXED;
PUT EDIT ("ENTER ARITHMETIC BLOCK") (SKIP(3),A);
DO A=1 TO 10:
  X=A*1.33; Y=X/A;
  Z=IF A<5 THEN A ELSE 0.0;
  PUT EDIT (X,Y,Z) (E(20));
  PUT EDIT (A) (1(5));
END;
PUT EDIT ("EXIT ARITHMETIC BLOCK") (SKIP,A);
END ARITH;

PROCEDURE CALLS AND RECURSION

/* RECURSIVE FACTORIAL EXAMPLE */

RPROC:BEGIN; DECLARE NFACT RETURNS(FIXED);
  NFACT: PROCEDURE (I); DECLARE I FIXED;
  IF I=0 THEN RETURN (1);
  RETURN (NFACT((I-1))*I);
END NFACT;
PUT EDIT ("ENTER RPROC") (SKIP(3),A);
PUT EDIT ("7 FACTORIAL ="+NFACT(NFACT(3)+1)) (SKIP,A,I(IO));
PUT EDIT ("EXIT RPROC") (SKIP,A);
END RPRGC;

/* TRANSLATION OF INFIX ARITHMETIC EXPRESSIONS TO POSTFIX FORM USING RECURSIVE PROCEDURES */

POSTF:BEGIN;
DECLARE (LITERAL, NUMBER, ID, EXP1, EXP2, NEXT, TERM, PRIMARY) RETURNS(LOGICAL),
(INPUT, OUTPUT) CHAR(80), (I, J) FIXED, CHAR CHAR(1);
OUT: PROCEDURE(OUTCHAR); DECLARE OUTCHAR CHAR(1);
SUBSTR(OUTPUT, J+1) = OUTCHAR; J = J+1;
END OUT;
NEXT: PROCEDURE;
DO WHILE (SUBSTR(INPUT, I, 1) = " ");
I = I+1;
IF I > 80 THEN RETURN(.F.);
END;
RETURN(.T.);
END NEXT;
NUMBER: PROCEDURE;
IF .NOT. NEXT THEN RETURN(.F.);
IF SUBSTR(INPUT, I, 1) < = "9" .AND. SUBSTR(INPUT, I, 1) > = "0" THEN DO;
CHAR = SUBSTR(INPUT, I, 1); I = I+1;
RETURN(.T.);
END;
RETURN(.F.);
END NUMBER;
ID: PROCEDURE;
IF .NOT. NEXT THEN RETURN(.F.);
IF SUBSTR(INPUT, I, 1) = "A" .AND. SUBSTR(INPUT, I, 1) < = "Z" THEN DO;
CHAR = SUBSTR(INPUT, I, 1); I = I+1;
RETURN(.T.);
END;
RETURN(.F.);
END ID;
LITERAL: PROCEDURE (TEST); DECLARE TEST CHAR(1);
  IF .NOT. NEXT THEN RETURN(.F.);
  IF SUBSTR(INPUT, I+1) = TEST THEN DO; I = I+1; RETURN(.T.); END;
  RETURN(.F.);
END LITERAL;

PRIMARY: PROCEDURE;
  IF LITERAL ("\(") THEN
    DO;
      IF .NOT. EXP1 THEN RETURN(.F.);
      IF TEST THEN RETURN(.F.);
      RETURN(.T.);
    END;
  IF NUMBER THEN DO; CALL OUT(CHAR); RETURN(.T.); END;
  IF ID THEN DO; CALL OUT(CHAR); RETURN(.T.); END;
  RETURN(.F.);
END PRIMARY;

TERM: PROCEDURE;
  IF .NOT. PRIMARY THEN RETURN(.F.);

MULT: IF LITERAL("*") THEN
  DO;
    IF .NOT. PRIMARY THEN RETURN(.F.);
    CALL OUT("*"); GO TO MULT;
  END;
  IF LITERAL("/") THEN
    DO;
      IF .NOT. PRIMARY THEN RETURN(.F.);
      CALL OUT("/"); GO TO MULT;
    END;
  RETURN(.T.);
END MULT;

EXP2: PROCEDURE;
  IF LITERAL("-") THEN
    DO;
      IF .NOT. TERM THEN RETURN(.F.);
      CALL OUT("-"); RETURN(.T.);
    END;
  IF LITERAL("+") THEN
DO;
    IF .NOT. TERM THEN RETURN(.F.);
    RETURN(.T.);
END;
IF TERM THEN RETURN(.T.); RETURN(.F.);
END EXP2;
EXP1:  PROCEDURE;
    IF .NOT. EXP2 THEN RETURN(.T.);
PLUS: IF LITERAL("+") THEN
    DO;
        IF .NOT. TERM THEN RETURN(.F.);
        CALL OUT("+"); GO TO PLUS;
    END;
    IF LITERAL("-") THEN
    DO;
        IF .NOT. TERM THEN RETURN(.F.);
        CALL OUT("-"); GO TO PLUS;
    END;
    RETURN(.T.);
END EXP1;
/* START HERE */
PUT EDIT ("ENTER POSTFIX") (SKIP(3),A);
GET EDIT (INPUT) (SKIP*A(80));
I*,J=1;
PUT EDIT ("INFIX EXPRESSION =",INPUT) (SKIP*A*X(2)*A(80));
OUTPUT=INPUT; /* CLEAR OUTPUT FIELD */
IF EXP1 THEN PUT EDIT ("POSTFIX EXPRESSION =",SUBSTR(OUTPUT,1,J-1)/";");
     (SKIP*A*X(2)*A);
ELSE PUT EDIT ("****ERROR****") (SKIP*A);
PUT EDIT ("EXIT POSTFIX") (SKIP*A);
END POSTF;

/*
PROCEDURE PARAMETER EXAMPLE TO TEST GLOBAL DISPLAY*/

GLBL: BEGIN;
P: PROCEDURE(X,Y);
   DECLARE X ENTRY, Y FIXED;
   DECLARE I FIXED;
BEGIN;
   Q: PROCEDURE(Z);
      DECLARE Z ENTRY;
      DECLARE F(I:10) FIXED;
      F(1)=13;
      CALL Z(F(1)+Y));
   END Q;
   CALL Q(X);
END;
END P;

R: PROCEDURE;
   DECLARE (I,G(I:10)) FIXED;
BEGIN;
   U: PROCEDURE(W); DECLARE W FIXED;
      G(I)=W;
   END U;
   DO I=1 TO 10;
      G(I)=23;
      CALL P(U,I);
   END;
END;

PUT EDIT ("GLOBAL DISPLAY TEST") (SKIP(3),A);
PUT EDIT((G(I) DO I=1 TO 10 ) ) (1(7));
END R;
CALL R;
PUT EDIT ("EXIT GLOBAL TEST") (SKIP,A);
END GLBL;

/*
DEMONSTRATION OF LABEL RESOLUTION IN A BLOCK STRUCTURE
*/

LABEL: BEGIN; DECLARE (Y,Z(3)) LABEL, (I,J,K) FIXED;
LBL: PROCEDURE(LABEL); DECLARE LABEL(*) LABEL;

GO TO LABEL(3);
END LBL;
PUT EDIT ("ENTER LABEL") (SKIP(3)*A);
Y=LBL2;
BEGIN;
  I=1;
  GO TO LBL1;
  K=I/2;
LBL1: PUT EDIT ("LABEL TEST", I) (SKIP* A, 5);  
  J=I+1;
  GO TO Y;
LBL2: PUT EDIT("INCORRECT LABEL TEST") (A);
END;
LBL1: PUT EDIT("INCORRECT LABEL TEST") (A);
LBL2: PUT EDIT("LABEL TEST", J) (SKIP* A, 5);
  Z(1)=BADLAB; *
  Z(2)=BADLAB;
  Z(3)=GOODLAB;
CALL LBL(Z);
BADLAB: PUT EDIT("INCORRECT LABEL RETURN") (SKIP* A);
GOODLAB: PUT EDIT("CORRECT LABEL RETURN") (SKIP* A);
PUT EDIT("EXIT LABEL") (SKIP* A);
END LBL;
END TEST;
****COMPiled PROGRAM SIZE = 6,325; METAx INSTRUCTION COUNT = 55,991****
****SYMTAB SEARCH COUNT = 1,484; SYMTAB COMPARE COUNT = 78,626****
****SYMBOL TABLE ENTRY COUNT = 370****
DYNAM $001
JUMP $002
STK TP $STK0 , $STK1
JUMP $003
FMT $004 , "I"
LDA "BEGIN STRINGS"
EDIT
PUT
JUMP $005

***FORMAT CODE***
2000380000
LDA T , "OL"
LDA "THIS IS A STRING."
STK
FMT $006 , "I"
LDA S , "OL"
EDIT
PUT
JUMP $007

***FORMAT CODE***
80000
LDA T , "OL"
LD "0001"
LD "0004"
SUBSTR
LDA "THIS"
STK
LDA T , "OL"
LD "0005"
LD "0006"
SUBSTR
LDA S , "OL"
LD "0005"
LD "0006"
SUBSTR
LDA "CONCATENATED SUBSTRING."
00210: STO
00211: FMT $008 ,"1"
00227: LDA T ,"OL"
00237: EDIT
00239: PUT
00245: JUMP $009

**FORMAT CODE**

00247  $008 "8000J"
00247  $009 LDA S ,"OL"
00255  LD "0001"
00261  LD "0005"
00267  SBSTR
00268  LDA "THIS"
00274  COMPC "1"
00277  JUMPF $010
00281  FMT $011 ,"1"
00287  LDA "STRING COMPARE 1 WORKS"
00311  EDIT
00312  PUT
00318  JUMP $010

**FORMAT CODE**

00318  $011 "80000"
00325  $010 LDA S ,"OL"
00333  LD "0001"
00339  LD "0004"
00345  SBSTR
00346  LDA T ,"OL"
00354  LD "0001"
00360  LD "0004"
00366  SBSTR
00367  COMPC "1"
00369  JUMPF $013
00374  FMT $014 ,"1"
00380  LDA "STRING COMPARE 2 WORKS"
00404  EDIT
00405  PUT
00406  JUMP $013
***FORMAT CODE***
00445 $016 "2000180000"
00447 $017 STKTP $STK10, $STK1
00468 ALLOC $STK1

***DOPE VECTOR CODE***
"10*2000120005"

00492 JUMP $018
00497 $018 FMT $019 "1"
00503 LDA "BEGIN I/O BLOCK"
00520 EDIT
00521 PUT

00522 JUMP $020

***FORMAT CODE***
00527 $019 "2000360000"
00539 $020 FMT $021 "Z"
00545 LDA S "0D"
00553 GET
00554 JUMP $022

***FORMAT CODE***
00559 $021 "80006"
00565 $022 FMT $023 "2"
00572 LDA A
00575 GET
00579 LDA B
00585 GET
00586 LDA C
00592 GET
00593 JUMP $024

***FORMAT CODE***
00598 $023 "30001000500050005"
00620  $024  LDA  T ,"0D"
00626  FMT $025 ,"0"
00632  LD  A
00638  EDIT
00644  LD  B
00650  EDIT
00656  LD  C
00664  EDIT
00670  PUT
00676  JUMP $026

***FORMAT CODE***
00678  LDA S ,"0D"
00684  EDIT
00690  LDA T ,"0D"
00696  EDIT
00702  LD  A
00708  EDIT
00714  EDIT
00720  LD  B
00726  EDIT
00732  LD  C
00738  EDIT
00744  PUT
00750  JUMP $028

***FORMAT CODE***
00752  LDA 5 ,"0D"
00758  FMT $029 ,"Z"
00764  LDA S ,"0D"
00770  GET
00776  LDA T ,"0D"
00782  GET
00788  JUMP $030

***FORMAT CODE***
00790  LDA 5 ,"0D"
00796  FMT $031 ,"Y"
0082:  LDA X
0082:  GET
0082:  LDA Y
0083:  GET
0083:  JUMP $032

***FORMAT CODE***
0084:  $031 "9000"
0084:  $032 LDA T ,"0D"
0085:  FMT $033 ,"Y"
0086:  LDA Z
0087:  GET
0088:  JUMP $034

***FORMAT CODE***
0087:  $033 "9000"
0088:  $034 FMT $035 ,"1"
0089:  LD X
0090:  EDIT
0091:  LD Y
0092:  EDIT
0093:  LD Z
0094:  EDIT
0095:  LDA S ,"0D"
0096:  EDIT
0097:  LDA T ,"0D"
0098:  EDIT
0099:  PUT
0100:  JUMP $036

***FORMAT CODE***
0093:  $035 "200029000D9000D9000D9000D200018000D8000D"
0096:  $036 LDA A
0097:  LD "0001"
0098:  SST
0099:  $038 LD "0005"
0099:  $038 STCKC "4"
0100:  JUMPT $037
0101:  LDA M
0102:  LD A
01006  INDXA "1"
01008  LD A
01014  STO
01015  LDA A
01021  LD A
01027  LD "0001"
01033  ADD
01034  SST
01035  JUMP $038
01040  $037  FMT $039 "1"
01046  LDA A
01052  LD "0001"
01058  SST
01059  $041  LD "0005"
01065  STCKC "4"
01067  JUMPT $040
01072  LDA M
01073  LD A
01084  INDXR "1"
01085  EDIT
01087  LDA A
01093  LD A
01099  LD "0001"
01105  ADD
01106  SST
01107  JUMP $041
01112  $040  PUT
01113  JUMP $042

***FORMAT CODE***
01118  $039  "0005"
01125  $042  FMT $043 "1"
01131  LDA "EXIT I/O BLOCK"
01147  EDIT
01148  PUT
01149  JUMP $044

***FORMAT CODE***
01154  $043  "2000180000"
01166 $044 STKTP $STK0 , $STK1
01177' ALLOC $STK1 , M
***COPE VECTOR CODE***
       "204ZccccZ000Z200012000"
01211 JUMP $045
01216 $045 FMT $046 , "1"
01222' LDA "BEGIN DO GROUPS"
01239 EDIT
01240' PUT
01244 JUMP $047
***FORMAT CODE***
01246 $046 "2000380000"
01259 $047 LDA I
01264' LD "0000"
01270' SST
01271 $049 LD "0004"
01277' STCKC "4"
01279' JUMPT $048
01286' LD "0004"
01290' LD I
01295' SUB
01297' LD "0005"
01303 MULT
01304' LD $051
01310' ADD
01311 JUMPA
01312 $053 FMT $054 , "1"
01318 LDA "CASE 0"
01326 EDIT
01327' PUT
01328 JUMP $055
***FORMAT CODE***
01333 $054 "80000"
01340 $055 JUMP $052
01345 $056 FMT $057 , "1"
01351 LDA "CASE 1"
01359 EDIT
01360  PUT
01361  JUMP $058

***FORMAT CODE***
01366 $057  "80000"
01373 $058  JUMP $052
01378 $059  FMT $060  "1"
01384  LDA  "CASE 2"
01392  EDIT
01393  PUT
01394  JUMP $061

***FORMAT CODE***
01395 $060  "80000"
01406 $061  JUMP $052
01411 $062  FMT $063  "1"
01417  LDA  "CASE 3"
01425  EDIT
01426  PUT
01427  JUMP $064

***FORMAT CODE***
01432 $063  "80000"
01439 $064  JUMP $052
01444 $065  FMT $066  "1"
01450  LDA  "CASE 4"
01458  EDIT
01459  PUT
01460  JUMP $067

***FORMAT CODE***
01465 $066  "80000"
01472 $067  JUMP $052
01477 $068  LD $050
01483 $050  JUMP $053
01488  JUMP $056
01493  JUMP $059
01493  JUMP $062
01503  JUMP $065
01503 $052  LDA I
01514  LD I
01520: LD "0001"
01526: ADD
01527: SST
01528: JUMP $049
01535: $048 LDA I
01535: LD "0001"
01545: SST
01546: $070 LD "cccc"
01552: STCKC "2"
01554: JUMPT $069
01559: LDA M
01565: LD I
01571: LD "0005"
01577: INDXA "2"
01579: LD "0000"
01585: STO
01586: LDA A
01592: LD "0003"
01598: LD "0002"
01604: MULT
01605: LD "0005"
01611: SUB
01612: SST
01613: $072 LD "0001"
01619: STCKC "4"
01621: JUMPT $071
01626: LD A
01632: LD "0005"
01638: STCKC "2"
01640: JUMPF $071
01646: LDA M
01651: LD I
01657: LD A
01663: INDXA "2"
01665: LD A
01671: STO
01672: LDA A
01678 LD A
01684 LD "0001"
01690 ADD
01691 SST
01692 JUMP $072
01697 $071 FMT $073 "1"
01703 LDA "I ="
01705 EDIT
01710 LD I
01716 EDIT
01717 LDA "M(I,1) ="
01722 EDIT
01723 LDA M
01728 LD I
01733 LD "0001"
01738 INDXR "2"
01743 EDIT
01744 LDA "M(I,5) ="
01749 EDIT
01750 LDA M
01755 LD I
01760 LD "0005"
01765 INDXR "2"
01770 EDIT
01775 PUT
01780 JUMP $074
01793 $073 "8000000005"
01805 $074 LDA I
01810 LD I
01816 LDA "0001"
01821 SUB
01826 SST
01831 JUMP $070
01836 $069 FMT $075 "1"
01841 LDA "EXIT DO GROUPS"
01846 EDIT
01851 EDIT
01853  PUT
01854  JUMP $076

***FORMAT CODE***
01855  $075  "2000180000"
01856  $076  STKTP $STKO ,*STKT1
01857  JUMP $077
01858  $077  FMT $078 ,"1"
01859  LDA "ENTER ARITHMETIC BLOCK"
01917  EDIT
01918  PUT
01919  JUMP $079

***FORMAT CODE***
01924  $078  "2000380000"
01935  $079  LDA A
01942  LD "0001"
01943  STK
01949  $081  LD "0001"
01953  STCKC "4"
01957  JUMPT $080
01962  LDA X
01968  LD A
01974  LD "EA@YD?01"
01984  MULT
01985  STO
01992  LDA Y
01998  LD X
02004  DIV
02005  STO
02006  LDA Z
02012  LD A
02018  LD "0005"
02024  STCKC "2"
02026  JUMPF $082
02031  LD A
02037  JUMP $083
02042  $082  LD "00000000"
02052 $083  STO
02053  FMT $084 "1"
02059  LD  X
02065  EDIT
02066  LD  Y
02072  EDIT
02073  LD  Z
02079  EDIT
02086  PUT
02081  JUMP $085

***FORMAT CODE***
02086 $084 "9000D"
02093 $085  FMT $086 "1"
02096  LD  A
02105  EDIT
02106  PUT
02107  JUMP $087

***FORMAT CODE***
02112 $086 "0005"
02119 $087  LDA  A
02125  LD  A
02132  LD  "0001"
02137  ADD
02138  SST
02139  JUMP $081
02144 $080  FMT $088 "1"
02150  LDA  "EXIT ARITHMETIC BLOCK"
02173  EDIT
02174  PUT
02175  JUMP $089

***FORMAT CODE***
02180 $088 "2000180000"
02192 $089  STkTP $STkTO $STkT1
02203  JUMP $090
02208 NFACT ENTPR "2"
02210  DYNAM $091
02215  JUMP $092
02220  $092  LD  I
02226  LD  "0000"
02232  STCKC  "1"
02234  JUMPF  $093
02239  LD  "0001"
02245  SWAP
02246  RETRN
02247  $093  LDA  $094
02253  LD  $ACTIVE
02259  LD  $ACTIVE
02265  FLAG
02266  LDA  $095
02272  LDA  $095
02278  LD  I
02284  LD  "0001"
02290  SUB
02291  STO
02292  LD  $STK T2
02298  LDA  NFACT
02304  JUMPA
02305  $094  LD  I
02311  MULT
02312  SWAP
02313  RETRN
02314  RETRN
02315  $090  FMT  $096  "1"
02321  LDA  "ENTER RPROC"
02334  EDIT
02335  PUT
02336  JUMP  $097

***FORMAT CODE***
02341  $096  "2000380000"
02353  $097  FMT  $098  "1"
02359  LDA  "7 FACTORIAL ="
02374  EDIT
02375  LDA  $099
02381  LD  $ACTIVE
02387  LD  $ACTIVE
02393  FLAG
02394  LDA  $100
02400  LDA  $100
02406  LDA  $101
02412  LD  $ACTIVE
02418  LD  $ACTIVE
02424  FLAG
02425  LDA  $102
02431  LDA  $102
02437  LD  "0003"
02442  STO
02444  LD  $STKT1
02450  LDA  NFAC1
02456  JUMPA
02457  $101  LD  "0001"
02463  ADD
02464  STO
02465  LD  $STKT1
02471  LDA  NFAC1
02477  JUMPA
02478  $099  EDIT
02479  PUT
02480  JUMP  $103
  ***FORMAT CODE***
02485  $098  "2000180000"$0001
02502  $103  FMT  $104  ,"1"
02508  LDA  "EXIT RPROC"
02520  EDIT
02521  PUT
02522  JUMP  $105
  ***FORMAT CODE***
02527  $104  "2000180000"
02533  $105  STKTP  $STKTO  ,$STKT1
02550  JUMP  $106
02555  OUT  ENTPK  "2"
02557  DYNAM  $095
02562 JUMP $108
02567 LDA T "1+
02575 LD J
02581 LD "0001"
02587 SBSTH
02588 LDA OUTCHAR "01"
02594 STO
02597 LDA J
02603 LD J
02609 LD "0001"
02615 ADD
02616 STO
02617 RETRN
02618 NEXT ENTPR "2"
02620 DYNAM I
02625 JUMP $110
02630 LDA INPUT "1+
02636 LD I
02642 LD "0001"
02644 SBSTR
02650 LDA ""
02651 COMPC "1"
02656 JUMPF $112
02661 LDA I
02667 LD I
02673 LD "0001"
02676 ADD
02680 STO
02681 LD I
02687 LD "001+"
02693 STCKC "4"
02695 JUMPF $113
02700 LD "F"
02703 SWAP
02706 RETRN
02705 LDA INPUT "1+
02711 LD I
02719  LD    "0001"
02725  SBSTK
02726  LDA    "1"
02729  COMPC   "1"
02731  JUMPF   $114
02736  LD    "F"
02739  SWAP
02740  RETRN
02741  $114  JUMP   $110
02746  $112  LD    "T"
02749  SWAP
02750  RETRN
02751  RETRN
02752  NUMBFR  ENTR "2"
02754  DYNAM   I
02759  JUMP   $116
02764  $116  LDA   $117
02770  LDA   $ACTIVE
02776  LDA   $ACTIVE
02782  FLAG
02783  LD    $STKT2
02786  LDA  NEXT
02795  JUMPA
02796  $117  NOT
02797  JUMPF   $118
02802  LD    "F"
02805  SWAP
02806  RETRN
02807  $118  LDA  INPUT   "1+"
02815  LD  "1"
02821  LD  "0001"
02827  SBSTR
02828  LDA   "9"
02831  COMPC   "3"
02833  LDA  INPUT   "1+"
02841  LD  "1"
02847  LD  "0001"
02853  SB:STR
02854  LDA "0"
02855  CMPC "5"
02856  AND
02857  JUMPF $119
02860  LDA CHAR "01"
02864  LDA INPUT "1+
02873  LD I
02886  LD "0001"
02893  SB:STR
02894  STO
02895  LDA I
02901  LD I
02907  LD "0001"
02913  ADD
02914  STO
02915  LD "T"
02918  SWAP
02919  RETRN
02920  $119  LD "F"
02923  SWAP
02924  RETRN
02925  RETRN
02926  ID  ENTPR "2"
02928  DYNAM I
02933  JUMP $121
02933  $121  LDA $122
02944  LD $ACTIVE
02950  LD $ACTIVE
02956  FLAG
02957  LD $STKT2
02963  LDA NEXT
02969  JUMPA
02970  $122  NOT
02971  JUMPF $123
02976  LD "F"
02979  SWAP
02980 RETRN
02981 $123 LDA INPUT ,"1+
02985 LD I
02995 LD "0001"
03001 SBSTR
03002 LDA "A"
03005 COMPC "5"
03007 LDA INPUT ,"1+
03015 LD I
03021 LD "0001"
03027 SBSTR
03028 LDA "Z"
03031 COMPC "3"
03033 AND
03034 JUMPF $124
03039 LDA CHAR ,"01"
03047 LDA INPUT ,"1+
03055 LD I
03061 LD "0001"
03067 SBSTR
03068 STO
03069 LDA I
03075 LD I
03081 LD "0001"
03087 ADD
03088 STO
03089 LD "T"
03092 SWAP
03093 RETRN
03094 $124 LD "F"
03097 SWAP
03098 RETRN
03099 RETRN
03100 LITERAL DYNAM $095
03102 JUMP $126
03112 $126 LDA $127
03253 LDA $133
03259 LDA $133
03265 LDA "(
03266 STO
03269 LD $STK2
03275 LDA LITERAL
03281 JUMPA
03282 $132 JUMPF $134
03287 LDA $135
03292 LD $ACTIVE
03296 LD $ACTIVE
03305 FLAG
03306 LD $STK2
03312 LDA EXP1
03316 JUMPA
03316 $135 NOT
03320 JUMPF $136
03325 LD "F"
03326 SWAP
03329 RETRN
03330 $136 LDA $137
03336 LD $ACTIVE
03342 LD $ACTIVE
03348 FLAG
03349 LDA $138
03355 LDA $138
03361 LDA ")"
03364 STO
03365 LD $STK2
03377 LDA LITERAL
03378 JUMPA
03378 $137 NOT
03379 JUMPF $139
03384 LD "F"
03387 SWAP
03388 RETRN
03389 $139 LD "T"
03392  SWAP
03393  RETRN
03394  $134  LDA  $140
03400  LD  $ACTIVE
03406  LD  $ACTIVE
03412  FLAG  
03413  LD  $STKT2
03419  LDA NUMBER
03425  JUMPA
03426  $140  JUMPF $141
03431  LDA $142
03437  LD  $ACTIVE
03443  LD  $ACTIVE
03449  FLAG  
03450  LDA CHAR  "01"
03456  LD  $STKT2
03462  LDA OUT
03470  JUMPA
03471  $142  LD  "T"
03477  SWAP
03478  RETRN
03484  $141  LDA  $143
03488  LD  $ACTIVE
03494  LD  $ACTIVE
03494  FLAG  
03495  LD  $STKT2
03501  LDA ID
03507  JUMPA
03508  $143  JUMPF $144
03513  LDA $145
03519  LD  $ACTIVE
03525  LD  $ACTIVE
03531  FLAG  
03532  LDA CHAR  "01"
03540  LD  $STKT2
03546  LDA OUT
03552  JUMPA
0355  $145  LD  "T"
0356  SWAP
0357  RETRN
0358  $144  LD  "F"
0359  SWAP
0360  RETRN
0361  RETRN
0362  TERM  ENTRPR "2"
0363  DYNAM  $146
0364  JUMP  $147
0365  $147  LDA  $148
0366  LD  $ACTIVE
0367  LD  $ACTIVE
0368  LDA  $147
0369  $147  JUMPA
0370  $148  NOT
0371  JUMPF  $149
0372  LD  "F"
0373  SWAP
0374  RETRN
0375  $149  LDA  $150
0376  LD  $ACTIVE
0377  LD  $ACTIVE
0378  FLAG
0379  LDA  $133
0380  LDA  $133
0381  LDA  "*
0382  STO
0383  LDA  $STKT2
0384  LDA  LITERAL
0385  JUMPA
0386  $150  JUMPF  $152
0387  LDA  $152
0388  LDA  $153
0389  LD  $ACTIVE
0390  LD  $ACTIVE
03690  FLAG
03691  LD  $STKT2
03697  LDA  PRIMARY
03703  JUMPA
03704  $153  NOT
03705  JUMPF  $154
03710  LD  "F"
03713  SWAP
03714  RETRN
03715  $154  LDA  $155
03721  LD  $ACTIVE
03727  LD  $ACTIVE
03733  FLAG
03734  LDA  $138
03740  LDA  $138
03745  LDA  "*"
03749  STO
03750  LD  $STKT2
03755  LDA  OUT
03762  JUMPA
03763  $155  LDA  $149
03769  JUMPA
03770  $152  LDA  $157
03776  LD  $ACTIVE
03782  LD  $ACTIVE
03788  FLAG
03789  LDA  $158
03795  LDA  $153
03801  LDA  "/"
03804  STO
03805  LD  $STKT2
03811  LDA  LITERAL
03817  JUMPA
03818  $157  JUMPF  $159
03823  LDA  $160
03829  LD  $ACTIVE
03835  LD  $ACTIVE
03841   FLAG
03842   LD $STKT2
03844   LDA PRIMARY
03854   JUMPA
03855   $160 NOT
03856   JUMPF $161
03861   LD "F"
03864   SWAP
03865   RETRN
03866   $161 LDA $162
03872   LD $ACTIVE
03873   LD $ACTIVE
03884   FLAG
03885   LDA $163
03891   LDA $163
03897   LDA "/"
03900   STO
03901   LD $STKT2
03907   LDA OUT
03913   JUMPA
03914   $162 LDA $149
03920   JUMPA
03921   $159 LD "T"
03924   SWAP
03925   RETRN
03926   RETRN
03927   EXP2 ENTPR "2"
03929   DYNAM $163
03934   JUMP $165
03939   $165 LDA $166
03945   LD $ACTIVE
03951   LD $ACTIVE
03957   FLAG
03958   LDA $133
03964   LDA $133
03970   LDA "←"
03973   STO
03974  LD  $STKT2
03980  LDA  LITERAL
03986  JUMPA
03987  JUMPF $166
03992  LDA $169
03996  LD  $ACTIVE
04000  LD  $ACTIVE
04010  FLAG
04011  LD  $STKT2
04017  LDA  TERM
04023  JUMPA
04026  $169  NOT
04025  JUMPF $170
04030  LD  "F"
04038  SWAP
04039  RETRN
04035  $170  LDA $171
04041  LD  $ACTIVE
04047  LD  $ACTIVE
04053  FLAG
04054  LDA $138
04060  LDA $138
04066  LDA "M"
04069  STO
04070  LD  $STKT2
04075  LDA  OUT
04082  JUMPA
04083  $171  LD  "T"
04088  SWAP
04087  RETRN
04083  $168  LDA $173
04094  LD  $ACTIVE
04100  LD  $ACTIVE
04105  FLAG
04107  LDA $158
04113  LDA $158
04119  LDA "*"
04122     STO
04123     LD   $STKT2
04129     LDA   LITERAL
04135     JUMPA
04136     $173     JUMPF $175
04141     LDA   $176
04147     LD   $ACTIVE
04153     LD   $ACTIVE
04159     FLAG
04160     LD   $STKT2
04166     LDA   TERM
04172     JUMPA
04173     $176     NOT
04174     JUMPF $177
04175     LD   "F"
04182     SWAP
04188     RETRN
04184     $177     LD   "T"
04187     SWAP
04188     RETRN
04189     $175     LDA   $178
04195     LD   $ACTIVE
04201     LD   $ACTIVE
04207     FLAG
04208     LD   $STKT2
04214     LDA   TERM
04220     JUMPA
04221     $178     JUMPF $179
04226     LD   "T"
04229     SWAP
04230     RETRN
04231     $179     LD   "F"
04234     SWAP
04235     RETRN
04236     RETRN
04237     Exp1     ENTPR "2"
04239     DYNAM $146
JUMP $181
LDA $182
LD $ACTIVE
LD $ACTIVE
FLAG
LD $STKT2
LDA EXP2
JUMPA
$181
LDA $182
LD $ACTIVE
LD "T"
SwAP
RETRN
$182
LDA $183
LD $ACTIVE
LD $ACTIVE
FLAG
LDA $133
LDA $133
LDA "*
STO
LD $STKT2
LDA LITERAL
JUMPA
$184
JUMPF $186
LDA $187
LD $ACTIVE
LD $ACTIVE
FLAG
LD $STKT2
LDA TERM
JUMPA
$187
NOT
JUMPF $188
LD "F"
SwAP
RETRN
04386 $188  LDA  $189
04394    LD  $ACTIVE
04400    LD  $ACTIVE
04406    FLAG
04407    LDA  $138
04413    LDA  $138
04419    LDA  "+
04422    STO
04423    LD  $STKT2
04429    LDA  OUT
04435    JUMPA
04436 $189  LDA  $183
04442    JUMPA
04443 $186  LDA  $191
04449    LD  $ACTIVE
04455    LD  $ACTIVE
04461    FLAG
04462    LDA  $158
04468    LDA  $158
04474    LDA  "-
04477    STO
04478    LD  $STKT2
04484    LDA  LITERAL
04490    JUMPA
04491 $191  JUMPF  $193
04495    LDA  $194
04502    LD  $ACTIVE
04503    LD  $ACTIVE
04514    FLAG
04515    LD  $STKT2
04521    LDA  TERM
04527    JUMPA
04528 $194  NOT
04529    JUMPF  $195
04534    LD  "F"
04537    SWAP
04538    RETRN
04539 $195   LDA   $196
04545   LD   $ACTIVE
04551   LD   $ACTIVE
04557   FLAG
04558   LDA   $163
04564   LDA   $163
04570   LDA   "-"
04573   STG
04574   LD   $STKT2
04580   LDA   OUT
04586   JUMPA
04587 $196   LDA   $183
04593   JUMPA
04594 $193   LD   "T"
04597   SWAP
04598   RETRN
04599   RETRN
04600 $106   FMT   $198   "1"
04606   LDA   "ENTER POSTFIX"
04621   EDIT
04622   PUT
04623   JUMP   $199
**:FORMAT CODE**: 04628 $198 "2000380000"
04640 $199   FMT   $200   "2"
04645   LDA   INPUT   "1+"
04654   GET
04655   JUMP   $201
**:FORMAT CODE**: 04660 $200 "200018001+
04672 $201   LDA   I
04678   LDA   J
04684   LD   "0001"
04690   SST
04691   STO
04692   FMT   $202   "1"
04698   LDA   "INFX EXPRESSION ="
04718 EDIT
04719 LDA INPUT "1+"
04727 EDIT
04728 PUT
04729 JUMP $203

***FORMAT CODE***
04734 $202 "200018000000028001+"
04756 $203 LDA T "1+"
04764 LDA INPUT "1+"
04772 STO
04775 LDA $204
04785 LDA $ACTIVE
04791 FLAG
04792 LD $STKTI
04796 LDA EXP1
04804 JUMPA
04805 $204 JUMPF $205
04810 FMT $206 "1"
04816 LDA "POSTFIX EXPRESSION ="
04830 EDIT
04839 LDA T "1+"
04847 LD "0001"
04853 LD J
04859 LD "0001"
04865 SUB
04866 SBSTR
04867 LDA ";"
04870 CAT
04871 EDIT
04872 PUT
04873 JUMP $207

***FORMAT CODE***
04873 $206 "2000180000000280000"
04900 $207 JUMP $208
04905 $205 FMT $209 "1"
04911 LDA "****ERROR****"
04926 EDIT
04927 PUT
04928 JUMP $209

***FORMAT CODE***
04933 $209 "2000160000"
04945 5208 FMT $211 "1"
04951 LDA "EXIT POSTFIX"
04965 EDIT
04966 PUT
04967 JUMP $212

***FORMAT CODE***
04972 $211 "2000180000"
04984 $212 STKTP $STKT0 , $STKT1
04995 JUMP $213
05000 P ENTPR "2"
05002 DYNAM $163
05007 JUMP $215
05012 $215 STKTP $STKT2 , $STKT3
05023 JUMP $216
05029 Q ENTPR "3"
05030 DYNAM $217
05035 ALLOC $STKT4 , F

***DOPE VECTOR CODE***
"104Z0001Z0001"
05059 JUMP $219
05064 $218 LDA F
05070 LD "0001"
05075 INDXA "1"
05079 LD "000 "
05084 STO
05085 LDA $219
05091 LD $ACTIVE
05097 LDA $220
05103 LD "300+" ****
05109 LD "0004"
05115 ADD
05116 STO
05117  LD    "0000"  *****
05123  FLAG
05124  LDA  $221
05130  LDA  $221
05136  LDA  F
05142  LD   "0001"
05148  INDXR "1"
05150  LD   Y
05156  ADD
05157  STO
05158  LD   $STKT4
05164  LDA  Z
05170  JUMPA
05171  $219  RETRN
05172  $216  LDA  $222
05178  LD   $ACTIVE
05184  LD   $ACTIVE
05190  FLAG
05191  LDA  $223
05197  LDA  $223
05203  LD   I
05209  STO
05210  LDA  $224
05216  LDA  $225
05222  LD   G
05228  LD   "0004"
05234  ADD
05235  STO
05236  LD   "0000"  *****
05242  STO
05243  LD   $STKT3
05249  LDA  Q
05255  JUMPA
05256  $222  RETRN
05257  R    ENTPR "2"
05259  DYNAM $226
05264  ALLOC $STKT2  G
***DOPE VECTOR CODE***

"104Z0001Z0001"

05288     JUMP   $227
05293   $227 STkT$ STkT2 STkT3
05304     JUMP   $228
05306   U     ENTPR "3"
05311  DYNAM F
05316     JUMP   $230
05321   $230 LDA   G
05327     LD    I
05333  INDXA "1"
05335     LD    W
05341  STO
05342  RETRN
05343   $228 LDA    I
05349     LD    "0001"
05355     SST
05356   $232 LD    "0001"
05362  STCKC "4"
05364  JUMPT $231
05369  LDA   G
05375     LD    I
05381  INDXA "1"
05383     LD    "000G"
05389  STO
05390  LDA   $233
05396     LD    $ACTIVE
05402  LD $ACTIVE
05408  FLAG
05409  LDA   $224
05415  LDA   $224
05421  LDA    "0185" *****
05427  STO
05428  LDA   $235
05434  LD $ACTIVE
05440  STO
05441  LDA   $236
**05447** LDA $236
**05453** LD "0001"
**05459** STO
**05460** LD $STKT3
**05466** LDA P
**05472** JUMPA
**05473** $233 LDA I
**05479** LD I
**05485** LD "0001"
**05491** ADD
**05492** SST
**05493** JUMP $232
**05493** $231 FMT $237 "1"
**05504** LDA "GLOBAL DISPLAY TEST"
**05525** EDIT
**05526** PUT
**05527** JUMP $238

---

*****FORMAT CODE***

**05532** $237 "2000360000"
**05544** $238 FMT $239 "1"
**05550** LDA I
**05556** LD "0001"
**05562** SST
**05563** $241 LD "0001"
**05569** STCKC "4"
**05571** JUMPT $240
**05576** LDA G
**05582** LD I
**05588** IDX "1"
**05590** EDIT
**05591** LDA I
**05597** LD I
**05603** LD "0001"
**05609** ADD
**05610** SST
**05611** JUMP $241
**05616** $240 PUT
05617 JUMP $242

***FORMAT CODE***
05622 $239 "0007"
05629 $242 RETRN
05630 $213 LDA $243
05636 LD $ACTIVE
05642 LD $ACTIVE
05648 FLAG
05649 LD $STKT1
05655 LDA R
05661 JUMPA
05662 $243 FMT $244 "1"
05668 LDA "EXIT GLOBAL TEST"
05676 EDIT
05687 PUT
05688 JUMP $245

***FORMAT CODE***
05693 $244 "2000180000"
05705 $245 STKTP $STKT0 , $STKT1
05715 ALLOC $STKT1 , Z

***DOPE VECTOR CODE***
"104Z0001Z0003"
05740 JUMP $246
05745 LBL ENTPK "2"
05747 DYNAM $095
05752 JUMP $248
05758 $248 POPUP
05758 LDA LABEL
05764 LD "0003"
05770 INDXR "1"
05772 RETRN
05772 RETRN
05774 $246 FMT $249 "1"
05780 LDA "ENTER LABEL"
05793 EDIT
05794 PUT
05795 JUMP $250
05800 $249  "2000380000"
05812 $250   LDA Y
05816   LD  "01I"
05824   STD
05825   STKTP $STKT1 $STKT2
05836   JUMP $251
05841 $251   LDA I
05847   LD  "0001"
05858   STD
05854   LD  "01Z"
05860   JUMPA
05863   LDA B
05867   LD  I
05873   LD  "0002"
05879   DIV
05880   STD
05881 LBL1   FMT $252 ,"1"
05887   LDA "LABEL TEST"
05899   EDIT
05900   LD  I
05906   EDIT
05907   PUT
05908   JUMP $253
05913 $252  "2000180000:0005"
05930 $253   LDA A
05935   LD  I
05942   LD  "0001"
05948   ADD
05949   STO
05950   LD  Y
05956   JUMPA
05957 LBL2   FMT $254 ,"1"
05963   LDA "INCORRECT LABEL TEST"
05985   EDIT
05986   PUT
05987 JUMP $255
***FORMAT CODE***
05992 $254 "80000"
05994 $255 FMT $256 "1"
06005 LDA "INCORRECT LABEL TEST"
06024 EDIT
06026 PUT
06026 JUMP LBL2
***FORMAT CODE***
06035 $256 "80000"
06041 LBL2 FMT $258 "1"
06047 LDA "LABEL TEST"
06059 EDIT
06060 LD A
06066 EDIT
06067 PUT
06067 JUMP $259
***FORMAT CODE***
06073 $258 "20001800000005"
06090 $259 LDA Z
06096 LD "0001"
06102 INDXA "1"
06104 LD "01-\%"
06110 STO
06111 LDA Z
06117 LD "0002"
06123 INDXA "1"
06125 LD "01-\%"
06131 STO
06132 LDA Z
06133 LD "0003"
06144 INDXA "1"
06146 LD "01J-"
06152 STO
06153 LDA BADLAB
06159 LD $ACTIVE
06165 LD $ACTIVE
06171 FLAG
06172 LDA Z
06173 LD $STKTI
06174 LDA LBL
06190 JUMPA
06191 BADLAB FMT $261 "1"
06197 LDA "INCORRECT LABEL RETURN"
06211 EDIT
06222 PUT
06228 JUMP GOODLAB
***FORMAT CODE***
06228 $261 "2000180000"
06240 GOODLAB FMT $263 "1"
06246 LDA "CORRECT LABEL RETURN"
06268 EDIT
06269 PUT
06270 JUMP $264
***FORMAT CODE***
06275 $263 "2000180000"
06287 $264 FMT $265 "1"
06293 LDA "EXIT LABEL"
06305 EDIT
06306 PUT
06307 JUMP $266
***FORMAT CODE***
06312 $265 "2000180000"
06324 $266 STOP
06325 END PROGRAM
BEGIN STRINGS
THIS IS A STRING.
THIS IS A CONCATENATED SUBSTRING.
STRING COMPARE 1 WORKS
STRING COMPARE 2 WORKS
EXIT STRINGS

BEGIN I/O BLOCK

123 -12 20 123 -12 20 123 -12 20

5.12340 .4300000000E+01 -.1120000000E-007
5.1234 4.3E+14 -11.2E-9
1 2 3 4 5

EXIT I/O BLOCK

BEGIN DO GROUPS
CASE 4
CASE 3
CASE 2
CASE 1
CASE 0
I = 10, M(I,1) = 1, M(I,5) = 0
I = 9, M(I,1) = 1, M(I,5) = 0
I = 8, M(I,1) = 1, M(I,5) = 0
I = 7, M(I,1) = 1, M(I,5) = 0
I = 6, M(I,1) = 1, M(I,5) = 0

268
EXIT DO GROUPS

ENTER ARITHMETIC BLOCK

1.330
1
2.660
2
3.990
3
5.320
4
6.650
5
7.980
6
9.310
7
10.640
8
11.970
9
13.30
10

EXIT ARITHMETIC BLOCK
ENTER RPROC
7 FACTORIAL = 5040
EXIT RPROC

ENTER POSTFIX
INFIX EXPRESSION = (A + B) / (C*D) + (-3) / L/M;
POSTFIX EXPRESSION = AB+CD*/3*L/M/+;
EXIT POSTFIX

GLOBAL DISPLAY TEST
14  14  14  14  14  14  14  14  14  14
EXIT GLOBAL TEST

ENTER LABEL
LABEL TEST 1
LABEL TEST 2
CORRECT LABEL RETURN
EXIT LABEL
****INSTRUCTION COUNT = 8,209****
// CONTROL RECORD.
FUNCTION PLXCPL.
INTERPRETER=MTXINT04.
GO=NO.
STORE=YES.

//
ESYLST: PROCEDURE MAIN;

PROCEDURE MAIN;

DECLARE INPUT CHAR(80)»(CRDCNT, ITSYMCNT) FIXED, SYMBOL(500) CHAR(6),
(CRDREF(500), DECNT) CHAR(5);

I.CRDCNT=0;
SYMCNT=1;

GETCRD: GET EDIT(INPUT) (A(80));
IF SUBSTR(INPUT,1,4)="****" THEN GO TO SORT;
CRDCNT=CRDCNT+10;
PUT STRING(DECNT) EDIT (CRDCNT) (I(5));
/* INSERT HIGH ORDER ZEROES */
DO I=1 TO 3;
IF SUBSTR(DECNT,1,1)="" THEN SUBSTR(DECNT,1,1)="0";
END;
PUT EDIT (DECNT, SUBSTR(INPUT,6,75)) (A(5), A(75));
/* TEST FOR COMMENT CARD */
IF SUBSTR(INPUT,6,1)="*" THEN GO TO GETCRD;
/* TEST FOR SYMBOL DEFINITION. ENTER SYMBOL AND CARD
REFERENCE IF FOUND.

* /

IF SUBSTR(INPUT,8,7)=" " THEN GO TO GETCRD;
IF SUBSTR(INPUT,8,1)=" " THEN
SYMBOL(SYMCNT)=SUBSTR(INPUT,9,6);
ELSE SYMBOL(SYMCNT)=SUBSTR(INPUT,8,6);
CRDREF(SYMCNT)=DECNT;
SYMCNT=SYMCNT+1;
GO TO GETCRD;

SORT: BEGIN; DECLARE (I,J,K,L,M) FIXED, SYMTMP CHAR(6), REFTMP CHAR(5);
SYMCNT=SYMCNT-1;
M=SYMCNT;
LBL20: M=M/2;
    IF M=0 THEN GO TO LBL40;
    K=SYMCNT-M;
    J=1;
LBL41: I=J;
LBL49: L=I+M;
    IF SYMBOL(I) < SYMBOL(L) THEN GO TO LBL60;
    SYMTMP=SYMBOL(I);
    REFTMP=CRDREF(I);
    SYMBOL(I)=SYMBOL(L);
    CRDREF(I)=CRDREF(L);
    SYMBOL(L)=SYMTMP;
    CRDREF(L)=REFTMP;
    I=I-M;
    IF I > 0 THEN GO TO LBL49;
LBL60: J=J+1;
    IF J > L THEN GO TO LBL20; ELSE GO TO LBL41;
END SORT;
LBL40: PUT EDIT ("SYMBOL DEFINITION - CARD REFERENCE INDEX"," ")
(SKIP(3),COL(20),A,SKIP(2),A);
PUT EDIT ((SYMBOL(I),CRDREF(I)," "); " DO I=1 TO SYMCNT))
(A(6)*X(2),A(5)*A,A(6),X(2),A(5)*A,A(6)*X(2),A(5)*A,A,
A(6)*X(2),A(5)*A,A(6),X(2),A(5)*A,SKIP);
STOP;
END ESYLST;
****COMPILED PROGRAM SIZE = 1,397; METAX INSTRUCTION COUNT = 14,274****
****SYMTAB SEARCH COUNT = 393; SYMTAB COMPARE COUNT = 4,819****
****SYMBOL TABLE ENTRY COUNT = 54****
CONTROL RECORD

FUNCTION PlXACPL
INTERPRETER=MIXINT04

//
ERROR: PROCEDURE MAIN;
/******************************************************************************/
/* IDENTIFICATION: */
/* PROGRAM-ID: ERROR. */
/* AUTHOR: J. R. VAN DOREN. */
/* SOURCE LANGUAGE: PLEX. */
/* */
/* PURPOSE: */
/* ERROR DEMONSTRATES THE ERROR DIAGNOSTICS OF PLXCPL. */
/* */
BEGIN; DECLARE (I,J) FIXED, N(10,15) FLOAT;
XYZ: PROCEDURE (A,B,C); DECLARE A ENTRY, B FIXED;

*****ERROR***** W: INCORRECT ARG DCL COUNT
A=0;

*****ERROR***** F: SYNTAX
END AbC;

*****ERROR***** W: POSS PROC CLOSING ERR
XYZ: PROCEDURE(A); DECLARE A FLOAT;

*****ERROR***** W: DUP PROC DCL
A=I;
END XYZ;
LABEL: CALL XYZ(N(1, J*3/I+14), I);

*****ERROR***** W: INCORRECT PARM CNT
J=I****-1;

*****ERROR***** F: SYNTAX
END;
END ERROR;
FATAL ERROR(S) ENCOUNTERED, JOB ABORTED
IDENTIFICATION:

PROGRAM-ID: PLXINT

AUTHOR: J. R. VAN DOREN

SOURCE LANGUAGE: EASYCODER

SOURCE COMPUTER: H-1200

OBJECT COMPUTER: H-1200

PURPOSE:

PLXINT INTERPRETIVELY EXECUTES OBJECT PROGRAMS PRODUCED BY THE PLEX COMPILER FOR THE PLEX LANGUAGE.

ASSEMBLE IN FOUR CHAR ADDRESSING MODE

EXECUTION LOCATION

OCTAL ADDRESS DEFINITIONS OF PERTINENT SYMBOLS IN THE RESIDENT INPUT/OUTPUT ROUTINE.
COMMUNICATION AREA FIELD LOCATION DEFINITIONS

INSTCT EQU 209 INSTRUCTION COUNTER
STKSTR EQU 223 STACK START
STKEND EQU 227 STACK END
LODLOC EQU 219 STARTING ADDRESS FOR EXECUTION
DYNSTR EQU 231 STARTING ADDRESS FOR DYNAMIC STORAGE
DYNEND EQU 235 ENDING ADDRESS FOR DYNAMIC STORAGE

INDEX REGISTER LOCATION DEFINITIONS AND USAGE DESCRIPTIONS

IR1 EQU 4 CONVERSION SUBROUTINE USAGE AND WORK REGISTER
IR2 EQU 6 PUSH DOWN STACK POINTER
IR3 EQU 12 PROGRAM COUNTER
IR4 EQU 16 RESULT REGISTER FOR ADDRESS COMPUTATION
IR5 EQU 20 RESULT REGISTER FOR ADDRESS COMPUTATION
IR6 EQU 24 INPUT BUFFER POINTER
IR7 EQU 28 OPERATION CODE REGISTER
IR8 EQU 32 OUTPUT BUFFER POINTER
IR9 EQU 36 WORK REGISTER
IR10 EQU 40 WORK REGISTER
IR11 EQU 44 WORK REGISTER
IR12 EQU 48 WORK REGISTER
IR13 EQU 52 WORK REGISTER
IR14 EQU 56 ACTIVE DISPLAY POINTER FOR DYNAMIC STORAGE
IR15 EQU 60 CONVERSION SUBROUTINE USAGE AND WORK REGISTER

MACRO CALLS TO SOURCE LIBRARY TO ESTABLISH CONVERSION SUBROUTINES

L F8/HD 4,BD, FLOATING BINARY TO FLOATING DECIMAL
L FD/FB 4, DB, FLOATING DECIMAL TO FLOATING BINARY
IFIX  EQU  INT  RESOLVE CONVERSION NAME MISMATCH

***************************************************************************

* *
PROGRAM INITIALIZATION:
* SET REGISTERS.
* INITIALIZE DYNAMIC STORAGE DISPLAY.
* *
***************************************************************************

START  EQU  *

CAM  60
Sw  STKEND-2
MCw  STKSTR,IR2
Sw  IR2-2
Si  IR2
MCw  LODLOC,IR3
Si  IR3
LCA  =4B0,IR5
LCA  =4B0,IR4
Si  IR4,IR5
BS  IR7
Sw  IR7-1
LCA  =4B0,IR14
MCw  DYNSTR,IR14
Si  IR14
Sw  DYEND-2
MLwdr  IR14,3,X14
MLwdr  IR14,7,X14
MCw  :;OUTPUT+132
MCw  OUTPUT+132
MCw  =ICZ1,OUTPUT
Si  INPUT+80
:SKIP  PRINT+57,
LCA  *PRINTBF,IR8
B  GETIPT

SET FOUR CHAR ADDRESSING FOR EXECUTION
WORD MARK FOR MOVING AND TESTING
STACK POINTER
ITEM MARK FOR RIGHT MOVE
INITIALIZE INSTRUCTION COUNTER
ACCOMMODATE RIGHT MOVE
PROPERLY
PUNCTUATE
INDEX REGISTERS IR4,IR5
CLEAR OP CODE REGISTER
SHORTEN FETCH ARITHMETIC
INITIALIZE ACTIVE STORAGE POINTER
ACCOMMODATE RIGHT MOVE
WORD MARK FOR COMPARISON
INIT DYNAMIC STOR STACK TOP POINTER
LEVEL ONE STORAGE POINTER
CLEAR PRINT BUFFER
CARRIAGE CONTROL
RESTORE LOST ITEM MARK ON INPUT BUFFER
SKIP TO TOP OF PAGE
OUTPUT BUFFER POINTER
INITIALIZE INPUT BUFFER
EXECUTION ADDRESS AND ADDRESS PARAMETER
ELSE IF NO ADDRESS COMPUTATION THEN EXECUTE
INSERT EXECUTION ADDRESS
SHIFT LEFT BY ENTRY SIZE
BUMP SEQUENCE COUNTER
INSERT OP CODE
CLEAR ZD AND CHAR

* FETCH BA = IBI, INSTRCT
EXECUTION COUNTER
**************************************************************************
ADDRESS COMPUTATIONS ARE PERFORMED BY THE ADCOMP SUBROUTINE. ONE OR TWO ADDRESSES ARE COMPUTED DEPENDING ON ADDRESSING PARAMETER IN TVEC WHICH IS ADDRESSED BY THE OP CODE REGISTER IR7. RETURN ADDRESS IN IR13.

ADCOMP SST 0+X3,STYPE,70 SAVE STORAGE TYPE
SST 0+X3,DTYPE,07 SAVE DATA TYPE
01860 LCOMP BCE NOCOMP 1+X3,00 CHECK STATIC LEVEL
01880 BS IR9 CLEAR
01890 MRSD 1+X3,IR9 INSERT STORAGE LEVEL
01900 BA IR9 BITS FOR DISPLAY ADDRESSING
01910 BA IR9 SHIFT LEFT TWO
01920 BA IR14,IR9 DISPLAY ADDRESS OF STORAGE LEVEL ADDR
01930 MRID 0+X9,IR4-3 LEVEL BASE ADDRESS
01940 BA 4+X3,IR4 ADD DISPLACEMENT
01950 BS IP4-3 CLEAR HIGH BITS
01960 BA =1B5,IR3 SEQUENCE COUNTER
01970 BCE INDIRA,SType,60 CHECK FOR INDIRECT ADDRESS
01980 BCE PRCADD+SType,20 CHECK FOR REMOTE PROCEDURE ADDR
01990 BCE 0+X13,TVEC+X7,01 IF ONE ADDRESS THEN RETURN
02000 BS 1R9 ELSE COMPUTE 2ND (DYNAMIC ONLY)
02010 MRSD 1+X3,IR9 CLEAR HIGH BITS
02020 BA 1R9
02030 BA 1R9
02040 BA IR14,IR9
02050 BA 1R14,IR9
02060 BA 1R14,IR9
02070 BA 1R14,IR9
02080 BA =1B5,IR3 RETURN
02090 B 0+X13 LITERAL
02100 LITADD MRID IR3-2,IR4-2 LITERAL
02110 BA =1B1,IR4 ADDRESS
02120 MRIN 1+X3,0 SEQUENCE COUNTER
02130 SAR IR3 RETURN
02140 B 0+X13 DO IT TWICE FOR REMOTE PROCEDURE
02150 PRCADD MRID 0+X4,IR4-3 GET ACTUAL ADDRESS
02160 INDIRA MRID 0+X4,IR4-3 RETURN
02170 B 0+X13
02180 NOCOMP MRID 2+X3,IR4-2 INSERT STATIC STORAGE ADDRESS
02190 SAR IR3 SEQUENCE COUNTER
02200 B 0+X13 RETURN
02210 STYPE DCw :0:
02220 DTYPE DCw :0:
**DYNAM - ALLOCATE FIXED DYNAMIC STORAGE (PROCEDURE ENTRY)**

```
DYNAM BA 3+X3,3+X14 INCREMENT STACKTOP
C 3+X14, DYNEPIND CHECK FOR DYNAMIC STORAGE
BL DYNFL OVERFLOW
BA =1B4, IR3 INSTRUCTION COUNTER
B FETCH
```

**STKTP - INITIALIZE NEW STACKTOP (BLOCK ENTRY)**

```
STKTP MRIDR 0+X4,0+X5
B FETCH
```

**ALLOC - SET DOPE VECTOR FOR INDICATED ARRAY**

```
ALLOC MRSD 0+X3, IR10 INSERT SUBSCRIPT COUNT
SAR IR3
MLWD IR4, IR11 SAVE LOCATION OF STACK TOP VALUE
MLWD IR5, IR12 SAVE DYNAMIC DOPE VECTOR ADDRESS
MRIDI 0+X3, LNGTH-1 MOVE IN ELEMENT LENGTH SIZE
SAR IR3
MCW LNGTH, SIZE INIT ARRAY SIZE COUNTER
BS CONPRT CLEAR
MCW =2C0120, IR7 SET OP CODE REGISTER TO ONE ADDRESS CODE
MORSUB MCW +ALCRT1, IR13 RETURN ADDRESS FROM ADCOMP
B ADCOMP CALL ADCOMP
```
02600 ALCRT1 MCW +ALCRT2,IR13    RETURN ADDRESS FROM SECOND CALL
02610 MCW IR4,IR5                SAVE FIRST ADDRESS
02620 B ADCOMP                   CALL ADCOMP
02630 ALCRT2 MRIDR 0+X4*0+X12    MOVE UPPER BOUND TO DYNAMIC D.V.
02640 SBR IR12                   NEXT MULTIPLIER LOCATION
02650 MCW IR5,IR4                RESTORE FIRST ADDRESS
02660 SW 0-4+X12                 PUNCTUATION TO STOP ARITHMETIC
02670 BS 3+X4+0-1+X12           SUBTRACT LOWER BOUND
02680 BA =1B1*0-1+X12           FINISH MULTIPLIER
02690 BIM 0-1+X12*,SIZE          UPDATE ARRAY SIZE COMPUTATION
02700 EIM 0-1+X12*,CONPRT        UPDATE CONSTANT PART COMPUTATION
02710 BA 3+X4*,CONPRT            ADD IN LOWER BOUND
02720 BS =1B1,IR10
02730 BCE DVIDNE,IR10,00
02740 B MORSUB
02750 DVDNE MRIDR LNGTH=3,0+X12 ELEMENT LENGTH FACTOR
02760 SBR IR12
02770 MRIDR 0+X11,0+X12          BASE ARRAY ADDRESS
02780 BIM LNGTH,CONPRT           LENGTH FACTOR
02790 BS CONPRT,3+X12           FINISH CONSTANT PART
02800 BA SIZE,3+X11             BUMP STACK TOP LOCATION BY ARRAY SIZE
02810 C 3+X11,DYNEND           CHECK FOR DYNAMIC STORAGE ALLOC OVERFLOW
02820 BL DYNENDFL
02830 B FETCH
02840 LNGTH DCW =4BO
02850 SIZE DCW =4BO
02860 CONPRT DCW =4BO
02870***************************************************************************
02880*
02890* LDA - LOAD ADDRESS TO THE STACK, INCLUDING THE*
02900* DATATYPE CODE. IF A CHARACTER STRING ALSO LOAD*
02910* THE MAX STRING LENGTH.*
02920*
02930***************************************************************************
02940 LDA MLWDR IR4,4+X2          ADDRESS TO STACK
02950 LCA DTYPE,0+X2             DATA TYPE
02960 BCE LDLNG,DTYPE,04        IF A CHARACTER STRING LOAD LENGTH
02970 UPSTCK BA =1B9,IR2  STACK POINTER
02980 B FETCH ELSE FETCH NEXT INSTRUCTION
02990 LDLNG BCE LITLNG,STYPE,70 IF A LITERAL COMPUTE LENGTH
03000 MRIDR 0\times3\times5\times2 ELSE EXTRACT FROM INSTRUCTION
03010 SAR IR3 BUMP SEQUENCE COUNTER
03020 B UPSTCK
03030 LITLNG BS IR5
03040 MRIN 0\times4\times0
03050 SAR IR5
03060 BS IR4
03070 MRIDR IR5\times1\times5\times2
03080 B UPSTCK
03090**************************************************************************
03100* LD - LOAD AN OPERAND TO THE STACK, INCLUDING DATA TYPE. *
03110***************************************************************************
03120
03130 LD MRIDR 0\times4\times1\times2 LOAD DATA
03140 BCE CHKTYP,DTYPE,07 IF UNDETERMINED TYPE CHECK IT
03150 LCA DTYPE,0\times2 SET TYPE
03160 B UPSTCK
03170 CHKTYP BI 3\times4\timesSETINT INTEGER
03180 BI 7\times4\timesSETFLT FLOAT
03190 B TYPERR
03200 SETINT LCA =1B1,0\times2
03210 B UPSTCK
03220 SETFLT LCA =1B2,0\times2
03230 B UPSTCK
03240**************************************************************************
03250**************************************************************************
03260* STO - STORES THE ITEM AT THE TOP OF THE STACK AT THE ADDRESS *
03270* NEXT TO THE TOP. NOTE SPECIAL HANDLING OF CHARACTER *
03280* STRINGS. POP TWO STACK ELEMENTS UPON COMPLETION. *
03290* SST - SAME AS STO EXCEPT ADDRESS IS DISCARDED BUT THE *
03300* STACK TOP IS RETAINED. *
03310* 285
0334C********************************************************************
0335C
0336C
0337C
0338C
0339C
0340C
0341C
0342C
0343C
0344C
0345C
0346C
0347C
0348C
0349C
0350C
0351C
0352C
0353C
0354C
0355C
0356C
0357C
0358C
0359C
0360C
0361C
0362C
0363C
0364C
0365C
0366C
0367C
0368C
0369C
0370C

STO EUU *
SST EUU *
BCE CNVRTN,0-18+x2,04 BYPASS IF STRING OR SUBSTRING
BCE CNVRTN,0-18+x2,34
C 0-9+x2,0-18+x2 IF TYPES NOT EQUAL
BCE CNVRTN,0-18+x2,34
BCE CNVRTN,0-9+x2,34 CHECK FOR SEND STRING
C 0-9+x2,0-18+x2 IF TYPES NOT EQUAL
BNE SCNV DO CONVERSION
BCE CNVRTN MCW 0-14+x2,IR4 RECEIVING ADDRESS
BCE CHSTR,0-9+x2,04 SPECIAL HANDLING IF CHARACTER STRING
BCE CSBSTR,0-9+x2,34 CHECK FOR SENDING SUBSTRING
MRIDR 0-8+x2,0+x4 STORE IT
SSTCHK BCE PRMOVE,0-9+x2,34
PNCRTN BCE SAVTOP,0-1+x3,23 CHECK FOR SAVE AND STORE (SST)
BS =1B18,IR2 POP STACK TWO ELEMENTS
MCW +ENDPRG,ENDADR RESTORE STRING WORKING STORAGE POINTER
B FETCH
SAVTOP 0-9+x2 PUNCTUATION FOR MOVE
MLRDR 0-1+x2,0-10+x2 DISCARD ADDRESS
CI 0-9+x2,0-18+x2 CLEAR ITEM MARKS
BS =1B9,IR2 AND SAVE STACK TOP ON TOP
B FETCH
SCnv BCE CNVRTN,0-18+x2,07 UNIVERSAL TYPE O.K.
BCE FIXFLT,0-9+x2,01 CHECK FOR CONVERT
TMA 0-1+x2,00 GO CONVERT
B IFIX
R B CNVERR CONVERSION OVERLOW INSTRUCTION
MCW IR15,0-5+x2 INTEGER RESULT
SI 0-5+x2 TO
LCA =1B1,0-9+x2 STACK
B CNVRTN
FIXFLT MLWDI =4C0000000027,0-1+x2 NORMALIZED LOAD
AMA 0-1+x2,70 CLEAR EXTRANEOUS PUNCTUATION
MLWDI =8B0,0-1+x2 FLOATING
TAM 0-1+x2,00 RESULT
SI 0-1+x2 TO STACK
LCA =1B2,0-9+x2 B CNVRTN
0371D
03710  CHKSTR BS  IR5
03720  MCw =1C10,IR5
03730  BCE CHTRN,0-18+X2+07
03740  M Rid 0-13+X2,IR5-1
03750  CHTRN MCw 0-5+X2,IR9
03760  Sw 0+X9
03770  BS  IR10
03780  M R1n 0+X9,0+X4
03790  SBR  IR10
03800  EA  IR4,IR5
03810  C  IR5,IR10
03820  BH  TMOVE
03830  BCE SUBMV,0-18+X2,34
03840  M RIdR 0+X9,0+X4
03850  SUBMV M R1d 0+X9,0+X4
03860  B  SSTCHK
03870  B  SSTCHK
03880  TMOVE BS  IR4,IR5
03890  BA  IR5,IR9
03900  BA  IR4,IR5
03910  MLWDR 0-1+X9,0-1+X5
03920  B  SSTCHK
03930  CSHSTR EQU *
03940  MCw 0-5+X2,IR15
03950  BS  IR9
03960  M R1u 0-4+X2,IR9-1
03970  BA  IR9,IR15
03980  E1 NOSET,0-1+X15
03990  SI 0-1+X15
04000  B  CHRSTR
04010  NOSET SST =1B0,0-9+X2,70
04020  B  CHRSTR
04030  PRMOVE CI 0-1+X15
04040  MCw 0-5+X2,IR9
04050  Cw 0+X9
04060  B  PNCRTN
04070***************************************************************************
04080* FLAG- MARK STACK FOR THE LIMIT OF PARAMETER ADDRESSES
04090* PRIOR TO A PROCEDURE CALL
04100*
04120*FLAG LCA =1C77,0+X2
041400 B UPSTCK
04150***************************************************************************
04160* ENTPRO- ESTABLISH DYNAMIC STORAGE DISPLAY, STORE PARAMETER
04180* ADDRESSES IN NEW DYNAMIC STORAGE AREA AND THEN PASS
04190* CONTROL TO THE CALLED PROCEDURE.
04210***************************************************************************
04230 ENTPRO MRID 0-8+X2,IR14-3 NEW DISPLAY AND STORAGE AREA
04240 BS =1B9,IR2
04250 BS IR13
04260 MRSN 0+X3,IR13 LEVEL NUMBER OF PROCEDURE
04270 SAR IR3 BUMP SEQUENCE COUNTER
04280 BIM =4B4,IR13
04290 BA =1B4,IR13
04300 BA IR14,IR13 LOCATION FOR PARAMETER ADDRESSES
04310 FLGCK BS =1B9,IR2 SEARCH DOWN
04320 BCE PRMSTO+0+X2,77 AND
04330 BA =1B1,PRMCNT COUNT
04340 B FLGCK PARGS
04350 PRMSTO MCw IR2+IR9
04360 PRMCHK BCE PRMDNE,PRMCNT,00 IF DONE GO ELSEWHERE
04370 MRIDR 10+X9,0+X13 MOVE ADDRESS TO DYNAMIC STORAGE
04380 SBR IR13
04390 BA =1B9,IR9
04400 BS =1B1,PRMCNT
04410 B PRMCHK
04420 PRMDNE MRID 0-8+X2,IR9-3 GLOBAL DISPLAY ADDRESS
04430 BA =1B4,IR9 JUMP OVER STACK TOP VALUE
04440 MRIDR 0-17+X2,0-22+X2 PACK CALLING PROC DISPLAY ADDRESS WITH
04450  BS   =1818,IR2   RETURN ADDRESS
04460  MRIUR IR14-3*0+X14 1ST DISPLAY ENTRY (STACK TOP VALUE)
04470  SBR IR12  NEXT DISPLAY LOCATION
04480  MRSI 0-1*3,IR13
04490  DISPY MRIUR 0+X9,0+X12  MOVE GLOBALLY ACTIVE STORAGE LEVEL
04500  SAR IR9  ADDRESSES TO CURRENT DISPLAY
04510  SBR IR12
04520  BS  =181,IR13
04530  BCE CURLEV+IR13,01
04540  B DISPY
04550  CURLEV MRIUR IR14-3*0+X12  CURRENT DISPLAY BASE ADDRESS
04560  B FETCH
04570  PRMCNT DCw =1B0
04580***************************************************************************
04590*    SWAP - SWAP THE TOP TWO ELEMENTS OF THE STACK  *
04600*    *  
04610*  ***************************************************************************
04620***************************************************************************
04630  SWAP SI 0-9+X2,0-18+X2
04640  MLRUR 0-1+X2,HOLD
04650  MLRUR 0-10+X2,0-1+X2
04660  MLRUR HOLD,0-10+X2
04670  CI 0-9+X2,0-18+X2
04680  B FETCH
04690  HOLD DCw =9
04700***************************************************************************
04710*    RETURN - RESTORE DISPLAY ADDRESS FOR CALLING PROCEDURE  *
04720*    *  
04730*    *  
04740*    ***************************************************************************
04750***************************************************************************
04760  RETRN MRID 0-8+X2,IR3-3  RETURN ADDRESS TO INST COUNTER
04770  MRID 0-4+X2,IR14-3  DISPLAY AT POINT OF CALL
04780  BS  =189,IR2
04790  B FETCH
04800***************************************************************************
04810*    
04820*
04820* JUMPA - JUMP TO THE ADDRESS IN THE STACK; POP THE STACK
04830* POPUP - POP THE STACK
04840*
04850* JUMPA MRID 0-B+X2+IR3-3 SET INSTRUCTION COUNTER
04860* JUMPA MRID 0-B+X2+IR3-3 SET INSTRUCTION COUNTER
04870* POPUP BS =189+IR2
04880* B FETCH
04890* JUMP - JUMP TO THE ADDRESS FOLLOWING THE OP CODE
04900*
04910* JUMPT - JUMP IF TOP OF STACK IS TRUE; POP THE STACK
04920* JUMPT - JUMP IF TOP OF STACK IS TRUE; POP THE STACK
04930* JUMPF - JUMP IF TOP OF STACK IS FALSE; POP THE STACK
04940*
04950* JUMP MRID 0+X3+IR3-3
04960* JUMP MRID 0+X3+IR3-3
04970* B FETCH
04980* JUMPT BCE JUMPP,0-B+X2,T
04990* BA =184+IR3
05000* B POPUP
05010* JUMPP MRID 0+X3+IR3-3
05020* B POPUP
05030* JUMPF BCE JUMPP,0-B+X2,F
05040* BA =184+IR3
05050* B POPUP
05060* STCKC - TESTS 2ND STACK ITEM AGAINST 1ST ACCORDING TO THE LITERAL
05070*
05080* STCKC - TESTS 2ND STACK ITEM AGAINST 1ST ACCORDING TO THE LITERAL
05090* CHARACTER FOLLOWING THE OP CODE.
05100* RESULTS IN A BOOLEAN VALUE ON TOP OF THE STACK.
05110*
05120* STCKC BCE CHK2ND,0-9+X2,02
05130* STCKC BCE CHK2ND,0-9+X2,02
05140* MLWDI =4C00000027,0-1+X2 CONVERT TO FLOATING INTEGER
05150* CHK2ND BCE SETCND,0-18+X2,02
05160* MLWDI =4C00000027,0-10+X2 CONVERT TO FLOATING INTEGER
05170* SETCND MRSD 0+X3,COND-1 SET CONDITION TO BE TESTED
05180* SAR IR3 BUMP INSTRUCTION COUNTER
05190  AMA  0-10+x2,70    LOAD
05200  SMA  0-1x2,00     SUBTRACT
05210  COND FBA SETT,00  BRANCH ON FLOATING REGISTER COND
05220  SETF MCW :F;0-17+x2  PUNCTUATION FOR POSSIBLE STORE
05230  SI  0-17+x2     SET STACK
05240  MCW =1C03,0-18+x2
05250  B  POPUP
05260  SETT MCW :T;0-17+x2  SET STACK
05270  SI  0-17+x2     SET STACK
05280  MCW =1C03,0-18+x2
05290  B  POPUP
05300  ***************************************************************
05310  *  COMPC-D-PARE THE CHARACTER STRING WHOSE ADDRESS IS 2ND IN THE  *
05320  *  STACK AGAINST THE STRING WHOSE ADDRESS IS 1ST. IF THE  *
05330  *  STRINGS ARE OF UNEQUAL LENGTH THE SHORTER IS MOVED AND  *
05340  *  PADDLED ON THE RIGHT WITH BLANKS.  *
05350  *  THE COMPARISON CONDITION IS CONTAINED IN THE CHARACTER  *
05360  *  FOLLOWING THE OP CODE.  *
05370  *  ***************************************************************
05390  ***************************************************************
05400  COMPC BS IR9  CLEAR
05410  BS IR10
05420  BS IR15
05430  MCW 0-14+x2,IR13  LEFT ADDRESSES OF STRINGS
05440  BCE SUBMV1,0-18+x2,34  IF SUBSTRING PUT IN WORKING STORE
05450  MVRT1 MCW 0-5+x2,IR12  TO INDEX REGISTERS
05460  BCE SUBMV2,0-9+x2,34  IF SUBSTRING PUT IN WORKING STORE
05470  MVRT2 MRIN 0+x13,0
05480  SAR IR9  DETERMINE
05490  MRIN 0+x12,0
05500  SAR IR10  STRING
05510  BS IR13,IR9
05520  BS IR12,IR10  LENGTHS
05530  SW IR9=2
05540  C IR10,IR9  TEST
05550  CW IR9=2
05560  BL   MOVNXT
05570  BH   MOVTOP
05580  TSTRNG BS IR15     SET PROPER
05590  MRSD 0+x3,IR15     CONDITION
05600  SAR IR3
05610  MRSD CNDTBL+x15,CNDTST CODE
05620  Sw 0+x13 WORK MARK FOR COMARE
05630  MRIN 0+x13,0+x12 POSITION
05640  SAR IR13 INDEX REGISTERS
05650  SBR IR12 TO RIGHT END
05660  MCw +ENDPRG+ENDADR RESTORE STRING WORKING STORAGE POINTER
05670  C 0-1+x12,0-1+x13 COMPARE
05680  CNDTSTBCT SETT,00 CONDITIONAL BRANCH
05690  B SETF
05700  MOVNXT EQU *
05710  MRIDR 0+x13,(ENDADR-3) MOVE TO TEMPORARY LOCATION
05720  SBR IR15
05730  CI 0-1+x15
05740  MCw ENADDR+IR13
05750  MCw ENADDR+IR11
05760  BA IR10,IR11 RIGHT END OF PADDED FIELD
05770  MRLNK MRSDR 6LNK,0+x15 PAD
05780  SBR IR15
05790  C IR11,IR15 WITH
05800  BEH SETPNC
05810  B MRBLNK BLANKS
05820  SETPNC SI 0-1+x11
05830  B TSTRNG
05840  MOVTOP MRIDR 0+x12,(ENDADR-3) RIGHT END OF PADDED FIELD
05850  SBR IR15
05860  CI 0-1+x15
05870  MCw ENADDR+IR12
05880  MCw ENADDR+IR11
05890  BA IR9,IR11
05900  B MRBLNK
05910  SUBMV1 MRID 0-13+x2,IR10-1 GET LENGTH
05920  Sw 0+x13 WORD MARK FOR MOVE
ADD - REPLACE THE TOP TWO NUMBERS ON THE STACK WITH THEIR SUM
SUB - REPLACE THE TOP TWO NUMBERS ON THE STACK WITH THEIR DIFFERENCE (2ND - 1ST)
MULT - REPLACE THE TOP TWO NUMBERS ON THE STACK WITH THEIR PRODUCT
DIV - REPLACE THE TOP TWO NUMBERS ON THE STACK WITH THEIR RATIO (2ND / 1ST)
NEG - UNARY MINUS OPERATION ON THE ELEMENT ON THE TOP OF THE STACK
NOTE THE TYPE CONVERSIONS WHICH MAY TAKE PLACE.
06300 BCE INTOAD, ARI, 01
06310 TNA 0-1*X2+00
06320 AMA 0-10+X2+00
06330 SETBC FBJ FERR+06
06340 BS =189+1R2
06350 MLWDR =8800+0-1+X2
06360 SI 0-1*X2
06370 TAM 0-1*X2+00
06380 LCA =1R2+0-9+X2
06390 B FETCH
06400 INTOAD Sw 0-17+X2
06410 BA 0-5+X2+0-14+X2
06420 BS =189+1R2
06430 B FETCH
06440 SUB B CNVCHK
06450 BCE INTOAD, ARI, 01
06460 TMA 0-10+X2+00
06470 SMA 0-1+X2+00
06480 B SETBC
06490 INTOAD Sw 0-17+X2
06500 BS 0-5+X2+0-14+X2
06510 BS =189+1R2
06520 B FETCH
06530 MUL B CNVCHK
06540 BCE INTOAD, ARI, 01
06550 TMA 0-1+X2+00
06560 MAM 0-10+X2+00
06570 B SETBC
06580 INTOAD BIM 0-5+X2+0-14+X2
06590 BS =189+1R2
06600 B FETCH
06610 DMRD MRSD 0-9+X2+ARI
06620 BA 0-18+X2+ARI
06630 BCE NUMER, 0-9+X2, 02
06640 MLWDL =4C00000027, 0-1+X2
06650 MLWDL =4C00000027, 0-10+X2
06660 MLWDL =4C00000027, 0-10+X2
DODIV AMA $0-1^2+X2,71$  
NORMALIZED LOAD TO FR1

AMA $0-10^2+X2,70$  
NORMALIZED LOAD TO FRO

DAA 10  
RESULT IN FRO

BCE INTDIV,ARI,02  
CHECK FOR INTEGER RESULT

B SETBCK

INTDIV B IFIX  
GO CONVERT

R B CNVERR  
CONVERSION OVERFLOW INSTRUCTION

MLWD IR15,0-14^2+X2  
PUT RESULT IN STACK

BS =1B9,IR2

B FETCH

NEG BCE INTNEG,0-9^2+X2,01  
IF INTEGER THEN BRANCH

SMA 0-1^2+X2,70  
ELSE SUBTRACT FROM NORMAL ZERO

TAM 0-1^2+X2,00  
AND STORE

B FETCH

INTNEG LCA =480,IR15  
CLEAR REGISTER

BS 0-5^2+X2,IR15  
SUBTRACT

MLWD IR15,0-5^2+X2  
PUT BACK

B FETCH

ARI DCW :0:

CNVCHK SbR CNVTR+4  
SET RETURN

MRSD 0-9^2+X2,ARI  
SET

SST 0-18^2+X2,ARI,02  
INDICATOR

BCE (CNVTR+1),ARI,01  
IF ALL INTEGER RETURN

BCE (CNVTR+1),ARI,02  
IF ALL FLTNG PT RETURN

BCE CNVTOP,0-9^2+X2,01  
DO

MLWDI =4C0000027^0-10+X2  
NECESSARY

CNVTOP MLWDI =4C0000027^0-1^2+X2

CNVTR B *  

***********************************************************************

OR -REPLACE TOP TWO ELEMENTS WITH LOGICAL SUM  

AND-REPLACE TOP TWO ELEMENTS WITH LOGICAL PRODUCT  

NOT-LOGICAL NEGATION OF TOP ELEMENT  

***********************************************************************

OR BS =1B9,IR2
**INDXA** - COMPUTE ADDRESS ACCORDING TO DOPE VECTOR (ADDRESS IN STACK) AND INDECES ON THE STACK. RESULTING ADDRESS ON TOP OF THE STACK.

**INDXR** - SAME AS INDXA EXCEPT THAT ADDRESSED ITEM IS LOADED TO THE STACK.

**INDXA** EQU *
**INDXR** EQU *

BS IR13 CLEAR
MRSD 0+X3,IR13 INSERT SUBSCRIPT COUNT
SUBCHK BCE INTSBS+0-9+X2,01 CHECK FOR
TMA 0-1+X2,00 SUBSCRIPT CONVERSION
B IFIX
B CNVERR
MCW IR15+0-5+X2 MOVE INTEGER RESULT TO THE STACK
INTSBS BS =1B9,IR2
BS =1B1,IR13
BCE CMPADD+IR13,00
B SUBCHK
MCW IR2+IR9 SAVE STACK POINTER
MCW 0-5+X2,IR10 ADDRESS OF DOPE VECTOR
MRSD 0+X3,IR13 SUBSCRIPT COUNT
07410    SAR    IR3
07420    MRI    1+X2,IR15-3
07430    MORIDX  BS =IR1,IR13
07440    BCE    SUBDNE,IR13,00
07450    BA =IR4,IR10
07460    BA =IR9,IR2
07470    BIM    3+X10,4+X2
07480    BA 4+X2,IR15
07490    B    MORIDX
07500    SUBDNE    BIM 7+X10,IR15
07510    BA 11+X10,IR15
07520    MCW    IR9,IR2
07530    C    IR15,DYNEND
07540    BL    IDXERR
07550    MLWD    IR15,0-5+X2
07560    BCE    FETCH,0-2+X3,67
07570    MRIDR    0+X15,0-8+X2
07580    B    FETCH

07590*******************************************************************************
07600*******************************************************************************
07610*   CAT -CONCATENATE TWO STRINGS WHOSE ADDRESSES ARE IN THE TOP
07620*   TWO POSITIONS OF THE STACK. THE RESULTING STRING IS PLACED
07630*   IN WORKING STRING STORAGE AND THE TWO ADDRESSES ARE REPLACED
07640*   BY THE ADDRESS IN WORKING STORAGE.
07650*

07660*******************************************************************************
07670   CAT    MCW 0-14+X2,IR13
07680   BCE    CTSUB1,0-18+X2,34
07690   MRIDR    0+X13,(ENDADR-3)
07700   SBR    IR12
07710   CI    0-1+X12
07720   CAT2    MCW 0-5+X2,IR13
07730   BCE    CTSUB2,0-9+X2,34
07740   MRIDR    0+X13,0+X12
07750   SBR    IR13
07760   CAT3    CW 0+X12
07770   MCW    ENDADR,0-14+X2
NEXT LOCATION IN WORKING STORAGE

POP STACK

MOVE

SUBSTRING

TO

WORKING

STORAGE

REMOVE SUB STRING MARK

CAT2

CATENATE

SUBSTRING

TO

WORKING

STORAGE

REMOVE SUB STRING MARK

CAT3

SUBSTR - PL/I SUBSTRING OPERATION

STARTING

POINT OF STRING

IF NO LENGTH GIVEN

COMPUTE IT

ELSE STORE LENGTH

MARK AS SUBSTRING ELEMENT

FIND
FMT - SETS THE ADDRESS OF THE FORMAT CODE AND INDICATORS FOR STRING OR UNIT RECORD AND WHETHER INPUT OR OUTPUT.

IF STRING I/O THE ADDRESS AND LENGTH ARE EXTRACTED FROM THE STACK AND THE APPROPRIATE BUFFER POINTER IS SET TO THE BEGINNING OF THE STRING.

FMT MRID 0+X3, FMTCD1-3 SET FORMAT CODE ADDRESS
SAR IR3 BUMP SEQUENCE COUNTER
BA =1B1, FMTCD1 JUMP OVER BEGINNING MARKER
MCW FMTCD1+FMTCD2 SAVE FOR REPETITION
SST 0+X3, INOROT, 70 INPUT OR OUTPUT CODE
SST 0+X3, DEVTYP, 07 STRING OR UNIT RECORD
BA =1B1, IR3 BUMP SEQUENCE COUNTER
BCE STRSET, DEVTYP, 00 IF STRING INITIALIZE
BCE FETCH, INOROT, 70 ELSE INITIALIZE FOR UNIT RECORD
MCW +PRNTBF, IR8 OUTPUT PRINT BUFFER
MCW +PRNTBF+132, STREND
B FETCH
FMTCD1 DCW =4
FMTCD2 DCW =4
INOROT DCW :0:
DEVTP DCW :0:
STADD DCW =4
LNGSTR DCW =4B0
STREND DCW =4
STRSET MRID 0-8+X2, STRADD-3 STRING ADDRESS
MRID 0-4+X2, LNGSTR-1 LENGTH
MCW STRADD, STREND
BA LNGSTR, STREND STRING END ADDR (PLUS ONE)
08520  BS  1B9,IR2       POP STACK
08530  BCE  FETCH,INOROT,70  IF INPUT STRING ALL DONE
08540  MCW  STRADD,IR8       ELSE SET OUTPUT POINTER
08550  MCW  STRADD,IR9
08560  CLRMOR MRSDR BLNK,0+X9  CLEAR OUTPUT
08570  SBR  IR9       STRING
08580  C  IR9,STREND  TO
08590  BH  CLRMOR  BLANKS
08600  SI  0-I+X9
08610  B  FETCH
08620***************************************************************************
08630*  PUT- IF OUTPUT IS TO THE PRINTER PRINT AND RESET
08640*                               OUTPUT BUFFER POINTER
08650*  OUTPUT BUFFER POINTER
08660*  IF OUTPUT IS TO A STRING SET RIGHT END PUNCTUATION
08670*  AND RESET BUFFER POINTER TO PRINTER BUFFER.
08680*  *
08690***************************************************************************
08700  PUT  BCE  STRPUT,DEVTY,P,00
08710L :PUT  PRINT,
08720  MCW  : :OUTPUT+132    CLEAR PRINT
08730  MCW  OUTPUT+132       BUFFER
08740  MCW  1C21,OUTPUT      CARRIAGE CONTROL
08750  SI  INPUT+80          RESTORE ITEM MARK ON INPUT BUFFER
08760  PBUFF  MCW  +PRINTBF,IR8  PRINT BUFFER POINTER
08770  B  FETCH
08780  STRPUT  SI  0-I+X8     RIGHT END TERMINATOR
08790  B  PBUFF
08800***************************************************************************
08810*  EDIT- CONVERT INTERNAL DATA TO OUTPUT FORMAT ACCORDING
08820*  TO FORMAT CODE. ALSO EXECUTE CONTROL FORMAT INSTRUCTIONS.
08830*  ADDRESS OF ITEM IS AT THE TOP OF THE STACK.
08840*  *
08850*  *
08860***************************************************************************
08870  EDIT  BCE  FMTSTR, (FMTCD1-3),77  CHECK IF RESET IS REQUIRED
08880  BBE  DATFMT, (FMTCD1-3),70  CHECK FOR DATA FORMAT
ELSE IT MUST BE CONTROL FORMAT

SPACE THE REQUIRED NUMBER OF COLUMNS

SET THE BUFFER

THE PROPER CHARACTER

ONE ORIGIN INDEX FOR BUFFER POINTER

IF STRING THEN ERROR

IF STRING THEN ERROR

DECREMENT LINE COUNT

SET BUFFER POINTER

THE
09260L :PUT PRINT, DUMMY, REQUIRED
09270 BS =1B1, IR10 NO OF
09280 B SKPTST BLANK LINES
09290 DUMMY DCw =A :
09300 L DCw =1C45
09310 FMTRST MCw FMTCD2, FMTCD1 RESET FORMAT POINTER
09320 B EDIT
09330 DATFMT EQU *
09340 BCE AFORM, (FMTCD1-3) * 10 CHECK FOR A FORMAT CODE
09350 BCE LFORM, (FMTCD1-3) * 11
09360 BCE IFORM, (FMTCD1-3) * 12
09370 BCE LFORM, (FMTCD1-3) * 13
09380 B FMTERR
09390 AFORM BA =1B1, FMTCD1 JUMP OVER CODE TYPE
09400 MRID (FMTCD1-3) * IR10-3 A FIELD LENGTH
09410 SAR FMTCD1 NEXT FORMAT CODE LOCATION
09420 MRID 0-8*X2*IR9-3 ADDRESS OF STRING
09430 C IR10, :00: IF NO A FORMAT LENGTH GET FROM STRING
09440 BE GTALN
09450 AFRMA BA IR10, IR9 RIGHT END OF SENDING FIEKD
09460 BA IR8, IR10 BUFFER POINTER PLUS LENGTH
09470 CHKbff C IR10, STREND CHECK FOR
09480 BL BUFOFL OVERFLOW PROBLEMS
09490 SW 0*X8 MOVE STOPPER IN RECEIVING FIELD
09500 MCw 0-1*X9, 0-1*X10 MOVE DATA TO RECEIVING FIELD
09510 CI 0-1*X10 CLEAR POSSIBLE ITEM MARK
09520 CW 0*X8 REMOVE MOVE STOPPER
09530 MCw IR10, IR8 NEXT BUFFER LOCATION
09540 BS =1B9, IR2 POP STACK
09550 B FETCH
09560 GTALN BCE GTSLBN, 0-9*X2, 34 IF SUBSTRING GET LENGTH FROM STACK
09570 MRIN 0*X9, 0*X8 ELSE COMPUTE END LOCATIONS
09580 SAR IR9 STRING RIGHT END ( PLUS ONE )
09590 SBR IR10 BUFFER RIGHT END ( PLUS ONE )
09600 B CHKbff
09610 GTSLBN MRLU 0-4*X2, IR10-1 LENGTH FROM STACK
09620 B AFRMA
**BUFFER OVERFLOW**:

- **BUMP FORMAT CODE POINTER**
  - EQU 1C45
- **I FIELD LENGTH**
  - FORMED = 1B1,FMTCD1
- **NEXT FORMAT CODE POINTER**
  - MRID = (FMTCD1-3),IR10-3
- **BUFFER POINTER PLUS LENGTH**
  - BA IR8,IR10
- **CHECK**
  - C IR10,STREND
- **OVERFLOW**
  - BL BUFOFL
- **CONVERT**
  - BS 0-8+X2
- **MARK FOR MOVE**
  - BBE IFMNEG,0-8+X2,40
- **INTEGER**
  - IFMCNV TMA CNVFLD,00
- **TO DECIMAL**
  - BTD CNVFLD,00
- **ZERO SUPPRESSION SYMBOL**
  - MCW :0:0-2+X10
- **LEFT END OF EDIT CONTROL**
  - SW 0+X8
- **MOVE AND EDIT DECIMAL INTEGER**
  - MCE CNVFLD,0-1+X10
- **NEXT BUFFER LOCATION**
  - MCW IR10,IR8
- **POP STACK**
  - BS =189,IR2
- **CHECK FOR NEGATIVE SIGN PLACEMENT**
  - BBE IFMSGN,1+X2,40
- **CONVERT TO POSITIVE**
  - Sw CNVFLD-5
- **GO CONVERT TO DECIMAL**
  - BS 0-5+X2,CNVFLD-2
- **FIND BLANK**
  - CW CNVFLD-5
- **NEXT POSITION TO THE LEFT**
  - BCE IFMSST,0-1+X10,15
- **NEXT FORMAT CODE POINTER**
  - BA =1B1,FMTCD1
- **NEXT FORMAT CODE POINTER**
  - MRID = (FMTCD1-3),IR10-3
  - SAR FMTCD1
  - BA IR10,IR8
10000  C  IR8,STREND
10010  BL  BUFOFL
10020  BCE  FOUT,0-8+x2,F
10030  MCw :T:0-1+x8  MOVE TRUE
10040  B  FETCH
10050  FOUT MCw :F:0-1+x8  MOVE FALSE
10060  B  FETCH
10070  EFORM  BA  =1B1,FMTCD1
10080  MRID (FMTCD1-3) ,IR10-3  OUTPUT LENGTH
10090  SAR  FMTCD1  NEXT FORMAT CODE
10100  C  IR10,STREND
10110  BL  BUFOFL
10120  TMA  0-1+x2,00  OPERAND TO FRO
10130  BS  =1B9,IR2  POP STACK
10140  B  F6/FD  CONVERT TO FLOATING DECIMAL
10150  R  DSA  DECRES  ADDRESS OF FLOATING DECIMAL FIELD
10160  MLWD  IR8,IR5  PREPARE FOR BASIC E FORMAT
10170  BBE  ROUND,DECRES-3,20  NO SIGN IF PLUS
10180  SST :0::DECRES-3,60  CLEAR SIGN
10190  MCw ::0+x5  MINUS SIGN
10200  ROUND LCA :0::DECRES-14  PREPARE FOR ROUNING OVERFLOW
10210  Cw  DECRES-13
10220  A  :5::DECRES-3  ROUND RESULT
10230  BCE  MARK,DECRES-14,00  TEST NO OVERFLOW
10240  MCw  DECRES-4,DECRES-3  MOVE RIGHT ONE CHARACTER ON OVERFLOW
10250  A  :1::DECRES  ADJUST EXPONENT
10260  MARK Sw  DECRES-13  RESTORE FIELD MARKER
10270  SI  19+x5  RIGHT END OF OUTPUT FIELD
10280  BSN  EFRM,DECRES  BRANCH IF NEGATIVE EXPONENT
10290  SST :0::DECRES-60  STRIP SIGN
10300  C  DECRES::009:  TEST EXPONENT
10310  BL  EFRM
10320  MCw  DECRES,IR1
10330  BA  IR1,IR5
10340  SI  DECRES-4  RIGHT MOVE STOPPER
10350  BCE  MVPT,DECRES,00  ZERO EXPONENT TEST
10360  MCw  DECRES-14+x1,0+x5  MOVE DIGITS LEFT OF .
10370 MVPT MCW :..:1+X5 MOVE IN .
10380 MRID DECRES=-13+X1.2+X5 MOVE DIGITS TO THE RIGHT OF .
10390 SBR IR5
10400 NZTST BBE NXTFLD,0-1+X5.77 REPLACE
10410 MCW :..:0+X5 ORDER ZEROS
10420 SBR IR5 WITH ELNKS
10430 B NZTST
10440 EFRM MCW :..:1+X5 MOVE IN DECIMAL PT.
10450 MCW DECRES=-4,11+X5 MOVE TEN FRACTION DIGITS
10460 MCw :..:12+X5 MOVE EXPONENT SYMBOL
10470 MCw :..:13+X5 EXPONENT
10480 BBE LDIGIT,DECRES,40 SIGN
10490 MCw :..:13+X5 LOGIC
10500 LDIGIT SST DECRES,-16+X5,17 LOW DIGIT BUT NO SIGN
10510 MCw MOVE TWO HIGH DIGITS OF EXP
10520 B NXTFLD
10530 DCw :0:
10540 DCw =11
10550 DECRES DCw =3
10560 NXTFLD MRIN 0+X5.0
10570 SAR IR5
10580 CI 0-1+X5 REMOVE ITEM MARK
10590 BA IR10,IR3 NEXT OUTPUT BUFFER LOCATION
10600 B FETCH
10610*************************************************************************
10620* GET - PROCESS INPUT EITHER FROM CARD BUFFER OR STRING *
10630* ACCORDING TO INFORMATION SET BY FMT. *
10640* *************************************************************************
10650*************************************************************************
10660************************************************************************* GET BCE GETFM,DEVTYP,00 IF STRING INPUT BYPASS EOF TEST
10670 BCE ENDFTY,ENDF,T TEST FOR CLOSED FILE
10680 BCE IFMRST,(FMTCD1-3),77 CHECK FOR FORMAT RESET
10690 GETFM BCE IFMRST,(FMTCD1-3),77 CHECK FOR FORMAT RESET
10700 BBE IDTFTMT,(FMTCD1-3),70 CHECK FOR DATA FORMAT
10710 BCE ISPCF,(FMTCD1-3),00 ELSE IT MUST BE CONTROL FORMAT
10720 BCE ISKP,(FMTCD1-3),02
10730 BCE ICOL,(FMTCD1-3),03
10740  B  FMTERR
10750  IFMRST MCW FMTCD2,FMTCD1
10760  B  GET
10770  NXTIFM LA =1B1,FMTCD1
10780  B  GET
10790  ISPCE BA =1B4,FMTCD1
10800  MCW (FMTCD1-3),IR9
10810  SPCCNT C IR9,000;
10820  BE NXTIFM
10830  BCE STRSPC,DEV,00
10840  EA =1B1,IR6
10850  C +INPUT+79,IR6
10860  BH GETIPT
10870  BS =1B1,IR9
10880  B  SPCCNT
10890  STRSPC BA IR9,STRADD
10900  C  STREN,STRADD
10910  BEH BUFOFL
10920  B  NXTIFM
10930  ICOL BCE STRCOL,DEV,00
10940  MCW +INPUT,IR6
10950  BA =1B4,FMTCD1
10960  BA (FMTCD1-3),IR6
10970  BS =1B1,IR6
10980  C +INPUT+79,IR6
10990  BH BUFOFL
11000  B  NXTIFM
11010  STRCOL MCW STREN,STRADD
11020  BS LNGST,STRADD
11030  BA =1B5,FMTCD1
11040  BA (FMTCD1-3),STRADD
11050  BS =1B1,STRADD
11060  C  STREN,STRADD
11070  BEH BUFOFL
11080  B  NXTIFM
11090  ISKP BCE FMTERR,DEV,00
11100  BA =1B1,FMTCD1

IF NONE OF THE ABOVE THEN ERROR
RESET FORMAT POINTER
NEXT FORMAT CODE
POINT TO SPACE COUNT
MOVE IT TO INDEX REG.
TEST FINISH
TEST FOR STRING INPUT
TEST FOR STRING OVER FLOW
TEST FOR STRING
TEST FOR STRING
TEST
CHECK FOR ERROR
ESTABLISH
ORIGINAL ADDRESS
POINT TO COL INDICATOR
ONE ORIGIN INDEX FOR COLUMN POINTER
TEST
FOR ERROR
IF STRING INPUT THEN ERROR
11110    MRIL    (FMTCD1-3), IR10-3  SKIP COUNT TO INDEX REG
11120    SAR    FMTCD1  NEXT FORMAT CODE POINTER
11130    ISKPTS   BCE    GET, IR10, 00  TEST FINISH
11140    B    GETIPT  GET NEXT INPUT RECORD
11150    BS    =1B1, IR10  GO CHECK FOR SKIP COUNT
11160    B    ISKPTS  SET RETURN ADDRESS
11170    GETIPT   SBR    GETRTN+4  SET RETURN ADDRESS
11180    BCE    ENDFMS, ENDF, T  TEST FOR CLOSED FILE
11190    :GET    READ,  TEST FOR CLOSED FILE
11200    MCW    +INPUT, IR6  RESET BUFFER POINTER
11210    C    INPUT+3, :1EOF:  END OF FILE TEST
11220    BNE    GETRTN  END OF FILE TEST
11230    MCW    :T:, ENDF  SET END OF FILE FLAG
11240    GETRTN   B    *  END OF FILE TEST
11250    ENDF   DCW    :F:  TEST FOR CLOSED FILE
11260    ENDFMS  EQU    *  TEST FOR CLOSED FILE
11270    :PUT    PRINT, EOFMES,  TEST FOR CLOSED FILE
11280    B    ERRLOC  TEST FOR CLOSED FILE
11290    EOFMES   DCW    :A  END OF FILE ON INPUT UNIT****:
11300    L    DCW    =1C45  END OF FILE ON INPUT UNIT****:
11310    IDTFMT  EQU    *  END OF FILE ON INPUT UNIT****:
11320    BCE    AFRMI, (FMTCD1-3), 10  JUMP OVER CODE TYPE
11330    BCE    EFRMI, (FMTCD1-3), 11  JUMP OVER CODE TYPE
11340    BCE    IFRMI, (FMTCD1-3), 12  JUMP OVER CODE TYPE
11350    BCE    LFRMI, (FMTCD1-3), 13  JUMP OVER CODE TYPE
11360    B    FMTERR  JUMP OVER CODE TYPE
11370    AFRMI   BA    =1B1, FMTCD1  JUMP OVER CODE TYPE
11380    MRID    (FMTCD1-3), IR10-3  A FIELD LENGTH
11390    MRID    (FMTCD1-3), IR10-3  A FIELD LENGTH
11400    SAR    FMTCD1  NEXT FORMAT CODE LOCATION
11410    B    CNVRR  NEXT FORMAT CODE LOCATION
11420    AOK    SW    0-4+X2  ADDRESS OF STRING
11430    C    1R10, 0-5+X2  RIGHT END OF RECEIVING FIELD(PLUS ONF)
11440    BL    FMTERR  RIGHT END OF RECEIVING FIELD(PLUS ONF)
11450    MRID    0-8+X2, IR9-3  RIGHT END OF RECEIVING FIELD(PLUS ONF)
11460    BA    IR10, IR9  RIGHT END OF RECEIVING FIELD(PLUS ONF)
11470    AMORE   SW    0+X6  MOVE STOPPER
BUFFER POINTER(PLUS ONE)
CHECK FOR STRING INPUT
CHECK SPLIT RECORD INPUT
MOVE CHARACTERS
CLEAR
WORD
TEST FOR NEW RECORD REQUIRED
MARKS
CHECK FOR RECEIVING SUBSTRING
ELSE SET ITEM MARK ON RIGHT
AND EXIT
DETERMINE EXCESS CHARACTERS
REDUCE RECEIVING FIELD
MOVE WHAT IS AVAILABLE
CLEAR
WORD
MARKS
NEW LENGTH
GET NEXT RECORD
RESTORE END OF RECEIVING FIELDS
GO FINISH MOVE
END ADDRESS OF INPUT BUFFER
CHECK LENGTH
MOVE STOPPER
NEXT STRING INPUT LOCATION
FINISH PUNCTUATION AND EXIT
JUMP OVER CODE TYPE
I FIELD LENGTH
NEXT FORMAT CODE LOCATION
CHECK FOR MISMATCH
11850  IOK   BS   IR9
11860  MCw :+:+:ISGN
11870  BS   CNVFLD
11880  BCE  ISTRING+DEVTYP+00  CLEAR CONVERSION FIELD
11890  IMORE BCE  STISGN+0+X6,=  CHECK FOR STRING INPUT
11900  BCE  STISGN+0+X6,=  TEST FOR MINUS SIGN
11910  SST  0+X6+DFLD+X9+17  TEST FOR PLUS SIGN
11920  BA   =1B1,IR9
11930  IENDT BA   =1B1,IR6  MOVE NUMERIC BITS ONLY
11940  C    IR6+INPEND  BUMP
11950  BL   GETIPT  COUNTERS
11960  C    IR10+IR9  TEST FOR STRING INPUT
11970  BL   IMORE
11980  IDECMV MCw  DFLD+1+X9,CNVFLD
11990  SST  ISGN,CNVFLD+60
12000  DTB  CNVFLD+00
12010  TAM  CNVFLD+00
12020  MRID  0-8+X2+IR9-3
12030  SI   CNVFLD+2
12040  MRID  CNVFLD+5,0+X9  PUNCTUATION FOR MOVING
12050  CI   CNVFLD+2
12060  B    POPUP
12070  DFLD Dcw =30
12080  ISGN Dcw :+
12090  STISGN MRSU  0+X6+ISGN
12100  BS   =1B1,IR10
12110  E    IENDT
12120  ISTRING SST  (STRADD-3)+DFLD+X9+17
12130  BCE  ISTISGN+(STRADD-3),+
12140  BCE  ISTISGN+(STRADD-3),-
12150  ISRUP EA   =1B1,IR9
12160  BA   =1B1,STRADD
12170  C    1K10+IR9
12180  EH   IDECMV
12190  C    STREND+STRADD
12200  BEH  BUFQFL
12210  B    ISTRING

309
12220 ISTSGN MRSD (STRADD-3)*ISGN
12230 B ISTRUP
12240 EFRMI EA =1B1,FMTCD1
12250 MRID (FMTCD1-3)*IR10-3
12260 SAR FMTCD1
12270 BCE EOK*0-9+x2,02
12280 B CNVERR
12290 EOK BS DEXP
12300 BS IR9
12310 BCE ESTRNG,DEVTYP,00
12320 BS IR11
12330 EMORE MRSD 0+x6*EFHLD+x9
12340 SAR IR6
12350 BA =1B1,IR9
12360 C IR6,INPEND
12370 BL GETIPT
12380 C IR10,IR9
12390 BL EMORE
12400 B ECNV
12410 EFHLD EOU DFLD
12420 ESTRNG MCW STRADD,IR9
12430 BA IR10,IR9
12440 C IR9,STREND
12450 BL BUFOFL
12460 MCW 0-1+x9*EFHLD=1+x10
12470 BA IR10,STRADD
12480 ECNV EOU *
12490 BS DFRACT+10
12500 MCW ++:FSGN
12510 MCW :::ESGN
12520 BS DECEXP
12530 BS IR9
12540 BS IR11
12550 ECKBLK BCE ECNT1,EFHLD+x9,15
12560 BCE FSGNST*EFHLD+x9,**
12570 BCE FSGNST*EFHLD+x9,-
12580 BCE FRACT,EFHLD+x9,**
12590  BCE       EXPON:*EFHLD+X9,E
12600  SST       EFHLD+X9,DFRACT+X11,17
12610  BA =1B1,IR11  DECIMAL FRACTION POINTER
12620  A  :001:,DEXP  INCREMENT EXPONENT (DECIMAL)
12630  ECNT1 BA =1B1,IR9   INPUT CHAR COUNTER
12640  C  IR9,IR10  IF MORE
12650  BH ECKBLK   THEN GO CHECK IT
12660  B  CFDFB    ELSE CONVERT FD/FB
12670  FSGNST MRSD EFHLD+X9,FSGN
12680  B  ECNT1
12690  FRACT BA =1B1,IR9   BUMP CHAR POINTER
12700  C  IR9,IR10  IF NO MORE
12710  BEL CFDFB   THEN CONVERT FD/FB
12720  BCE EXPON:*EFHLD+X9,E  ELSE CHECK EXPONENT
12730  BCE EXPON:*EFHLD+X9,15 IF BLANK THEN LOOK FOR EXPONENT
12740  SST EFHLD+X9,DFRACT+X11,17 ELSE MOVE DIGIT
12750  BA =1B1,IR11
12760  B  FRACT  GO LOOK FOR MORE
12770  EXPON BCE EXPON:*EFHLD+X9,E
12780  BA =1B1,IR9   BUMP CHAR POINTER
12790  C  IR9,IR10  IF NO MORE
12800  BL CFDFB   THEN CONVERT FD/FB
12810  B  EXPON
12820  EXP WR :++:ESGN   DEFAULT SIGN
12830  BA =1B1,IR9
12840  BCE ESGNST:*EFHLD+X9,- CHECK FOR
12850  BCE ESGNST:*EFHLD+X9,+ EXPONENT SIGN
12860  B  EXPNUM  GET EXPONENT
12870  ESGNST MRSD EFHLD+X9,FSGN SET SIGN
12880  BA =1B1,IR9
12890  EXP NUM SW EFHLD+X9
12900  BS DECEXP
12910  MCW EFHLD-1+X10,DECEXP
12920  CW EFHLD+X9
12930  CFDFB SST ESGN,DECEXP,60  SET SIGN
12940  SST FSGN,DFRACT+10,60  SET FRACTION SIGN FOR CONVERSION
12950  A  DECEXP,DEXP    ACCUMULATE DEC EXP FOR CONVERSION
CALL CONVERSION ROUTINE
ADDRESS OF RIGHT END OF 14 CHAR
CALL FLOATING DECIMAL FIELD
ERROR INSTRUCTION
ADDRESS OF RESULT
CLEAR ANY EXTRANEOUS PUNCTUATION
STORE RESULT
ITEM MARK RIGHT
POP STACK AND FETCH NEXT INSTR

DFRACTDCW =11
DExp DCW =3
DECEXP DCW =3
ESGN DCW =1
FSGN DCW =1
LFRMI BA =1B1,FMTC1)
FMTC1-3),IR10-3
JUMP OVER CODE TYPE
FIELD LENGTH
SAR FMTC1
NEXT FORMAT CODE LOCATION
BCE LOK,0-9+X2,03
CHECK FOR TYPE MISMATCH
B CNVERR
BCE BS IR9
CHECK FOR STRING INPUT
LRECP BA IR10+IR6
CHECK
C IR6+INPUT+80
NEW RECORD REQUIREMENT
BL LNWREC
MOVE ADDRESS
MRID 0-8+X2,IR9-3
MRSDR 0-1+X6,0+X9
BE GETIPT
B POPUP
LNWREC BS +INPUT+80,IR6
DETERMINE EXCESS CHAR
NEW LENGTH
MCW IR6,IR10
GETIPT
B LRECP
LSTR MCW STRADD,IR9
CHECK OVERFLOW
BA IR10,IR9
C IR9,STREND
BL BUF0FL
MRU 0-8+X2,IR10-3
ADDRESS OF RECEIVING FIELD
1333C    MRSDR 0=1*x9=0=x10           MOVE INPUT
1334C    MCw 1K9=STRADD                 NEXT STRING INPUT
1335C    B  POPUP                      POP STACK AND FETCH
1336C    ****************************  *
1337C*    STOP - PRINT INSTRUCTION COUNT MESSAGE AND EXIT.  *
1338C*    ****************************  *
1339C*    ****************************  *
1340C*    ****************************  *
1341C    STOP  BS  CNVFLD            
1342C    MCw  INSICT,CNVFLD-2       BINARY COUNT TO CONVERSION FIELD
1343C    TMA  CNVFLD+00             CONVERT IT
1344C    BTD  CNVFLD+00             TO DECIMAL
1345C    LCA  EWORD,PRTCNT          EDIT WORD
1346C    MCE  CNVFLD,PRTCNT         MOVE IT TO PRINT FIELD (EDITED)
13470L  :PUT  PRINT,CNTMES,       
13480    B  (164)                   EXIT
13490   CNTMESDCw :A  *****INSTRUCTION COUNT = :
13500   PRTCNT DC  =9               
13510   DC  :****:
13520   L  DCw  =1C45               
13530  ****************************  *
13540*    ERROR MESSAGES           *
13550*    ERROR EQU *              *
13560*    BADOP DCw :A  *****ILLEGAL OP CODE***:
13570  ****************************  *
13580  ERROR EQU *                *
13590L  :PUT  PRINT,BADOP,        
13600    B  ERRLOC                  
13610   BADOP DCw :A  *****ILLEGAL OP CODE***:
13620   L  DCw  =1C45               
13630  DYNOFLEQU *                *
13640L  :PUT  PRINT,STROFL,       
13650    B  ERRLOC                  
13660  STROFLDCW :A  *****DYNAMIC STORAGE EXHAUSTED****:
13670   L  DCw  =1C45               
13680  CNVERR EQU *               *
13690L  :PUT  PRINT,CVEMS,
13700   B     ERRLOC
13710   CVEMS DCW :A   ****DATA TYPE CONVERSION ERROR****:
13720   L     DCw =1C45
13730   FERR EQU *
13740   L :PUT PRINT,FERMES,
13750   B     ERRLOC
13760   FERMESDCW :A   ****FLTNG PT OVFLW OR ZERO DIVIDE****:
13770   L     DCw =1C45
13780   IDXERR EQU *
13790   L :PUT PRINT,IDXMES,
13800   B     ERRLOC
13810   IDXMESDCW :A   ****INDEXED ADDRESS BEYOND DYNAMIC STORAGE****:
13820   L     DCw =1C45
13830   TYPERR EQU *
13840   L :PUT PRINT,TYPMES,
13850   B     ERRLOC
13860   TYPMESDCW :A   ****INCONSISTENT DATA TYPE ERROR****:
13870   L     DCw =1C45
13880   FMTERR EQU *
13890   L :PUT PRINT,FMTMES,
13900   B     ERRLOC
13910   FMTMESDCW :A   ****FORMAT CODE ERROR****:
13920   L     DCw =1C45
13930   ERRLOC BS LUDLOC,IR3       FIND RELATIVE LOCATION
13940   MCW IR3,CNVFLD-2
13950   TMA CNVFLD+00       CONVERT
13960   BTD CNVFLD+00       TO
13970   LCA EWORD,ELOC       DECIMAL
13980   MCE CNVFLD+ELOC       FOR PRINTING
13990   L :PUT PRINT,LOCMES,
14000   BCT HALT,01       IF SENSE SWITCH ONE THEN DUMP REQUEST
14010   B     STOP       ELSE EXIT
14020   HALT H         DUMP REQUEST
14030   EWORD DCw : , 0     :
14040   LOCMESDCW :A   ****ERROR AT RELATIVE LOCATION :
14050   ELOC DC =9
14060   DC :****:
Symbol Definition - Card Reference Index

ADCQP 01840;  ADD 06290;  AFIN 11560;  AFORM 09390;  AFRMA 09450;
AFRMI 11370;  ALCRT1 02600;  ALCRT2 02630;  ALLOC 02490;  AMORE 11470;
AND 07060;  AOX 11420;  AOK 11420;  ASPL 03610;  ASPLT 11600;  ASTRNG 07130;
BADUP 13610;  BFJFL 09660;  BGSTR 09050;  BLNK 06160;  BSTF 07130;
BSTT 07100;  BUFJFL 09630;  CAT2 07720;  CAT3 07760;  CAT 07670;
CFDFB 12930;  CHK2ND 05150;  CHKBF 09470;  CHKTYP 03180;  CHRSTR 07670;
CNTMES 13490;  CNVCHK 06860;  CNVFLD 14080;  CNVRTN 03410;
CNVRTR 06950;  CNVTUP 06940;  COLMN 08990;  COLSET 09010;  COMP 05400;
COND 05210;  CONPRT 02860;  CBSTR 03930;  CTSUB1 07810;  CTSUB2 07920;
CURLEV 04550;  CVEMS 13710;  DATFMT 09330;  DECEXP 13070;  DECRE 10550;
DIV 06610;  DODIV 06670;  DTYPE 08420;  DUMMY 09290;  DVDNE 02750;
DYNAME 02280;  DYNEND 00460;  DYNFL 13630;  DYNS 00450;  ECKBLK 12550;
ECNTI 12630;  ECNV 12480;  EDIT 08870;  EFOR 12410;  EFORM 10070;
EFMD 01440;  EFRMI 12240;  ELOC 14050;  EMORE 12330;  END 06020;
ENF 11250;  ENDFM 11260;  ENDPRG 14100;  ENTPRO 04220;  EOT 11290;
EOK 12290;  ERRLOC 13930;  ERROR 13580;  ESGN 13080;  ESIGN 12870;
ESTRNG 12420;  EWORD 14030;  EXPNUM 12890;  EXP 12770;  EXPR 12820;
FERMES 13760;  FERR 13730;  FETCH 01170;  FIXFLT 03640;  FLG 04130;
FJCHK 04310;  FMT 08290;  FMTCD 08410;  FMTCD2 08420;  FMTF 13880;
FMMES 13910;  FMRST 09310;  NDLNG 08140;  FOUT 10050;  FRACT 12690;
FSGN 13090;  FSGN 12670;  GET 10670;  GETFLM 10690;  GETI 11170;
GTRTN 11240;  GTALN 09560;  GTSL 09610;  HALT 14020;  HOLD 06190;
ICL 10930;  IDCMV 11980;  IDTFMT 11310;  IDXERR 13780;  IDX 13810;
IFND 11930;  IFIX 00780;  IFMCNV 09780;  IFMN 0870;  IFMRST 10750;
**Instruction Count = 296,018**
IDENTIFICATION:

PROGRAM-ID: MTXMCP03.
AUTHOR: J. R. VAN DOREN.
SOURCE LANGUAGE: EASYCODER.
SOURCE COMPUTER: H-1200
OBJECT COMPUTER: H-1200

PURPOSE:

MTXMCP02 PROVIDES THE METAX SYSTEM CONTROL FUNCTIONS AND SYSTEM SERVICES. SEE THE CHAPTER ON THE METAX SYSTEM FOR A DETAILED DESCRIPTION.

INDEX REGISTER LOCATION DEFINITIONS

IR1 EQU 4
IR2 EQU 8
IR3 EQU 12
IR4 EQU 16
IR5 EQU 20
IR6 EQU 24
IR7 EQU 28
IR13 EQU 52
IR14 EQU 56
IR15 EQU 60

OCTAL ADDRESS DEFINITIONS OF PERTINENT SYMBOLS IN THE RESIDENT INPUT/OUTPUT ROUTINES.

RDWR CEQU =4C00000754
READ CEQU =4C00005430
INPUT CEQU =4C00006144
OUTPUT CEQU =4C00006265
PRINT CEQU =4C00005647
#SKP CEQU =4C00000756

COMMUNICATION AREA FIELD LOCATION DEFINITIONS

GENFLD EQU 215
LODFLD EQU 219
STCKF1 EQU 223
STCKF2 EQU 227
SYMFL EQU 231
(SYMFL) EQU 235
SYMFL EQU 235
CMPLCD EQU 236
DSKLOD EQU 237
EXCPPG EQU 238
PSILST EQU 239
PRGNAME EQU 290
INTNAME EQU 298
SYMBOL EQU 243
INSTCT EQU 209

CONTAINS ADDRESS OF CODE GENERATION LOCAL CONTAINS METAX PROGRAM LOADING ADDRESS BEGINNING PUSH DOWN STACK ADDRESS CONTAINS STACK LIMIT ADDRESS ADDRESS OF SYMBOL TABLE START (START OF DYNAMIC STORAGE FOR PLEX OBJECT PROGRAMS) CONTAINS SYMBOL TABLE LIMIT ADDRESS (LIMIT OF DYNAMIC STORAGE FOR PLEX OBJECT PROGRAMS) COMPLETION CODE FIELD SET BY COMPILERS DISK LOADING OPTION FIELD GO OPTION FIELD POST LISTING OPTION FIELD METAX PROG NAME FIELD INTERPRETER NAME FIELD SYMBOL FIELD USED BY COMPILERS INTERPRETER INSTRUCTION COUNT FIELD
LOCATION FOR RESIDENT METAX PROGRAM
LOCATION FOR RESIDENT CONTROL RECORD ANALYZER
SET FOUR CHAR ADDRESSING MODE
GET NAMES OF RESIDENT METAX PROGRAMS
SAVE FOR LATER USE
SET LOCATION FOR CONTROL RECORD ANALYZER
NAME TO METAX COMMUNICATIONS FIELD
SEGMENT AND
NAME OF DISK TO MEMORY LOAD PROGRAM
SET UP RETURN FOR RETURN START
FETCH AND EXECUTE
SET UP RETURN FOR RETURN START
EXIT POINT
RETURN TO LOADER
SET NAME OF RESIDENT METAX PROGRAM
SET LOCATION FOR RESIDENT METAX PROGRAM
EXIT POINT
FETCH RESIDENT METAX PROGRAM
ZERO INSTRUCTION COUNT
COMMUNICATIONS AREA
FIELDS
FOR EXECUTING
CONTROL RECORD ANALYZER
INTERPRETER SECT
AND NAME
RETURN POINT
FETCH AND EXECUTE
ZERO INSTRUCTION COUNT
INITIALIZE COMPLETION CODE
SET UP
MEMORY
CLEAR
OPERATION

DO IT

TEST EXECUTION OF RESIDENT MFTAX PROGRAM
IF NOT GO GET THE RIGHT ONE
ALTER EXECUTION LOCATION TO RESIDENT PROG
INTERPRETER SEGMENT
AND NAME
RETURN POINT
FETCH AND EXECUTE
TEST FOR FATAL ERROR ACTION
LOAD COMPILED PROGRAM TO DISK IF REQUESTED
IF NO GO THEN SEARCH FOR END OF FILE
ELSE MOVE AND RELOCATE COMPILLED
PROGRAM FOR EXECUTION

COMPILLED PROGRAM LOCATION
LOADING LOCATION

MOVE REMAINDER OF ADDRESS
RELOCATE THE ADDRESSES
TEST POSSIBLE BLOCK PSEUDO OP CODE
PREPARE FOR CLEARING REMAINING MEMORY
CLEAR
01490  MCw SYMF2,IR15
01500  MRSBR =4B0.0+X14
01510  SBR IR14
01520  Sw 0-1+X15
01530  SI 0-1+X15
01540  B CLEAR
01550  LCA =4B0+INSTCT
01560  B EXINT
01570  BLKCNT HNP ENDTST,0-5+X14
01580  BNP ENDTST,0-4+X14
01590  MRID 0-3+X14,IR14-2
01600  B ENDTST
01610  MTXSTR DCw :MTXSTR:
01620  STHSEG DCw :00:
01630  MTXLDR DCw :MTXLDR:
01640  LDRSEG DCw :00:
01650  MTXINT DCw :MTXINT:
01660  INTSEG DCw :02:
01670  SAVNME DCw =16B0
01680  LODSAV DCw =4B0
01690  CLEAR SBR CLRRTN+4
01700  MOPE MRIDw 0-1+X14,0+X14
01710  SBR IR14
01720  CI 0-2+X14
01730  Cw 0-2+X14
01740  C IR14+IR15
01750  BH MORE
01760  CLRRTN B *
01770  EXINT MCw INTNME,75
01780  MCw
01790  MCw +EOFTST*167
01800  B (168)
01810  EOFTST C INPUT+3,:1EOF:
01820  BE CRACLL
01830  :GET READ*
01840  B EOFTST
01850  FATAL EQU *

SYMBOL (OR DYNAMIC STORAGE FOR PLFX)

TABLE AREA
ZERO INSTRUCTION COUNT
GO LOAD INTERPRETER
MAKE SURE IT IS A BLOCK CODE
ADJUST MEMORY POINTER BY SIZE OF BLOCK

CORE TO DISK UPDATE PROGRAM NAME
METAX SYSTEM LOADER PROGRAM NAME
MTXCRA INTERPRETER NAME
INITIAL METAX PROGRAM NAMES SAVE LOC
LODFLD SAVE LOCATION
MEMORY CLEAR SUBROUTINE

SEGMENT AND NAME
FOR EXECUTING COMPILED PROGRAM
RETURN POINT
FETCH AND EXECUTE
TEST END OF FILE
NEXT JOB
SEARCH UNTIL IT IS FOUND
01860L PUT PRINT,FTLMES,
01870 B EOFTST
01880 FTLMESDCw :1 FATAL ERROR(S) ENCOUNTERED, JOB ABORTED:
01890 L DCw =1C45
01900 LTODSK SBR 167 SET RETURN POINT
01910 MCw IR5,IR14 AVOID SUPERVISOR USE OF IR5
01920 MCw STRSEG,75 SEGMENT AND NAME
01930 MCw OF MEMORY TO DISK PROGRAM
01940 B (168) FETCH AND EXECUTE
01950 MFETCH MCw LDRSEG,75 SEGMENT AND NAME
01960 MCw OF DISK TO MEMORY PROGRAM
01970 MCw +FTRTST,167 SET UP RETURN FOR RETURN START
01980 B (168) FETCH AND EXECUTE
01990 FTRTST SBR FTRTN+4 SET UP RETURN START
02000 MCw +GETINT,167 SET EXIT
02010 FTRTN B *
02020 LITORG
02030 ENDP RG EQU *
02040 END START

SYMBOL DEFINITION - CARD REFERENCE INDEX

BLKCNT 01570; CLEAR 01690; CLRRTH 01760; CMPLCD 00660; CPACLL 00960;
DSKLOD 00670; ENDP RG 02030; ENDTST 01410; EOFTST 01810; EXCPPG 00680;
EXINT 01770; FATAL 01850; FTLMES 01880; FTRTST 02010; FTXTST 01990;
GENFLD 00560; GETINT 01180; GO 01250; INPUT 00470; INSTCT 00730;
INTNAME 00710; INTRT1 01060; INTRT2 01220; INTSEG 01660; IR13 00360;
INSTRUCTION COUNT = 47,601***
XIII. APPENDIX E

Some of the pertinent hardware and software characteristics of the host computer system, an H-1200, are presented below. Comments about machine dependent characteristics of the METAX system are also included.

Basically the host system is a variable word length two address computer. An eight bit character consisting of six data bits and two punctuation bits is the unit of addressable storage. Normally only the data bits participate directly in data manipulation operations, the punctuation bits being used to delimit the respective fields. Punctuation may participate in data moving instructions, however.

The two punctuation bits are referenced as a word mark and an item mark. For the most part the H-1200 instruction set expects delimiting word marks on the left of a data field with addresses being given on the right. A specific exception to the punctuation requirements occurs with the floating point hardware option in that floating point instructions do not utilize these bits in any way since all operands are fixed length. However, no boundary alignments are required which simplifies certain translation or interpreter factors.

Floating point instructions also represent a departure from the two address scheme in that floating point registers are used in a one address fashion.
The internal data representations for the respective pseudo-machines correspond to the host computer with the exceptions that addressing is always on the left and item marks are used as right end delimiters. The respective interpreters make the necessary adjustments for addressing and may insert word marks on the left during execution. However, word marks are never generated for data fields during translation or loading.

The only explicit use of word marks in object code is to mark the left hand character of an 18 bit address field as a relocatable address or pseudo-address. During loading by either the control program or MTXLDR these word marks provide a convenient scheme for marking addresses to be relocated.

The RESOLVE primitive, discussed in Chapter III, also utilizes word marks to examine object code for potential pseudo-addresses. A pseudo-address is marked by a one in the left most bit of an 18 bit address in addition to the word mark. The remaining 17 bits comprise a symbol table address as described.

The use of punctuation bits represents a significant dependence on the structure of the host machine for all of the METAX processors.

The addressing structure of the H-1200 is binary. Address modification may be effected with either indirect addressing or indexing. There are three addressing modes
based on the amount of storage to be addressed and the number of index registers to be used. The mode used in all the assembler programs in the METAX system is the four character or 24 bit mode which allows a 19 bit address and a five bit address modifier. The latter is used to specify one of fifteen index registers or indirect addressing.

The index registers are resident in main storage and are thus manipulated with standard storage-to-storage arithmetic and data moving instructions. Assembly control statements are used to equate the symbols IR1, IR2,...,IR15 to the proper addresses for purposes of symbolic reference. Thus

\[ BA = 1B1, IR13 \]

specifies that a one character binary constant of one is to be added (in binary) to index register 13.

The specification of indexing is exemplified by

\[ MCW TVEC+3+X7, IR14 \]

which specifies that the first operand is to be moved to index register 14. The address computation TVEC+3 is effected at translation time while the indexing via index register seven (specified by +x7) takes place at execution time.

There are two address registers, the A and B address registers, which are referenced frequently for updating index registers. Thus

\[ SAR IR1 \]
SBR IR10

specify that the A and B registers are to be stored in index registers one and ten, respectively. Such instructions are used frequently in the interpreters immediately following an extended move instruction as discussed below.

The B register may also be used for subroutine linkage.

Specific forms of the generic EXM (extended move) instruction are used extensively for data and punctuation moving and for scanning purposes. With this instruction one may establish three categories of options. The first is the direction of the move, left or right. This is important because the A and B registers will be set one position beyond the last character position processed for the first and second operands, respectively. On completion of an EXM instruction SAR and SBR may be used to store the contents of the address registers.

The second category is the terminating condition which may be a single character move or any one of three combinations of punctuation bits. The third category specifies which combination of data and/or punctuation bits to move, if any.

Then

MRIDI 0+x6,SYMBOl

specifies the data and item mark bits of the first operand
are to be moved from left to right to SYMBOL with the move terminated by the first item mark in the sending field while

\text{MRIN} \ 0\times14,0\times13

does nothing more than position the A and B registers according to the first item mark found in the first operand.

Item marks are used extensively in the object code of the pseudo-machines to delimit address fields and literal operands. This scheme is not essential for addresses because the address size is fixed but it does speed up interpretation in that arithmetic instructions for updating index registers are not required in many cases.

With respect to symbolic addressing within the respective assembler programs instructions are normally addressed on the left and data fields on the right.

A reversal of these rules is used on occasion by indenting the location field by one position.

The reader is referred to the appropriate Honeywell publications (27,28) for more information on the assembler language and hardware characteristics.

The system supervisor (24) under which the METAX system operates utilizes its own communications region. Several field in this region are used by the METAX system. Decimal positions 67-75 are used to communicate the name of a program to be loaded. An indirect branch to the address in positions 168-171 (B (168)) is then a supervisor call to fetch and ex-
execute the named program. A return address may be set in positions 164-167 which is used by programs loaded into the transient region to return to the METAX control program.

All input and output operations are coded using macro routines outlined in (26). These include unit record and disk I/O functions. The METAX library is maintained in a partitioned sequential data file on disk. Additional information about certain aspects of Honeywell's version of this type of data file may be found in (25).