1971

An experimental compiler-compiler system

James Robert Van Doren

Iowa State University

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

Part of the Computer Sciences Commons

Recommended Citation

https://lib.dr.iastate.edu/rtd/4516

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
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
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 META9 metalanguage with which the PLZX translator was written, PLEX being the PL/I-like language mentioned. A review of META9 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 META9. 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

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

\[ \langle \text{SUBLIST} \rangle ::= \langle \text{SUBLIST} \rangle , \langle \text{EXP} \rangle | \langle \text{EXP} \rangle; \]

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 "\( I \)" 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:

```plaintext
.SYNTAX PROGRAM
PROGRAM := "SYNTAX" .ID $ ST "END";
ST := .ID "=" EX1 ";";
EX1 := EX2 $ ( "I" 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-
development 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 McNeeley (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.
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:

```
SUBLIST:="(" 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.
PORTRAN-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(DYNAMB>DYNAMP)

.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

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

as opposed to

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

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.

\[
\text{ENDING} ::= \text{"END" (.ID/.EMPTY)};
\]
The slash is used as an alternation symbol and corresponds to "|" in BNF.

4. **Semantic commands**

**.OUT(...) (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 METAX9 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, ENLOC):**

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 de-
pending on whether the specified symbol is already in the ta-
ble or whether it must be entered.

.STKSYM (STKSYM):
The symbol contained in SYMBOL is pushed onto the con-
trol 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 META9 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,DYNAMB:**
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 II. 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 contained 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 METAX 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 .TYPEG 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 translator were then written.

The ability to declare compile time variables was included 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 necessary or desirable and retested by translating its own translator.

Of course revisions and additions were made to the corresponding 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. MTLDR, 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.

MTXCPA 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.

INSTRCT (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.

STCKF1 (221-223)* and STCKF2 (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 respective 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 MTXCR 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
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 FLEX 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 METAAX 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 implementation 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 upper most 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 procedure 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 beyond 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 procedure.

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 value 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-
acter string the maximum length is extracted from the in-
struction (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, MUL, 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.

CALLPET 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 aesthetically pleasing from a formal syntactical point of view but they may be rather effective.
As PLXCPL 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.

DECLAR 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 \( \text{OUT}(\text{LD}, 76, **(04, 01)) \) is issued, being equivalent to \( \text{OUT}(\text{LDA}, 06, **(04, 01)) \), which identifies the address as a literal address constant. Recalling the label resolution function of BLKEXT, primitive \text{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 undergo 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 \texttt{.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 PLXCPP effectively represents the capability of META9 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
further investigation.

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 METAX8.
INTERPRETER=MTXINT03.
STORE=YES.
GO=NO.

//
*PROG MTX009;

*****************************************************************************/
/*/ 
/*/ IDENTIFICATION: 
/*/ 
/*/ """""""""""
/*/ PROGRAM-ID: METAX9.
/*/ AUTHOR: J. R. VAN DOREN.
/*/ SOURCE LANGUAGE: METAX8.
/*/ OBJECT LANGUAGE: METAX8 PSEUDO-MACHINE CODE.
/*/ OBJECT INTERPRETER: MTXINT03.
/*/ 
/*/ PURPOSE: 
/*/ """"""""
/*/ METAX9 IS A REVISION OF THE METAX8 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; 
TYPTST .OEQU 34 .TYPEO; ENTTYP .OEQU 35 .TYPEO; ENTFR .OEQU 72 .TYPEO; 
DECNUM .OEQU 75 .TYPEO;
/* OPERATION CODES COMMON TO METAX8 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; ENTL .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;
SAVE .OEQU 41 .TYPEB; RESTORE .OEQU 45 .TYPEB; ERASE .OEQU 31 .TYPEB;

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

BEF .OEQU 16 .TYPEG; COMP .OEQU 32 .TYPEG; TSTTBL .OEQU 34 .TYPEG;
TSTTBA .OEQU 35 .TYPEG; ENTA .OEQU 52 .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 .TYPEP; SYMOL .OEQU 000363 .TYPEP;
CMPLCD .OEQU 000354 .TYPEP; PRGNME .OEQU 000433 .TYPEP;

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

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

PRGHD := "PROG" .ID ";"
.DO(MOVSYM "PRGNME"
.OUT(B**) $ (DECLPT/COMMENT);
DECLPT ::= "DECLARE" : ( .ID .DEFLAB(*) .STRING / .ONUM ) .OUT(*) ( ":" / ";" .RETURN ) ) ;

PRG1DY ::= / 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 )) ";"

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

MTXEXP ::= NEWLAB NEWLAB ALTERN $ ( ":" .DEFLAB(*1) .DO(POP) .NEWLAB ALTERN ) .DO(MOVI,SYMBOL,"4",ERASE) .DEFLAB(*1) .DEFLAB(*2) ;
/* THE METALANGUAGE CONSTRUCTS CONTAINED THEREIN ARE CLASSIFIED AS ELEMENTARY SYNTAX OR SEMANTIC ACTION. */

ALTERN ::= $ SENACT ( ELMSTX *OUT(BF,*1) / *EMPTY )

$ ( SENACT / ELMSTX ERRACT ) *OUT(E,*2) ;

/* THE ELEMENTARY SYNTACTICAL CONSTRUCTS ARE OUTLINED BELOW. OBSERVE THAT "STKCHR", "STTBA", "STTBL" 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(CLAS,**) / *STRING *OUT(TEST**) / "ID" *OUT(ID) / "STRING" *OUT(STRTST) / "ONUM" *OUT(INUM) / "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) / "STKCHR" *OUT(CHRSYM)

"LATCH(" ID *OUT(LATCH,**) ")"

"TSTTBA(" *OUT(STTBA) ONUM *OUT(*) ", ID *OUT(**) ")"

"TSTTBL(" *OUT(STTBL) ONUM *OUT(*) ", ( ONUM *OUT(*) / STRING *OUT(*) ) ")"

"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 $ ("","OUT")")"
            /"\*DO(" OUTDO $ ("","OUTDO")")"

OUTDO :=\*ID \*OUT(**) / \*NUM \*OUT(*) / \*STRING \*OUT(*)

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

SEMI :=="*NEWLAB" \*OUT(PUSHLB)
            /"*DEFLAB" ("*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 / \*NUM ) \*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) \*NUM \*OUT(***)
                                      ( \*STRING / \*NUM ) \*OUT(* )")"
            /"*ENTERA(" \*OUT(ENTA) \*NUM \*OUT(* )" \*ID \*OUT(***) ")"
            /"*SEARCH(" ("(" \*STRING \*OUT(SEARCH,**) ")")
                                      / \*EMPTY \*OUT(SEARCH,0) )
```

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

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

ERRACT := "ERR(NEWLAB.OUT(BT,*1).STRING.OUT(BM,*)

$ ("","OUTDO ")" .DEFLAB(*1)

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

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

END)

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

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

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

//
.PROG PLXCPL;

/**************************************************************************
* 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.

**************************************************************************/

/DECLEARE 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. */

.DECLAIM DYNAMP "1008", DYNAMB "0000", STACKTP "$STKTO ", ONELEV 01000000,
   LEVNO 01, ONELNG "0000100", DOPFIX 0008;

/* VARIABLES USED FOR SYNTACTIC ANALYSIS */

.DECLAIM DOSYM "DO ", IFSYM "IF ", CALLCON "CALL ",
   RPAREN "]", SYMSAV " ", STMLAB " ";
/* 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, PEIXTT 00;
/* VARIABLES USED FOR TESTING I/O ITERATION LOOPS */
.DEclare IOIITER 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;
B .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; MOV I .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  OEUQ 43 TYPEO;  PUSHLB  OEUQ 46 TYPEO;  POP  OEUQ 47 TYPEO;
LBI  OEUQ 50 TYPEO;  LBZ  OEUQ 51 TYPEO;  ENTA  OEUQ 55 TYPEO;
ENTL  OEUQ 56 TYPEO;  OUT  OEUQ 62 TYPEO;  OUTSYM  OEUQ 63 TYPEO;
TEST  OEUQ 64 TYPEO;  ID  OEUQ 65 TYPEO;  ONUM  OEUQ 66 TYPEO;
STRTST  OEUQ 67 TYPEO;  EVAL  OEUQ 70 TYPEO;  ENTEOC  OEUQ 71 TYPEO;
INUM  OEUQ 73 TYPEO;  FNUM  OEUQ 74 TYPEO;  BLKENT  OEUQ 75 TYPEO;
BLKEXT  OEUQ 76 TYPEO;  STKSYM  OEUQ 52 TYPEO;  CHKSYM  OEUQ 53 TYPEO;
SWAP  OEUQ 54 TYPEO;  MAPK  OEUQ 40 TYPEO;  SAE  OEUQ 41 TYPEO;
RESTORF  OEUQ 45 TYPEO;  ERASE  OEUQ 31 TYPEO;

;/* OPERATION CODES FOR PLEX OBJECT PROGRAMS */

DYNAM  OEUQ 10 TYPEG;  STKTOP  OEUQ 11 TYPEG;  ALLOC  OEUQ 12 TYPEG;
LDA  OEUQ 20 TYPEG;  LD  OEUQ 21 TYPEG;  STC  OEUQ 22 TYPEG;
SST  OEUQ 32 TYPEG;  JUMP  OEUQ 30 TYPEG;  JUMPT  OEUQ 31 TYPEG;
ADD  OEUQ 33 TYPEG;  STCKC  OEUQ 33 TYPEG;  COMPC  OEUQ 34 TYPEG;
DIV  OEUQ 40 TYPEG;  MUL  OEUQ 41 TYPEG;  SUB  OEUQ 42 TYPEG;
GET  OEUQ 51 TYPEG;  PUT  OEUQ 52 TYPEG;  EDIT  OEUQ 53 TYPEG;
OR  OEUQ 60 TYPEG;  AND  OEUQ 61 TYPEG;  NOT  OEUQ 62 TYPEG;
INDXR  OEUQ 66 TYPEG;  INDXA  OEUQ 67 TYPEG;  CAT  OEUQ 70 TYPEG;
SUBSTR  OEUQ 71 TYPEG;  JUMPA  OEUQ 77 TYPEG;  STOP  OEUQ 76 TYPEG;
ENTPRO  OEUQ 25 TYPEG;  RETURN  OEUQ 41 TYPEG;  FLAG  OEUQ 24 TYPEG;
SWP  OEUQ 35 TYPEG;  POPUP  OEUQ 36 TYPEG;

;/* THE DEFINITION OF COMMUNICATIONS AREA FIELD LOCATIONS FOLLOWS. */

SYMBOL  OEUQ 000363 TYPEN;  PRGNAME  OEUQ 000433 TYPEN;
INTNAME  OEUQ 000443 TYPEN;  SYMPTWO  OEUQ 000365 TYPEN;
ELATCH  OEUQ 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=SYMBOL) /* SET PROG NAME FOR STORE OPTION */
*SET(INTNAME="PLXINT00") /* SET INTERPRETER NAME FOR "GO" OPT. */
*SET(SYMBOL="$ACTIVE") /* SYMBOL AND ADDRESS VALUE OF INDEX */
*SEARCH *ENTERL(00,0100000065) /*REGISTER POINTING TO ACTIVE */
*SEARCH *ENTERL(00,0501000000) /*TYPE AND INIT STACKTOP LOCATION */
$ COMMENT /* ADMIT HEADER COMMENTS */
BHDBDY /* PROCESS BLOCK HEAD AND BODY */
"END" *OUT(STOP) /* POSITION SYMTAB POINTER */
*DEFLAB(*1) /* 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 := $ (*SET(STMLAB=BLNK8) STMENT ) ;
STMENT := *ERRLATCH $ COMMENT & *LATCH(LABEL) /* PROCESS COMMENTS AND LABELS */
( ENDTST *DO(SETF,R) /* DO NOT MISTAKE "END" AS AN ID */
/ CONST / BSCSTM ) *ERR("F: BAD STATEMENT","SET,SCAN,"; ");
BSCSTM := ( *LATCH(DOGROUP) / *LATCH(BLOCK) / UNCOND / *EMPTY ) "; ";
COMMENT := "/*" *SCAN("*/") ;
LABEL := *ID ":" *SET(STMLAB=SYMBOL) /* SAVE LABEL */
*SEARCH("1") *DO(ENTLOC) /* ENTER LABEL AND VALUE */
*ENTERL(00,06) ; /* LABEL CONSTANT TYPE */
ENDTST := *LATCH(ENDSTMT) *DO(SETF) / *EMPTY 
ENDSTMT := "END" *DO(SETF,BEF) / *EMPTY ;
\begin{verbatim}
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)
*DO(A,ONEELNG,STACKTP) /* POTENTIAL RECURSIVE CALL OF "BLOCK"*/
*DO(A,ONEELNG,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 */
*SET(DYNAMP=DYNAMB) /* INCREMENT DYNAMIC STORAGE COUNTER */
*OUT(**) /* PROCESS BLOCK HEAD AND BODY */
*UNSTACK(DYNAMB,STACKTP) /* RESTORE PREVIOUS STACK TOP SYMBOL */
"END"•BLKEXT /* AND BLOCK DYNAMIC STORAGE COUNTER */
( •ID •STKCHK •ERR("W: POSSIBLE BLOCK CLOSING ERROR",SET)
/ •EMPTY ) ;

BHDEDY := BLKHD BLKEDY /* DECLARATIONS AND BLOCK BODY */
( •TEST(DYNAMB > DYNAMP) /* DETERMINE LEVEL OF STORAGE REQUIRED */
•SET(DYNAMP=DYNAMB) /* SET NEW LEVEL */
/ •EMPTY) /* OLD LEVEL O.K. */

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

PROCDEF:=•ID "" "PROCEDURE" •CANCEL •NEWLAB /* 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,DYNAMP,LEVNO,STACKTP) /* SAVE PREVIOUS PROCEDURE INFO */
•DO(A,ONE,LEVNO) /* SET UP */
•SET(DYNAMP=ONELEV) /* DYNAMIC STORAGE */
\end{verbatim}
\*DO(M\*LEVNO\*DYNAMP) \* COUNTERS
\*SET(DYNAM\*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
\*DO(M\*LEVNO\*SYMBOL\*A\*FOUR,SYMBOL\*A\*SYMBOL\*DYNAMP)
\*OUT(ENTPRO) \* PROCEDURE ENTRY
\*SET(SYMBOL=LEVNO) \* 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
\*TEST(ARGCNT=ADECNT) \* ERR("W: INCORRECT ARG DCL COUNT",SET)
/ \* EMPTY \*/
BHDECY \* 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\*R) / \* EMPTY;
ARGID := ID • SEARCH("I") • ENTERA(01, DYNAMB) /* ACTUAL PARAMETER ADDRESSES */
• ENTERL(00, 60) • DO(A, ONE, ARGCNT) /* IN DYNAMIC STORAGE. MARK */
• DO(A, FOJR, DYNAMB, SFT) ; /* DUMMY ARGS INDIRECT REF */

ARGDEC := ID • SEARCH("I") • ERPLATCH • TSTTBL(00, 60)
• ERR("F: NON-EXISTNT ARG") ARGARY ARGATR • DO(A, ONE, ADECN) ;

ARGARY := "(" • 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, TypE)
/ • 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(STMLAB=SYMBOL) /* SAVE PROC NAME FOR CHECKING */
• NEWLAB • OUT(LDA, 06, 1) /* CODE TO LOAD RETURN ADDRESS */
• OUT(LD, 0100000065) /* CODE TO LOAD CURRENT DISPLAY ADDR */
• SET(SYMBOL=STMLAB) • 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(*)
.*OUT(LD,71,"0004",ADD)
.*CUT(STO)
.*OUT(LD,01) .DEFLAB(*1)
.*ENTERL(00,01)
.*OUT("0000")
.*DO(POP)
.*DO(SEL)
.*ERR("F: INVALID PROC CALL")

CLLP RM :=.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) ;

PARM ID :=.LATCH(ALFTPT) / .LATCH(BLFTPT) / .LATCH(SLFTPT) / .LATCH(LLFTPT) / .LATCH(PROCHK) ;

PROCHK :=.ID ( "(" .DO(SETF,BEF) / .EMPTY )
SEARCH("O")
( TSTBL(00,10) CANCEL GPROC /* CHECK FOR GLOBAL PROC PARM */
/ TSTBL(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 / DPMRY )
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,**,LD0100000065,STO) /* CODE TO PUT CURRENT DISPLAY ADDR */
/* IN DYNAMIC STORAGE AS GLOBAL DISPLY*/
/* AT POINT OF CALL */
DO(A,FGJR,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,**)

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

•DO(A*FOUR,DYMB)
•NEWLAB
•OUT(LDA,01,*1)

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

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

/* CODE TO STORE IN CURR DYNAMIC STOR */

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

DECL1 :=IDELMNT / IDGROUP;

IDGFOUP:="" LISTPT ;

LISTPT :=•ID •STKSYh

/* CODE TO STORE IN NEXT INSTk */

ATTRIBUTE :="" LISTPT / ";"" ATTRIBT ; /* NOTE MUTUAL RECURSION WITH LISTPT */

ATTRIBT :=""CHARACTER(" / "CHAR(" )"INUM ")"
*SET(LENGTH=SYPMTWO) /* TWELVE BIT (TWO CHARACTER) LENGTH */ *SET(TYPE= 04) /* CHARACTER TYPE CODE */ / "RETURNS( " ATTRIBT ")" *DO(A,OCTEN,TYPEx) *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 */ INTBDN ("":*OUT(#) INTBDN .OUT(#)) ; /* DEFAULT LOW BOUND ONE */

INTBDN :=*ID .SEARCH .TSTTBL{00»01) .ERR("W: INVALID ARRAY BOUND") .SAV(**) /* INUM .SAV(71,*); */

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

ARRYSEM :=*SET(SYMBOL=STACKTP)

/* INITIALIZE DIMENSION COUNT */
/* SAVE BOUND PAIR CODE */
/* CATENATE BOUND PAIR CODE */
/* DEFAULT LOW BOUND ONE */
/* INUM .SAV(71,*); */
OUT(ALLOC,**)
* OP CODE AND STACK TOP ADDRESS */
* FOLLOWED BY SDV */
SET(SYMBOL=BLNK8)
* CLEAR SYMBOL TO BLANKS */
SET(SYMBOL=SYM5AV)
* SEARCH("1") */ POSITION SYMTAB POINTER */
ENTERA(07,DIMCNT)
* ENTER DIMENSION COUNT */
IDSLM
* ENTER TYPE, LENGTH AND ADDRESS */
OUT(**)
* DYNAMIC DOE VECTOR ADDRESS */
OUT(**(01,07))
* DIMCNT ALSO TO SDV */
OUT(**(02,05))
* LENGTH TO SDV */
OUT(?)
* FINISH SDV WITH BOUND PAIR CODE */
DO(A,DOPF1X,DYNAMB)
* ADD FIXED DOE VECTOR SIZE */
DO(M,DIMCNT,SYMBOL)
* ADJUST FOR MULTIPLIER STORAGE */
DO(SET);

IDSLM := ENTERA(00,TYPE)
* ENTER TYPE CODE */
ENTERA(05,LENGTH)
* ENTER LENGTH */
( TEST(LENGTH=FUNCT) RETUPN
* ENTER(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 MULTI BY TV ELEMENT SIZE */
DO(LB1) SEARCH("1") ENTERL(00,01) /* MARK ADCON 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 */
OUT(JUMP,*2)     /* CODE TO JUMP OUT OF CASE GROUP */
SAV(JUMP,*1)    /* FIRST TV ENTRY */
NEWLAB .DEFLAB(*1) SMENT    /* LABEL AND Compile Statement */
OUT(JUMP,*2)    /* CODE TO JUMP OUT OF CASE GROUP */
CAT(JUMP,*1)    /* CATEenate CODE TO TRANSFER VECTOR */
DO(Pop)        /* Pop ExTREMEOUS LABEL */
LD(*6)         /* DUMMY FOR POSTLISTING */
DEFLAB(*2)     /* ADDR OF ADCON AND MARK INTEGER */
DO(Swap,Pop)   /* DISCARD ADCON LABEL AND PUT EXIT */
DO(Pop)        /* LABEL ON TOP */
OUT(*2)        /* GENERATE ADCON */
DEFLAB(*2)     /* DEFINE TV ADDRESS */
OUT(*)         /* OUTPUT TV TO CODE STRING */
ENDING       /* DEFINE CASE GROUP EXIT ADDRESS */

WHILE := "WHILE" NEWLAB .DEFLAB(*1) NEWLAB WHLPT ";" TAIL
OUT(JUMP,*2) .DEFLAB(*1)
WHILE := "WHILE" WHLPT ";" ;
WHLPT := "(" BOOLEXP .ERR("F: LOGICAL EXPRESSION ERROR","SCAN",";","R"
OUT(JUMPF,*1) ")" ;
LOOP := ID UNDTST
NEWLAB  /* LOOP EXIT LABEL */
OUT(LDA,**)    /* CODE TO LOAD ADDRESS FOR INIT VALUE */
STKSYM        /* SAVE IDENTIFIER FOR LATER USE */
ITERPRT
OUT(JUMPFT,*1) /* LOOP EXIT CODE */
WHILE / ";" ) .ERR("M: EXPECTED ";",SET)
TAIL
DO(Pop)        /* RETRIEVE LOOP INDEX */
OUT(LDA,**,LD,**) /* CODE TO PREPARE FOR INCREMENT */
OUT(*)         /* AND TEST */
DEFLAB(*1)     /* CODE STACK */
/* DEFINE LOOP EXIT ADDRESS */
```
ITERPT:="=" ERR(LATCH EXP
  ERR("F: INVALID INITIAL INDEX") OUT(SST)
  "TO" /* INITIAL VALUE */
  NEWLAB DEFLAB(*1) /* ITERATION LABEL AND POSSIBLE VALUE */
  MARK /* POSSIBLE CODE REORDERING POINT */
(LATCH(LOCPRM) /* POSSIBLE CODE REORDERING POINT */
 / NEWLAB DEFLAB(*1) /* LABEL FOR TEMPORARY IN DYNAMIC */
  ENTERA(01•DYNAMB) /* STORAGE AND VALUE */
  ENTERL(00•07) /* DATA TYPE ANY */
  DO(A•EIGHT•DYNAMB) /* SPACE FOR INTEGER OR REAL */
  OUT(LDA) DO(LB1) OUT(**) /* CODE TO LOAD DYNAM STOR ADDRESS */
  EXP OUT(STO) /* COMPIL LIMIT EXPRESSION */
  MARK /* NEW REORDERING POINT */
  DEFLAB(*2) /* REDEFINE ITERATION LABEL */
  OUT(LD) DO(LB1) OUT(**) /* CODE TO LOAD EXPRESSION VALUE */
  DO(SwAP,POP) /* DISCARD LABEL AND LEAVE CODE */
) ERR("F: INVALID INDEX LIMIT")
  /* MARKER ON TOP */
("BY" OUT(STCKC) /* COMPIL LOOP INDEX TESTING CODE */
  (="#" OUT(02) IOCHK SAV(SUB) /* AND STACK PROPER OP CODF FOR */
  EMPTY OUT(04) IOCHK SAV(ADD) ) /* INCREMENTING LOOP INDEX */
  MARK /* POSSIBLE CODE REORDERING POINT */
(LATCH(LOCPRM) /* POSSIBLE CODE REORDERING POINT */
 / DO(FOP) SAV /* DISCARD CODE MARK, SAVE TEST CODE */
  NEWLAB DEFLAB(*1) /* LABEL FOR TEMPORARY IN DYNAMIC */
  ENTERA(01•DYNAMB) /* STORAGE AND VALUE */
  ENTERL(00•07) /* DATA TYPE ANY */
  DO(A•EIGHT•DYNAMB) /* SPACE FOR INTEGER OR REAL */
  OUT(LDA) DO(LB1) OUT(**) /* CODE TO LOAD DYNAM STOR ADDRESS */
  EXP OUT(STO)
  DEFLAB(*2) /* REDEFINE ITERATION LABEL */
  OUT(#) /* RESTORE STACKED TEST CODE */
  MARK /* LOOP END CODE FOLLOWING TO BE SAVED*/
  OUT(LD) DO(LB1) OUT(**) /* CODE TO LOAD EXPRESSION VALUE */
  DO(SwAP,POP) /* CODE MARK ON TOP, DISCARD LABEL */
) ERR("F: INVALID INCREMENT")
```
*OUT(#) /* RESTORE INCREMENT UP CODE */
*EMPTY OUT(STCKC+04) /* DEFAULT TEST */
ICCHK = MARK
*OUT(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 ;

LOOPRM := (*ID *OUT(LD*)) /*INUM *OUT(LD+71,*) /*FNUM *OUT(LD+72**) 
( "+" / "-" / "/" / "/" / "/" )
*DO(SETF,BEF) /* FORCE BACKUP IF NO SIMPLE PRIMARY */
/*EMPTY) ;

UND'TST := *SEARCH TSTTBL(00,00)
*DO(BM:"W: UNDECLARED IDENTIFIER") /*EMPTY ;

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

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

BOOL EXP := (*LATCh(IFCLSE) *NEWLAB *OUT(JUMPF+1)
BTERM *NEWLAB *OUT(JUMPF+1)
"ELSE" *DEFLAB(*2) BOOL EXP
*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 / "(" BOOLEAN "")" .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(03) / "<" .SAV(02) / ">=" .SAV(05) / ">" .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,**#) ;

VARIABLE :=ID .SEARCH("O") ( .TSTTBL(00,01) / .TSTTBL(00,02) / .TSTTBL(00,06) / .TSTTBL(00,02) ) .CANCEL ( "(" .OUT(LDA,**) SUBLIST ")" .OUT(INDXR*)) / .EMPTY .OUT(LD,**#) ;

EXP :=.LATCH(IFCLSE) .NEWLAB .OUT(JUMPF,*1) SAE "ELSE" .NEWLAB .OUT(JUMPF,*1)
.DEFLAB(*2) EXP.DEFLAB(*1)
/ SAE ;
SEXF := .LATCH(IFCLOSE) .NEWLAB .OUT(JUMPF*1) SSE
"ELSE" .NEWLAB .OUT(JUMP*1)
.DEFLAB(*2) SEXP .DEFLAB(*1)
/ SSE ;
SSE := STERM $( "//'" STERM .OUT(CAT) ) ;
STEPM := SBSTRNG / .STRING .OUT(LDA,74,*) / .LATCH(SVARABLE) / .LATCH(SFUNCT) ;
SFUNCT := .ID .SEARCH("O") ( .TSTTBL(00,14) / .TSTTBL(00,24) ) .CANCEL CALLPRT ;
SBSTRNG := "SUBSTR(" SEXP "," EXP
( "," EXP / .EMPTY .OUT(LD,71,"0000"))")" .OUT(SUBSTR) ;
SVARABLE := .ID .SEARCH("O") ( .TSTTBL(00,04) / .TSTTBL(00,64) ) .CANCEL
.OUT(LDA,**,**(02,05)) ( "(" SUBLIST ")" .OUT(INDXR,*)
/ .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 COUNT FOR CODE GEN */
LTERM := .ID .SEARCH("1")
( LBLCON / LBLVAR
/ ( "(" .OUT(LDA,05,**(04,01)) EXP $. ( "," EXP")" .OUT(INDXR,*)
/ .EMPTY .OUT(LD,76,**(04,01)) ) ) ) ;
LBLCON := .TSTTBL(00,06) .OUT(LDA,**);
LBLVAR := ( \text{TSTTBL}(00,65) \text{SET}(\text{PEXITT}=\text{PEXITF}) / \text{SET}(\text{PEXITT}=00) \text{TSTTBL}(00,05) )
( "(" \text{OUT}(\text{LDA}/**) \text{SUBLIST} ")" \text{OUT}(\text{INDXR}/**) \\
/ \text{EMPTY} \text{OUT}(\text{LDA}/**) )

\text{COND} := \text{LATCH}(\text{IFCLSE}) \text{NEWLAB} \text{OUT}(\text{JUMP}/**)
\text{BMSTM} ( "ELSE" \text{NEWLAB} \text{OUT}(\text{JUMP}/**)
\text{DEFLAB}(**2) \text{STMENT} \text{DEFLAE}(**1) \\
/ \text{EMPTY} \text{DEFLAB}(**1) ) ;
\text{IFCLSE} := \text{ID} \text{TEST}(\text{SYMBOL}=\text{IFSY}1) \text{CANCEL} \text{BOOLEXP} "\text{THEN}" ;
\text{IDENT} := \text{ID} \text{SEARCH}("0") \text{TSTTBL}(04,00) \text{DO}(\text{BM},"W: UNDECLARED VARIABLE") ;
\text{UNCOND} := \text{LATCH}(\text{GOTOST}) / \text{INOUT} / \text{LATCH}(\text{CLLSTM}) / \text{LATCH}(\text{RTNSTMT}) \\
/ \text{STPSTM} / \text{ASSGNST} / \text{IDENT} ;
\text{STPSTM} := "\text{STOP}" \text{OUT}(\text{STOP}) ;
\text{RTNSTMT} := "\text{RETURN}" ( "(" ( \text{EXP} / \text{SSE} / \text{BPRMRY} ) ")"\text{OUT}(\text{SWP}) \\
/ \text{EMPTY} \text{OUT}(\text{RETURN}) ;
\text{GOTOST} := "\text{GO}" \text{TO} \text{CANCEL} \text{MARK} \text{LTERM}
( \text{TEST}(\text{PEXITT}=\text{PEXITF}) \text{SAV} \text{OUT}(\text{POPUP}/**) \text{OUT}(\text{RETURN}) \\
/ \text{EMPTY} \text{OUT}(\text{JUMPA}) ) ;
\text{ASSGNST} := \text{ASSGN} / \text{BASSGN} / \text{SASSGN} / \text{LASSGN} ;
\text{AASSGN} := \text{LATCH}(\text{ALFTPT}) \text{SAV}(\text{STO})
\$ ( "," \text{ALFTPT} \text{MARK} \text{OUT}(\text{SST}/**) \text{SAV} )
"=" \text{EXP} \text{OUT}(\#) ;
\text{BASSGN} := \text{LATCH}(\text{BLFTPT}) \text{SAV}(\text{STO})
\$ ( "," \text{BLFTPT} \text{MARK} \text{OUT}(\text{SST}/**) \text{SAV} )
"=" \text{BOOLEXP} \text{OUT}(\#) ;
\text{SASSGN} := \text{SLFTPT} \text{SAV}(\text{STO})
\$ ( "," \text{SLFTPT} \text{MARK} \text{OUT}(\text{SST}/**) \text{SAV} )
"=" \text{SEXPR} \text{OUT}(\#) ;
LASIGN := LATCH(LLFTPT) .SAV(STO)
  $ ( "*" LLFTPT .MARK .OUT(SSIT#) .SAV )
  "=" LFTRMR .TEST(PEXITT=ZERO) .ERR("F: INDIRECT LABEL ASSIGNMENT")
  .OUT($) ;

ALFTPT := ID .SEARCH("0")
  (.TSTTLB(00,01) / .TSTTBL(00,02)
   / .TSTTBL(00,03) / .TSTTBL(00,04) ) .CANCEL
  .OUT(LDA,**) SUBPART ;

BLFTPT := ID .SEARCH("0")
  (.TSTTBL(00,03) / .TSTTBL(00,04) ) .CANCEL
  .OUT(LDA,**) SUBPART ;

LLFTPT := ID .SEARCH("0")
  (.TSTTBL(00,05) / .TSTTBL(00,06) ) .CANCEL
  .OUT(LDA,**) SUBPART ;

SLFTPT := LATCH(SVARIABLE) / LSUBSTR ;

LSUBSTR := "SUBSTR(" VARIABLE "\" EXP"
  ( "\" EXP / .EMPTY .OUT(LD,71,"0000") ) "")
  .OUT(SUBSTR) ;

SUBPART := "(" SUBLIST "")" .OUT(INDEXA**) / .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 := IRPLIST
   /* .LATCH(SLFTPT) / .LATCH(ALFTPT) / .LATCH(BLFTPT) */ .OUT(GFT) ;

IRPLIST := "(" .MARK ILIST .SAV ERPLIST ;

ERPLIST := "DO" .ID UNDTST .NEWLAB /* "ERPLIST" IS SIMILAR TO "LOOP" */
   .OUT(LDA,**) .STKSYM
   .SET(IOSW=IOITER) /* SET SWITCH FOR INTERPRT TO TEST */
   INTERPRT
   .SET(IOSW=00) /* RESTORE IOSW */
   .DO(POP) .OUT(LDA,***,LD,**)
   .OUT(#)
   .DEFLAB(*1) ")" ;

OUTPUT := "PUT" ( "STRING" .SAV(00) "(" SLFTPT ")"
   / .EMPTY .SAV(01) )
   "EDIT" .CANCEL .NEWLAB .OUT(FMT,**)
   "(" OLIST ")" .OUT(PUT) .NEWLAB .OUT(JUMP,**)
   .DEFLAB(*2) FMTLIST .DEFLAB(*1) ;

OLIST := OELMNT & ( ".*" OELMNT ) ;

OELMNT := ORPLIST
   /* (EXP / SSE / .LATCH(BVARBLE) ) */ .OUT(EDIT) ;

ORPLIST := "(" .MARK OLST .SAV 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(*) ")" ;

*END

****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****
CLM PROGRA
BT $002
BM "F: COMPILER ABORT
EXIT

MOVI ELATCH "1"

PUSHLB
ID

BF $005
TEST ":"

BEF

STKSYM

MOVE SYM

PRGNME
INTNME

"PLXINT00"

SYM

"$ACTIVE"

SEARCH

"0"

ENTL "0"

I0CHK

MOVE DYNAMP

DYNAMB

TEST "PROCEDURE"

BEF

TEST "MAIN"

BEF

OUT DYNAM

LB1

EVAL

"4"

"1"
MOVE STACK SYMBOL
SEARCH "0"
ENTL "$1000"
$006 CLM COMMEN
BT $006
SET BEF CLM BHDBDY BEF TEST "END" BEF OUT STOP LB1 ENLOC ENTL
ENTA "0"
"6"
"1"
DYNAMP IDF BF $008 CHKSYM BT $009 BM "W: POSSI"
SET $009 B $007 $008 SET BF $007 $007 BEF BLKEXT "1"
RESOLV $005 R $005 BLKBDY MOVE BLNKBY STMLAB

$015 CLM STMENT
BF $015 BT BLKBDY
$012 BT SET BF $012

$018 CLM COMMEN
BT $018 SET BF $017

$019 LATCH LABEL
BT $019

$021 CLM ENDTST
BF $021 SETF R $021

$020 B $020

$021 CLM CONDST
BF $021

$022 CLM BSCSTM
BF $022

$020 BT $020

$020 BF $020

$029 LATCH BLOCK
BF $029

STATEMENT"
$029  CLM    UNCOND
       BF    $030
       B    $027

$030  SET    BF    $027
       BF    $026
       TEST  $026
       BEF

$026  R      TEST  "/\"
       BF    $033
       SCAN  DOSYM

$033  R      LABEL  ID
       BF    $035
       TEST  ":\"
       BEF
       MOVE  SYMBOL
       STKSYM
       OUT
       EVAL  "5"
       "0"
       A    ONELNG
       MOVE  STACKT
       SEARCH  "1"
       ENTA  "1"
       ENTL  "0"
       "5"

$035  R      ENDSTM  LATCH
       BF    $037
       SETF  $036
       B    $036

$037  SET    BF    $036
       BEF

$036  R      ENDSTM  TEST  "END"
       BF    $040
       SETF  $040
       BEF
       B    $039

$040  SET    CLM    BHDBDY
B SET $063

BEF $063

TEST ";"

BEF MOVE STMLAB SYMBOL

SEARCH "0"

ENTA "7"

MOVE ZERO ADECNT

CLM TARGCN

BF $070

TEST "DECLARE"

BF $072

CLM ARGDEC

BEF $073

TEST ";

BF $075

CLM ARGDEC

BEF $075

BT $073

SET

BEF TEST ";"

BEF $071

SET $071

BF $071

BEF COMP ARGCNT ADECNT "2"

BT $077

BM "W: INCORRECT ARG"

SET $069

B $069

BEF $070

SET $070

BF $069

BEF $072

CLMelah

BF $072

TEST "END"

BEF OUT RETURN

LB1 ENTLOC

ENTL

"0"

"6"

ENTA "1"

DYNAMP "1"

BLKEXT POP

MOVE SYMBOL

STACKT

POP

MOVE SYMBOL

LEVNO

POP

MOVE SYMBOL

DYNAMP

POP

MOVE SYMBOL

DYNAMP

ID

BT $079

BM "W: EXPECTED CLOS

SET $079

CHKSYM

BT $080
<table>
<thead>
<tr>
<th>ENTAL</th>
<th>TYPE</th>
<th>&quot;0&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>BF</td>
<td>$096</td>
<td>$095</td>
</tr>
<tr>
<td>TEST</td>
<td>&quot;ENTRY&quot;</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$095</td>
<td>$095</td>
</tr>
</tbody>
</table>
| TEST  | "RETURNS("
| BF    | $099   |      |
| CLM   | ATTRIB |      |
| BEF   |        |      |
| TEST  | ")"    |      |
| BF    | $098   |      |
| A     | OCTL20 |      |
| ENTAL | "0"    |      |
| BF    | $098   |      |
| ENTL  | "0"    |      |
| "4"   |        |      |
| BEF   | $095   |      |
| R     | $095   |      |
| CLLSTM| ID     |      |
| BF    | $102   |      |
| COMP  | SYMBOL |      |
| CALLCO| "2"    |      |
| BEF   | $108   |      |
| ID    |        |      |
| BEF   | $108   |      |
| CANCEL| SEARCH |      |
| BEF   | $109   |      |
| CLM   | CALLPR |      |
| BEF   | $110   |      |

CALLPR MARK
OUT  LDA
EVAL "5"
"0"
OUT  JUMPA
SAVE SET
BF  $104
MOVE SYMBOL
STMLAB
PUSHLB
OUT  LDA
OUT  "6"
LB1  EVAL
"4"
"1"
OUT  LD
OUT  IOCHK
MOVE STMLAB
SYMBOL
SEARCH "0"
TSTTBL "0"
"8"
BF  $108
B   $107
TSTTBL "0"
"9"
BF  $109
B   $107
TSTTBL "0"
"#"
BF  $110
B   $107
TSTTBL "0"
BF $111
B $107
TSTTLB "0"
B $106
OUT LD
OUT IOCK
B $105
TSTTLB "0"
B $115
BF $114
TSTTLB "0"
B $114
BF $116
B $114
TSTTLB "0"
B $117
B $114
TSTTLB "0"
B $118
B $114
TSTTLB "0"
B $114
BF $105
MARK OUT LD OUT "1" EVAL "4" "1"
SAVE PUSHLB

LDA OUT "1"
LB1
EVAL
"4"
"1"

RESTOR OUT LD OUT "Z" OUT "0004"
OUT ADD OUT STO OUT LD OUT "1"
LB1
ENTLOC
ENTL

BF $116
B $114
TSTTLB "0"
B $117
B $114
TSTTLB "0"
B $118
B $114
TSTTLB "0"
B $114
BF $105
MARK OUT LD OUT "1" EVAL "5"
"0"

"6"
ENTL "0"
"1"
OUT "0000"
POP SET

"0"

BF $111
B $107
TSTTLB "0"
B $106
OUT LD
OUT IOCK
B $105
TSTTLB "0"
B $115
BF $114
TSTTLB "0"
B $114
BF $116
B $114
TSTTLB "0"
B $117
B $114
TSTTLB "0"
B $118
B $114
TSTTLB "0"
B $114
BF $105
MARK OUT LD OUT "1" EVAL "5"
"0"

"6"
ENTL "0"
"1"
OUT "0000"
POP SET

"0"

BF $111
B $107
TSTTLB "0"
B $106
OUT LD
OUT IOCK
B $105
TSTTLB "0"
B $115
BF $114
TSTTLB "0"
B $114
BF $116
B $114
TSTTLB "0"
B $117
B $114
TSTTLB "0"
B $118
B $114
TSTTLB "0"
B $114
BF $105
MARK OUT LD OUT "1" EVAL "5"
"0"

"6"
ENTL "0"
"1"
OUT "0000"
POP SET

"0"

BF $111
B $107
TSTTLB "0"
B $106
OUT LD
OUT IOCK
B $105
TSTTLB "0"
B $115
BF $114
TSTTLB "0"
B $114
BF $116
B $114
TSTTLB "0"
B $117
B $114
TSTTLB "0"
B $118
B $114
TSTTLB "0"
B $114
BF $105
MARK OUT LD OUT "1" EVAL "5"
"0"

"6"
ENTL "0"
"1"
OUT "0000"
POP SET

"0"
ENTL "0" "6"
R CLIPRM OUT FLAG PARMCN SYMBOL
MOVE PARMCN SYMBOL
STKSYM MOVE ZERO PARMCN SYMBOL
MOVE PARMCN SYMBOL
STKSYM CLM PARMCR
BF $124
B $123
SET
BF $123
BF $122
POP
SEARCH "0"
TSTT6L "0"
BF $129
B $128
TSTT6L "0"
BF $130
B $128
TSTT6L "0"
BF $131
B $128
TSTT6L "0"
BF $128
BF $127
TSTT6A "7"
PARAMCN
BT $133
BM "W: INCORRECT PAR
SET
B $126
BF $126
BEF
POP
MOVE SYMBOL PARAMCN
PARAMCR TEST "(
BF $136
CLM PARM
BEF
TEST ","
BF $139
CLM PARM
BEF
TEST ")"
BF $137
SET
BEF
TEST PARM CLM PARMID
B $143
B $142
CLM PARMEX
BF $142
BF $141
A ONE
PARMCN
PARAM LATCH ALFTPT
BF
B
$146
LATCH
BF
B
$147
LATCH
BF
8
$148
LATCH
BF
B
$149
LATCH
BF
$145
R
PROCHK ID
BF
TEST
BF

$146
$145
BLFTPT
$147
$145
SLFTPT
$148
$145
LLFTPT
$149
$145
PROCHK
$145

DEF
BEF
R
$152
PARMEX PUSHLB
LBL
SEARCH
ENTA
il56 .

DYNAMB
ENTL

HQ II

A

EIGHT
DYNAMB
LDA

II7 It

OUT
EVAL

$152

$154
$153

TSTTbL

BF
CANCEL
CLM
BEF
B
$157

TSTTbL

BF
CANCEL
CLM

$154

CLM
BF

$153
$153

E

5162
"0"
"0"
II8II
$157

Î163

GPROC

$164

$156

$161

HQ II
•I +II

$160

$156

GPROC

IPROC

II ^ II
" 0 "

OUT
EVAL

II ^ II

SETF

BEF
B
SET
BF
BEF
SEARCH

II1II
II II

CLM
BF
B
CLM
BF
B
CLM
BF
BF
OUT
R
MARK
OUT
OUT

LDA
II5II
HQ II

EXP
$162
$161
SSE
$163
$161
LTERM
$164
$161
BPRMRY
$161
$160
STO
LDA
II2^ II

«r
a\


EVAL "4"
"1"
SAVE
PUSHLB
LB1
SEARCH "1"
ENTL "0"
"1"
ENTA "1"
DYNAMB
OUT LDA
EVAL "5"
"0"
OUT LDA
EVAL "5"
"0"
RESTOR
OUT STO
A FOUR
DYNAMB
PUSHLB
LB1
SEARCH "1"
ENTA "1"
DYNAMB "0"
"1"
OUT LDA
EVAL "5"
"0"
OUT LDA
EVAL "5"
"0"
RESTOR
OUT STO
A FOUR
DYNAMB
PUSHLB
LB1
SEARCH "1"
ENTA "1"
<table>
<thead>
<tr>
<th>Command</th>
<th>Operation</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTLOC</td>
<td>DYNAMB</td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td>ENTL</td>
<td>&quot;0&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;1&quot;</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>LDA</td>
<td></td>
</tr>
<tr>
<td>EVAL</td>
<td>&quot;5&quot;</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>FOUR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DYNAMB</td>
<td></td>
</tr>
<tr>
<td>PUSHLB</td>
<td>LDA</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;1&quot;</td>
<td></td>
</tr>
<tr>
<td>LB1</td>
<td>EVAL</td>
<td>&quot;4&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;1&quot;</td>
<td></td>
</tr>
<tr>
<td>RESTOR</td>
<td>LD</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;2&quot;</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;0004&quot;</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>ADD</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>STO</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>LD</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;1&quot;</td>
<td></td>
</tr>
<tr>
<td>LB1</td>
<td>ENTLOC</td>
<td></td>
</tr>
<tr>
<td>ENTL</td>
<td>&quot;0&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;6&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;1&quot;</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;0000&quot;</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>STO</td>
<td></td>
</tr>
<tr>
<td>$168</td>
<td>R</td>
<td></td>
</tr>
</tbody>
</table>
| DECLAR   | TEST                    | "DECLARE"
|         | BF                      | $171     |
| MOVI     | ELATCH                  |          |
| $172     | BT                      |          |
|         | CLM                     | DECL1    |
|         | BT                      | $172     |
|         | BM                      | "F"      |
|         | SCAN                    | ";"      |
|         | SET                     |          |
| $172     | TEST                    | ";"      |
|         | BF                      | $175     |
|         | CLM                     | DECL1    |
|         | BT                      | $175     |
|         | BM                      | "F"      |
|         | SCAN                    | ";"      |
|         | SET                     |          |
| $175     | BT                      | $172     |
|         | SET                     |          |
|         | TEST                    | ";"      |
|         | BEF                     |          |
| $171     | R                       |          |
|         | DECL1                   |          |
|         | CLM                     | IDELMN   |
|         | BT                      | $178     |
|         | B                       | $177     |
| $178     | CLM                     | IDGROU   |
|         | BT                      | $177     |
| $177     | R                       |          |
|         | IDGROU                  | "("      |
|         | BF                      | $181     |
|         | CLM                     | LISTPT   |
|         | BEF                     |          |
| $181     | R                       |          |
|         | LISTPT                  | ID       |
|         | BF                      | $183     |
|         | STKSYM                  |          |
|         | CLM                     | ARRYPT   |
|         | BF                      | $185     |
|         | MOVE                    | DIMCNT   |
|         | SYMBOL                  |          |
STKSYM
CLM  IDLIST
BEF
POP
MOVE  SYMBOL
POP
MOVE  SYMBOL
CLM  IDLIST
BEF
B  $184
$185
CLM  IDLIST
BF  $184
POP
SEARCH  "1"
CLM  IDSEM
BEF
A  LENGTH
BF  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
Test  ")"
BEF
TEST  "RETURNS"
BF  $190
$191
TEST  "")
BF  $191
BEF
TEST  ""
BF  $191
BEF
TEST  "RETURNS"
BF  $190
$195
TEST  "FIXED"
BF  $196
MOVI  LENGTH
LEVNO
BF  $190
$196
TEST  "FLOAT"
BF  $197
MOVI  LENGTH
LEVNO
BF  $190
$197
TEST  "LOGICAL"
BF  $198
MOVI LENGTH LEVNO
MOVI TYPE "3"
B $190
$198 TEST "LABEL"
BF $190
MOVI LENGTH LEVNO
MOV TYPE "5"

$198 R ARRYPT TEST "(" BF $201
MOVI DIMCNT "0"
CLM BDPRLS BEF
TEST ")"
BEF

$201 R BDPRLS MARK CLM BNDPAI BF $203
SAVE TEST "," BF $206
MARK RESTOR CLM BNDPAI BEF
SAVE

$204 BT $204
SET BEF

$203 R BNDPAI A ONE $212

DIMCNT
CLM INTBND
BF $208
TEST ":;"
BF $210
RESTOR CLM INTBND
BF
RESTOR B $209

SET BF $209
OUT "Z0001"
RESTOR

BEF

BF $204
OUT "Z0001"

SEARCH "0"
TSTTL "0"
"1"

BT $214
BM "W: INVALID ARRAY"

MARK "5"
EVAL "0"
"0"

SAVE B $212
INUM BF $212
MARK OUT "Z"
OUTSYM SAVE R

150
COMP SYMBOLO
DOSYM "2"
BEF MOV I ELATCH "1"
TEST ";" $231
BF $230 TAIL
BEF
B $230

$231 CLM CASE
B $232 LOOP
BF $230

$232 DOWHIL
B $230

$233 CLM EXP
BT $241
BM "F: BAD CASE EXP"
SCAN ";"
CLM TAIL
R

$230 TAIL

$237 CASE
TEST "CASE"
BF $240
PUSHLB
BF $240
PUSHLB
MOV I ELATCH "1"
PUSHLB

$231 CLM BEF
OUT LD
OUT "Z0065"
OUT MULT
LB1
SEARCH "1"
ENTL "0"

$232 CLM BEF
OUT LD
EVAL

$233 CLM BEF
OUT ADD
OUT JUMPA
PUSHLB
PUSHLB

$230 CLM BEF
OUT ADD
OUT JUMPA
PUSHLB
PUSHLB

$232 CLM BEF
OUT ADD
OUT JUMPA
PUSHLB
PUSHLB
CLM  IOCHK  RESTOR
BEF  MARK  OUT  LD
MARK  OUT  LB1  EVAL
OUT  ADD
SAVE  BEF  "5"
MARK  "0"

LATCH  LOOPRM  SWAP
BF  POP  $271  $273  BM  "F: INVALID INCRE"
RF  $270  $270  BT  $273
EVAL  "$0"

POP  SAVE  $270  RESTOR  "$0"
SAVE  $273  B  $265
PUSHLB  $266  SET  "$0"
LB1  BF  $265
ENTLOC  OUT  STCKC  "$0"
ENTL  OUT  "$4"
ENTL  OUT  "$0001"
"0"
"6"
"1"
DYNAMB
"6"
"7"
"0"
"0"
EIGHT
"2"
DYNAMB
ADD
OUT  LDA  $265  BEF
LB1  OUT  SST  "$4"
EVAL  OUT  JUMP  "$1"
"5"
"0"
CLM  EXP  "$4"
BF  $270  "$1"
OUT  STO  SAVE
OUT  "$0"
LB2  "0"
$259  R  BF  "$6"
ENTLOC  IOCHK  COMP  IOITER
ENTL  "$2"
ENTL  "$6"
ENTL
<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT</td>
<td>JUMPT</td>
<td>$289</td>
<td>BF</td>
</tr>
<tr>
<td>LB2</td>
<td></td>
<td>$286</td>
<td>B</td>
</tr>
<tr>
<td>EVAL</td>
<td></td>
<td>$289</td>
<td>TEST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$290</td>
<td>&quot;/&quot;</td>
</tr>
<tr>
<td>RESTOR</td>
<td>SET</td>
<td>$286</td>
<td>BF</td>
</tr>
<tr>
<td></td>
<td>BF</td>
<td>$285</td>
<td>$275</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>$275</td>
<td>LOOPRM</td>
</tr>
<tr>
<td>INUM</td>
<td>BF</td>
<td>$281</td>
<td>OUT</td>
</tr>
<tr>
<td></td>
<td>LD</td>
<td>$284</td>
<td>BEF</td>
</tr>
<tr>
<td>$281</td>
<td></td>
<td>$285</td>
<td>SET</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$284</td>
<td>B</td>
</tr>
<tr>
<td>$279</td>
<td>R</td>
<td>$284</td>
<td>UNDTST</td>
</tr>
<tr>
<td>TSTTBL</td>
<td>&quot;0&quot;</td>
<td>$284</td>
<td>SEARCH</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$282</td>
<td></td>
<td>$284</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$293</td>
<td>BM</td>
</tr>
<tr>
<td>$282</td>
<td>FNUM</td>
<td>$284</td>
<td>&quot;W: MULTIPLE DECLA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ENDING TEST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$293</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$297</td>
<td>BM</td>
</tr>
<tr>
<td>$280</td>
<td></td>
<td>$297</td>
<td>&quot;W: UNDECLARED IDE</td>
</tr>
<tr>
<td></td>
<td>TEST</td>
<td>$297</td>
<td>&quot;END&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ENDING TEST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$302</td>
<td>BF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$302</td>
<td>ID</td>
</tr>
<tr>
<td>Line</td>
<td>Instructions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>--------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$323</td>
<td>LATCH BVARBL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$324</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>$322</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$324</td>
<td>LATCH BFNCT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$325</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>$322</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$325</td>
<td>CLM RELATN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$326</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>$322</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$326</td>
<td>TEST &quot;(&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$322</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td>BOOLEAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td>TEST &quot;(&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BT</td>
<td>$322</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EM</td>
<td>&quot;w: EXPE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SET</td>
<td>$322</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$322</td>
<td>RELATN SAE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$330</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td>RELOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td>CLM SAE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td>OUT STCKC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESTOR</td>
<td>$329</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>$329</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$330</td>
<td>CLM SSE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$329</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td>RELOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td>CLM SSE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td>OUT COMPC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESTOR</td>
<td>$329</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>$341</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$325</td>
<td>R BFNCT ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$342</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$333</td>
<td>SEARCH &quot;0&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TSTTBL</td>
<td>&quot;0&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;=&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$335</td>
<td>BEF $334</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$334</td>
<td>BEF CANCEL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td>CALLPR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$333</td>
<td>R BVALUE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td>&quot;T&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;F&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$337</td>
<td>B $337</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>LD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;T&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>$337</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;F&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$337</td>
<td>R BVARBL ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$341</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$341</td>
<td>SEARCH &quot;0&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TSTTBL</td>
<td>&quot;0&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;3&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$343</td>
<td>BEF $342</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$342</td>
<td>B $342</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$343</td>
<td>TSTTBL &quot;0&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;T&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CANCEL TEST "(" BF $346 OUT LDA $351 EVAL "5" "0" CLM SUBL15 BEF TEST ")" $352 BEF OUT INDXR OUTSYM B $345 $346 SET BF $345 $353 OUT LD EVAL "5" "0" $346 $341 BEF R $348 RELOP TEST ";" BF $349 MARK OUT "1" SAVE B $348 $349 TEST ";<=" BF $350 MARK OUT "3" SAVE B $348 $350 TEST ";<" BF $351 MARK $351 OUT "2" SAVE B $348 OUT ">=" BF $352 MARK OUT "5" CLM TERM BEF OUT "<=" BF $357 MARK OUT "<" BF $357 MARK CLM TERM BEF
CLM  TERM
BEF
OUT  ADD
$362
B  $362
TEST  
BF  $362
CLM  TERM
BEF
OUT  SUB
$362
BT  <$361
AFUNCT  ID
$371  R
BF
TERM
CLM  PRIMRY
BF  $366
TEST  
$367
BF  $369
CLM  PRIMRY
BEF
OUT  MULT
$368
B  $368
TEST  
$369
BF  $368
CLM  PRIMRY
BEF
OUT  DIV
$368
BT  $367
SET  
BEF
$366  R
PRIMRY
TEST  
$366
BF  $372
CLM  EXP
BEF
TEST  
$366
BF  $371
ENTLC
ENTL
"0"
"6"
B $397
CLM $396
BF $399
$397
CLM SAE
BF $396
$396
R LATCH IFCLSE
BF $400
PUSHLB OUT JUMPF
LB1 EVAL
"4"
"1"
CLM SSE
BEF TEST "ELSE"
BEF PUSHLB OUT JUMP
LB1 EVAL
"4"
"1"
LB2 ENTLOC ENTLC
ENTL
"0"
"6"
CLM SEXP
BEF LB1 ENTLOC ENTLC
ENTL
"0"
R CLM SSE
BF $399
$400
CLM SSE
BF $399
$399
R CLM STERN
BF $403
$404 TEST "//"
BF $406
CLM STERN
BF $404
OUT CAT
$406 BT $404
SET BEF
$403 R
CLM SBSTRN
BF $408
$408 STRTST
BF $409
OUT LDA
OUT "("
OUTSYM E $407
$409 LATCH SVARBL
BF $410
R $407
$410 LATCH SFUNCT
BF $407
$407 R
SFUNCT ID
BF $413
SEARCH "0"
TSTTBL "0"
BF $415
$415
TSTTBL $414
"0"
"0"

$414
BEF
CANCEL
CLM CALLPR
BEF

$413
R
SASTRN TEST "SUBSTR(" BF
$418
CLM SEXP
BEF
TEST ","
BEF
CLM EXP
BEF
TEST ","
BEF
CLM EXP
BEF
E $419

$420
SET $428
BF
OUT LD
OUT "Z"
OUT "0000"

$419
BEF
TEST ")"
BEF
OUT SUBSTR

$418
R
SVARBL ID
BF $423
SEARCH "," TSTTBL "4"

BF $425
B $424
BF $424
BEF CANCEL
CLM CALLPR
BEF

$424
BEF
CANCEL
CLM CALLPR
BEF

$428
BF $428
CLM SUBLIS
BEF

$427
B $427
BF $427
BEF

$431
BF $431
CLM EXP
BEF

$434
BF $434
CLM EXP
BEF

ONE
DIMCNT

$431
BF $431
CLM EXP
BEF

ONE
DIMCNT
<table>
<thead>
<tr>
<th>$435</th>
<th>BT</th>
<th>$432</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET</td>
<td>TEST</td>
<td>&quot;&quot;)</td>
</tr>
<tr>
<td>BEF</td>
<td>BEF</td>
<td>OUT</td>
</tr>
<tr>
<td>POP</td>
<td>OUT</td>
<td>INDXR</td>
</tr>
<tr>
<td>SEARCH</td>
<td>OUTSYM</td>
<td>B</td>
</tr>
<tr>
<td>&quot;$0&quot;</td>
<td></td>
<td>$442</td>
</tr>
<tr>
<td>TSTTA</td>
<td>&quot;7&quot;</td>
<td></td>
</tr>
<tr>
<td>DIMCNT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BT</td>
<td>$435</td>
<td></td>
</tr>
<tr>
<td>BM</td>
<td>OUT</td>
<td></td>
</tr>
<tr>
<td>&quot;$: INCOR&quot;</td>
<td>BF</td>
<td>&quot;4&quot;</td>
</tr>
<tr>
<td>MOVE</td>
<td>DIMCNT</td>
<td></td>
</tr>
<tr>
<td>SYMBOL</td>
<td>B</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>$442</td>
<td>BF</td>
<td>$438</td>
</tr>
<tr>
<td>$431</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>LTERM</td>
<td>ID</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$437</td>
<td></td>
</tr>
<tr>
<td>SEARCH</td>
<td>LBLCON</td>
<td></td>
</tr>
<tr>
<td>&quot;1&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td>LBLCON</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$439</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>$438</td>
<td></td>
</tr>
<tr>
<td>$439</td>
<td>CLM</td>
<td></td>
</tr>
<tr>
<td>LBLVAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$440</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>$438</td>
<td></td>
</tr>
<tr>
<td>$440</td>
<td>TEST</td>
<td></td>
</tr>
<tr>
<td>&quot;(&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$443</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>LDA</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;5&quot;</td>
<td></td>
</tr>
<tr>
<td>EVAL</td>
<td>&quot;4&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;1&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td>EXP</td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$444</td>
<td>TEST</td>
<td></td>
</tr>
<tr>
<td>&quot;,&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$446</td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td>EXP</td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$446</td>
<td>BT</td>
<td></td>
</tr>
<tr>
<td>SET</td>
<td>$444</td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

165
$466 R UNCOND "A: UNDEC"
BM "B: $467"
BF $468
BM B
BF $467
$466 CLM INOUT $469
BF $469
BM B
BF $467
$469 LATCH CLLSTM $470
BF $470
BM B
BF $467
$470 LATCH RTNSSTM $471
BF $471
BM B
BF $467
$471 CLM STPSTM $472
BF $472
BM B
BF $467
$472 CLM ASSGN3 $473
BF $473
BM B
BF $467
$473 CLM IDENT $474
BF $474
BM B
BF $467
$467 R STPSTM "STOP"
BF $476
OUT STOP
$476 R RTNSSTM "RETURN"
BF $478
TEST "("
BF $480
CLM EXP
BF $482
$482 CLM SSE $483
ASSGN3 CLM AASSGN
BF $483
B $481
CLM BPRPHY
BF $481
B $479
OUT SWP
B $479
OUT RETURN
B $479
OUT RETURN
B $487
GOTOST TEST "GO"
BF $487
B $487
OUT RETURN
B $487
GOTOST TEST "TO"
BF $487
B $487
OUT RETURN
B $487
GOTOST TEST "GO"
BF $487
B $487
OUT RETURN
B $487
GOTOST TEST "TO"
BF $487
B $487
OUT RETURN
B $487
GOTOST TEST "GO"
BF $487
B $487
OUT RETURN
B $487
GOTOST TEST "TO"
BF $487
B $487
OUT RETURN
B $487
GOTOST TEST "GO"
BF $487
B $487
OUT RETURN
B $487
GOTOST TEST "TO"
BF $487
B $487
OUT RETURN
B $487
GOTOST TEST "GO"
BF $487
B $487
OUT RETURN
B $487
GOTOST TEST "TO"
BF $487
B $487
OUT RETURN
B $487
GOTOST TEST "GO"
BF $487
B $487
OUT RETURN
B $487
GOTOST TEST "TO"
BF $487
B $487
OUT RETURN
B $487
GOTOST TEST "GO"
BF $487
B $487
OUT RETURN
B $487
GOTOST TEST "TO"
BF $487
B $487
OUT RETURN
B $487
GOTOST TEST "GO"
BF $487
B $487
OUT RETURN
B $487
GOTOST TEST "TO"
<table>
<thead>
<tr>
<th>BF</th>
<th>$492</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>$491</td>
</tr>
<tr>
<td>CLM</td>
<td>BASSGN</td>
</tr>
<tr>
<td>BF</td>
<td>$493</td>
</tr>
<tr>
<td>B</td>
<td>$491</td>
</tr>
<tr>
<td>CLM</td>
<td>SASSGN</td>
</tr>
<tr>
<td>BF</td>
<td>$494</td>
</tr>
<tr>
<td>B</td>
<td>$491</td>
</tr>
<tr>
<td>CLM</td>
<td>LASGN</td>
</tr>
<tr>
<td>BF</td>
<td>$491</td>
</tr>
<tr>
<td>R</td>
<td>$491</td>
</tr>
<tr>
<td>AASSGN</td>
<td>LATCH</td>
</tr>
<tr>
<td>BF</td>
<td>$497</td>
</tr>
<tr>
<td>MARK</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>STO</td>
</tr>
<tr>
<td>SAVE</td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td>&quot;,&quot;</td>
</tr>
<tr>
<td>BF</td>
<td>$500</td>
</tr>
<tr>
<td>CLM</td>
<td>ALFTPT</td>
</tr>
<tr>
<td>BEF</td>
<td></td>
</tr>
<tr>
<td>MARK</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>SST</td>
</tr>
<tr>
<td>RESTOR</td>
<td></td>
</tr>
<tr>
<td>SAVE</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$493</td>
</tr>
<tr>
<td>SET</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>$502</td>
</tr>
<tr>
<td>TEST</td>
<td>&quot;,&quot;</td>
</tr>
<tr>
<td>BEF</td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td>EXP</td>
</tr>
<tr>
<td>BEF</td>
<td></td>
</tr>
<tr>
<td>RESTOR</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$492</td>
</tr>
<tr>
<td>LATCH</td>
<td>BLFTPT</td>
</tr>
<tr>
<td>BF</td>
<td>$502</td>
</tr>
<tr>
<td>MARK</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>STO</td>
</tr>
<tr>
<td>BF</td>
<td>$505</td>
</tr>
<tr>
<td>CLM</td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td></td>
</tr>
<tr>
<td>MARK</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>SST</td>
</tr>
<tr>
<td>RESTOR</td>
<td></td>
</tr>
<tr>
<td>SAVE</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$503</td>
</tr>
<tr>
<td>CLM</td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td></td>
</tr>
<tr>
<td>MARK</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>SST</td>
</tr>
<tr>
<td>RESTOR</td>
<td></td>
</tr>
<tr>
<td>SAVE</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$508</td>
</tr>
<tr>
<td>CLM</td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td></td>
</tr>
<tr>
<td>MARK</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>SST</td>
</tr>
<tr>
<td>RESTOR</td>
<td></td>
</tr>
<tr>
<td>SAVE</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$510</td>
</tr>
<tr>
<td>CLM</td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td></td>
</tr>
<tr>
<td>MARK</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>SST</td>
</tr>
<tr>
<td>RESTOR</td>
<td></td>
</tr>
<tr>
<td>SAVE</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$508</td>
</tr>
<tr>
<td>CLM</td>
<td>SEXP</td>
</tr>
<tr>
<td>Instruction</td>
<td>Address</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>$507</td>
<td>LA5SGN</td>
</tr>
<tr>
<td>$507</td>
<td>R</td>
</tr>
<tr>
<td>$513</td>
<td>LATCH</td>
</tr>
<tr>
<td>$512</td>
<td>BF</td>
</tr>
<tr>
<td>$513</td>
<td>BF</td>
</tr>
<tr>
<td>$515</td>
<td>BT</td>
</tr>
<tr>
<td>$516</td>
<td>RESTOR</td>
</tr>
<tr>
<td>$512</td>
<td>R</td>
</tr>
<tr>
<td>$512</td>
<td>ALFPTPT ID</td>
</tr>
<tr>
<td>$520</td>
<td>TSTTBL</td>
</tr>
<tr>
<td>$521</td>
<td>BF</td>
</tr>
<tr>
<td>$522</td>
<td>TSTTBL</td>
</tr>
<tr>
<td>$519</td>
<td>BEF</td>
</tr>
<tr>
<td>$519</td>
<td>BEF</td>
</tr>
<tr>
<td>$519</td>
<td>BEF</td>
</tr>
<tr>
<td>$518</td>
<td>R</td>
</tr>
<tr>
<td>$517</td>
<td>CLM</td>
</tr>
<tr>
<td>$517</td>
<td>CLM</td>
</tr>
<tr>
<td>$517</td>
<td>CLM</td>
</tr>
<tr>
<td>$517</td>
<td>CLM</td>
</tr>
<tr>
<td>$517</td>
<td>CLM</td>
</tr>
<tr>
<td>$516</td>
<td>RESTOR</td>
</tr>
<tr>
<td>$512</td>
<td>R</td>
</tr>
<tr>
<td>$512</td>
<td>ALFPTPT ID</td>
</tr>
<tr>
<td>$520</td>
<td>TSTTBL</td>
</tr>
<tr>
<td>$521</td>
<td>BF</td>
</tr>
<tr>
<td>$522</td>
<td>TSTTBL</td>
</tr>
<tr>
<td>$519</td>
<td>BEF</td>
</tr>
<tr>
<td>$519</td>
<td>BEF</td>
</tr>
<tr>
<td>$519</td>
<td>BEF</td>
</tr>
<tr>
<td>$518</td>
<td>R</td>
</tr>
<tr>
<td>$517</td>
<td>CLM</td>
</tr>
<tr>
<td>$517</td>
<td>CLM</td>
</tr>
<tr>
<td>$517</td>
<td>CLM</td>
</tr>
<tr>
<td>$517</td>
<td>CLM</td>
</tr>
<tr>
<td>$517</td>
<td>CLM</td>
</tr>
</tbody>
</table>
SAVE
TEST "(" ENTLOC ENTL "0"
BEF CLM SLFTPT BEF TEST ")"
BEF SET $550 BF $550
MARK OUT "Z" $549 R
SAVE BEF TEST "EDIT"
BEF CANCEL
PUSHLB OUT FMT $557 BT $555
LB1 EVAL "4" $554 R
"1" IELMNT CLM IRPLIS
RESTART
TEST "(" SET $559 LATCH SLFTPT
BEF CLM ILIST BEF $558 B
TEST ")" $562 LATCH ALFTPT
BEF PUSHLB OUT JUMP $563 LATCH BLFTPT
LB1 EVAL "4" $561 BF $558
"1" OUT GET
LB2  $558 R IRPLIS TEST "("
$589  R
FMTITE  CLM   CTRLFM
BF      $594   BF    $607  CLM   SPEC
B       $593   B     $606

$594  CLM   DATAFM
BF      $593   BF    $606

$593  R
CTRLFM  TEST  "$X"
BF      $597   SET    "$E"
OUT     "$0"
CLM     SPEC
BEF
B       $596   PER    "$9"

$597  TEST  "PAGE"
BF      $598   BF    $609
OUT     "$1"
B       $596   OUT    "$9"

$598  TEST  "SKIP"
BF      $599   CLM    SPEC
OUT     "$2"
CLM     SPEC
BEF
B       $596   BEF

$601  SET    "$0001"
BF      $600   BF    $604
OUT     "$0001"
CLM     SPEC
BEF

$600  BEF
E       $596   SPEC    TEST  "("$613  R

$599  TEST  "COL"
BF      $596   INUM   BEF
OUT     "$3"
OUTSYM  TEST  "")"
BEF

$596  R
DATAFM  TEST  "A"
BF      $605   BEF
OUT     "$8"

END
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.

ASSEMBLING 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
PSILST EQU 239 POST LISTING OPTION FIELD
INSTCT EQU 209 INSTRUCTION COUNT FIELD
GENLOC EQU 215 BEGINNING CODE GENERATION POINTFR
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 ARFA
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 IFSFR 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 DCW :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 =1B4,SYMEND INITIALIZE
MCw SYMEND,IR14
LCA =1C77+1+X14 BLOCK
LCA NEWSYM+4+X14 LIST
MCw SYMEND,CRLKLT INIT CURRENT BLOCK LIST POINTER
MCw NEWSYM,IR15 INIT SYMTAB POINTER
MCw SYMEND,SVSIME SAVE INITIAL SYMTAB END AS START OF BLOCK
MCw :OUTPUT+132 CLEAR PRINT
MCw OUTPUT+132 BUFFER
MCw =1C21 CARRIAGE CONTROL
SI INPUT+80 RESTORE LOST ITEM MARK ON INPUT BUFFER
MCw :SKIP PRINT+57, SKIP TO TOP OF PAGE AND INIT INPUT BUFFER
MCw +INPUT+IR6
B FIRST
MCw OUTPUT+132 CLEAR PRINT
MCw OUTPUT+132 BUFFER
INSTRUCTION FETCHING

FETCH3 BA =1B3,IR1
FETCH BA =1B1,INSTCT
FIRST BS IR7-1
MRSD 0+X1,IR7
SAR IR1
BA IR7
SAR IR7
MC w TVEC+3*IR7,IR14
MCw TVEC+3*IR7,IR14
B 0+X14
TVEC EQU *
REP 10
DSA ERROR
DSA MOVE
DSA ADD
DSA MULT
DSA SETF
DSA BEF
DSA EXITI
DSA RESOLV
DSA BRANCH
DSA BRNCHT
DSA BRNCHF
DSA BM
DSA SET
DSA SCANI
DSA ERROR
DSA MOVLT
DSA ERASE
DSA COMP
DSA SRCHP
DSA TSTTBL
DSA TSTTBA
DSA LATCH
BRANCH, BRANCH TRUE, AND BRANCH FALSE PRIMITIVES
01860 BRANCH MR1D 0+X1,2
01870 B FETCH
01880 BRNCHT BCE FETCH3,TF,F
01890 B BRANCH
01900 BRNCHF BCE BRANCH,TF,F
01910 B FLCH3
01920******************************************************************************************
01930* THE SET AND SETF PRIMITIVES SET THE TRUF-FALSE INDICATOR.
01940* ******************************************************************************
01950* 01960******************************************************************************************
01970 SETF MCW :F,TF
01980 B FETCH
01990 SET MCW :T,TF
02000 E FETCH
02010******************************************************************************************
02020* 02030* THE CLM PRIMITIVE (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 MRd 5+X2,ELATCH RESTORE ERROR LATCH UPON RETURN
02290 B CANCEL RETURN CANCELS ANY BACKUP LATCH
02300***************************************************************************
02310* PUSHL6 GENERATES A NEW INTERNAL LABEL AND PUSHES IT ON THE
02320* CONTROL STACK.
02330****************************************************************************
02340* PUSHLR A LABEL
02350 MRdI LABEL-4\,*0+X2
02360 BA =1B9\,*IR2
02370 B FETCH
02380 DCw :6$:
02390 RLABEL DCw :000: ITEM MARK RIGHT
02400***************************************************************************
02410* POP POPS THE CONTROL STACK RESTORING THE VALUE TO SYMBOL IF THE
02420* TOP OF THE STACK IS MARKED AS A SYMBOL.
02430****************************************************************************
02440 Pop BS =1B9\,*IR2
02450 BCE POPSYM,0+X2,S BRANCH IF STACK TOP IS SYMBOL
02460 B FETCH
02470 POPSYM MCW 8+X2,S SYMBOL+7 STACK SYMBOL TO SYMBOL AREA
02480***************************************************************************
02490* LB1 SEARCHES THE CONTROL STACK TO FIND THE FIRST LABEL SYMBOL
02500* WHICH IS THEN MOVED TO SYMBOL. THE STACK IS NOT AFFECTED.
02510****************************************************************************
02520 LB1 LCA IR2,IR14
**HDIBd a**

**HDIBd 0£££0**

**EEEO**

**aVHD VHdlV HOd iS3i EIX+GViOI'NViSi dd OÏEEO**

**Etyi*t%X+0 OSWW OOEEO**

**VJdV lOaWAS WV313 A + 109WAS*: : MOW OifZEO**

**ixgy a ai O9ZEO**

**********************************************************************'*****»*0?ZE0**

**ONnOd dl lOOWAS 01 y3IdIiN3ai 3Hi 9NIiiVI 3H1 GiS3i 01 **

***************************************************************************** DOZE0**

**HDibd**

**G**

**D6U0**

**9yi»oiai**

**V3"l**

**39IE0**

**dl':i:**

**MD/I**

**11931**

**Oil£Q**

**HDild**

**9**

**09TE0**

**di*:d:**

**MDW**

**OSTEO**

**iiSli**

**39**

**OtTE0**

**9X+0*TX+0**

**NI aw**

**O6OE0**

**iX3N**

**3**

**1931**

**0«0E0**

**%yvw W31I dOd NV3S**

***************************************************************************** OlOEO**

***************************************************************************** OZOEO**

**HDIBd**

**G**

**D6U0**

**9yi»oiai**

**V3"l**

**39IE0**

**dl':i:**

**MD/I**

**11931**

**Oil£Q**

**HDild**

**9**

**09TE0**

**di*:d:**

**MDW**

**OSTEO**

**iiSli**

**39**

**OtTE0**

**9X+0*TX+0**

**NI aw**

**O6OE0**

**iX3N**

**3**

**1931**

**0«0E0**

**%yvw W31I dOd NV3S**

**03200**

**03196**

**03190**

**0318C**

**03180**

**0317C**

**03170**

**03160**

**03156**

**03150**

**03140**

**03130**

**03120**

**03110**

**03100**

**03090**

**03080**

**03070**

**03060**

**03050**

**03040**

**03030**

**03020**

**03010**

**03000**

**02990**

**02980**

**02970**

**02960**
0334C TSTAN BA =181,1R14
0335C MR50 0+X14,1R13
0336C BI TSTAN, IDTAB+X13 TEST ALPHANUMERIC
0337C SI 0-1+X14 SAVE ID
0338C MRID 0+X6+SYMBOL IN
0339C SAR IR6 SYMBOL
0340C CI 0-1+X6 AREA
0341C SI SYMBOL+7
0342C MCW :T:,TF
0343C B FETCH
0344C DCW ;0:
0345C IDTAB EQU *
03460 REP 10
03470 L DC ': 0-9
03480 DC ': :
03490 REP 9
03500 L DCW ': A-I
03510 DC ': :
03520 REP 9
03530 L DCW ': J-K
03540 DC ': :
03550 REP 8
03560 L DCW ': S-Z
03570 DC ': :
03580********************************************************************'^******
03590 ONUM B NEXT
03600 ONUM TESTS THE INPUT STRING FOR A VALID OCTAL NUMBER SETTING THE *
03610 TRUE-FALSE INDICATOR AND CONVERTING THE NUMBER TO BINARY IN SYMBOL *
03620 IF TRUE.
03630*
03640********************************************************************
03650 ONUM B NEXT
03660 HS SYMBOL+32 CLEAR TO ZEROES
03670 MCW :F:,TF
03680 LCA =380,1R13
03690 HS IR14
03700 ONMTST MR50 0+X6,1R14
| 03710 | 03720 | 03730 | 03740 | 03750 | 03760 | 03770 | 03780 | 03790 | 03800 | 03810 | 03820 | 03830 | 03840 | 03850 | 03860 | 03870 | 03880 | 03890 |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 03890 | 03890 | 03900 | 03910 | 03920 | 03930 | 03940 | 03950 | 03960 | 03970 | 03980 | 03990 | 04000 | 04010 | 04020 | 04030 | 04040 | 04050 | 04060 | 04070 |

**STRST EXAMINES THE INPUT STRING FOR A CHARACTER STRING LITERAL SETTING THE TRUE-FALSE INDICATOR AND MOVING THE STRING TO SYMBOL**

MOVE STRING TO SYMBOL
THE EVAL PRIMITIVE CAUSES A SEARCH OF THE SYMBOL TABLE AND THEN OUTPUTING OF A SYMBOL TABLE VALUE TO THE CODE STREAM ACCORDING TO TWO ONE CHARACTER LITERAL PARAMETERS:

1) THE LENGTH OF THE FIELD
2) THE RELATIVE POSITION WITHIN THE TABLE ENTRY

THE ENTLOC PRIMITIVE CAUSES A FULL SEARCH OF THE SYMBOL TABLE AND THEN CAUSES THE PROGRAM COUNTER TO BE ENTERED AS A VALUE FOR THE ADDRESS OF THE SPECIFIED SYMBOL. LEVEL 0 (STATIC) IS ASSIGNED FOR THE STORAGE LEVEL.
ENTLOC MCw :0:,SRCHTP
B SEARCH
MRID IR3-2,ADDR-2+X15 MOVE IN ADDRESS
Sw ADDR-2+X15 MARK RELOCATABLE
MCw :0:,LEVEL+X15 STATIC STORAGE INDICATOR
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. *
* *
* COMPUTE LEFTMOST
* ADDRESS
* OF
* RECEIVING FIELD
* CLEAR POSSIBLE WORD MARK
* TEST FOR LITERAL ENTRY
* ADDRESS TO IR13
* UPDATE INSTRUCTION COUNTER
* AVOID INADVERTENT RELOCATION MARKER
* ENTER, CLEAR ANY WORD MARKS
* ENTER LITERAL
* UPDATE INSTRUCTION COUNTER
* *
* SEARCH IS A SUBROUTINE FOR SEARCHING A BLOCK STRUCTURED SYMBOL *
* 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.*
06290 MRIN 0+XI.0+X6
06290 SW 0+XI
06270 SCAN B NEXT
06260 SCAN B NEXT
06260 SCAN B NEXT
06250 * F:BCUP
06240 * FOLLOWING THE OP CODE*
06230 * SCAN THE INPUT STRING FOR THE SPECIFIED LITERAL STRING*
06220 * NEXT INSTRUCTION
06210 * NEXT INSTRUCTION
06200 SCAN B NEXT
06190 CANCEL MCW: F:*BCUP
06180 * TURN OFF BACKUP SWITCH
06170 * CANCEL TURNS OFF ANY BACK UP LATCH*
06160 * CANCEL TURNS OFF ANY BACK UP LATCH*
06150 * CANCEL TURNS OFF ANY BACK UP LATCH*
06140 * BACKUP INITIALLY TURNED OFF*
06130 BCKUP DCM: F:
06120 SAVIN DCM = 3
06110 SAVOUT DCM = 3
06100 CM
06090 BCPUP
06080 T:BCKUP
06070 MCW: I:BCUP
06060 LATCH & NEXT
06050 LATCH & NEXT
06040 LATCH & NEXT
06030 LATCH & NEXT
06020 LATCH & NEXT
06010 LATCH & NEXT
06000 LATCH & NEXT
05990 LATCH & NEXT
05980 SETB TE
05970 COMP BCT SET*90
05960 SET TE
05950 SBR 141
05940 SBR 142
05930 MRIN 0+X13+0+X4
05920 POSITION A AND B ADDRESS REGISTERS
MOVLIT MOVES THE LITERAL CHARACTER STRING FOLLOWING THE ADDRESS (WHICH FOLLOWS THE OP CODE) TO THE ADDRESSED LOCATION.

MOVLIT MRIU 0+X1,IR13-2 ADDRESS OF RECEIVING CHAR FIFO

SAR IR14

MOVLIT MRIU 0+X14,0+X13 MOVE LITERAL DATA

SAR IR1 UPDATE LOCATION COUNTER

SW 0+X14

B FETCH

INUM CALLS INM FOR AN ATTEMPTED RECOGNITION OF AN INTEGER NUMBER.

INM SBR INMRTN+4

B NEXT
0667C  MCW   IR6,IR10           SAVE INPUT POINTER
0668C  MCW   :F:,TF
0669C  LCA   =380,IR13
0670C  BS    IR14
0671C  BCE   STISGN+0*X6,-     TEST FOR MINUS SIGN
0672C  BCE   STISGN+0*X6,++    TEST FOR PLUS SIGN
0673C  MCW   :+*:ISGN          MUST BE POSITIVE
0674C  B     INMTST
0675C  ISGN  DCW   =1
0676C  STISGN MRSD  0*X6,ISGN  SAVE SIGN
0677C  SAR   IR6               UPDATE INPUT POINTER
0678C  INMTST MRSD  0*X6,IR14
0679C  BCE   NOINT,0*X6,..      TEST FOR POSSIBLE FLTNG PT NM
0680C  BIO   MVNUM,IDTAB+X14   TEST FOR INTEGER DIGIT
0681C  BCE   NOINT,TF,F        HAVE WE FOUND AN INTEGER
0682C  B     CONVRT             YES, GO CONVERT TO BINARY
0683C  NOINT MCW   :F:,TF       SET TRUE-FALSE INDICATOR
0684C  MCW   IR10,IR6          RESTORE INPUT POINTER
0685C  B     INMRTN             RETURN
0686C  MVNUM MRSDI 0*X6,SYMBOL+X13
0687C  SAR   IR6
0688C  =1B1,IR13        
0689C  BCE   INMERR,IR13,10    TEST TOO MANY DIGITS
0690C  MCW   :T:,TF            WE HAVE PART OF AN INTEGER
0691C  B     INMTST             GO LOOK FOR MORE
0692C  CONVRT BS   CVRFLD       CLEAR HOLD FIELD
0693C  MCW   SYMBOL=1*X13,CVFLD MOVE IN DECIMAL INTEGER
0694C  SST   ISGN,CVFLD,60     SET SIGN IN CONVERSION FIELD
0695C  DTB   CVFLED+00         BINARY MANTISSA IN FR 0
0696C  TAM   CVFLED+00         STORE IT
0697C  C    CVFLED=6,=2C7777   TEST NUMBER TOO LARGE FOR 24 BITS
0698C  BE    INMOK
0699C  C    CVFLED=6,=2C0000
0700C  BL    INMERR
0701C  INMOK EQU   *            
0702C  MCW   CVFLED=2*SYMBOL+3  SAVE 24 BITS
0703C  SI    SYMBOL+3
ERASE ERASES THE SPECIFIED NUMBER OF CHARACTERS FROM THE CODE STRING.

ERASE BS

MOVE ERASE COUNT TO INDEX REG

ADJUST PROGRAM COUNTER

AND OUTPUT POINTERS

ERASE A CHARACTER

NEXT CHARACTER TO ERASE

DECREMENT LOOP COUNT

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

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 *

IF TRUE CONTINUE

IF NO BACKUP BYPASS BACKUP MECHANISM

RESTORE INPUT POINTER

COMPUTE ERASE

COUNT

ADJUST PROGRAM COUNTER

RESTORE OUTPUT POINTER
07410 MCW IR5,IR14
07420 MCW +ERSRTN,ERTST+4 SET UP RETURN FROM CODE ERASURE
07430 B ERTST ERASE CODE
07440 ERSRTN MCW +FETCH,ERTST+4 RESTORE INSTRUCTION IN ERASE ROUTINE
07450 B RETURN BACKUP CANCELS AND RETURNS
07460 DFMESE BCE ERRRTN,OUTPUT+20,F TEST PREVIOUS PENDING MESSAGE
07470 B ERMRPR
07480 MCW =9AF: SYNTAX,OUTPUT+28 DEFAULT MESSAGE
07490 B ERRHD FINISH TESTING AND MESSAGE
07500 ERMES B ERMRPR NO BACKUP SO CONTINUE WITH ERROR MESSAGE
07510 MRID 0+X1,OUTPUT+20
07520 SAR IR1
07530 ERPRTE EQU *
07540 MCW :****ERROR****:OUTPUT+16
07550L :PUT PRINT,OUTPUT,
07560 MCW ::OUTPUT+132 CLEAR
07570 MCW OUTPUT+132 PRINT LINE
07580 MCW =1C21,OUTPUT CARRIAGE CONTROL
07590 SI INPUT+80 RESTORE LOST ITEM MARK ON INPUT BUFFER
07600 B FCH FETCH
07610 HM EQU *
07620 BCE BEF,BCKUP,T TEST FOR BACKUP ACTION
07630 BCE FERR,0+X1,F IF FATAL CONTINUE TESTING
07640 B ERMES ELSE PRINT MESSAGE AND CONTINUE
07650 ERPASS MRID 0+X1,0 SCAN BY ERROR MESSAGE
07660 SAR IR1
07670 B ERRRTN
07680 FERR BCE ERPASS,OUTPUT+20,F TEST PREVIOUS FATAL MESSAGE PENDING
07690 B ERMRPR NO, SO SET UP
07700 MRID 0+X1,OUTPUT+20 MOVE IN MESSAGE
07710 SAR IR1
07720 ERRHD MCW :F:,CMPLCD SET FATAL COMPLETION CODE
07730 ERRRTN BCE RETURN,ECHT+F IF NO LATCH RETURN
07740 B ERPRTE ELSE PRINT AND CONTINUE
07750 ERMRPR SBR PRPRTN+4
07760 MCW ::OUTPUT+132 CLEAR PRINT LINE
07770 MCW OUTPUT+132 CHAINED MOVE
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>07780</td>
<td>SI</td>
<td>INPUT+80</td>
</tr>
<tr>
<td>07790</td>
<td>MCw</td>
<td>IR6,IR14</td>
</tr>
<tr>
<td>07800</td>
<td>BS</td>
<td>+INPUT,IR14</td>
</tr>
<tr>
<td>07810</td>
<td>MCw</td>
<td>::OUTPUT+1+X14</td>
</tr>
<tr>
<td>07820</td>
<td>MCw</td>
<td>::OUTPUT</td>
</tr>
<tr>
<td>07830</td>
<td>:PUT</td>
<td>PRINT,OUTPUT,</td>
</tr>
<tr>
<td>07840</td>
<td>MCw</td>
<td>::OUTPUT+1+X14</td>
</tr>
<tr>
<td>07850</td>
<td>PRPRTN B</td>
<td>*</td>
</tr>
<tr>
<td>07860</td>
<td>EXITI EQU</td>
<td>ERRFLG</td>
</tr>
<tr>
<td>07870</td>
<td>OVFLw EQU</td>
<td>*</td>
</tr>
<tr>
<td>07880</td>
<td>:PUT</td>
<td>PRINT,OVFMES,</td>
</tr>
<tr>
<td>07900</td>
<td>ERRFLG MCw</td>
<td>:F:,CMPLCD</td>
</tr>
<tr>
<td>07930</td>
<td>L DCw</td>
<td>=1C45</td>
</tr>
<tr>
<td>07930</td>
<td><em><strong>#</strong></em>********* **********************************************</td>
<td></td>
</tr>
<tr>
<td>07940</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>07950</td>
<td>ERROR IS EXECUTED IF AN ATTEMPT IS MADE TO INTERPRET AN INVALID</td>
<td></td>
</tr>
<tr>
<td>07960</td>
<td>OP CODE, THE JOB IS ABORTED.</td>
<td></td>
</tr>
<tr>
<td>07970</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>08000</td>
<td>L DCw</td>
<td>=1C45</td>
</tr>
<tr>
<td>08010</td>
<td>:PUT</td>
<td>PRINT,OPCDMS,</td>
</tr>
<tr>
<td>08020</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>08030</td>
<td>B EXIT</td>
<td></td>
</tr>
<tr>
<td>08030</td>
<td>OPCDMSDCw</td>
<td>:1INVALID OP CODE, JOB ABORTED:</td>
</tr>
<tr>
<td>08050</td>
<td>L DCw</td>
<td>=1C45</td>
</tr>
<tr>
<td>08060</td>
<td>**THE ENUM PRIMITIVE EXAMINES THE INPUT STRING FOR A FLOATING POINT</td>
<td></td>
</tr>
<tr>
<td>08070</td>
<td>NUMBER SETTING THE TRUE-FALSE INDICATOR AND CONVERTING THE NUMBER</td>
<td></td>
</tr>
<tr>
<td>08090</td>
<td>TO BINARY IN SYMBOL IF TRUE.</td>
<td></td>
</tr>
<tr>
<td>08100</td>
<td>**</td>
<td></td>
</tr>
<tr>
<td>08120</td>
<td>ENUM MCw</td>
<td>+DHOLD,IR12</td>
</tr>
<tr>
<td>08130</td>
<td>BS SCALE</td>
<td>CLEAR SCALE EXPONENT FIELD</td>
</tr>
<tr>
<td>08140</td>
<td>B NEXT</td>
<td></td>
</tr>
</tbody>
</table>
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 POINTER FOR MOVING FRACTION
DETERMINE FRACTION FOLLOWING
IF NONE MOVE DECIMAL FIELD FOR CONVERSION
UPDATE INPUT POINTER
IF RECORD END GO GET MORE
LOOK FOR DECIMAL FRACTION
CONCATENATE FRACTION WITH INTEGER PART
SCALE EXPONENT ADJUSTMENT
ADJUST IT
RESTORE X12
CLEAR SYMBOL AREA
TEST IF FIELD TOO LONG
WARNING MESSAGE
LOOK FOR DECIMAL FRACTION
CONCATENATE FRACTION WITH INTEGER PART
COMPUTE SCALE EXPONENT ADJUSTMENT
ADJUST IT
RESTORE X12
CLEAR SYMBOL AREA
TEST IF FIELD TOO LONG
WARNING MESSAGE
LOOK FOR DECIMAL FRACTION
CONCATENATE FRACTION WITH INTEGER PART
COMPUTE SCALE EXPONENT ADJUSTMENT
ADJUST IT
RESTORE X12
CLEAR SYMBOL AREA
TEST IF FIELD TOO LONG
WARNING MESSAGE
NO SIGN, TREAT AS PLUS
SET SUBTRACTION OP CODE
UPDATE INPUT POINTER
IF END OF RECORD GO GET MORE
LOOK FOR DECIMAL EXPONENT
IF INVALID THEN SIGNAL ERROR
ADJUST SCALE FACTOR EXPONENT
BININARY FORM OF EXPONENT
GO CONVERT
SET ADDITION OP CODE
SAVE RETURN ADDRESS
CLEAR INDEX
INITIAL TF SWITCH
INPUT CHARACTER TO INDEX REGISTER
TEST FOR NUMERIC CHARACTER
IF NO NUMERICS RETURN
MARK RIGHT END OF NUMERIC FIELD
RETURN TO CALLEER
MOVE NUMERIC CHAR TO SYMBOL FIELD
UPDATE INPUT POINTER
TEST END OF RECORD
UPDATE CHARACTER COUNT
SET TF FLAG TRUE
LOOK FOR MORE
CONVERT DECIMAL FIELD TO BINARY IN FRO
NORMALIZE IT
SET EXPONENT SIGN FLAG
TEST SCALE EXPONENT FOR VALID RANGE
CONVERT NEGATIVE
SCALE EXPONENT
TO POSITIVE
SET EXPONENT SIGN FLAG
TEST SCALE EXPONENT FOR VALID RANGE
B ERMPRP SET UP
MCW :EXPONENT OUT OF RANGE; OUTPUT+25 ERROR MESSAGE
MCW :F; CMLLCD FATAL COMPLETION CODE
B ERRPRT PRINT IT
SCLOK C SCALE, =380 IF EXPONENT ZERO CONVERSION FINISHED
BE FCVEND CLEAR INDEX TO ZERO
BS IR14 INSERT LOW 4 BITS FOR INDEXING
SST SCALE, IR14, 17 LEFT 3 BITS TO INDEX CVTTAB
EIM =48B, IR14 CONVERSION FACTOR TO FR1
TMA CVTTAB+X14, 01 CLEAR EIGHT CHAR FLOATING POINT FIELD
MCW SCALE, FHOLD+5 3 CHAR FIELD TO 8 CHAR FLOATING PT FIELD
TMA FHOLD+7, 03 LOAD IT TO FR3
BMS 31, 04 SHIFT RIGHT 4 BITS
SST FHOLD+7, 30 STORE IT
TAM FHOLD+7, 03 CLEAR INDEX TO ZERO
BS IR14 INSERT LOW 4 BITS FOR INDEXING
SST FHOLD+5, IR14, 17 LEFT 3 BITS TO INDEX CVTTAB
BIM =48B, IR14 CONVERSION FACTOR TO FR2
TMA CTAB16+X14, 02 COMPUTE INTERMEDIATE FACTOR
FINTM MAA 21 LOAD IT TO FR3
TLA 02 SAVE LOW ORDER FOR DOUBLE PRECISION
BMS 31, 04 NEXT 4 BITS
TAM FHOLD+7, 30 STORE IT
BS IR14 CLEAR INDEX TO ZERO
SST FHOLD+5, IR14, 03 MAX 10 BITS FOR SCALE EXPONENT
BIM =48B, IR14 SHIFT FOR INDEX
TMA CTB256+X14, 03 CONVERSION FACTOR TO FR3
FLOW MAA 32 LOW ORDER FACTOR
MAA 31 HIGH ORDER FACTOR
TAM FHOLD+7, 03 STORE IT
BS IR14 CLEAR INDEX TO ZERO
SST FHOLD+5, IR14, 03 MAX 10 BITS FOR SCALE EXPONENT
BIM =48B, IR14 SHIFT FOR INDEX
TMA CTB256+X14, 03 CONVERSION FACTOR TO FR3
AAA 32 ACCUMULATE LOW ORDER FACTORS
TAA FDIV, EXPSGN, M TEXT EXPONENT SIGN
AAA 32 ACCUMULATE LOW ORDER FACTORS
TAA 03 SAVE LOW ORDER
AAA 01 MULTIPLY BY HIGH ORDER SCALE FACTOR
TAA 03 SAVE LOW ORDER
AAA 02 LOW ORDER SCALE FACTOR
AAA 32 ACCUMULATE LOW ORDER FACTORS
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
THE MARK PRIMITIVE PUSHES THE ADDRESS OF THE NEXT OUTPUT STRING LOCATION ON THE CONTROL STACK FOR LATER USE BY THE SAVE PRIMITIVE.

SAVE PUSHES THE CODE GENERATED SINCE THE LAST MARK OPERATION ON THE VARIABLE LENGTH CODE STACK AND RESETS THE OUTPUT LOCATION BACK TO THE MARKED LOCATION.
THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE
CONTROL STACK.
THE RESTOR PRIMITIVE RESTORES THE TOP OF THE VARIABLE LENGTH CODE STACK TO THE OUTPUT STRING.

```
10410 RESTOR LCA STKEND,IR14
10420 BCE DORES,1+X14,77 TREAT AS NO-OP IF NULL
10430 B FETCH STK
10440 DORES MCw 4+X14,STKEND RESTORE PREVIOUS STK END POINTER
10450 MCw 4+X14,IR13 ALSO TO IR13 FOR LOOP TEST
10460 BA =1B5,IR14 POINT TO CODE TO MOVE
10470 RESTST C IR13,IR14 TEST MOVE NOTE POSSIBLE
10480 BH FETCH COMPLETION NULL RESTORE
10490 MRSDR 0+X14,0+X5 MOVE CODE
10500 SAR IR14 AND ADJUST
10510 SBR IR5 POINTERS
10520 BA =1B1,IR3
10530 B RESTST
```

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

```
10600 CHKSYM BS =1B9,IR2 ADJUST STACK POINTER
10610 C 8+X2,SYMBOL+7 TEST AGAINST SYMBOL VALUE
10620 BE SET IF EQUAL SET TRUE
10630 B SETF ELSE SET FALSE
```

SWAP SWAPS THE TOP TWO ELEMENTS ON THE CONTROL STACK.

```
10690 SWAP MCw 0-1+X2,SWPTMP MOVE TOP TO TEMPORARY
10700 SI 0-18+X2 ITEM MARK FOR NEXT MOVE
10710 MLIDI 0-10+X2,0-1+X2 SWAP
10720 MCw SWPTMP,0-10+X2
10730 B FETCH
```
THE ADD AND MULT PRIMITIVES COM普PRISE THE BINARY ARITHMETIC CAPABILITIES (TWO ADDRESS) OF THE META9 PSEUDO-MACHINE.

ADD
- Go get operand addresses
- Word mark to stop addition
- Find right position
- Set index
- Registers
- Binary add

MULT
- Go get operand addresses
- Word mark to stop move
- Find
- Right end
- Clear receiving field
- Move multiplier
- Word mark to stop move
- Find
- Right end
- Clear receiving field
- Move multiplicand
- Binary multiply requires 24 bits
- Put back result

SET RETURN
- First operand address
- Update location counter
- Second operand address
- Update location counter
11110  GTRTN B  *    RETURN TO CALLER

11170  MOVE B    GTOPRA  GET ADDRESSES IN IR10 AND IR11
11180  MRIDI 0+X10,0+X11  MOVE DATA AND TERMINATING ITEM MARK
11190  B    FETCH

11260  BLKCNT DCW :0:  BLOCK COUNTER
11270  PRVBLK DCw =1C00  PREVIOUS BLOCK NUMBER
11280  CRBLKT DCW =3  INITIAL SYMTAB SEARCH ENTRY
11290  BLKENT BA =1B1*BLKCNT  BLOCK COUNT
11300  BS =1B4*SYMEND  REDUCE SYMBOL TABLE END LOCATION
11310  TO ACCOMMODATE NEW BLOCK LIST ENTRY
11320  MCw SYMEND+IR13  SET SURROUNDING BLOCK NUMBER
11330  LCA PRVBLK+1+X13  SET UP FOR NEXT BLOCK ENTRY
11340  MCw BLKCNT+PRVBK  SPACE FOR DUMMY ENTRY
11350  BA =1B20*NEWSYM  STORE IN BLOCK LIST
11360  LCA NEWSYM+4+X13  SET FOR CURRENT SYMTAB SEARCH
11370  MCw SYMEND+CRBLKT  TO ACCOMMODATE NEW BLOCK LIST ENTRY
11380  B    FETCH

11400  THE BLOCK EXIT PRIMITIVE RESTORES CRBLKT POINTER AND PRVBLK
11420  NUMBER FOR THE SURROUNDING BLOCK.  THE SYMBOL TABLE ENTRIES FOR
11430  THE TERMINATING BLOCK ARE SCANNED FOR UNRESOLVED SYMBOLS.
11440  UNRESOLVED ENTRIES ARE ADDED TO THE SURROUNDING BLOCK IF NOT
11450  FOUND IN THAT PORTION OF THE TABLE.  APPROPRIATE LINKING
11460  PARAMETERS ARE SET FOR THE RESOLVE PRIMITIVE TO USE.  THUS
11470  DIABOLICAL LABEL REFERENCES IN A BLOCK STRUCTURE ARE RESOLVABLE.
1148C*

1149C**************************************************************************

1150C  BLKSAV  DCW  =3
1151C  BLKPRM  DCW  =1
1152C  BLKEXT  MCW  CRBLKT*BLKSAV  SAVE FOR UNRESOLVED SEARCH
1153C  MRSR  0+X1*BLKPRM  BLOCK EXIT PARAMETER
1154C  SAR  IR1
1155C  BS  IR13  CLEAR
1156C  MCW  CRBLKT*IR14  INSERT PREVIOUS BLOCK NUMBER
1157C  MCW  1+X14*IR13  NEW SURROUNDING BLOCK NUMBER
1158C  MCW  1+X14(PRVBLK  NEW SURROUNDING BLOCK NUMBER
1159C  BIM  =4B4*IR13  COMPUTE
1160C  MCW  SVSYME*IR14  BLOCK LIST
1161C  BS  IR13*IR14  LOCATION FOR SURROUNDING BLOCK
1162C  MCW  IR14*CRBLKT  BLOCK LIST POINTER
1163C  MCW  BLKSAV*IR12  TERMINATING BLOCK LIST POINTER
1164C  MCW  4+X12*IR12  SYMBOL TABLE POINTER
1165C  BCE  FETCH*BLKPRM,01  TEST NO LABEL LINK UP
1166C  CHKUND  C  CHAIN+X12:000:  CHECK FOR END OF
1167C  BE  FETCH  BLOCK TABLE ENTRIES
1168C  MCW  CHAIN+X12*IR12  NOTE 1ST TIME JUMP OVER DUMMY ENTRY
1169C  UDCOX  BCE  CHKPRV+DTYPE+X12:00  CHECK FOR UNDEFINED SYMBOLS
1170C  B  CHKUND  LOOK FOR MORE
1171C  CHKPRV  MCW  NAME+X12*SYMBOL+7  SET NAME TO USE SEARCH SUBROUTINE
1172C  MCW  1*SRCHTP  BLOCK ONLY SEARCH MODE
1173C  B  SEARCH
1174C  BCE  ADDSYM,TF,F  IF FALSE SYMBOL ADDED TO SURROUNDING
1175C  BCE  PRVUN,0+X15,00  IF FOUND BUT STILL UNDEFINED SET MARKERS
1176C  SI  0+X15
1177C  MLIDW  DIMCNT+X15,DIMCNT+X12  FOUND, SET VALUES FOR RESOLVE
1178C  CI  0+X15
1179C  B  CHKUND
1180C  ADDSYM  EQU  *
1181C  PRVUN  Sw  IR15-2
1182C  MCW  IR12*IR9  SAVE CURRENT ENTRY POINTER
1183C  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 BE EXIT, CMP LCD, F exit if fatal compilation to this point.

BS CVBFLD CLEAR

MCW IR3, CVBFLD-2 MOVE PROGRAM SIZE

TMA CVBFLD, 00 LOAD TO FR0

BTD CVBFLD, 00 CONVERT TO DECIMAL AND STORE

LCA EW0RD, PSlZE EDIT CONTROL WORD

MCE CVBFLD, PSlZE MOVE AND EDIT

CW PSlZE-8 CLEAR WORD MARK

BS CVBFLD CLEAR

MCW INSTCT, CVBFLD-2 MOVE INSTRUCTION COUNT

TMA CVBFLD, 00 LOAD TO FR0

BTD CVBFLD, 00 CONVERT TO DECIMAL AND STORE

LCA EW0RD, ICOUNT EDIT CONTROL WORD

MCE CVBFLD, ICOUNT MOVE AND EDIT

CW ICOUNT-8 CLEAR WORD MARK

PUI PRINT, EXITMS PRINT IT

BS CVBFLD

MCW SRCHCT, CVBFLD-2 MOVE SEARCH COUNT

TMA CVBFLD, 00

BTD CVBFLD, 00

LCA EW0RD, SCOUNT

MCE CVBFLD, SCOUNT

BS CVBFLD
START OF COMPILED CODE

SCAN FOR WORD MARK

SAVE NEXT POSITION

DETERMINE COMPLETION

TEST POSSIBLE LABEL VARIABLE - LABEL CONSTANT RESOLUTION

UNRESOLVED ADDRESS IS POINTER TO SYMBOL TABLE. MOVE IT TO IR14.*

REMOVE UNRESOLVED MARKER

VALID

ADDRESS

CHAIN TO SURROUNDING BLOCK

GO TEST FOR ADDITIONAL CHAINING

SET LEVEL AND DISPLACEMENT

TEST POSSIBLE LABEL VARIABLE - LABEL CONSTANT RESOLUTION

MOVE COMPARISON COUNT

MOVE TABLE ENTRY COUNT

MOVE AND EDIT
12590  LBVTST BCE LBV,DTYPE+X14,05  TEST LABEL VARIABLE IN SYMTAB
12600  B SCAN              ELSE CONTINUE
12610  LBV BNP SCAN,0-4+X15  MAKE SURE OP CODEF 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,CMPLMS5,
12660  H                    
12670  B EXIT
12680  CMPLMDCW :ACOMPILER ERROR DISCOVERED DURING RESOLVE:
12690  L DCW =1C45
12700  NTDEFN MCW NAME+X14,PSYM
12710L :PUT PRINT,NDFMES5,
12720  MCW :F:,CMPLCD      SET COMPLETION CODE
12730  B SCAN
12740  NDFMESDCW :1UNDEFINED SYMBOL :
12750  PSYM DCW : :    
12760L  L DCW =1C45
12770  TABCNTDCW :A  ****SYMBOL TABLE ENTRY COUNT =:
12780  ECOUNT DC =9    
12790  DC :****:        
12800  L DCW =1C45
12810  EXITMSDCW :2  ****COMPILED PROGRAM SIZE =:
12820  PSIZE DC =9     
12830  DC :: METAX INSTRUCTION COUNT =:
12840  ICOUNT DC =9    
12850  DC :****:        
12860L  L 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  ICOUNT DC =9      
12930  DC :****:        
12940L  L 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 NXRTRN+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
EOFMESDCW :1UNEXPECTED END OF FILE, JOB ABORTED:
DCw =1C45 RECORD MARK
PLIST IS EXECUTED IF A POST LISTING IS REQUESTED.

EXIT, PSTLST, N
EXIT IF POST LIST NOT REQUESTED

:SKIP PRINT, 57,
PROGRAM COUNTER
START OF GENERATED CODE

EXIT IF POST LIST NOT REQUESTED

PRINT, 57,
CLEAR
PRINT LINE
PRINT LINE
PRINT POST LISTING
CLEAR
PRINT LINE
CLEAR CONVERSION FIELD
LOAD TO FR0
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 FMTCDE=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  MRSD 0-1×X15,IR12  INSERT OP CODE
13770  BIM =4BS,IR12  MULT BY TABLE ENTRY SIZE
13780  MCW OPTAB+4×12,OUTPUT+21 OP CODE TO PRINT
13790  BCE PLSTPR+OPTAB+5×12,00 TEST FOR NO OPERANDS
13800  BBE TLITRL+OPTAB+5×12,60 TEST POSSIBLE LITERAL
13810  LITERL EQU LITRL
13820  BCE LITERL+OPTAB+5×12,01 TEST SINGLE CHARACTER LITERAL
13830  BCE ADDR+OPTAB+5×12,03 TEST FOUR CHAR ADDRESS
13840  BCE TWOOP+OPTAB+5×12,10 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 SET PRINT
14010  MCW : : : OUTPUT+132 CLEAR
14020  MCW OUTPUT+132
14030  MCW =1C21,OUTPUT CARRIAGE CONTROL
14040L :PUT PRINT+DPVMES DOPE VECTOR MESSAGE
14050  BS IR12 CLEAR
14060  MRSD 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 PROGRAM COUNTER
BA = IR15, IR12
END OF DOPE VECTOR
SW = 0-1+X12
MARK FOR MOVE
MCW = :; OUTPUT+15
MRRD = 0+X15, OUTPUT+16
MOVE IT
SAR IR15
SBR IR11
CW = 0-1+X12
MCW = ; ; 0+X11
B PLSTPR
GO PRINT
DCW = 1C45
LDA MCW = :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 LNCDE, OTYPE, 04
TEST FOR STRING TYPE
B PLSTPR
GO PRINT
MCW = ; ; OUTPUT+32
MRSD = 0+X15, OUTPUT+33
MOVE IN
EXM
LENGTH CODE
SAR IR15
MCW = ; ; OUTPUT+35
BA = 1B2, IR13
B PLSTPR
O>Type DCW = 0:
ADD4 B ADDFR
BCE IOTYPE, 0-5+X15, 50
CHECK I/O SETUP
B PLSTPR
MCW = ; ; OUTPUT+32
I/O TYPE FOR PRINTING
MRSD 0+X15, OUTPUT+33
SAR IR15
BA = 1B1, IR13
MCW = ; ; OUTPUT+34
14440  B  PLSTPR
14450  ADDR5  B  ADDFV
14460  B  PLSTPR
14470  ADDFR  SBR  AFRRTN+4  SET RETURN ADDRESS
14480  MRID  0X15,ADD4CN-3  MOVE ADDRESS
14490  SAR  IR15
14500  BA  =1B4,IR13  BUMP CODE COUNTER
14510  MCw  SYMSTR,IR14  TABLE START
14520  A4COMP  C  4X14,ADD4CN  TEST
14530  BE  ADFRFD
14540  BA  =1B20,IR14  NEXT
14550  C  IR14,NEWSYM  TEST
14560  BEH  A4COMP  TABLE END
14570  MISSAD  MCw  :*****:,OUTPUT+35  MISSING SYMBOL MARKER
14580  B  LITOPR
14590  ADFRFD  MCw  NAME+X14,OUTPUT+30  MOVE SYMBOL
14600  AFRRTN  B  *  RETURN
14610  ADD4CN  DCw  =4
14620  ADDFV  SBR  AFVRTN+4
14630  MRSD  0X15,ADD5CN-4  GET ALL FIVE CHARACTERS
14640  EXM
14650  EXM
14660  EXM
14670  EXM
14680  SAR  IR15
14690  BA  =1B5,IR13
14700  MCw  SYMSTR,IR14
14710  A5COMP  SI  4X14  ITEM MARK FOR MOVE
14720  MRID  0X14,AD5CN1-4  MOVE DATA, NO WORD MARKS
14730  C  ADD5CN+AD5CN1  TEST EQUALITY
14740  BE  ADFVFD
14750  BA  =1B20,IR14
14760  C  IR14,NEWSYM  TEST
14770  BEH  A5COMP
14780  BS  =1B5,IR15
14790  BS  =1B5,IR13
14800  B  MISSAD
14810 ADFVFD MCW NAME+X14, OUTPUT+30
14820 AFVRTN B *
14830 ADD5CN DCw =5
14840 AD5CN1 DCw =5
14850 SYMADD B OPCODE FIRST TIME ONLY
14860 MCW SYMSTR, IR14 TABLE START
14870 REL0CT BNP NOSYM, 2+X14 IGNORE NON-RELOCATABLE SYMBOLS
14880 C 4+X14, IR13 TEST
14890 BE SYMFND EQUALITY
14900 NOSYM BA =1B20, IR14 NEXT
14910 C SYMEND, IR14 TEST
14920 BH OPCODE TABLE END
14930 B REL0CT TEST NEXT
14940 SYMFND MCW NAME+X14, OUTPUT+15
14950 R OPCODE
14960 PEXIT MCW :END PROGRAM, OUTPUT+32
14970L :PUT PRINT,
14980 L EXIT
14990 TLITRL S$T 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 *
15040L :PUT PRINT, FMTMES, MESSAGE
15050 MCW "::OUTPUT+15 BUILD
15060 MCW +OUTPUT+16, IR12 FORMAT
15070 NFMTCH MRSD 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 DCw :A ***FORMAT CODE***:
OP CODE OPERAND TABLE FOR POSTLISTING

15180 L DCw =1C45
15190 OPTAB EQU *
15200 REP 8
15210 DCw :ERRORO:
15220 DCw :DYNAM4:
15230 DCw :STKTP8:
15240 DCw :ALLOC4:
15250 REP 5
15260 DCw :ERRORO:
15270 DCw :LDA 5:
15280 DCw :LD 5:
15290 DCw :STO 0:
15300 DCw :SST 0:
15310 DCw :FLAG 0:
15320 DCw :ENTPR1:
15330 DCw :RETRNO:
15340 DCw :JUMPA0:
15350 DCw :JUMP 4:
15360 DCw :JUMPT4:
15370 DCw :JUMPF4:
15380 DCw :STCKC1:
15390 DCw :COMPC1:
15400 DCw :SWAP 0:
15410 DCw :POPUPO:
15420 DCw :ERRORO:
15430 DCw :ADD 0:
15440 DCw :MULT 0:
15450 DCw :SUB 0:
15460 DCw :DIV 0:
15470 DCw :NEG 0:
15480 REP 3
15490 DCw :ERRORO:
15500 DCw :FMT 4:
15510 DCw :GET 0:
15520 DCw :PUT 0:
15530 DCw :EDIT 0:
15540 REP 4
15550 DCw :ERROR0:  OP CODE 60
15560 DCw :OR 0:  OP CODE 61
15570 DCw :AND 0:  OP CODE 62
15580 DCw :NOT 0:  OP CODE 63
15590 REP 3
15600 DCw :ERROR0:  OP CODE 64
15610 DCw :INDEXR1:  OP CODE 65
15620 DCw :INDEXX1:  OP CODE 66
15630 DCw :CAT 0:  OP CODE 70
15640 DCw :SBSTRO:  OP CODE 71
15650 REP 4
15660 DCw :ERROR0:  OP CODE 76
15670 DCw :STOP 0:  LAST
15680 DCw :ERROR0:  END START
15690 LITORG*
15700

SYMBOL DEFINITION - CARD REFERENCE INDEX

A4C0MP 14520; A5C0MP 14710; ADD5CN1 14840; ADD4CN 14610; ADD5CN 14830;
ADD 10810; ADDFR 14470; ADDFV 14620; ADDR4 14360; ADDR5 14450;
ADDR 04950; ADDRS5 12540; ADDSYM 11810; ADFRFD 14590; ADFVFD 14810;
AFRRTN 14600; AVFRRTN 14820; ALLOC 13930; BCTRP 06130; BFT 07330;
BLKCNT 11260; BLKEXT 11290; BLKEX 11520; BLKPRM 11510; BLKSAV 11500;
BLKTST 13180; BM 07610; BRANCH 01860; BRANCHF 01900; BRANCHT 01880;
CANCEL 06190; CHAIN 04920; CHKPRV 11710; CHKSYM 10600; CHKUND 11660;
CHNADD 12520; CHNTST 12490; CLM 02080; CMPCNT 12880; CMPLCD 00420;
CMPLMS 12680; CMP 05850; COMPT 05970; CONUM 03760; CONVR 06920;
CRBLKT 11280; CTAB16 09640; CTB256 09800; CVBFLD 07050; CVT 09480;
DECMVE 08340; DEFMES 07460; DHOLD 09450; DIMCNT 04930; DMVE 08200;
DNMRTN 08710; DNMTST 08670; DNUM 08630; DOENT 05130; DORES 10440;
DORRT 02260; DOSAVE 10020; DPVMES 14190; DTYPE 04970; ECOUNT 12780;
ELATCH 00400; ENDTST 13170; ENTA 04600; ENTL 04610; ENTLIT 04730;
ENTLOC 04450; ENUM 08120; EOFMES 13310; ERASE 07140; ERMES 07500;
**INSTRUCTION COUNT = 310489**
XI. APPENDIX C
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.
TEST: 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.            */
/*                                                                            */
/*******************************************************************************/

STRINGS AND SUBSTRINGS

STRNCI:BEGIN; DECLARE (S,T) CHAR (35);
  PUT EDIT ("BEGIN STRINGS") (SKIP(3),A);
  S="THIS IS A STRING."
  PUT EDIT (S) (A);
  SUBSTR(T,1,4)="THIS"
  SUBSTR(T,5,29)=SUBSTR(S,5,6) // "CONCATENATED SUBSTRING."
  PUT EDIT (T) (A(33));
  IF SUBSTR(S,1,5)="THIS" THEN PUT EDIT("STRING COMPARE 1 WORKS") (A);
  IF SUBSTR(S,1,4)=SUBSTR(T,1,4) THEN PUT EDIT("STRING COMPARE 2 WORKS") (A);
  PUT EDIT ("EXIT STRINGS") (SKIP,A);
END STRNCI;
/*
  INPUT / OUTPUT (INCLUDING STRING I/O)
*

IOBLK: BEGIN;
  DECLARE (A, B, C, M(5)) FIXED, (X, Y, Z) FLOAT, (S, 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, A(20) A(20));
  GET STRING(S) EDIT(X, Y) (E(10));
  GET STRING(T) EDIT(Z) (E(10));
  PUT EDIT(X, Y, Z, S, T) (SKIP(2) E(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;
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 = "«I", 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;

ABIHNCAT;
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(10));
PUT EDIT ("EXIT RPROC") (SKIP,A);
END RPRGC;

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

POSTF:BEG1N;
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(I);
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.);
 IF SUBSTR(INPUT,I,1)=";" 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 EXPR THEN RETURN(\*F\*);
        IF NOT LITERAL(")") 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 TERM;
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(2),A(80));
OUTPUT=INPUT; /* CLEAR OUTPUT FIELD. */
IF EXP1 THEN PUT EDIT ("POSTFIX EXPRESSION =",SUBSTR(OUTPUT,1,J-1)//";")
    (SKIP,A(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(I)=13;
   CALL Z((F(I)+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 )) (I(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*I(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*I(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 LABEL;
END TEST;
****COMPILED PROGRAM SIZE = 6,325; MAX INSTRUCTION COUNT = 55,991****
****SYMTAB SEARCH COUNT = 1,484; SYMTAB COMPARE COUNT = 78,626****
****SYMBOL TABLE ENTRY COUNT = 370****
00000          DYNAM $001
00005          JUMP $002
00010  $002    STKP $STK0 ,&STK1
00025          JUMP $003
00026  $003    FMT $004 ,"1"
00032          LDA "BEGIN STRINGS"
00047          EDIT
00048          PUT
00049          JUMP $005
   ***FORMAT CODE***
00054  $004    "2000380000"
00068  $005    LDA S ,"OL"
00074          LDA "THIS IS A STRING."
00093          ST0
00094          FMT $006 ,"1"
00100          LDA S ,"OL"
00103          EDIT
00109          PUT
00110          JUMP $007
   ***FORMAT CODE***
00115  $006    "80000"
00122  $007    LDA T ,"OL"
00130          LD "0001"
00136          LD "0004"
00142          SBSTR
00143          LDA "THIS"
00149          ST0
00150          LDA T ,"OL"
00158          LD "0005"
00164          LD "0006"
00170          SBSTR
00171          LDA S ,"OL"
00179          LD "0005"
00185          LD "0006"
00191          SBSTR
00192          LDA "CONCATENATED SUBSTRING."
00217          CAT
00210:  STO
00219:  FMT $008 "1"
00223:  LDA T "OL"
00230:  EDIT
00232:  PUT
00235:  JUMP $009

**FORMAT CODE**
00240: $008 "8000J"
00247: $009 LDA S "OL"
00255: LD "0001"
00261: LD "0005"
00267: SBSTR
0026b: LDA "THIS"
00274: COMPC "1"
00275: JUMPF $010
00281: FMT $011 "1"
00287: LDA "STRING COMPARE 1 WORKS"
00311: EDIT
00312: PUT
00313: JUMP $010

**FORMAT CODE**
00318 $011 "80000"
00325 $010 LDA S "OL"
00333: LD "0001"
00339: LD "0004"
00345: SBSTR
00348: 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***
00411 $014 "80000"
00418 $013 FMT $016 "1"
00424 LDA "EXIT STRINGS"
00430 EDIT
00436 PUT
00440 JUMP $017
***:FORMAT CODE***
00445 $016 "2000180000"
00457 $017 STKTP $STK0 $STK1
00462 ALLOC $STK1 ,M
***DOPE VECTOR CODE***
"10*Z0001Z0005"
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 "2000380000"
00539 $020 FMT $021 "2"
00545 LDA S "0D"
00553 GET
00554 JUMP $022
***:FORMAT CODE***
00559 $021 "80006"
00565 $022 FMT $023 "2"
00572 LDA A
00578 GET
00579 LDA B
00585 GET
00586 LDA C
00592 GET
00593 JUMP $024
***:FORMAT CODE***
00598 $023 "30001000500050005"
00620 $024 LDA T ,"OD"
00628 FMT $025 ,"0"
00634 LD A
00640 EDIT
00641 LD B
00647 EDIT
00648 LD C
00654 EDIT
00655 PUT
00656 JUMP $026
***FORMAT CODE***
00661 $025 "0005"
00668 $026 FMT $027 ,"1"
00674 LDA S ,"0D"
00682 EDIT
00683 LDA T ,"0D"
00691 EDIT
00692 LD A
00693 EDIT
00699 LD B
00705 EDIT
00705 LD C
00712 EDIT
00713 PUT
00714 JUMP $028
***FORMAT CODE***
00719 $027 "2000180000003000D8000030000*0005*0005*0005"
00761 $029 FMT $029 ,"Z"
00767 LDA S ,"0D"
00775 GET
00776 LDA T ,"0D"
00784 GET
00785 JUMP $030
***FORMAT CODE***
00790 $029 "2000180000D8000D"
00807 $030 LDA S ,"0D"
00815 FMT $031 ,"Y"
0082: LDA X
0082: GET
0082: LDA Y
0083: GET
0083: JUMP $032

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

***FORMAT CODE***
0087: $033 "9000"
0088: FMT $035 ,"1"
0088: LD X
0089: EDIT
0089: LD Y
0089: EDIT
0090: LD Z
0090: EDIT
0090: LDA S ,"0D"
0091: EDIT
0091: LDA T ,"0D"
0092: EDIT
0092: PUT
0092: JUMP $036

***FORMAT CODE***
0093: $035 "200029000D9000D9000D200018000D8000D"
0096: LDA A
0097: LD "0001"
0098: SST
0098: LD "0005"
0098: STCKC "4"
0098: JUMPT $037
0099: LDA M
0100: 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 $STKTO , $STKTI
01177' ALLOC $STKTI , M

***DOPE VECTOR CODE***
"204ZcccZ2000Z20001Z000:
01211 JUMP $045
01216 $045 FMT $046 , "1"
01227 LDA "BEGIN DO GROUPS"
01230 EDIT
01240 PUT
01244: JUMP $047

***FORMAT CODE***
01246 $046 "2000380000"
01258 $047 LDA I
01264 LDA "0000"
01270 STK
01271 $049 LDA "0004"
01277 STCKC "4"
01279 JUMPT $048
01284 LDA "0004"
01290 LDA I
01295 SUB
01297 LDA "0005"
01303 MULT
01304 LDA $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***
01399  S060 "80000"
01406  S061 JUMP $052
01411  S062 FMT $063 "1"
01417  LDA "CASE 3"
01425  EDIT
01426  PUT
01427  JUMP $064

***FORMAT CODE***
01432  S063 "80000"
01439  S064 JUMP $052
01444  S065 FMT $066 "1"
01450  LDA "CASE 4"
01458  EDIT
01459  PUT
01460  JUMP $067

***FORMAT CODE***
01465  S066 "80000"
01472  S067 JUMP $052
01477  S068 LD $050
01483  S050 JUMP $053
01488  JUMP $056
01493  JUMP $059
01498  JUMP $062
01503  JUMP $065
01503  $052 LDA 1
01514  LD 1
01520  LD  "0001"
01526  ADD
01527  SST
01528  JUMP  $049
01530  LDA  I
01531  LD  "0001"
01541  S51
01542  LD  "cccc"
01543  STCKC  "2"
01544  JUMPT  $069
01550  LDA  M
01551  LD  I
01552  LD  "0005"
01553  INDXA  "2"
01554  LD  "0000"
01555  STO
01556  LDA  A
01557  LD  "0003"
01558  LD  "0002"
01559  MUL
01604  LD  "0005"
01605  SUB
01606  SST
01607  LD  "0001"
01608  STCKC  "4"
01613  JUMPT  $071
01623  LD  A
01633  LD  "0005"
01634  STCKC  "2"
01639  JUMPF  $071
01644  LDA  M
01645  LD  I
01646  LD  A
01647  INDXA  "2"
01648  LD  A
01649  STO
01650  LDA  A
01678  LD A
01684  LD "0001"
01690  ADD
01691  SST
01692  JUMP $072
01697  ADD $071 FMI $073 "1"
01703  LDA "I = "
01705  EDIT
01710  LD I
01716  EDIT
01717  LDA ", M(I,1) = "
01720  EDIT
01723  LDA M
01727  LD I
01733  LD "0001"
01736  INDEXR "2"
01741  EDIT
01743  LDA ", M(I,5) = "
01747  EDIT
01752  LDA M
01755  LD I
01761  LD "0005"
01763  INDEXR "2"
01766  EDIT
01767  PUT
01778  JUMP $074
***FORMAT CODE***
01793  $073 "80000*0005"
01805  1074 LDA I
01810  LD I
01817  LD "0001"
01823  SUB
01828  SST
01830  JUMP $070
01839  FMT $075 "1"
01836  LDA "EXIT DO GROUPS"
01852  EDIT
01853   PUT
01854   JUMP $076
***FORMAT CODE***
01859   $075 "2000180000"
01872   $076 STKTP $STKTO $STKTL
01882   JUMP $077
01887   $077 FMT $078 "1"
01893   LDA "ENTER ARITHMETIC BLOCK"
01917   EDIT
01918   PUT
01919   JUMP $079
***FORMAT CODE***
01924   $078 "2000380000"
01935   $079 LDA A
01942   LD "0001"
01943   SST
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
01986   LDA Y
01992   LD X
01998   LD A
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
02087 JUMP $085

***FORMAT CODE***
02088 $084 "0000D"
02095 $085 FMT $086 "1"
02099 LD A
02105 EDIT
02106 PUT
02107 JUMP $087

***FORMAT CODE***
02113 $086 "0005"
02117 $087 LDA A
02122 LDA A
02126 LD "0001"
02127 ADD
02132 SST
02139 JUMP $081
02146 $080 FMT $088 "1"
02152 LDA "EXIT ARITHMETIC BLOCK"
02158 EDIT
02162 PUT
02165 JUMP $089

***FORMAT CODE***
02180 $088 "2000180000"
02187 $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  JUMP  $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  LDA  "0001"
02290  SUB
02291  STO
02292  LD  $STKT2
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
02335  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 NFACT
02456   JUMPA
02457   $101 LD "0001"
02463   ADD
02464   STO
02465   LD $STKT1
02471   LDA NFACT
02477   JUMPA
02478   $099 EDIT
02479   PUT
02480   JUMP $103
    **FORMAT CODE***
02485   $098 "2000180000*000*"
02502   $103 FMT $104 "1"
02508   LDA "EXIT RPROC"
02520   EDIT
02521   PUT
02522   JUMP $105
    **FORMAT CODE***
02527   $104 "2000180000"
02539   $105 STKTP $STKTO $STKT1
02550   JUMP $106
02555   OUT ENTPK "2"
02557   DYNAM $095
02562  JUMP $108
02567  $108  LDA T  "1+"
02575  LD  J
02581  LD  "0001"
02587  SBSTH
02588  LDA OUTCHAR  "01"
02596  STO
02597  LDA  J
02603  LD  J
02605  LD  "0001"
02615  ADD
02616  STO
02617  RETRN
0261E  NEXT  ENTRP "2"
02620  DYNAM I
02625  JUMP $110
02630  $110  LDA INPUT  "1+"
02638  LD  I
02644  LD  "0001"
02650  SBSTR
02651  LDA  ""
02654  COMPC "1"
02656  JUMPF $112
02661  LDA  I
02667  LD  I
02673  LD  "0001"
02675  ADD
02679  STO
02680  LD  I
02687  LD  "001+"
02693  STCKC "4"
02695  JUMPF $113
02700  LD  "F"
02703  SWAP
02706  RETRN
02705  $113  LDA INPUT  "1+"
02713  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  NUMBFN  ENTPR  "2"
02754  DYNAM   I
02759  JUMP  $116
02764  $116  LDA   $117
02770  LD   $ACTIVE
02776  LD   $ACTIVE
02782  FLAG
02783  LD   $STKT2
02789  LDA   NEXT
02795  JUMPA
02796  $117  NOT
02797  JUMPF  $118
02802  LD   "F"
02805  SWAP
02806  RETRN
02807  $118  LDA   INPUT   ",14"
02815  LD   I
02821  LD   "0001"
02827  SBSTR
02828  LDA   "9"
02831  COMPC   "3"
02833  LDA   INPUT   ",14"
02841  LD   I
02847  LD   "0001"
02853  SbSTR
02854  LDA "0"
02857  COMPC "5"
02859  AND
02860  JUMPF $119
02865  LDA CHAR "01"
02873  LDA INPUT "1+"
02881  LD I
02887  LD "0001"
02893  SbSTR
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+
02982 LD I
02985 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
03069 STO
03069 LDA I
03079 LD I
03089 LD "0001"
03087 ADD
03089 STO
03092 SWAP
03093 RETRN
03094 $124 LD "F"
03097 SWAP
03098 RETRN
03099 RETRN
03100 LITERAL ENTPR "2"
03103 DYNAM $095
03107 JUMP $126
03112 $126 LDA $127
03118    LD  $ACTIVE
03124    LD  $ACTIVE
03130    FLAG
03131    LD  $STK_T2
03137    LDA  NEXT
03143    JUMPA
03144 $127  NOT
03145    JUMPF  $128
03150    LD  "F"
03153    SWAP
03154    RETRN
03155 $128  LDA  INPUT  "1+
03162    LD  I
03165    LD  "0001"
03175    SBSTR
03176    LDA  OUTCHAR  "01"
03184    COMPC  "1"
03186    JUMPF  $129
03191    LDA  I
03197    LD  I
03203    LL  "0001"
03209    ADD
03210    STO
03211    LD  "T"
03214    SWAP
03215    RETRN
03216 $129  LD  "F"
03219    SWAP
03220    RETRN
03222 PRIMARY ENTRP  "2"
03224    DYNAM  $130
03229    JUMP  $131
03234 $131  LDA  $132
03240    LD  $ACTIVE
03246    LD  $ACTIVE
03252    FLAG
<table>
<thead>
<tr>
<th>Address</th>
<th>Command</th>
<th>Immediate</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>03253</td>
<td>LDA</td>
<td>$133</td>
<td></td>
</tr>
<tr>
<td>03259</td>
<td>LDA</td>
<td>$133</td>
<td></td>
</tr>
<tr>
<td>03265</td>
<td>LDA</td>
<td>&quot;(&quot;</td>
<td></td>
</tr>
<tr>
<td>03266</td>
<td>STO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03269</td>
<td>LD</td>
<td>$STK T2</td>
<td></td>
</tr>
<tr>
<td>03275</td>
<td>LDA</td>
<td>LITERAL</td>
<td></td>
</tr>
<tr>
<td>03281</td>
<td>JUMPA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03282</td>
<td>JUMPF</td>
<td>$132</td>
<td></td>
</tr>
<tr>
<td>03287</td>
<td>LDA</td>
<td>$135</td>
<td></td>
</tr>
<tr>
<td>03292</td>
<td>LD</td>
<td>$ACTIVE</td>
<td></td>
</tr>
<tr>
<td>03295</td>
<td>LD</td>
<td>$ACTIVE</td>
<td></td>
</tr>
<tr>
<td>03305</td>
<td>FLAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03306</td>
<td>LD</td>
<td>$STK T2</td>
<td></td>
</tr>
<tr>
<td>03312</td>
<td>LDA</td>
<td>EXP1</td>
<td></td>
</tr>
<tr>
<td>03316</td>
<td>JUMPA</td>
<td>$135</td>
<td></td>
</tr>
<tr>
<td>03319</td>
<td>NOT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03320</td>
<td>JUMPF</td>
<td>$136</td>
<td></td>
</tr>
<tr>
<td>03325</td>
<td>LD</td>
<td>&quot;F&quot;</td>
<td></td>
</tr>
<tr>
<td>03326</td>
<td>SWAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03329</td>
<td>RETRN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03330</td>
<td>LDA</td>
<td>$136</td>
<td></td>
</tr>
<tr>
<td>03336</td>
<td>LD</td>
<td>$ACTIVE</td>
<td></td>
</tr>
<tr>
<td>03342</td>
<td>LD</td>
<td>$ACTIVE</td>
<td></td>
</tr>
<tr>
<td>03348</td>
<td>FLAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03349</td>
<td>LDA</td>
<td>$138</td>
<td></td>
</tr>
<tr>
<td>03355</td>
<td>LDA</td>
<td>$138</td>
<td></td>
</tr>
<tr>
<td>03361</td>
<td>LDA</td>
<td>&quot;)&quot;</td>
<td></td>
</tr>
<tr>
<td>03364</td>
<td>STO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03368</td>
<td>LD</td>
<td>$STK T2</td>
<td></td>
</tr>
<tr>
<td>03377</td>
<td>LDA</td>
<td>LITERAL</td>
<td></td>
</tr>
<tr>
<td>03378</td>
<td>JUMPA</td>
<td>$137</td>
<td></td>
</tr>
<tr>
<td>03379</td>
<td>NOT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03384</td>
<td>JUMPF</td>
<td>$139</td>
<td></td>
</tr>
<tr>
<td>03385</td>
<td>LD</td>
<td>&quot;F&quot;</td>
<td></td>
</tr>
<tr>
<td>03387</td>
<td>SWAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03388</td>
<td>RETRN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03389</td>
<td>LD</td>
<td>&quot;T&quot;</td>
<td></td>
</tr>
</tbody>
</table>
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"
03474  SWAP
03475  RETRN
03476  $141  LDA $143
03482  LD $ACTIVE
03488  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
03553 $145  LD  "T"
03556  SWAP
03557  RETRN
03558 $144  LD  "F"
03561  SWAP
03562  RETRN
03563  RETRN
03564  TERM  ENTRPR "2"
03566  DYNAM $146
03567  JUMP $147
03576 $147  LDA $148
03582  LD  $ACTIVE
03588  LD  $ACTIVE
03594  FLAG
03595  LD  $STKT2
03601  LDA PRIMARY
03607  JUMPA
03608 $148  NOT
03609  JUMPF $149
03614  LD  "F"
03617  SWAP
03613  RETRN
03619 $149  LDA $150
03625  LD  $ACTIVE
03631  LD  $ACTIVE
03637  FLAG
03638  LDA $133
03644  LDA $133
03650  LDA "*
03653  STO
03654  LD  $STKT2
03660  LDA LITERAL
03666  JUMPA
03667 $150  JUMPF $152
03672  LDA $153
03678  LD  $ACTIVE
03684  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  $158
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
03848  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
LD $STKT2
LDA LITERAL
JUMPA
JUMPF $168
LDA $169
LD $ACTIVE
LD $ACTIVE
FLAG
LD $STKT2
LDA TERM
JUMPA
$169
JUMPF $170
LD "F"
Swap
RETRN
$170
LDA $171
LD $ACTIVE
LD $ACTIVE
FLAG
LDA $138
LDA $138
LDA "m"
STO
LD $STKT2
LDA OUT
JUMPA
$171
LD "T"
Swap
RETRN
$168
LDA $173
LD $ACTIVE
LD $ACTIVE
FLAG
LDA $158
LDA $158
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
04183  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
04244  JUMP $181
04249  $181  LDA $182
04255  LD $ACTIVE
04261  LD $ACTIVE
04267  FLAG
04268  LD $STKT2
04274  LDA EXP2
04280  JUMPA
04281  $182  NOT
04282  JUMPF $183
04287  LD "T"
04290  Swap
04291  RETRN
04292  $183  LDA $184
04298  LD $ACTIVE
04304  LD $ACTIVE
04310  FLAG
04311  LDA $133
04317  LDA $133
04323  LDA "*
04326  STO
04327  LD $STKT2
04333  LDA LITERAL
04339  JUMPA
04340  $184  JUMPF $186
04345  LDA $187
04351  LD $ACTIVE
04357  LD $ACTIVE
04363  FLAG
04364  LD $STKT2
04370  LDA TERM
04376  JUMPA
04377  $187  NOT
04378  JUMPF $188
04383  LD "F"
04386  Swap
04387  RETRN
LDA $188
LD $ACTIVE
LD $ACTIVE
FLAG
LDA $138
LDA $138
LDA "+"
STO
LD $STKT2
LDA OUT
JUMPA
LDA $189
LDA $183
JUMPA
LDA $186
LDA $191
LD $ACTIVE
LD $ACTIVE
FLAG
LDA $158
LDA $158
LDA "-"
STO
LD $STKT2
LDA LITERAL
JUMPA
JUMPF $193
LDA $194
LD $ACTIVE
LD $ACTIVE
FLAG
LD $STKT2
LDA TERM
JUMPA
JUMPF $194
LDA TERM
LD "F"
SwAP
RETRN
04539 $195  LDA $196
04545  LD  $ACTIVE
04551  LD  $ACTIVE
04557  FLAG
04558  LDA $163
04564  LDA $163
04570  LDA "-"
04576  STG
04577  LD  $STKT2
04580  LDA OUT
04586  JUMPA
04587 $196  LDA $183
04593  JUMPA
04594 $193  LD  "T"
04598  SWAP
04599  RETRN
04600  RETRN
04600 $106  FMT $198  "1"
04606  LDA "ENTER POSTFIX"
04622  EDIT
04622  PUT
04623  JUMP $199

***FORMAT CODE***
04623 $199  "2000380000"
04640 $199  FMT $200  "Z"
04646  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 "INFIX EXPRESSION ="
04718  EDIT
04719  LDA  INPUT  "1+"
04727  EDIT
04728  PUT
04729  JUMP $203
***FORMAT CODE***
04734  $202  "2000180000000028001+
04756  $203  LDA  T  "1+"
04764  LDA  INPUT  "1+"
04772  STO
04775  LDA $204
04779  LD  $ACTIVE
04785  LD  $ACTIVE
04791  FLAG
04792  LD  $STKTI
04796  LDA  EXP1
04804  JUMPA
04805  $204  JUMPF $205
04810  FMT $206  "1"
04816  LDA  "POSTFIX EXPRESSION ="
04838  EDIT
04839  LDA  T  "1+"
04847  LD  "0001"
04853  LD  J
04859  LD  "0001"
04865  SUB
04866  SUBSTR
04867  LDA  ";"
04870  CAT
04871  EDIT
04872  PUT
04873  JUMP $207
***FORMAT CODE***
04878  $206  "20001800000000280000"
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"
04985 $212 STKTP $STK0 , $STK1
04995 JUMP $213
05005 P ENTPR "2"
05007 DYNAM $163
05012 JUMP $215
05017 STKTP $STK2 , $STK3
05023 JUMP $216
05029 Q ENTPR "3"
05037 DYNAM $217
05039 ALLOC $STK4 , F

***DOPE VECTOR CODE***
"104Z0001Z0001"
05052 JUMP $219
05064 $218 LDA F
05070 LD "0001"
05078 INDXA "1"
05078 LD "0001"
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 $STK T4
05166  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 $STK T3
05249  LDA Q
05255  JUMPA
05256 $222  RETRN
05257  ENT PR "2"
05259  DYNAM $226
05264  ALLOC $STK T2 G
***DOPE VECTOR CODE***

"104200012000*"

05288 JUMP $227
05293 $227 ST<TP $STK2 ,$STK3
05304 JUMP $228
05309 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 "01B%" *****
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
05527:  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:  INDXR "1"
05590:  EDIT
05591:  LDA I
05597:  LD I
05603:  LD "0001"
05609:  ADD
05610:  SST
05611:  JUMP $241
05616:  $240 PUT
056177  JUMP  $242

***FORMAT CODE***
056222  $239  "0007"
056229  $242  RETRN
056300  $213  LDA  $243
056306  LD  $ACTIVE
056422  LD  $ACTIVE
056418  FLAG
056490  LD  $STKT1
056555  LDA  R
056611  JUMPA
056622  $243  FMT  $244  "1"
056688  LDA  "EXIT GLOBAL TEST"
056888  EDIT
056888  JUMP  $245

***FORMAT CODE***
056932  $244  "2000180000"
057055  $245  STKTP  $STKT0  $STKT1
057156  ALLOC  $STKT1  ,Z

***DOPE VECTOR CODE***
"104Z0001Z00003"
057400  JUMP  $246
057455  LBL  ENTPK  "2"
057477  DYNAM  $095
057522  JUMP  $248
057575  $248  POPUP
057585  LDA  LABEL
057644  LD  "0003"
057704  INDEX  "1"
057723  RETRN
057733  RETRN
057744  $246  FMT  $249  "1"
057800  LDA  "ENTER LABEL"
057933  EDIT
057944  PUT
057955  JUMP  $250
**FORMAT CODE**

05800 $249 "2000380000"
05812 $250 LDA Y
05818 LD "01,1"
05824 STD
05825 STKTP $STKT1, $STKT2
05836 JUMP $251
05841 $251 LDA I
05847 LD "0001"
05853 STD
05854 LD "01,2"
05860 JUMPA
05864 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

**FORMAT CODE**

05913 $252 "2000180000*0005"
05930 $253 LDA A
05935 LD I
05942 LD "0001"
05948 ADD
05949 STD
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"
05999: $255 FMT $256 "1"
06005: LDA "INCORRECT LABEL TEST"
06027: EDIT
06028: PUT
06029: JUMP LBL2
***FORMAT CODE***
06034: $256 "80000"
06041: LBL2 FMT $258 "1"
06047: LDA "LABEL TEST"
06059: EDIT
06060: LD A
06066: EDIT
06067: PUT
06068: 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"
06145: LD "01J-"
06152: STO
06153: LDA BADLAB
06159: LD $ACTIVE
06165: LD $ACTIVE
06171: FLAG
06172: LDA Z
06173: LD $STK1
06174: LDA LBL
06179: JUMPA
06180: BADLAB FMT $261 "1"
0618E: LDA "INCRREFT LABEL RETURN"
06191: LDA "INCORRECT LABEL RETURN"
06192: EDIT
06193: PUT
06223: 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"
06303: EDIT
06305: 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 4.300000000E+015 -1.120000000E-007
5.1234 4.3E+14 -1.2E-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
\begin{align*}
I &= 5, \ M(I,1) = 1, \ M(I,5) = 0 \\
I &= 4, \ M(I,1) = 1, \ M(I,5) = 0 \\
I &= 3, \ M(I,1) = 1, \ M(I,5) = 0 \\
I &= 2, \ M(I,1) = 1, \ M(I,5) = 0 \\
I &= 1, \ M(I,1) = 1, \ M(I,5) = 0 \\
I &= 0, \ M(I,1) = 1, \ M(I,5) = 0 \\
I &= -1, \ M(I,1) = 1, \ M(I,5) = 0 \\
I &= -2, \ M(I,1) = 1, \ M(I,5) = 0 \\
\end{align*}

EXIT DO GROUPS

ENTER ARITHMETIC BLOCK
\begin{align*}
1.330 & \quad 1.330 & \quad 1.0 \\
1 & \quad & \\
2.650 & \quad 1.330 & \quad 2.0 \\
2 & \quad & \\
3.990 & \quad 1.330 & \quad 3.0 \\
3 & \quad & \\
5.320 & \quad 1.330 & \quad 4.0 \\
4 & \quad & \\
6.650 & \quad 1.330 & \quad 5.0 \\
5 & \quad & \\
7.980 & \quad 1.330 & \quad 6.0 \\
6 & \quad & \\
9.310 & \quad 1.330 & \quad 7.0 \\
7 & \quad & \\
10.640 & \quad 1.330 & \quad 8.0 \\
8 & \quad & \\
11.970 & \quad 1.330 & \quad 9.0 \\
9 & \quad & \\
13.30 & \quad 1.330 & \quad 10.0 \\
10 & \quad & \\
\end{align*}

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*/3L/M/+;
EXIT POSTFIX

GLOBAL DISPLAY TEST

14 14 14 14 14 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 = 8209****
// CONTROL RECORD.
FUNCTION PLXCP.
INTERPRETER=MTXINT04.
GO=NO.
STORE=YES.

//
DECLARE INPUT CHAR(80), (CRDCNT, I, SYMCNT) 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, I, 1) = "" THEN SUBSTR (DECNT, I, 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 > < 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(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 PLXCLI
INTERPRETER=MIXINT04

//
ERROR: PROCEDURE MAIN;

IDENTIFICATION:

PROGRAM-ID: ERROR.

AUTHOR: J. R. VAN DOREN.

SOURCE LANGUAGE: PLEX.

PURPOSE:

ERROR DEMONSTRATES THE ERROR DIAGNOSTICS OF PLXCL.

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****-I;

****ERROR**** F: SYNTAX
END;
END ERROR;
FATAL ERROR(S) ENCOUNTERED, JOB ABORTED
00010  PROG PLXINT
00020  SEG  00
00030***********************************************************************
00040  * *
00050  IDENTIFICATION:
00060  -----------------
00070  *
00080  PROGRAM-ID:  PLXINT.
00090  AUTHOR:  J. R. VAN DOREN.
00100  SOURCE LANGUAGE:  EASYCODER.
00110  SOURCE COMPUTER:  H-1200
00120  OBJECT COMPUTER:  H-1200
00130  *
00140  PURPOSE:
00150  -------
00160  *
00170  PLXINT INTERPRETIVELY EXECUTES OBJECT PROGRAMS PRODUCED
00180  BY THE PLEX COMPILER FOR THE PLEX LANGUAGE.
00190  *
00200***********************************************************************
00210  ADMODE4 ASSEMBLE IN FOUR CHAR ADDRESSING MODE
00220  ORG  45056 EXECUTION LOCATION
00230***********************************************************************
00240  *
00250  OCTAL ADDRESS DEFINITIONS OF PERTINENT SYMBOLS IN THE RESIDENT
00260  INPUT/OUTPUT ROUTINE.
00270  *
00280***********************************************************************
00290  #RDWR CEQU =4C00000754
00300  READ CEQU =4C00005430
00310  INPUT CEQU =4C00006144
00320  OUTPUT CEQU =4C00006265
00330  PRINT CEQU =4C00005647
00340  #SKP CEQU =4C00000756
00350  PRNTBF EQU OUTPUT+1
00360***********************************************************************
00370  *
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 8 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

FB/BD 4, DB, FLOATING BINARY TO FLOATING DECIMAL
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

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 DYNEND-2
MLWDR IR14,3+X14
MLWDR IR14,7+X14
MCw :;OUTPUT+132
MCw OUTPUT+132
MCw =1C21,OUTPUT
SI INPUT+80
:Skip PRINT+57,
LCA +PRNTBF,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
<table>
<thead>
<tr>
<th>OpCode</th>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0000</td>
<td>0070</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>0040</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>0044</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>003C</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>0038</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>0034</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>0030</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>002A</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>0026</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>0022</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>0020</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>001F</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>001E</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>001D</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>001B</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>0019</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>0017</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>0015</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>0013</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>0011</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>000F</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>000E</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>000C</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>000A</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>0008</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>0006</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>0004</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>0002</td>
</tr>
<tr>
<td>00</td>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>
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 BCE LITADD,STYPE,70   CHECK FOR LITERAL OPERAND
01870 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 IR4-3   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 IR9   ELSE COMPUTE 2ND (DYNAMIC ONLY)
02010 MRSD 1×X3,IR9
02020 BA IR9
02030 BA IR9
02040 BA IR14,IR9
02050 MRID 0×X9,IR5-3
02060 BA 4×X3,IR5
02070 BS IR5-3
02080 BA =1B5,IR3   CLEAR HIGH BITS
02090 B 0×X13   RETURN
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
02160 INDIRA MRID 0×X4,IR4-3   GET ACTUAL ADDRESS
02170 B 0×X13   RETURN
02180 NOCOMP MRID 2×X3,IR4-3   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)

<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>02280</td>
<td>DYNAM BA 3+X3,3+X14</td>
<td>INCREMENT STACKTOP</td>
</tr>
<tr>
<td>02290</td>
<td>C 3+X14,DYNEND</td>
<td>CHECK FOR DYNAMIC STORAGE</td>
</tr>
<tr>
<td>02300</td>
<td>BL DYNQFL</td>
<td>OVERFLW</td>
</tr>
<tr>
<td>02310</td>
<td>BA =1B4,IR3</td>
<td>INSTRUCTION COUNTER</td>
</tr>
<tr>
<td>02320</td>
<td>B FETCH</td>
<td></td>
</tr>
</tbody>
</table>

STKTP - INITIALIZE NEW STACKTOP (BLOCK ENTRY)

<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>02380</td>
<td>STKTP MRIDR 0+X4,0+X5</td>
<td></td>
</tr>
<tr>
<td>02390</td>
<td>B FETCH</td>
<td></td>
</tr>
</tbody>
</table>

ALLOC - SET DOPE VECTOR FOR INDICATED ARRAY

<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>02490</td>
<td>ALLOC MRSD 0+X3,IR10</td>
<td>INSERT SUBSCRIPT COUNT</td>
</tr>
<tr>
<td>02500</td>
<td>SAR IR3</td>
<td></td>
</tr>
<tr>
<td>02510</td>
<td>MLWD IR4,IR11</td>
<td>SAVE LOCATION OF STACK TOP VALUE</td>
</tr>
<tr>
<td>02520</td>
<td>MLWD IR5,IR12</td>
<td>SAVE DYNAMIC DOPE VECTOR ADDRESS</td>
</tr>
<tr>
<td>02530</td>
<td>MRIDI 0+X3,LNGTH-1</td>
<td>MOVE IN ELEMENT LENGTH SIZE</td>
</tr>
<tr>
<td>02540</td>
<td>SAR IR3</td>
<td></td>
</tr>
<tr>
<td>02550</td>
<td>MCW LNGTH,SIZE</td>
<td>INIT ARRAY SIZE COUNTER</td>
</tr>
<tr>
<td>02560</td>
<td>BS CONPRT</td>
<td>CLEAR</td>
</tr>
<tr>
<td>02570</td>
<td>MCW =2C0120,IR7</td>
<td>SET OP CODE REGISTER TO ONE ADDRESS CODE</td>
</tr>
<tr>
<td>02580</td>
<td>MORSUB MCW +ALCRTL,IR13</td>
<td>RETURN ADDRESS FROM ADCOMP</td>
</tr>
<tr>
<td>02590</td>
<td>B ADCOMP</td>
<td>CALL ADCOMP</td>
</tr>
</tbody>
</table>
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+CONPR  UPDATE CONSTANT PART COMPUTATION
02710 BA 3+X4+CONPR  ADD IN LOWER BOUND
02720 BS =1B1+IR10  
02730 BCE DVDNE,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,CONPR  LENGTH FACTOR
02790 BS CONPR+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 DYNFL  
02830 B FETCH  
02840 LNGTH DCW =4B0  
02850 SIZE DCW =4B0  
02860 CONPR DCW =4B0  
02870***************************************************************************

02880* LDA - LOAD ADDRESS TO THE STACK, INCLUDING THE *
02890* DATATYPE CODE. IF A CHARACTER STRING ALSO LOAD *
02900* THE MAX STRING LENGTH. *
02910* ***************************************************************************
02920* 02930***************************************************************************
02940 LDA MLWDR IR4+4+X2  ADDRESS TO STACK
02950 LCA DTYP+O+X2  DATA TYPE
02960 BCE LDLNG,DTYPE,*0  IF A CHARACTER STRING LOAD LENGTH
LD - LOAD AN OPERAND TO THE STACK, INCLUDING DATA TYPE.

LD MRIDR 0+X4,l+X2 LOAD DATA
BCE CHKTYP,DTYPE,07 IF UNDETERMINED TYPE CHECK IT
LCA DTYPE,0+X2 SET TYPE
B UPSTCK
CHKTYP BI 3+X4,SETINT INTEGER
BI 7+X4,SETFLT FLOAT
B TYPERR
SETINT LCA =1B1,0+X2
B UPSTCK
SETFLT LCA =1B2,0+X2
B UPSTCK

STO - STORES THE ITEM AT THE TOP OF THE STACK AT THE ADDRESS NEXT TO THE TOP. NOTE SPECIAL HANDLING OF CHARACTER STRINGS. POP TWO STACK ELEMENTS UPON COMPLETION.

STO B UPSTCK

SST - SAME AS STO EXCEPT ADDRESS IS DISCARDED BUT THE STACK TOP IS RETAINED.

SST B UPSTCK
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

0335C  STO  EUU  *
0336C  SST  EUU  *
0337C  BCE  CNVRTN+0-18*+X2,04  BYPASS IF STRING OR SUBSTRING
0338C  BCE  CNVRTN+0-18*+X2,34
0339C  C    0-9*+X2,0-18*+X2  IF TYPES NOT EQUAL
0340C  BNE  SCNV  DO CONVERSION
0341C  CNVRTN  MCW  0-14*+X2,IR4  RECEIVING ADDRESS
0342C  BCE  CHRSTR+0-9*+X2,04  SPECIAL HANDLING IF CHARACTER STRING
0343C  BCE  CSBSTR+0-9*+X2,34  CHECK FOR SENDING SUBSTRING
0344C  MRIDR  0-8*+X2,0+X4  STORE IT
0345C  SSTCHK  BCE  PRRMO+0-9*+X2,34  SPECIAL HANDLING IF CHARACTER STRING
0346C  PNCRTN  BCE  SAVTOP+0-1*+X3,23  CHECK FOR SAVE AND STORE (SST)
0347C  BS   =1B18,IR2  POP STACK TWO ELEMENTS
0348C  MCW  +ENDPRG,ENDADR  RESTORE STRING WORKING STORAGE POINTER
0349C  B    FETCH
0350C  SAVTOP  SI  0-9*+X2  PUNCTUATION FOR MOVE
0351C  MLRDR  0-1*+X2,0-10*+X2  DISCARD ADDRESS
0352C  CI    0-9*+X2,0-18*+X2  CLEAR ITEM MARKS
0353C  BS   =1B9,IR2  AND SAVE STACK TOP ON TOP
0354C  B    FETCH
0355C  SCNV  BCE  CNVRTN+0-18*+X2,07  UNIVERSAL TYPE O.K.
0356C  BCE  FIXFLT+0-9*+X2,01  UNIVERSAL TYPE O.K.
0357C  TMA  0-1*+X2,00  GO CONVERT
0358C  B    IFIX
0359C  R    CNVERR  CONVERSION OVERLOW INSTRUCTION
0360C  MCW  IR15+0-5*+X2  INTEGER RESULT
0361C  SI    0-5*+X2  TO
0362C  LCA  =1B1,0-9*+X2  STACK
0363C  B    CNVRTN
0364C  FIXFLT  MLWDI  =4C00000027,0-1*+X2  NORMALIZED LOAD
0365C  AMA  0-1*+X2,70  CLEAR EXTRANEOUS PUNCTUATION
0366C  MLWDI  =8B0,0-1*+X2  FLOATING
0367C  TAM  0-1*+X2,00
0368C  SI    0-1*+X2  RESULT
0369C  LCA  =1B2,0-9*+X2  TO STACK
0370C  B    CNVRTN
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>03710</td>
<td>CHKSTR BS IR5</td>
<td>MAX LENGTH FOR UNIVERSAL TYPE</td>
</tr>
<tr>
<td>03720</td>
<td>MCW (=1C10,IR5)</td>
<td>CHECK FOR UNIVERSAL TYPE</td>
</tr>
<tr>
<td>03730</td>
<td>BCE CHRTRN,0-18+X2,07</td>
<td>MAX LENGTH OF RECEIVING FIELD</td>
</tr>
<tr>
<td>03740</td>
<td>MRID 0-10+X2,IR5-1</td>
<td>TRANSMITTING ADDRESS</td>
</tr>
<tr>
<td>03750</td>
<td>CHRTRN MCW 0-5+X2,IR9</td>
<td>POTENTIAL LEFT MOVE MARKER</td>
</tr>
<tr>
<td>03760</td>
<td>Sw 0(+X9)</td>
<td>CLEAR</td>
</tr>
<tr>
<td>03770</td>
<td>BS IR10</td>
<td>END OF PROPOSED MOVE</td>
</tr>
<tr>
<td>03780</td>
<td>MRI 0+X9,0+X4</td>
<td>RECEIVING ADDRESS PLUS MAX LENGTH</td>
</tr>
<tr>
<td>03790</td>
<td>SBR IR10</td>
<td>IF MOVE TOO LONG THEN TRUNCATE IT</td>
</tr>
<tr>
<td>03800</td>
<td>EA IR4,IR5</td>
<td>IF SUB STRING RECEIVING BE CARFUL</td>
</tr>
<tr>
<td>03810</td>
<td>C IR5,IR10</td>
<td>MOVE ENTIRE TRANSMITTING FIELD</td>
</tr>
<tr>
<td>03820</td>
<td>BH TMOVE</td>
<td>MOVE ONLY DATA, NO PUNCTUATION</td>
</tr>
<tr>
<td>03830</td>
<td>BCE (=1C10,IR5)</td>
<td></td>
</tr>
<tr>
<td>03840</td>
<td>MRIDR 0+X9,0+X4</td>
<td></td>
</tr>
<tr>
<td>03850</td>
<td>SUBMV MRI 0+X9,0+X4</td>
<td></td>
</tr>
<tr>
<td>03860</td>
<td>B SSTCHK</td>
<td></td>
</tr>
<tr>
<td>03870</td>
<td>SUBMV MRI 0+X9,0+X4</td>
<td></td>
</tr>
<tr>
<td>03880</td>
<td>B SSTCHK</td>
<td></td>
</tr>
<tr>
<td>03890</td>
<td>TMOVE BS IR4,IR5</td>
<td></td>
</tr>
<tr>
<td>03900</td>
<td>BA IR5,IR9</td>
<td></td>
</tr>
<tr>
<td>03910</td>
<td>BA IR4,IR5</td>
<td></td>
</tr>
<tr>
<td>03920</td>
<td>MLWDR 0-1+X9,0-1+X5</td>
<td></td>
</tr>
<tr>
<td>03930</td>
<td>CSSTR EQU *</td>
<td></td>
</tr>
<tr>
<td>03940</td>
<td>MCW 0-5+X2,IR15</td>
<td>TRANSMITTING FIELD IS A SUBSTRING</td>
</tr>
<tr>
<td>03950</td>
<td>BS IR9</td>
<td></td>
</tr>
<tr>
<td>03960</td>
<td>MRI 0-4+X2,IR9-1</td>
<td></td>
</tr>
<tr>
<td>03970</td>
<td>BA IR9,IR15</td>
<td></td>
</tr>
<tr>
<td>03980</td>
<td>EI NOSET,0-1+X15</td>
<td></td>
</tr>
<tr>
<td>03990</td>
<td>SI 0-1+X15</td>
<td></td>
</tr>
<tr>
<td>04000</td>
<td>B CHRSTR</td>
<td></td>
</tr>
<tr>
<td>04010</td>
<td>NOSET SST =1B0,0-9+X2,70</td>
<td></td>
</tr>
<tr>
<td>04020</td>
<td>B CHRSTR</td>
<td></td>
</tr>
<tr>
<td>04030</td>
<td>PRMOVE CI 0-1+X15</td>
<td></td>
</tr>
<tr>
<td>04040</td>
<td>MCW 0-5+X2,IR9</td>
<td></td>
</tr>
<tr>
<td>04050</td>
<td>CW 0+X9</td>
<td></td>
</tr>
<tr>
<td>04060</td>
<td>B PNCRTN</td>
<td></td>
</tr>
</tbody>
</table>
04080*    FLAG= MARK STACK FOR THE LIMIT OF PARAMETER ADDRESSES
          * PRIOR TO A PROCEDURE CALL
          *
04120*    ENTPRO- ESTABLISH DYNAMIC STORAGE DISPLAY, STORE PARAMETER
          * ADDRESSES IN NEW DYNAMIC STORAGE AREA AND THEN PASS
          * CONTROL TO THE CALLLD PROCEDURE.
04210*    ENTPRO MRID 0-8+X2,IR14-3    NEW DISPLAY AND STORAGE AREA
04230    BS     =1B9,IR2
04240    BS     IR13
04250    MRSID 0+X3,IR13    LEVEL NUMBER OF PROCEDURE
04260    SAR    IR3    BUMP SEQUENCE COUNTER
04270    BIM    =4B4,IR13
04280    BA     =1B4,IR13
04290    BA     IR14,IR13    LOCATION FOR PARAMETER ADDRESSES
04300    BS     PRMCNT
04310    FLGCHK BS     =1B9,IR2    SEARCH DOWN
04320    BCE    PRMSTO=0+X2,77 AND
04330    BA     =1B1,PRMCNT COUNT
04340    B      FLGCHK PARM5
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
BS = 1818, IR2
RETURN ADDRESS
MRIDR IR14 = 3 * IR14
1ST DISPLAY ENTRY (STACK TOP VALUE)
SBR IR12
NEXT DISPLAY LOCATION
MRSD 0 = 1 * X3, IR13
MOVE GLOBALLY ACTIVE STORAGE LEVEL
DISPLY MRI D R 0 = X9, 0 = X13
ADDRESSES TO CURRENT DISPLAY
SAR IR9
CURRENT DISPLAY BASE ADDRESS
SBR IR12
B DISPLY
MRSD 0 = 1 + X3, IR13
DISPLY MRIDS 0 = X9, 0 = X12
SAR IRQ
BCE CURLEV, IR13, 01
CURL V MRI D R IR14 = 3 * 0 = X12
B FETCH
PRMCNT DCw = 180
***************************************************************************
* *
SWAP - SWAP THE TOP TWO ELEMENTS OF THE STACK *
* *
***************************************************************************
SWAP SI 0 = 9 * X2 = 0 = 18 * X2
MRLUR 0 = 1 * X2 = HOLD
MLURD 0 = 10 = X2 = 0 = 1 = X2
MLRDR HOLD = 0 = 10 = X2
CI 0 = 9 * X2 = 0 = 18 = X2
B FETCH
HOLD DCw = 9
***************************************************************************
* *
RETURN - RESTORE DISPLAY ADDRESS FOR CALLING PROCEDURE *
AND SET INSTRUCTION COUNTER WITH RETURN ADDRESS *
***************************************************************************
RETN MRI D 0 = 8 = X2 = IR3 = 3
RETURN ADDRESS TO INST COUNTER
MRI D 0 = 4 = X2 = IR14 = 3
DISPLAY AT POINT OF CALL
BS = 189, IR2
B FETCH
***************************************************************************
*
04820* JUMPA - JUMP TO THE ADDRESS IN THE STACK; POP THE STACK
04830* POPUP - POP THE STACK
04840* wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
COMPC—COMPARE THE CHARACTER STRING WHOSE ADDRESS IS 2ND IN THE
STACK AGAINST THE STRING WHOSE ADDRESS IS 1ST. IF THE
STRINGS ARE OF UNEQUAL LENGTH THE SHORTER IS MOVED AND
PADDED ON THE RIGHT WITH BLANKS.
THE COMPARISON CONDITION IS CONTAINED IN THE CHARACTER
FOLLOWING THE OP CODE.

COMPC BS IR9 CLEAR
BS IR10
BS IR15
MCW 0-14+x2,IR13 LEFT ADDRESSES OF STRINGS
MCW 0-14+x2,IR13
BCE SUBMV1,0-18+x2,34 IF SUBSTRING PUT IN WORKING STORE
BCE SUBMV2,0-9+x2,34 IF SUBSTRING PUT IN WORKING STORE
MRIN 0+x13,0 DETERMINE
MRIN 0+x12,0
SAR IR9 STRING
SAR IR10
BS IR13,IR9
BS IR12,IR10 LENGTHS
SW IR9=2
C IR10,IR9 TEST
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  ENDR+IR13
05750       MCw  ENDR+IR11
05760       BA    IR10,IR11   RIGHT END OF PADDED FIELD
05770       MRBLNK MRSDR  BLNK,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)
05850       SBR  IR15
05860       CI    0-1+X15
05870       MCw  ENDR+IR12
05880       MCw  ENDR+IR11
05890       BA    IR9,IR11   RIGHT END OF PADDED FIELD
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 INTADD,ARI,01  
06310 TMA 0-1X2+00  
06320 AMA 0-10+X2+00  
06330 SETBCK FBI FERR+06  
06340 BS =189+IR2  
06350 MLWDR =8B0+0-1+X2  
06360 SI 0-1+X2  
06370 TAM 0-1+X2+00  
06380 LCA =1B2+0-9+X2  
06390 B FETCH  
06400 INTADD SW 0-17+X2  
06410 BA 0-5+X2+0-14+X2  
06420 BS =189+IR2  
06430 B FETCH  
06440 SUB B CNVCHK  
06450 BCE INTSUB,ARI,01  
06460 TMA 0-10+X2+00  
06470 SMA 0-1+X2+00  
06480 B SETBCK  
06490 INTSUB SW 0-17+X2  
06500 BS 0-5+X2+0-14+X2  
06510 BS =189+IR2  
06520 B FETCH  
06530 MULT B CNVCHK  
06540 BCE INTMLT,ARI,01  
06550 TMA 0-1+X2+00  
06560 MAM 0-10+X2+00  
06570 B SETBCK  
06580 INTMLT BIM 0-5+X2+0-14+X2  
06590 BS =189+IR2  
06600 B FETCH  
06610 DIV MRSD 0-9+X2+ARI  
06620 BA 0-18+X2+ARI  
06630 BCE NUMER,0-9+X2+02  
06640 MLWDI =4C00000027,0-1+X2  
06650 NUMER BCE DODIV,0-18+X2+02  
06660 MLWDI =4C00000027,0-10+X2  

STACK TOP TO FRO  
ADD NEXT  
BRANCH ON ANY ERROR  
REDUCE STACK  
CLEAR EXTRANEOUS PUNCTUATION  
PROPER PUNCTUATION  
STORE RESULT  
DATA TYPE  
WORD MARK TO STOP ADDITION  
BINARY ADD  
REDUCE STACK  
IF INTEGERS BRANCH  
FLOATING POINT  
WORD MARK TO STOP ARITHMETIC  
INTEGER SUBTRACTION  
IF INTEGERS BRANCH  
FLOATING POINT  
SUM OF TYPES FOR LATER TESTING  
CONVERT TO UNNORMALIZED FLTNG PT  
CONVERT TO UNNORMALIZED FLTNG PT
NORMALIZED LOAD TO FR1
NORMALIZED LOAD TO FRO
RESULT IN FRO
CHECK FOR INTEGER RESULT
SETBCK
GO CONVERT
CONVERSION OVERFLOW INSTRUCTION
PUT RESULT IN STACK
IF INTEGER THEN BRANCH
ELSE SUBTRACT FROM NORMAL ZERO
AND STORE
CLEAR REGISTER
SUBTRACT
PUT BACK
SET RETURN
SET
INDICATOR
DO
NECESSARY
CONVERSION
OR BS =1B9,IR2
0704C  BCE  BSTT,1+X2,T       ALREADY PROPERLY TRUE OR FALSE
0705C  B   FETCH
0706C  AND  BS =1B9,IR2
0707C  BCE  BSTF,1+X2,F
0708C  B   FETCH
0709C  NOT  BCE  BSTF,0-8+X2,T  ALREADY PROPERLY TRUE OR FALSE
0710C  BSTT  MCw :T:0-8+X2  SET TRUE ON STACK
0711C  SI  0-8+X2  PUNCTUATION FOR POSSIBLE STORE
0712C  B   FETCH
0713C  BSTF  MCw :F:0-8+X2  SET FALSE ON STACK
0714C  SI  0-8+X2  PUNCTUATION FOR POSSIBLE STORE
0715C  B   FETCH
0716C ****************************
0717C  *
0718C  INDEXA-COMPUTE ADDRESS ACCORDING TO DOPE VECTOR (ADDRESS IN STACK) AND INDECES ON THE STACK. RESULTING ADDRESS ON TOP OF THE STACK.
0719C  *
0720C  ON TOP OF THE STACK.
0721C  *
0722C  INDXR-SAME AS INDEXA EXCEPT THAT ADDRESSED ITEM IS LOADED TO THE STACK.
0723C  *
0724C  ****************************
0725C  INDEXA EQU  *
0726C  INDEXR EQU  *
0727C  BS  IR13  CLEAR
0728C  MRS0  O+X3,IR13  INSERT SUBSCRIPT COUNT
0729C  SUBCHK  BCE  INTSBS+0-9+X2+01  CHECK FOR
0730C  TMA  0-1+X2+00  SUBSCRIPT CONVERSION
0731C  B   IFIX
0732C  R   B  CNVERR
0733C  MCw  IR15+0-5+X2  MOVE INTEGER RESULT TO THE STACK
0734C  INTSBS  BS =1B9,IR2
0735C  BS  =1B1,IR13
0736C  BCE  CMPADD+IR13+00
0737C  B   SUBCHK
0738C  CMPADD  MCw  IR2+IR9  SAVE STACK POINTER
0739C  MCw  0+5+X2,IR10  ADDRESS OF DOPE VECTOR
0740C  MRS0  O+X3,IR13  SUBSCRIPT COUNT
07410 SAR IR3
07420 MRI0 1+X2,IR15-3
FIRST SUBSCRIPT
07430 MORIDX BS =IR13,IR13
REDUCE SUBSCRIPT COUNT
07440 BCE SUBDNE,IR13,00
IF NO MORE THEN FINISH
07450 BA =IR10,IR10
POINT TO NEXT MULTIPLIER
07460 BA =IR2,IR2
NEXT SUBSCRIPT
07470 BIM 3*X10,4*X2
SUBSCRIPT * MULTIPLIER
07480 BA 4*X2,IR15
ACCUMULATE VARIABLE PART
07490 B MORIDX
07500 SUBDNE BIM 7*X10,IR15
MULTIPLY TIMES ELEMENT SIZE
07510 BA 11*X10,IR15
ADD CONSTANT PART
07520 MCW IR9,IR2
RESTORE STACK POINTER
07530 C IR15,DYNEND
CHECK FOR EXTREMELY WILD INDEX
07540 BL IDXERR
07550 MLW0 IR15,0-5+X2
ADDRESS TO STACK
07560 BCE FETCH,0-2+X3,67
IF INDEXA THEN DONE
07570 MRIDR 0+X15,0-8+X2
ELSE LOAD RESULT
07580 B FETCH
07590******************************************************************************
07600******************************************************************************
076100 CAT -CONCATENATE TWO STRINGS WHOSE ADDRESSES ARE IN THE TOP
076200 TWO POSITIONS OF THE STACK. THE RESULTING STRING IS PLACED
076300 IN WORKING STRING STORAGE AND THE TWO ADDRESSES ARE REPLACED
076400 BY THE ADDRESS IN WORKING STORAGE.
076500******************************************************************************
076600******************************************************************************
076670 CAT MCW 0-14+X2,IR13
ADDRESS TO REGISTER
076800 BCE CTSUB1,0-18+X2,34
CHECK FOR SUBSTRING
076900 MRIDR 0+X13,(ENDADR-3)
077000 SBR IR12
077100 CI 0-1+X12
CLEAR ITEM MARK
077200 CAT2 MCW 0-5+X2,IR13
077300 BCE CTSUB2,0-9+X2,34
CHECK FOR SUBSTRING
077400 MRIDR 0+X13,0+X12
077500 SBR IR13
NEXT LOCATION IN WORKING STORAGE
077600 CAT3 CW 0+X12
CLEAR POSSIBLE WORD MARK
077700 MCW ENDADR,0-14+X2
ADDRESS OF RESULTING STRING
0778C  MCW IR13,ENDADR  NEXT LOCATION IN WORKING STORAGE
0779C  BS =1B9,IR2  POP STACK
0780C  B FETCH
0781C  CTSUB1 BS IR10
0782C  MRID 0-13+X2,IR10-1  MOVE
0783C  BA IR13,IR10  SUBSTRING
0784C  SI 0-1+X10  TO
0785C  MRIDR 0+X13,(ENDADR-3)  WORKING
0786C  SBR IR12  STORAGE
0787C  CI 0-1+X12
0788C  CI 0-1+X10
0789C  Sw (ENDADR-3)
0790C  SST =1B0,0-18+X2,70  REMOVE SUB STRING MARK
0791C  B CAT2
0792C  CTSUB2 BS IR10
0793C  MRID 0-4+X2,IR10  CATENATE
0794C  BA IR13,IR10  SUBSTRING
0795C  SI 0-1+X10  TO
0796C  MRIDR 0+X13,0+X12  WORKING
0797C  SBR IR13  NEXT LOCATION IN WORKING STORAGE
0798C  CI 0-1+X10  STORAGE
0799C  B CAT3

0800C***************************************************************************
0801C  SUBSTR - PL/I SUBSTRING OPERATION
0802C***************************************************************************
0803C***************************************************************************
0804C***************************************************************************
0805C  SBSTR Sw 0-26+X2  COMPUTE
0806C  BA 0-14+X2,0-23+X2  STARTING
0807C  BS =1B1,0-23+X2  POINT OF STRING
0808C  C 0-5+X2,00:  IF NO LENGTH GIVEN
0809C  BE FNDLNG  COMPUTE IT
0810C  MRIDI 0-6+X2,0-22+X2  ELSE STORE LENGTH
0811C  SBSTRL BS =1B18,IR2  POP STACK
0812C  LCA =1C34,0-9+X2  MARK AS SUBSTRING ELEMENT
0813C  B FETCH
0814C  FNDLNG MCW 0-23+X2,IR13  FIND
08150: MRIN 0+X13,0  CURRENT REMAINING
08160: SAR IR4  LENGTH
08170: BS IR13,IR4  OF
08180: MREDI IR4-1,0-22+X2  STRING
08190: B SBSTRRL
08200:***************************************************************************
08210: * FMT - SETS THE ADDRESS OF THE FORMAT CODE AND INDICATORS *
08220: * FOR STRING OR UNIT RECORD AND WHETHER INPUT OR OUTPUT. *
08230: * IF STRING I/O THE ADDRESS AND LENGTH ARE EXTRACTED FROM *
08240: * THE STACK AND THE APPROPRIATE BUFFER POINTER IS SET TO *
08250: * THE BEGINNING OF THE STRING. ( *
08260: **************************************************************************
08270: FMT MRID 0+X3,FMTCD1-3  SET FORMAT CODE ADDRESS
08280: SAR IR3  BUMP SEQUENCE COUNTER
08290: BA =1B1,FMTCD1  JUMP OVER BEGINNING MARKER
08300: MCW FMTCD1,FMTCD2  SAVE FOR REPETITION
08310: SST 0+X3,INOROT,70  INPUT OR OUTPUT CODE
08320: SST 0+X3,DEVTYP,07  STRING OR UNIT RECORD
08330: BA =1B1,IR3  BUMP SEQUENCE COUNTER
08340: BCE STRSET,DEVTYP,00  IF STRING INITIALIZE
08350: BCE FETCH,INOROT,70  ELSE INITIALIZE FOR UNIT RECORD
08360: MCW +PRNTBF,IR8  OUTPUT PRINT BUFFER
08370: MCW +PRNTBF+132,STREND
08380: B FETCH
08390: FMTCD1 DCW =4
08400: FMTCD2 DCW =4
08420: INOROT DCW ;0;
08430: DEVTYP DCW ;0;
08440: STRADD DCW =4
08450: LNGSTR DCW =4B0
08460: STREND DCW =4
08470: STRSET MRID 0=8+X2,STRADD-3  STRING ADDRESS
08480: MRID 0=4+X2,LNGSTR-1  LENGTH
08490: MCW STRADD,STREND
08500: BA LNGSTR,STREND  STRING END ADDR (PLUS ONE)
08520 BS =IR9,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 MRRDR BLNK,0+X9 CLEAR OUTPUT
08570 SBR IR9 STRING
08580 C IR9,STREND TO
08590 BH CLRMOR BLANKS
08600 SI 0-1+X9
08610 B FETCH
08620***************************************************************************
08630* PUT- IF OUTPUT IS TO THE PRINTER PRINT AND RESET
08640* OUTPUT BUFFER POINTER.
08650* IF OUTPUT IS TO A STRING SET RIGHT END PUNCUATION
08660* AND RESET BUFFER POINTER TO PRINTER BUFFER.
08670* *
08680* *
08690***************************************************************************
08700 PUT BCE STRPUT,DEVTYP,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 0-1+X8 RIGHT END TERMINATOR
08790 B PBUFF
08800***************************************************************************
08810* *
08820* EDIT- CONVERT INTERNAL DATA TO OUTPUT FORMAT ACCORDING
08830* TO FORMAT CODE. ALSO EXECUTE CONTROL FORMAT INSTRUCTIONS.
08840* ADDRESS OF ITEM IS AT THE TOP OF THE STACK.
08850* *
08860***************************************************************************
08870 EDIT BCE FMSTRST,(FMTCD1-3),77 CHECK IF RESET IS REQUIRED
08880 BCE DATFMT,(FMTCD1-3),70 CHECK FOR DATA FORMAT
ELSE IT MUST BE CONTROL FORMAT

SPACE THE REQUIRED NUMBER OF COLUMNS

SET THE BUFFER POINTER TO THE PROPER CHAR

POSITION TO PROPER CHARACTER

ONE ORIGIN INDEX FOR BUFFER POINTER

THE PROPER CHARACTER

TOP OF PAGE

FLUSH OUT CURRENT BUFFER

CLEAR FOR BCE TEST

BUMP FORMAT CODE POINTER

IF STRING THEN ERROR

FLUSH OUT CURRENT BUFFER

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  *  CHECK FOR A FORMAT CODE
09340  BCE  AFORM, (FMTCD1-3), 10  CHECK FOR A FORMAT CODE
09350  BCE  EFORM, (FMTCD1-3), 11  CHECK FOR A FORMAT CODE
09360  BCE  IFORM, (FMTCD1-3), 12  CHECK FOR A FORMAT CODE
09370  BCE  LFORM, (FMTCD1-3), 13  CHECK FOR A FORMAT CODE
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 FIELD
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  GTSBLN, 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  GTSBLN  MRID  0-4+X2*IR10-1  LENGTH FROM STACK
09620  B  AFRMA
09630  BUFOFL EQU  *
09640  :PUT PRINT,BFOFL*
09650  B ERRLOC
09660  BFOFL DCW :A  ****BUFFER OVERFLOW****:
09670 L DCW =1C45
09680 IFORM BA =1B1,FMTCD1 BUMP FORMAT CODE POINTER
09690 MR ID (FMTCD1-3),IR10-3 I FIELD LENGTH
09700 SAR FMTCD1 NEXT FORMAT CODE POINTER
09710 BA IR8,IR10 BUFFER POINTER PLUS LENGTH
09720 C IR10,STREND CHECK
09730 BL BUFOFL OVERFLOW
09740 BS CNVFLD CONVERT
09750 Sw 0-8+X2 MARK FOR MOVE
09760 BBE IFMNEG,0-8+X2,40 CHECK FOR MINUS SIGN
09770 MLWD 0-5+X2,CNVFLD-2 BINARY
09780 IFMCNV TMA CNVFLD,00 INTEGER
09790 GTD CNVFLD,00 TO DECIMAL
09800 MCW :0:,0-2+X10 ZERO SUPPRESSION SYMBOL
09810 Sw 0+X8 LEFT END OF EDIT CONTROL
09820 MCE CNVFLD,0-1+X10 MOVE AND EDIT DECIMAL INTEGER
09830 MCW IR10,IR8 NEXT BUFFER LOCATION
09840 BS =1B9,IR2 POP STACK
09850 BBE IFMSGN,1+X2,40 CHECK FOR NEGATIVE SIGN PLACEMENT
09860 Sw FETCH
09870 IFMNEG Sw CNVFLD-5 CONVERT TO POSITIVE
09880 BS 0-5+X2,CNVFLD-2 GO CONVERT TO DECIMAL
09890 Cw CNVFLD-5 FIND BLANK
09900 B IFMCNV NEXT POSITION TO THE LEFT
09910 IFMSGN BCE IFMSST,0-1+X10,15
09920 BS =1B1,IR10
09930 B IFMSGN
09940 IFMSST MCw :--;0-1+X10 NEXT POSITION TO THE LEFT
09950 B FETCH
09960 LF ORM BA =1B1,FMTCD1
09970 MR ID (FMTCD1-3),IR10-3 NEXT FORMAT CODE POINTER
09980 SAR FMTCD1
09990 BA IR10,IR8
<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>C IR8,STREN</td>
<td>MOVE TRUE</td>
</tr>
<tr>
<td>10010</td>
<td>BL BUFOFL</td>
<td>MOVE FALSE</td>
</tr>
<tr>
<td>10020</td>
<td>BCE FCUT+0-8+X2+F</td>
<td>MOVE TRUE</td>
</tr>
<tr>
<td>10030</td>
<td>MCw T:+0-1+X8</td>
<td>MOVE FALSE</td>
</tr>
<tr>
<td>10040</td>
<td>B FETCH</td>
<td>address</td>
</tr>
<tr>
<td>10050</td>
<td>FOUT MCw F:+0-1+X8</td>
<td>MOVE FALSE</td>
</tr>
<tr>
<td>10060</td>
<td>B FETCH</td>
<td>address</td>
</tr>
<tr>
<td>10070</td>
<td>EFORM BA =1B1,FMTCD1</td>
<td>OUTPUT LENGTH</td>
</tr>
<tr>
<td>10080</td>
<td>MRID (FMTCD1-3),IR10-3</td>
<td>NEXT FORMAT CODE</td>
</tr>
<tr>
<td>10090</td>
<td>SAR FMTCD1</td>
<td>address</td>
</tr>
<tr>
<td>10100</td>
<td>C IR10,STREN</td>
<td>MOVE TRUE</td>
</tr>
<tr>
<td>10110</td>
<td>BL BUFOFL</td>
<td>MOVE FALSE</td>
</tr>
<tr>
<td>10120</td>
<td>TMA 0-1+X2+00</td>
<td>OPERAND TO FRO</td>
</tr>
<tr>
<td>10130</td>
<td>BS =1B9+IR2</td>
<td>POP STACK</td>
</tr>
<tr>
<td>10140</td>
<td>B FB/FD</td>
<td>CONVERT TO FLOATING DECIMAL</td>
</tr>
<tr>
<td>10150</td>
<td>R DSA DECRES</td>
<td>ADDRESS OF FLOATING DECIMAL FIELD</td>
</tr>
<tr>
<td>10160</td>
<td>MLWD IR8+IR5</td>
<td>PREPARE FOR BASIC E FORMAT</td>
</tr>
<tr>
<td>10170</td>
<td>BBE ROUND+DECRES-3,20</td>
<td>NO SIGN IF PLUS</td>
</tr>
<tr>
<td>10180</td>
<td>SST 0:+DECRES-3,60</td>
<td>CLEAR SIGN</td>
</tr>
<tr>
<td>10190</td>
<td>MCw 0:+0+X5</td>
<td>MINUS SIGN</td>
</tr>
<tr>
<td>10200</td>
<td>ROUND LCA 0:+DECRES-14</td>
<td>PREPARE FOR Rounding OVERFLOW</td>
</tr>
<tr>
<td>10210</td>
<td>Cw DECRES-13</td>
<td>ADDRESS OF FLOATING DECIMAL FIELD</td>
</tr>
<tr>
<td>10220</td>
<td>A 5:+DECRES-3</td>
<td>ROUND RESULT</td>
</tr>
<tr>
<td>10230</td>
<td>BCE MARK+DECRES-14,00</td>
<td>TEST NO OVERFLOW</td>
</tr>
<tr>
<td>10240</td>
<td>MCw DECRES-4,DECRES-3</td>
<td>MOVE RIGHT ONE CHARACTER ON OVERFLOW</td>
</tr>
<tr>
<td>10250</td>
<td>A 1:+DECRES-13</td>
<td>ADJUST EXPONENT</td>
</tr>
<tr>
<td>10260</td>
<td>MARK Sw DECRES-13</td>
<td>RESTORE FIELD MARKER</td>
</tr>
<tr>
<td>10270</td>
<td>SI 19+X5</td>
<td>RIGHT END OF OUTPUT FIELD</td>
</tr>
<tr>
<td>10280</td>
<td>BSN EFRM+DECRES</td>
<td>BRANCH IF NEGATIVE EXPONENT</td>
</tr>
<tr>
<td>10290</td>
<td>SST 0:+DECRES,60</td>
<td>STRIP SIGN</td>
</tr>
<tr>
<td>10300</td>
<td>C DECRES+009:</td>
<td>TEST EXPONENT</td>
</tr>
<tr>
<td>10310</td>
<td>BL EFRM</td>
<td>address</td>
</tr>
<tr>
<td>10320</td>
<td>MCw DECRES+IR1</td>
<td>address</td>
</tr>
<tr>
<td>10330</td>
<td>BA IR1+IR5</td>
<td>address</td>
</tr>
<tr>
<td>10340</td>
<td>SI DECRES-4</td>
<td>RIGHT MOVE STOPPER</td>
</tr>
<tr>
<td>10350</td>
<td>BCE MVP+DECRES,00</td>
<td>ZERO EXPONENT TEST</td>
</tr>
<tr>
<td>10360</td>
<td>MCw DECRES-14+X1+0+X5</td>
<td>MOVE DIGITS LEFT OF</td>
</tr>
</tbody>
</table>
10370 MVPT MCW :..:1+X5
10380 MRID DECRES=-13+X1.2+X5
10390 SBR IR5
10400 NZTST BBE NXTFLD,0-1+X5,77
10410 MCW :..:0+X5
10420 SBR IR5
10430 B NZTST
10440 EFRM MCW :..:1+X5
10450 MCW DECRES=-4,11+X5
10460 MCW :..:12+X5
10470 MCW :..:13+X5
10480 BBE LDIGIT,DECRES,40
10490 MCW :..:13+X5
10500 LDIGIT SST DECRES,16+X5,17
10510 MCW :..:1+X5
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
10590 BA IR10,IR8
10600 B FETCH
10610
10620* GET - PROCESS INPUT EITHER FROM CARD BUFFER OR STRING
10640* ACCORDING TO INFORMATION SET BY FMT.
10650*
10660* GET BCE GETFM,*DEV_TYP*,00 IF STRING INPUT BYPASS EOF TEST
10680 BCE ENDMS,*ENDF,T TEST FOR CLOSED FILE
10690 GETFM BCE IFMRST,* (FMTCD1-3),77 CHECK FOR FORMAT RESET
10700 BBE IDTFMT,* (FMTCD1-3),70 CHECK FOR DATA FORMAT
10710 BCE ISPCE,* (FMTCD1-3),00 ELSE IT MUST BE CONTROL FORMAT
10720 BCE ISKP,* (FMTCD1-3),02
10730 BCE ICOL,* (FMTCD1-3),03
IFMRST MCw =FMTCD2,FMTCD1
B GET
NXTIFM LA =IB1,FMTCD1
B GET
ISPC B =IB4,FMTCD1
ISPCE BA (FMTCD1-3),IR9
SPCCNT C IR9,:000:
BE NXTIFM
BCE STRSPC,DEVTyp,00
EA =IB1,IR6
C +INPUT+79,IR6
BH GETIPT
BS =IB1,IR9
S =IB1,IR6
BEH BUFOFL
MCw +INPUT,IR6
BA =IB4,FMTCD1
BA (FMTCD1-3),IR6
BS =IB1,IR6
C +INPUT+79,IR6
BH BUFOFL
B NXTIFM
ICOL BCE STRCOL,DEVTyp,00
MCw +INPUT,IR6
BA =IB4,FMTCD1
BA (FMTCD1-3),IR6
BS =IB1,STRADD
C +INPUT+79,IR6
BH BUFOFL
B NXTIFM
STRCOL MCw STREN+STRADD
BS LNGSTR,STRADD
BA =IB5,FMTCD1
BA (FMTCD1-3),STRADD
BS =IB1,STRADD
C STREN+STRADD
BEH BUFOFL
B NXTIFM
ISKP BCE FMTERR,DEVTyp,00
BA =IB1,FMTCD1
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 NEXT FORMAT CODE LOCATION
11180 BCE ENDFMS, ENDF, T  TEST FOR CLOSED FILE
11190L :GET READ,  
11200 MCW *INPUT, IR6  RESET BUFFER POINTER
11210 C INPUT+3, :1EOF:  END OF FILE TEST
11220C BNE GETRTN  
11230C MCW :T, :ENDF  SET END OF FILE FLAG
11240 GETRTN B *  
11250 ENDF DCW :F:  
11260 ENDFMS EQU *  
11270L :PUT PRINT, EOFMES,  
11280 B ERRLOC  
11290 EOFMESDCW :A ****END OF FILE ON INPUT UNIT****:  
11300D DCW =1C45  
11310 IDTFMT EQU *  
11320 BCE AFRMI, (FMTCD1-3), 10  JUMP OVER CODE TYPE
11330 BCE EFRMI, (FMTCD1-3), 11  A FIELD LENGTH
11340 BCE IFRMI, (FMTCD1-3), 12  NEXT FORMAT CODE LOCATION
11350 BCE LFRMI, (FMTCD1-3), 13  
11360 B FMTERR  
11370 AFRMI BA =1B1, FMTCD1  
11380 MRID (FMTCD1-3), IR10-3  A FIELD LENGTH
11390 MRID FMTCD1  NEXT FORMAT CODE LOCATION
11400 BB E AOK, 0-9, X2, 04  
11410 B CNVERR  
11420 AOK SW 0-4, X2  
11430 C IR10, 0-5, X2  
11440 BL FMTE RR  
11450 MRID 0-8, X2, IR9-3  ADDRESS OF STRING
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 FIELD
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 PLUS SIGN
11910  SST    0+X6,DFLD+X9,17  MOVE NUMERIC BITS ONLY
11920  BA     =1B1,IR9
11930  IENDT  BA    =1B1,IR6  BUMP
11940  C      IR6,INPEND  COUNTERS
11950  BL     GETIPT  NEXT RECORD
11960  C      IR10,IR3
11970  BL     IMORE  TEST FIELD END
11980  IDECMV MCW    DFLD+1+X9,CNVFLD  MOVE DECIMAL DIGITS
11990  SST    ISGN,CNVFLD,60  SET SIGN IN CONVERSION FIELD
12000  DTB    CNVFLD+00
12010  TAM    CNVFLD+00
12020  MRID   0-8+X2,IR9-3  ADDRESS OF RECEIVING FIELD
12030  SI     CNVFLD+2  PUNCTUATION FOR MOVING
12040  MRID   CNVFLD-5,0+X9  MOVE RESULT
12050  CI     CNVFLD-2  CLEAR ITEM MARK
12060  B      POPUP  POP STACK AND EXIT
12070  DFLD   DCw    =30
12080  ISGN   DCw    :+;
12090  STISGN MRSU   0+X6,ISGN  SET SIGN
12100  BS     =1B1,IR10  ADJUST CHARACTER COUNT
12110  E      IENDT  CHECK FOR RECORD END
12120  ISTRING SST    (STRADD-3),DFLD+X9,17  MOVE DIGIT
12130  BCE    ISTISGN,(STRADD-3),+  MOVE DIGIT
12140  BCE    ISTISGN,(STRADD-3),-
12150  ISRUP  EA     =1B1,IR9
12160  BA     =1B1,STRADD
12170  C      IR10,IR9  IF FIELD END
12180  EH     IDECMV  THEN SET UP FOR CONVERSION
12190  C      STREND,STRADD  CHECK BUFFER
12200  BEH    BUFOFL  OVERFLOW
12210  B      ISTRING  GET ANOTHER DIGIT
JUMPOVER CODE TYPE  FIELD LENGTH  NEXT FORMAT CODE LOCATION
CHECK FOR TYPE MISMATCH
INITIALIZE DECIMAL EXPONENT
CHECK FOR STRING INPUT
FINISH
GO CONVERT
CLEAR DECIMAL MANTISSA
CLEAR DECIMAL EXPONENT FIELD
CLEAR CHAR COUNTER
MANTISSA COUNTER
CHECK FOR BLANKS
CHECK SIGNS
CHECK BEGINNING OF FRACTION
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  EXPR,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  Expr MCW  ++:ESGN  DEFAULT SIGN
12830  BA   =1B1,IR9
12840  BCE  ESGNST*EFHLD+x9,- CHECK FOR
12850  BCE  ESGNST*EFHLD+x9++ EXPONENT SIGN
12860  P    EXPNUM  GET EXPONENT
12870  ESGNST MRSD EFHLD+x9,FSGN  SET SIGN
12880  BA   =1B1,IR9
12890  ExpNum 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
FLOATING DECIMAL FIELD
ERROR INSTRUCTION
ADDRESS OF RESULT
CLEAR ANY EXTRANEOUS PUNCTUATION
STORE RESULT
ITEM MARK RIGHT
POP STACK AND FETCH NEXT INSTR
JUMP OVER CODE TYPE
FIELD LENGTH
NEXT FORMAT CODE LOCATION
CHECK FOR TYPE MISMATCH
CHECK FOR STRING INPUT
CHECK
NEW RECORD REQUIREMENT
MOVE ADDRESS
DETERMINE EXCESS CHAR
NEW LENGTH
CHECK OVERFLOW
ADDRESS OF RECEIVING FIELD
1333C MRSDR 0-1+X9+0+X10 MOVE INPUT
1334C MCW 1R9+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
1349C CNTMESDCw :A ****INSTRUCTION COUNT = :
1350C PRTCNT DC =9
1351C DC :****:
13520 L DCw =1645
1353C ******************************************
1354C* ERROR MESSAGES *
1355C* *
1356C* *
1357C* ******************************************
1358C ERROR EQU *
13590L :PUT PRINT,BADOP,
1360C B ERRLOC
1361C BADOP DCw :A ****ILLEGAL OP CODE***:
13620 L DCw =1645
1363C DYNFL EQU *
13640L :PUT PRINT,STROFL,
13650 B ERRLOC
1366C STROFLDCw :A ****DYNAMIC STORAGE EXHAUSTED****:
13670 L DCw =1645
1368C 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 FERMES DCW :A ****FLTNG PT OVFLW OR ZERO DIVIDE****:
13770 L DCW =1C45
13780 IDXERR EQU *
13790 L :PUT PRINT, IDXMES,
13800 B ERRLOC
13810 IDXMES DCW :A ****INDEXED ADDRESS BEYOND DYNAMIC STORAGE****:
13820 L DCW =1C45
13830 TYPERR EQU *
13840 L :PUT PRINT, TYPMES,
13850 B ERRLOC
13860 TYPMES DCW :A ****INCONSISTENT DATA TYPE ERROR****:
13870 L DCW =1C45
13880 FMTERR EQU *
13890 L :PUT PRINT, FMTMES,
13900 B ERRLOC
13910 FMTMES DCW :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 LOCMES DCW :A ****ERROR AT RELATIVE LOCATION :
14050 ELOC DC =9
14060 DC ;****:
14070 L  DCW  =1C45
14080  CNVFLD DCW =1180
14090   LITORG*
14100  ENDPRTG EQU *
14110      END  START

SYMBOL DEFINITION - CARD REFERENCE INDEX

ADCOMP  01840;  ADD  06290;  AFIN  11560;  AFORM  09390;  AFRMA  09450;
AFRMI   11370;  ALCRT1  02600;  ALCRT2  02630;  ALLOC  02490;  AMORE  11470;
AND     07060;  AOK  11420;  AFRMA  09390;  AFRMI  11370;  ALCRTL  02600;
BADUP   13610;  BFJFL  09660;  BGSTR  09050;  BLNK  06160;  BSTF  07130;
BSTT    07100;  BUFOFL  09630;  CAT2  07720;  CAT3  07760;  CAT  07670;
CFDFB   12930;  CHK2ND  05150;  CHKBF  09470;  CHKFTP  03180;  CHRTSR  03710;
CNTMES  13490;  CNVCCH  06860;  CNVER  13680;  CNVFLD  14080;  CNVRTN  03410;
CDN     05210;  CONPRT  02860;  CSBSTR  03930;  CTSUB1  07810;  CTSUB2  07920;
CURLEV  04550;  CVEMS  13710;  DATFMT  09330;  DECESP  13070;  DECRES  10550;
DEFYP   08440;  DEXP  13060;  DFLD  12070;  DRFCT  13050;  DISPLY  04490;
DIV     06610;  DODIV  06670;  DTYPE  02220;  DUMMY  09290;  DVMN  02750;
ECNTL   12630;  ECNVR  12480;  EDIT  08870;  EFHL  12410;  EFORM  10070;
EFRM    10440;  EFRMI  12240;  ELOC  14050;  EMMORE  12330;  ENDADR  06020;
ENDF    11250;  ENDFMS  11260;  ENDPRTG  14100;  ENTPRO  04220;  EOFM  11290;
EOK     12290;  ERRLOC  13930;  ERROR  13580;  ESGN  13080;  ESGNST  12870;
ESTRNG  12420;  EWORD  14030;  EXPNUM  12890;  EXPON  12770;  EXPR  12820;
FERMES  13760;  FERR  13730;  FETCH  01170;  FIXFLT  03640;  FLAG  04130;
FICCHK  04310;  FMT  08290;  FMTC  08410;  FMTC  08420;  FMTERR  13880;
FMVES   13910;  FMRST  09310;  FNLNG  08140;  FOUT  10050;  FRACT  12690;
FSGN    13090;  FSGNST  12670;  G  10670;  GETFM  10690;  GETIPT  11700;
GETRTN  11240;  GTLN  09560;  GTSBLN  09610;  HALT  14020;  HOLD  04690;
ICOL    10930;  IDECMV  11980;  IDFMT  11310;  IDXERR  13780;  IDMXE  13810;
IENDT   11930;  IFIX  00780;  IFMCNV  09780;  IFMNEG  09870;  IFMRST  10750;
IFMSGN 09910; IFMSST 09940; IFORM 09680; IFRMI 11800; IMORE 11890;
INDIRA 02160; INDXA 07250; INDXR 07260; INOROT 08430; INPEND 11700;
INPUT 00310; INSTCT 00410; INTCBF 03350; INTCBF 06580; INFRMI 11850;
INTNEG 06810; INTNEG 07090; INTRSD 07290; INTRSD 08110; INVSIGN 12080;
IR11 06620; IR12 06630; IR13 06640; IR14 06650; IR15 06660;
IR2 00520; IR7 00580; IR3 00540; IR4 00550; IR5 00560;
IR6 00570; IR8 00590; ISGN 12080; ISKP 11090; ISKPTS 11130;
JUMP 04960; JUMPA 04860; JUMPF 05030; JUMPP 05010; JUMPT 04980;
LDLG 02990; LFORMAT 09960; LFRMI 13100; LIREC 13170; LOK 13500;
LONGT 02840; MSRT1 05450; MSRT2 05470; MRBLNK 05770; MORSJB 02580;
NSEG 04010; NOT 07090; NUMER 06650; NXTFLD 10560; NRADD 12040;
NZTST 01400; OR 07030; OUTPUT 00320; PAGE 09070; PNCRTN 03460;
PRTCT 13500; PUT 08700; READ 00300; RETRN 04760; RDWR 00290;
SCNV 03550; SETBCK 06330; SETCND 05170; SETF 05220; SETL 03230;
SETINT 03210; SETPNC 05820; SETT 05260; SIZE 02850; #INSTR 00290;
SKPCLR 09200; #SKP 00340; SAVTOP 03500; SBSTR 08050; SBSTRL 08110;
SSC 03550; SETBCK 06330; SETCND 05170; SETF 05220; SETL 03230;
STDCH 03210; SETPNC 05820; SETT 05260; SIZE 02850; #SKP 00340;
STKSTR 00420; STKPS 09250; STCKC 05130; STISGN 12090; STKEND 00430;
STRCOL 11010; STREND 08470; STROFL 13660; STRPUT 08780; STRSET 08480;
STRSC 10890; STRP 02210; SUB 06440; SUBCH 07290; SUBDNE 07500;
SUBMB 05910; SUBMB 06030; SUBMV 03860; SwAP 04630; TMOVE 03880;
TSTRNG 05580; TVEC 01250; TYPERR 13830; TYPMES 13860; UPDPT 08940;
UPSTCK 02970;

****INSTRUCTION COUNT = 296,018****
XII. APPENDIX D
**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.
00380  IR15  EQU  60
00390*******************************************************************************
00400  *
00410  OCTAL ADDRESS DEFINITIONS OF PERTINENT SYMBOLS IN THE RESIDENT *
00420  INPUT/OUTPUT ROUTINES *
00430  *
00440*******************************************************************************
00450  #RDWR  CEQU  =4C00000754
00460  READ  CEQU  =4C00005430
00470  INPUT  CEQU  =4C00006144
00480  OUTPUT  CEQU  =4C00006265
00490  PRINT  CEQU  =4C00005647
00500  #SKP  CEQU  =4C00000756
00510*******************************************************************************
00520  *
00530  COMMUNICATION AREA FIELD LOCATION DEFINITIONS *
00540  *
00550*******************************************************************************
00560  GENFLD  EQU  215  CONTAINS ADDRESS OF CODE GENERATION LOC
00570  LODFLD  EQU  219  CONTAINS METAX PROGRAM LOADING ADDRESS
00580  STCKF1  EQU  223  BEGINNING PUSH DOWN STACK ADDRESS
00590  STCKF2  EQU  227  CONTAINS STACK LIMIT ADDRESS
00600  SYMF1  EQU  231  ADDRESS OF SYMBOL TABLE START
00610  (START OF DYNAMIC STORAGE FOR PLEX
00620  OBJECT PROGRAMS)
00630  SYMF2  EQU  235  CONTAINS SYMBOL TABLE LIMIT ADDRESS
00640  (LIMIT OF DYNAMIC STORAGE FOR PLEX
00650  OBJECT PROGRAMS)
00660  CMPLCD  EQU  236  COMPLETION CODE FIELD SET BY COMPILERS
00670  D5KLOD  EQU  237  DISK LOADING OPTION FIELD
00680  EXCPPG  EQU  238  GO OPTION FIELD
00690  PSTLST  EQU  239  POST LISTING OPTION FIELD
00700  PRGNME  EQU  290  METAX PROG NAME FIELD
00710  INTNME  EQU  298  INTERPRETER NAME FIELD
00720  SYMBOL  EQU  243  SYMBOL FIELD USED BY COMPILERS
00730  INSTCT  EQU  209  INTERPRETER INSTRUCTION COUNT FIELD
00740*******************************************************************************
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 START
RETURN TO LOADER
FETCH RESIDENT METAX PROGRAM
ZERO INSTRUCTION COUNT
COMMUNICATIONS AREA
FIELDS
FOR EXECUTING
CONTROL RECORD ANALYZER
INTERPRETER SEGMENT
AND NAME
RETURN POINT
FETCH AND EXECUTE
ZERO INSTRUCTION COUNT
INITIALIZE COMPLETION CODE
SET UP
MEMORY
CLEAR
01120 Sw O+X15 OPERATION
01130 B CLEAR DO IT
01140 MCw LODFLD,LODSAV TEST EXECUTION OF RESIDENT MFTAX PROGRAM
01150 C PRGNME,SAVNME IF NOT GO GET THE RIGHT ONE
01160 BNE MFETCH ALTER EXECUTION LOCATION TO RESIDENT PROG
01170 MCw MTXPRG,LODFLD INTERPRETER SEGMENT
01180 GETINT MCw INTNME,75 AND NAME
01190 BNE MFETCH RETURN POINT
01200 MCw +INTRT2,167 FETCH AND EXECUTF
01210 INTRT2 BCE FATAL,CMPLCD,F TEST FOR FATAL ERROR ACTION
01220 BCE LTODSK,DSKLOD,Y LOAD COMPILED PROGRAM TO DISK IF REQUESTED
01230 BCE EOFSTST,EXCPPG,N IF NO GO THEN SEARCH FOR END OF FILE
01240 GO EQU * ELSE MOVE AND RELOCATE COMPILED
01250 BCE LT0DSK,0SKL0D, PROGRAM FOR EXECUTION
01260 MCw LODSAV,LODFLD COMPILED PROGRAM LOCATION
01270 SI 1+X5 LOADING LOCATION
01280 Sw W+X5
01290 MLwD GENFLD,IR15
01300 MLwD LODFLD,IR14
01310 MORPRG MRwDR 0+X15,0+X14
01320 SAR IR15 MOVE REMAINDER OF ADDRESS
01330 SBR IR14
01340 MRIwR 0+X15,0+X14
01350 SAR IR15 RELOCATE THE ADDRFSS
01360 SBR IR14
01370 BA LODFLD,0-1+X14
01380 CW 0-3+X14
01390 CW 0-1+X14
01400 BCE BLKCTN,0-4+X14,00 TEST POSSIBLE BLOCK PSEUDO OP CODE
01410 ENDTST C IR15,IR5
01420 BH MORPRG
01430 MLwD =3C117776,IR15 PREPARE FOR CLEARING REMAINING MEMORY
01440 MRSDR =4B0,0+X14
01450 SBR IR14
01460 SI 0+X15
01470 B CLEAR
01480 MCw SYMF1,IR14 CLEAR
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
MTXCRRA INTERPRETER NAME
INITIAL METAX PROGRAM NAMES SAVE LOC
LODFLD SAVE LOCATION
MEMORY CLEAR SUBROUTINE
SYMBOL DEFINITION - CARD REFERENCE INDEX

BLKCN T 01570;  C LEAK 01690;  CLRRTN 01760;  CMPLCD 00660;  CPACLL 00960;  
D SKLOD 00670;  ENDPRG 02030;  ENDST 01410;  EOFST 01810;  EXCPPG 00680;  
EXNT 01770;  FATAL 01850;  FTLMES 01880;  FTRTN 02010;  FTPTST 01990;  
GENFLD 00660;  GETINT 01180;  GO 01250;  INPUT 00470;  INSTCT 00730;  
INTNME 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 MTXLD 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

\[
BA = \text{1B1,IR13}
\]

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

\[
\text{MCW TVEC+3+X7,IR14}
\]

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

\[
\text{SAR IR1}
\]
and

\[ SBR \ IR10 \]

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

```
MRIN  0+X14,0+X13
```

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).