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's 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
B. The PLEX Pseudo-Machine 71
C. The PLEX Translator 87
VI. CONCLUSIONS AND SUGGESTED FURTHER WORK 95
VII. BIBLIOGRAPHY 99
VIII. ACKNOWLEDGEMENTS 104
IX. APPENDIX A 105
X. APPENDIX B 113
XI. APPENDIX C 220
XII. APPENDIX D 317
XIII. APPENDIX E 325
I. INTRODUCTION

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

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

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

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

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

The overall system organization is presented in Chapter
IV including descriptions of the supporting assembler lan­
guage 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 lan­
guage, PLEX, the object language interpreter and the imple­
mentation of its compiler which illustrates most of the fea­
tures of METAX9. Run-time storage administration represents
an important aspect of the implementation and is considered
extensively.

In Chapter VI we review the results, present some con­
cclusions 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 META9 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

```
SDBLIST:= EXP $ ( "," EXP );
```
in META II as opposed to

```plaintext
< SUBLIST > ::= < SUBLIST > , < EXP > | < EXP >;
```
in BNF.

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

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

```plaintext
.SYNTAX PROGRAM
PROGRAM := ".SYNTAX" .ID $ ST ".END";
ST := .ID ":=" EX1 ";";
EX1 := EX2 $( "|" EX2 );
EX2 := ( EX3 | OUTPUT ) $( EX3 | OUTPUT );
EX3 := .ID | .STRING | ".ID" | ".NUMBER" | ".STRING"
| "(" EX1 ")" | ".EMPTY" | "$" EX3 ;
OUTPUT := ".OUT" "(" $ OUT1 ")" | ".LABEL" "(" OUT1 ")";
OUT1 := "*1" | "*2" | "*" | .STRING ;
.END .
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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'Neill'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 METAXO, 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 METAXO written in METAXO 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 METAXO 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 metacompilation time.

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

1. Compiler definition commands

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

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

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

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

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

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

Example: PUSHLB .OEOU 46 .TYPEO;

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

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

Example:

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

2. **Elementary syntactical commands**

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

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

"XYZ" (TEST):

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

PQR (CLM):

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

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

.LATCH(...) (LATCH):

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

.EMP TY (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.

.OMUX (OMUX):

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 .OMUX 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 (STRST):**

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 lattitude 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(DYNAHB>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

SUBLIST := EXP $ ( "," EXP ) ;

as opposed to

SUBLIST := SUBLIST "," EXP / 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.

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

4. Semantic commands

\[ \text{.OUT(...)} \ (LBI,LB2,eval,outsym,restore,OUT) ; \]

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: \[ .OUT(DYNAM,*1) \]

\[ \text{.DO(...)} \ (\text{Complete instruction set}) ; \]
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,ENTLOC):**

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

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

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

**.UNSTACK(...) (POP,MGE):**

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. Anomolous behavior may result if the receiving field is shorter than the sending field.

Example: `.SET(DYNAMP=DYNAMB)`

`.BLKENT (BLKENT):`

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

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

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

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

.CAT(...) (CAT):

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

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

**.ENTERL(...) (ENTL):**

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

Example: `.ENTERL(00,01)`

**.ENTERA(...) (ENTA):**

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

Example: `.ENTERA(07,DIMCNT)`

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

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

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

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

.STKSYM (STKSYM):

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

.SCAN(...) (SCAN):

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

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

5. \textbf{Output operators}

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

\texttt{*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.

\texttt{*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 .OEOU 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 (MOV):**

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
 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 META9 Pseudo-Machine

1. **Primitives**

There are 47 operation codes or primitives available on the META9 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,DYNAHB:**

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

**M FOUR,DYNAMP:**

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

EXIT (EXITI):

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

RESOLVE (RESOLV):

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

F. Chronological Development

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

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

The system control record analyzer (MTX.CRA) 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. MTX.CRA 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 META8 version which was then used for implementing the final version of the BASIC translator. Of course META9 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. MTXLDR, a supporting assembler program, is then loaded into the transient program area and called to load the two resident programs from the METAX library. Once these functions are complete the main control processing loop commences.

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

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

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

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

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

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

B. Control Record Analyzer

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

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

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

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

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

C. Communications Region

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

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.

**SYM1 (229-231)* and SYM2 (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.

**DSKLLOD (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 MTXCSR 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 con-
siderations are correct. If the object is a label variable
the contents must satisfy the above label constant require-
ments except that if the label variable is a dummy argument
in the formal parameter list of the currently active proce-
dure 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 behav-
ior if that function itself attempts an output operation.

Format lists may contain both data format items and con-
trol format items although somewhat limited in scope compared
to PL/I, the intent being to provide only enough format capa-
bility 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 \texttt{X}, \texttt{COL}, \texttt{PAGE} and \texttt{SKIP}, the latter two causing execution time errors if used in the wrong context.

Procedures must be declared in a block head in an \texttt{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 \texttt{RETURNS} attribute being required to establish the data attribute of the function value. Additional usage of the \texttt{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 \texttt{ALGOL-}like fashion. A \texttt{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 \texttt{DECLARE} statement prior to the declaration of any local variables.

A procedure may be invoked with a \texttt{CALL} statement if a \texttt{RETURNS} attribute has not been established or otherwise as a function reference. \texttt{LABEL} functions, however, are not permitted because of restrictions on label assignments delineated below. A return from a procedure may be effected by a \texttt{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 FLEX programs in Appendix C and the FLEX translator (PLXCPL) listing in Appendix B will provide adequate additional detail.

B. The FLEX 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.

Pun-time storage administration is concerned with the proper management of available dynamic storage. With respect to the META6 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 BNTPRO 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, MULT, SUB, DIV, NEG:**

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

**FMT:**

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

**GET:**

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

EDIT:

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

PUT:

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

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

OR, AND, NOT:

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

INDXR,INDXA:

The two indexing primitives expect the requisite number of arithmetic indexing values on the stack with the right most index on top. A single character literal value following 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 address 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 element 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 signifi-
cant dependence on the host machine structure for the imple-
mentation 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 PLXCP, 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 STMENT 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 and END statement and then backing up so that it may be processed as the end of a procedure, DO group or block. Although it isn't necessary to back over the END delimiter only to allow another procedure to test for it, it does perhaps make the procedures representing DO group, block or procedure syntax somewhat more readable.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Certainly the translator contains errors of comission 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 .SEARCH command would facilitate symbol table entries and searches for structured operands. The entry and testing primitives will easily accommodate manipulation of three contiguous 20 character entries should they be required for processing a particular structure component as the positioning parameter in the respective commands has an upper limit of 63. It remains to be shown, however, that such structures can be accommodated within the basic framework provided.

In any case it is hoped that PLXCPL effectively represents the capability of 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


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 THEPLEX */
/* 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 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;
TYPIST OEQU 34 TYPEO; ENTTYP OEQU 35 TYPEO; ENTPR OEQU 72 TYPEO;
DECNUM OEQU 75 TYPEO;
/* OPERATION CODES COMMON TO METAX8 AND METAX9 OBJECT PROGRAMS. */

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

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

BEF .OEUQ 16 .TYPEG; COMP .OEUQ 32 .TYPEG; TSTTBL .OEUQ 34 .TYPEG;
TSTTBA .OEUQ 35 .TYPEG; ENTA .OEUQ 55 .TYPEG; ENTL .OEUQ 56 .TYPEG;
BLKENT .OEUQ 75 .TYPEG; BLKEXT .OEUQ 76 .TYPEG;

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

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

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

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

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

DECLPT := "DECLARE" $( .ID .DEFLAB(*)
   $( STRING / .ONUM)
   .OUT(*) ("\n" / ";" .RETURN) ) ;

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

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

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

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

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

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

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

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

ELMSTX := ID *OUT(CLAS**2) / STRING *OUT(TEST**) 

/ "ID" *OUT(ID) / "STRING" *OUT(STRTST) 
/ "ONUM" *OUT(ONUM) / "INUM" *OUT(INUM) / "EMPTY" *OUT(SET) 

/* OBSERVE THAT THE NEXT ALTERNATIVE PERMITS FACTORING OF METALANGUAGE EXPRESSIONS. */

/ "(" MTXEXP ")" *ERR("W: EXPECTED") *OUT(SET) 

/* THE NEXT ALTERNATIVE PERMITS ITERATIVE EXPRESSIONS. */

/ "NEWLAB DEFLAB(*1) ELMSTX *OUT(BT,*1) *OUT(SET) 
/ "FNUM" *OUT(FNUM) / "STKCHK" *OUT(CHKSYM) 
/ "LATCH(" ID *OUT(LATCH**,")")" 
/ "TSTTBA(" OUT(TSTTBA) ONUM OUT(*) "," ID OUT(**) ")" 
/ "TSTTBL(" OUT(TSTTBL) ONUM OUT(*) "," ( ONUM OUT(*) 

/ "STRING OUT(*) ) ")" 
/ "TEST(" OUT(COMP) ID OUT(**) TESTOP ID OUT(**,*) ")" ;

TESTOP :="=" *SAV(02) / ";=" *SAV(03) / ">");=" *SAV(03) / ">" *SAV(01) / ">" *SAV(04) / ">=" *SAV(03) / ">");=" *SAV(02) / ">=" *SAV(01) ;

/* "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. */

```c
OUTPUT := "*OUT(" OUTL $ ( "", OUT1")")
   / "*DO(" OUTDO $ ( "", OUTDO")") ;

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

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

SEMI := ";NEWLAB" OUT(PUSHLB)
   / ";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 / ONUM ) .OUT(MOVI,**)) )")
   / ";BLKEXT" ( "(" STRING .OUT(BLKEXT,**)) ")"
   / EMPTY .OUT(BLKEXT,"0")
   / ";MARK" .OUT(MARK)
   / ( ";SAV(" OUT(MARK) OUT1 $ ( ";", OUT1 ) "") / ";SAV") .OUT(SAVE)
   / ";CAT(" .OUT(MARK,RESTORE) OUT1 $ ( ";", OUT1) .OUT(SAVE) "")
   / ";BLKENT" .OUT(BLKENT)
   / ";ENTERL(" .OUT(ENTL) .ONUM .OUT(*) ";"
   ( STRING / ONUM ) .OUT(*) )"
   / ";ENTERA(" .OUT(ENTA) .ONUM .OUT(*) ";" ID .OUT(**) "")
   / ";SEARCH" ( "(" STRING .OUT(SEARCH,**)) ")"
   / EMPTY .OUT(SEARCH,"0")
```

111
/ "STKSYM" .OUT(STKSYM)
/ "SCAN(" .STRING .OUT(SCAN,*) ")"
/ "ERKATCH" .OUT(MOVI,ELATCH,"1")
/ "RETURN" .OUT(R) / "CANCEL" .OUT(CANCEL) ;

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

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

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

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

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

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

END

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

/****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.
//
**IDENTIFICATION:**

**PROGRAM-ID:** PLXCPL.
**AUTHOR:** J. R. VAN DOREN.
**SOURCE LANGUAGE:** METAX9.
**OBJECT LANGUAGE:** METAX9 PSEUDO-MACHINE CODE.
**OBJECT INTERPRETER:** MTXINT04.

**PURPOSE:**

PLXCPL IS THE COMPILER FOR THE PLEX LANGUAGE.

**DECLARE VARIABLE NAMES AND INITIAL VALUES REQUIRED BY THE COMPILER.**
**NOTE THAT INITIAL VALUES ARE IN OCTAL OR CHARACTER STRING FORM.**
**THE LENGTH OF A VARIABLE BEING IMPLICIT IN ITS INITIAL VALUE.**

**VARIABLES USED FOR COMPILING DYNAMIC STORAGE ADMINISTRATION CODE.**

*DECLARE DYNAMP "1008", DYNAMB "0000", STACKTP "$STKTO ", ONELEV 01000000,
LEVNO 01, ONELNG "00000100", DOPFIX 0008;*

**VARIABLES USED FOR SYNTACTIC ANALYSIS**

*DECLARE DOSYM "DO  ", IFSYM "IF  ", CALLCON "CALL  ",
RPAREN ")", SYMSAV "  ", STMLAB "  ";*
/* VARIABLES USED FOR ATTRIBUTE PROCESSING */
.DECLARE TYPE "" LENGTH 0000 DIMCMT 00 FUNCT 0000;

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

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

/* VARIABLES USED FOR TESTING I/O ITERATION LOOPS */
.DECLARE IOITTR 77 IOITW 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; MOVI .OEQU 30 .TYPEO; SEARCH .OEQU 33 .TYPEO;
COMP .OEQU 32 .TYPEO; TSTTBL .OEQU 34 .TYPEO; TSTTHA .OEQU 35 .TYPEO;
LATCH .OEQU 36 .TYPEO; CANCEL .OEQU 37 .TYPEO; CLM .OEQU 42 .TYPEO;
R .OEU 43 .TYPEO; PUSHLB .OEU 46 .TYPEO; POP .OEU 47 .TYPEO;
LBI .OEU 50 .TYPEO; LB2 .OEU 51 .TYPEO; ENTA .OEU 55 .TYPEO;
ENTL .OEU 56 .TYPEO; OUT .OEU 62 .TYPEO; OUTSYM .OEU 63 .TYPEO;
TEST .OEU 64 .TYPEO; ID .OEU 65 .TYPEO; ONUM .OEU 66 .TYPEO;
STRST .OEU 67 .TYPEO; EVAL .OEU 70 .TYPEO; ENTOC .OEU 71 .TYPEO;
INUM .OEU 73 .TYPEO; FNUM .OEU 74 .TYPEO; BLKENT .OEU 75 .TYPEO;
BLKEXT .OEU 76 .TYPEO; STKSYM .OEU 52 .TYPEO; CHKSYM .OEU 53 .TYPEO;
SWAF .OEU 54 .TYPEO; MARK .OEU 40 .TYPEO; SAVE .OEU 41 .TYPEO;
RESTORF .OEU 45 .TYPEO; ERASE .OEU 31 .TYPEO;

/* OPERATION CODES FOR PLEX OBJECT PROGRAMS */

DYNAM .OEU 10 .TYPEG; STKTOP .OEU 11 .TYPEG; ALLOC .OEU 12 .TYPEG;
LDA .OEU 20 .TYPEG; LD .OEU 21 .TYPEG; STC .OEU 22 .TYPEG;
SST .OEU 23 .TYPEG; JUMP .OEU 30 .TYPEG; JUMPT .OEU 31 .TYPEG;
JUMPF .OEU 32 .TYPEG; STCKC .OEU 33 .TYPEG; COMPC .OEU 34 .TYPEG;
ADD .OEU 40 .TYPEG; MULT .OEU 41 .TYPEG; SUB .OEU 42 .TYPEG;
DIV .OEU 43 .TYPEG; NEG .OEU 44 .TYPEG; FMT .OEU 50 .TYPEG;
GET .OEU 51 .TYPEG; PUT .OEU 52 .TYPEG; EDIT .OEU 53 .TYPEG;
OR .OEU 60 .TYPEG; AND .OEU 61 .TYPEG; NOT .OEU 62 .TYPEG;
INDXR .OEU 66 .TYPEG; INDXA .OEU 67 .TYPEG; CAT .OEU 70 .TYPEG;
SUBSTR .OEU 71 .TYPEG; JUMPA .OEU 27 .TYPEG; STOP .OEU 76 .TYPEG;
ENTPRO .OEU 25 .TYPEG; RETURN .OEU 26 .TYPEG; FLAG .OEU 24 .TYPEG;
SWP .OEU 35 .TYPEG; POPUP .OEU 36 .TYPEG;

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

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

/* COMMENCE THE RECURSIVE PROCEDURES COMPRISING THE COMPILER PROPER. */

PLXCLPL := .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(INAME="PLXINT00") /* SET INTERPRETER NAME FOR "GO" OPT. */
.SET(SYMBOL="$ACTIVE ") /* SYMBOL AND ADDRESS VALUE OF INDEX */
.SEARCH .ENTERL(00,0100000065) /*REGISTER POINTING TO ACTIVE */
.SET(DYNAM=DYNAMP) /* DYNAMIC STORAGE AT RUN TIME */
.SET(SYMCP="LACTIVE") /* COUNTER FOR LEVEL AND DISPACEMENT */
"PROCEDURE" "MAIN" .BLKENT";" /* REQUIRED SYNTAX; START BLOCK LIST */
.OUT(DYNAM, *1) /* PROCEDURE DYNAMIC STORAGE CODE */
.SET(SYMBOL=STACKTP) /* TYPE AND INIT STACKTOP LOCATION */
.SEARCH .ENTERL(00,0501000000) /* MAXIMUM SIZE OF LEVEL 1 STORAGE */
$ COMMENT /* ADMIT HEADER COMMENTS */
BHDBDY /* PROCESS BLOCK HEAD AND BODY */
"END" .OUT(STOP) /* POSITION SYMTAB POINTER */
.DEFLAB(*1) /* FOR THIS PROCEDURE */
.ENTERA(01,DYNAMP) /* FOR THIS PROCEDURE */
( .ID .STKCHK .ERR("W: POSSIBLE PROC CLOSING ERROR",SET)
/ .EMPTY ) .BLKENT("1") .DO(RESOLVE) ;

BLKEDY := $ ( .SET(STMLAB=BLMK8) 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(ENDSTM) .DO(SETF) / .EMPTY ;

ENDSTM := "END" .DO(SETF,BEF) / .EMPTY ;
BLOC< :="BEGIN" ";" •BLKENT •ERRLATCH /* SET BLOCK LIST AND ERROR LATCH */

•STACK(STMLAB,DYNAMB,STACKTP) /* STACK VARIABLES TO PREPARE FOR */

•OUT(5TMLAB,**),DO(A,ONLLNG,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 */

•OUT(5TMLAB,**),DO(A,ONEELNG,DYNAMB) /* INCREMENT DYNAMIC STORAGE COUNTER */

•BHDEDY /* PROCESS BLOCK HEAD AND BODY */

•UNSTACK(DYNAMB,STACKTP) /* RESTORE PREVIOUS STACK TOP SYMBOL */

"END".BLKEXT /* AND BLOCK DYNAMIC STORAGE COUNTER */

.3DECLAR /* DECLARATIONS AND BLOCK BODY */

.3NEWLAB /* DETERMINE LEVEL OF STORAGE REQUIRED */

•OUT(JUMP,*1) /* 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 */

•STTB(00,10) •DO(BM,"W:"DUP PROC DCL",SET)

•STTB(00,00) •ENTERL(00,10) •DO(SET) )

•BLKENT •ERRLATCH

•STKSYM /* SAVE PROCEDURE ID FOR ENDCHECK */

•STACK(DYNAMB,DYNAMB,LEVNO,STACKTP) /* SAVE PREVIOUS PROCEDURE INFO */

•DO(A,ONE,LEVNO) /* SET UP */

•SET(DYNAMB=ONELEV) /* DYNAMIC STORAGE */

/* SFCT BLOCK LIST AND ERROR LATCH */
/* STACK VARIABLES TO PREPARE FOR */
/* POTENTIAL RECURSIVE CALL OF "BLOCK"*/
/* DECLARATIONS AND BLOCK BODY */
/* DETERMINE LEVEL OF STORAGE REQUIRED */
/* PROCESS IDENT AND PROC DECLARATIONS*/
/* LOCATION AND PROCEDURE TYPE */
/* SAVE PROCEDURE ID FOR ENDCHECK */
/* SAVE PREVIOUS PROCEDURE INFO */
/* SET UP */
/* DYNAMIC STORAGE */
*DO(M*LEVNO,DYNAMP)     */ COUNTERS */  
*SET(DYNA= DYNAMP)     */ FOR THIS PROCEDURE  */  
*DO(A*ONELG,STACKTP)     */ NEW STACKTOP SYMBOL FOR THIS PROC */  
*SET(SYMBOL=STACKTP)  
*SEARCH("1") .ENTERA(01,DYNA)     */ LOCATION IN FIRST WORD OF */  
*ENTERA(00,05)     */ THE DISPLAY FOR THIS PROC */  
*SET(SYMBOL=FOUR)     */ COMPUTE SIZE OF DISPLAY FOR */  
*DO(M*LEVNO,SYMBOL,A*FOUR,SYMBOL,A*SYMBOL,DYNA)  
*OUT(ENTPRO)     */ PROCEDURE ENTRY */  
*SET(SYMBOL=LEVNO) .OUT(*)     */ DYNAMIC STORAGE LEVEL NUMBER */  
*OUT(DYNA,*1)     */ CODE TO ALLOCATE DYNAMIC STORAGE */  
*SET(ARGCNT=ZERO)     */ DYNAMIC STORAGE COUNT */  
( "(" ARGID $ ( "," ARGID ) ")" / .EMPTY ) / .EMPTY ) / .DUMMY ARGUMENTS */  
*SET(SYMBOL=STMLAB) /* ENTER ARGUMENT COUNT */  
*SEARCH A ENTERA(07,ARGCNT)  
*SET(ARGCNT=ZERO)     */ TEST FOR ARGUMENT COUNT */  
( "DECLARE" ARGDEC $ ( "," ARGDEC ) "," / .EMPTY )  
*TEST(ARGCNT=ADECNT) ERR("W: INCORRECT ARG DCL COUNT",SET)  
/ .EMPTY )     */ BYPASS IF NO ARGS */  
BHEDEY  
"END",OUT(RETURN)     */ PROCESS THE REST OF THIS PROCEDURE */  
*DEFLAB(*1)     */ POSITION SYMTAB */  
*ENTERA(01,DYNA)     */ MAXSIZE OF FIXED DYNAMIC STORAGE */  
*FOR THIS PROCEDURE */  
*BLKEXT("1")     */ PREVENT UNRESOLVED LABEL LINKAGE */  
/ .OUTSIDE THIS PROCEDURE */  
*UNSTACK(DYNA,DYNA,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(SetFR)  
/ .EMPTY;
ARGID := ID SEARCH("1") ENTERA(01,DYNAMB) /* ACTUAL PARAMETER ADDRESSES */
   ENTERL(00,60) DO(A,ONE,ARGCNT) /* IN DYNAMIC STORAGE MARK */
   DO(A,F0UR,DYNAMA,SFT) /* DUMMY ARGS INDIRECT REF */

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

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

ARGATR := ATTRIBT ENTERA(05,LENGTH) DO(A,OCT60,TYPE)
   ENTERA(00,TYPE)
   "ENTRY" ( "RETRNS(" ATTRIBT ")" DO(A,OCTL20,TYPE) ENTERA(00,TYPF)
   EMPTY ENTERL(00,20) 

CLLSMT:= ID TEST(SYMB0L=CALLC0N) ID CANCEL SEARCH CALLPRT;

CALLPRT:= SAV(LDA,***,JUMPA) EMPTY /* SAVE CODE TO JUMP TO PROC */
   SET(SYMLAB=SYMBOL) /* SAVE PROC NAME FOR CHECKING */
   SET(STMAB=STMAB) /* CODE TO LOAD RETURN ADDRESS */
   OUT(LD,0100000065) /* CODE TO LOAD CURRENT DISPLAY ADDR */
   SET(SYMBOL=STMAB) 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)

OUT(STO)

OUT(LD,01) DEFLAB(*1)

ENTERL(00,01)

OUT("0000")

DO(POP)

DO(SET)

ERR("F: INVALID PROC CALL")

CLLPRM /* CODE TO STORE ADDR OF GLOBAL DISPLAY ADDR IN NEXT INSTR */

SET(SYMBOL=STACKTP)

OUT(LD,**)

OUT(#)

DEFLAB(*1) ;

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

/* COMPUTE ADDR OF GLOBAL DISPLAY ADDR*/

/* CODE TO LOAD GLOBAL DISPLAY ADDR */

/* CODE TO LOAD GLOBAL DISPLAY ADDR */

/* POP *1 LABEL */

/* SET TF CODE FOR COMPILER TESTING */

/* DEFINE RETURN ADDRESS */

CLLPRM := OUT(FLAG)

STACK(PARMCNT) SET(PARMCNT=ZERO)

STACK(STMLAB) /* SAVE PROC NAME */

( PARMPRT / *EMPTY ) /* PROC NAME OFF STACK FOR SEARCHING */

DO(PARMS) /* SEARCH("0") */

TSTTBL(00,11) / TSTTBL(00,12) / TSTTBL(00,13) / TSTTEL(00,10)

TSTTEL(07*PARMCNT)

ERR("W: INCORRECT PARM CNT",SET)

/ *EMPTY */ UNSTACK(PARMCNT);

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

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

PARMID := LATCH(ALFTPT) / LATCH(SLFTPT) / LATCH(LLFTPT)

/ LATCH(PROCHK) ;

PROCHK := ID ("(" DO(SETF,BEF) / *EMPTY )
* SEARCH("O")
  ( * TSTBL(00,10) CANCEL GOPROC /* 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,**,LD,0100000065,STO) /* CODE TO PUT CURRENT DISPLAY ADDR */
  /* IN DYNAMIC STORAGE AS GLOBAL DISPLAY */
  /* AT POINT OF CALL */
  *DO(A,FOUR,DYNAMB) ;

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

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

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

/* CODE TO LOAD ADDR FOR STORING GLOBAL DISPLAY */
/* CODE TO LOAD ADDR FOR STORING COMPUTED ADDR OF PASSFD GLOBAL DISPLAY ADDRESS */
/* RESTORE ADDR OF PROC ADDR CODE */
/* COMPUTE ADDR OF GLOBAL DISPLAY ADDR */
/* CODE TO STORE IN NEXT INSTR */
/* CODE TO LOAD GLOBAL DISPLAY ADDR */
/* MARK INTEGER */
/* CODE TO STORE IN CURR DYNAMIC STOR */

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

DECL1 :=IDELMNT / IDGROUP;

IDGROUP:="(" LISTPT ;

LISTPT :=•ID •STKSYM:
•STACK(DIMCNT)
•STACK(UIMCNC)
•IDLIST
•UNSTACK(SYMSAVE,DIMCNT)
•ARRAYSEM
•IDLIST
•DO(POP),SEARCH("1")
•IDSEM
•DO(A,LENGTH,DYNAMD) ) ; /* BUMP DYNAMIC STORAGE COUNTER */

IDLST :="." LISTPT / "") * ATTRIBT ; /* NOTE MUTUAL RECURSION WITH LISTPT */

ATTRIBT:="("CHARACTER(" / "CHAR(").INUM ")"
*/ TWELVE BIT (TWO CHARACTER) LENGTH */
*/ CHARACTER TYPE CODE */
/ "RETURNS (" ATTRIBT ")
*DO(A, OCTEN, TYPE)
*SET(LENGTH = 0000)
/ "FIXED" *SET(LENGTH = 0004)
*SET(TYPE = 01)
/ "FLOAT" *SET(LENGTH = 0010)
*SET(TYPE = 02)
/ "LOGICAL" *SET(LENGTH = 0001)
*SET(TYPE = 03)
/ "LABEL" *SET(LENGTH = 0004)
*SET(TYPE = 05);

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

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

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

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

IDE-MNT := *ID .STKSYM /* SIMILAR TO "LISTPT" BUT NO */
( ARRYPT ATTRIBT /* RECURSION WITH "IDLIST" */
*UNSTACK(SYMSAV)
ARRYSEM
/ ATTRIBT *DO(Popped) .SEARCH("1")
IDSEM *DO(A, LENGTH, DYNAMB) ) ;

ARRYSEM := *.SET(SYMBOL = STACKTP)
OUT (ALLOC, **)

SET (SYMBOL=BLNK8)

SET (SYMBOL=SYM5AV)  SEARCH("I")  /* POSITION SYMTAB POINTER */

ENTERA (07, DIMCNT)  /* ENTER DIMENSION COUNT */

OUT (**)

OUT (**(01, 07))

OUT (**(02, 05))

OUT ("")

DO (A, D0PFIX, DYNAMB)

SET (SYMBOL=FOUR)

DO (M, DIMCNT, SYMBOL)

DO (A, SYMBOL, DYNAMB)

DO (SET) ;

IDSIM := ENTERA (00, TYPE)  /* ENTER TYPE CODE */

ENTERA (05, LENGTH)  /* ENTER LENGTH */

. TEST (LENGTH=FUNCT)  . RETURN

/ * ENTER LEVEL AND DISPLACEMENT */

TDA := 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, 71000005, MULT)  /* CODE TO MULT 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) STMTEN  /* LABEL AND COMPILE STATEMENT */
OUT(JUMP,*2)  /* CODE TO JUMP OUT OF CASE GROUP */
CAT(JUMP,*1)  /* CATEGORIZE CODE TO TRANSFER VECTOR */
DO(P)  /* POP EXTRANEOUS LABEL */
OUT(LD+06)  /* DUMMY FOR POSTLISTING */
DEFLAB(*2)  ENTREL(00+01)  /* ADDR OF ADCON AND MARK INTEGER */
DO(SWAP,POP)  /* DISCARD ADCON LABEL AND PUT EXIT */
OUT(*2)  /* LABEL ON TOP */
DEFLAB(*2)  /* GENERATE ADCON */
OUT(*)  /* DEFINE TV ADDRESS */
ENDING  DEFLAB(*1)  /* DEFINE CASE GROUP EXIT ADDRESS */

WHILE := "WHILE" .NEWLAB .DEFLAB(*1) .0EFLAB(*1) .NEMLAB 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  /* LOOP EXIT LABEL */
.NEWLAB  /* CODE TO LOAD ADDRESS FOR INIT VALUE */
.OUT(LDA,**)  /* SAVE IDENTIFIER FOR LATER USE */
.STKSYM  /* CODE TO PREPARE FOR INCREMENT */
.INTERPRT  /* AND TEST */
.OUT(JUMPF,*1)  /* LOOP EXIT CODE */
(WHILE / ":;" )  ERR("H: EXPECTED ",SET)
.TAIL  /* RETRIEVE LOOP INDEX */
.DO(P)  /* CODE TO PREPARE FOR INCREMENT */
.OUT(LDA,**,LD,**)  /* CODE STACK */
.OUT(*)  /* RESTORE END OF LOOP CODE FROM */
.DEFLAB(*1) ;  /* DEFINE LOOP EXIT ADDRESS */
ITERPRT:="="  *ERRLATCH 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(LGCPRM))          /* COMPILABLE SIMPLE PRIMARY IF ANY */
/  *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(LBI)  OUT(**)   /* CODE TO LOAD DYNAM STOR ADDRESS */
  EXP  OUT(STO)              /* COMPILABLE LIMIT EXPRESSION */
  *MARK                             /* NEW REORDERING POINT */
  *DEFLAB(*2)                    /* REDEFINE ITERATION LABEL */
  OUT(LD)  DO(LBI)  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)          /* COMPILABLE LOOP INDEX TESTING CODE */
  "+"  OUT(02)  I0CHK  SAV(SUB)  /* AND STACK PROPER OP CODE FOR */
/  EMPTY  OUT(04)  I0CHK  SAV(ADD)  /* INCREMENTING LOOP INDEX */
  *MARK                             /* POSSIBLE CODE REORDERING POINT */
/  (*LATCH(LGCPRM))          /* COMPILABLE SIMPLE PRIMARY IF ANY */
/  *DO(S0P)  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(LBI)  OUT(**)   /* CODE TO LOAD DYNAM STOR ADDRESS */
  EXP  OUT(STO)              /* COMPILABLE LIMIT EXPRESSION */
  *DEFLAB(*2)                    /* REDEFINE ITERATION LABEL */
  OUT(*)                        /* RESTORE STACKED TEST CODE */
  *MARK                             /* LOOP END CODE FOLLOWING TO BE SAVED*/
  OUT(LD)  DO(LBI)  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(SJUMP+1) /* FINAL LOOP END CODE */
SAVE /* SAVE ALL LOOP END CODE */
/* RESTORE ILIST CODE IF I/O ITERATION */

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

LOOPRM := (.L .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 ) ;

BOOLEAN := (.LATC.IFCLSE) .NEWLAB .OUT(JUMPF+1)
 .BTERM .NEWLAB .OUT(JUMPF+1)
 ."ELSE" .DECLAB(2) BOOLEAN
 .DECLAB(1) )
 / BTERM ;

BTERM := BFAC$T ( ".OR." BFAC$T .OUT(OR) ) ;
BFAC$T := BCNDRY$ ( ".AND." BCNDRY$ .OUT(AND) ) ;
BCNDRY$ := PMRY/ ".NOT." PMRY .OUT(NOT) ;
BPRMRY := BVALUE / LATCH(BVARBLE) / LATCH(BFNCT) / RELATN / 
("" BOOLEXP "") .ERP(""W: EXPECTED "") *SET) ;

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

BFNCT := ID .SEARCH("0") ( TSTTBL(00,13) / TSTTBL(00,23) ) .CANCEL 
CALLPRT ;

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

BVARBLE := ID .SEARCH("0") ( TSTTBL(00,03) / TSTTBL(00,83) ) .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("0") ( 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("0") ( TSTTBL(00,01) / TSTTBL(00,02) 
/ TSTTBL(00,61) / TSTTBL(00,62) ) .CANCEL 
("(" .OUT(LDA,** SUBLIST ")" .OUT(INDXR#) / .EMPTY .OUT(LD,** ) ) ;

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

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

SSE :=STERM $( "//" STERM •OUT(CAT) ) ;

STEPM :=SBSSTRNG / •STRING •OUT(LDA,74,* ) / •LATCH(SVAR),BLE) / •LATCH(SFUNCT) ;

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

SBSSTRNG :="SUBSTR(" EXP "," EXP
( "," EXP / •EMPTY •OUT(LD,71,"0000"))")" •OUT(SUBSTR) ;

SVAR,ABLE :=•ID •SEARCH("O") ( •TSTTBL(00,04) / •TSTTBL(00,64) ) •CANCEL
•OUT(LDA,**,**(02,05)) ( "(" SUBLIST ")" •OUT(INDXA,**)
/ •EMPTY ) ;

SUBLIST :=•DO(MOVE,ONE,DIMCNT) /* INITIALIZE SUBSCRIPT COUNTER */
•STKSYM /* SAVE IDENTIFIER */
EXP /* COMPILE SUBSCRIPT EXPRESSION */
$ ( "," EXP •DO(A,ONE,DIMCNT) )
•DO(PGP) •SEARCH /* RESTORE IDENTIFIER AND SYMTAB POINT*/
•TSTTBA(07,DIMCNT) •ERR("W: INCORRECT SUBSCRIPT COUNT")
•DO(MOVE ,DIMCNT,SYMBO) ; /* 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 := ( .TSTTBL(00,65) .SET(PEXITT=PEXITF) / .SET(PEXITT=00) .TSTTBL(00,05) )
  ( "(" .OUT(LDA,** )SUBLIS T ")" .OUT(INDXR,* )
    / .EMPTY .OUT(LD,** ) ) ;

COND ST ::= LATCH(IFCLSE) .NEWLAB .OUT(JUMPF,*1)
  ELSE NEWLAB .OUT(JUMP,*1)
  .DEFLAB(*2) STMENT .DEFLAE(*1)
  / .EMPTY .DEFLAB(*1) ;
IFCLSE ::= ID TEST(SYMBOL=IFSYM) .CANCEL BOOLEXP "THEN" ;
IDENT ::= ID .SEARCH("0") .TSTTBL(00,00) .DO(BM,"W: UNDECLARED VARIABLE") ;
UNCOND ::= LATCH(GOTOST) / INOUT / LATCH(CLLSTMT) / LATCH(RTNSTMT)
  / STPSTM / ASSGNST / IDENT ;
STPSTM ::= "STOP" .OUT(STOP) ;
R TNSTMT ::= "RETURN"( "(" ( EXP / SSE / BPRMRY ) ")" .OUT(SWP)
  / .EMPTY ) .OUT(RETURN) ;
GOTOST ::= "GO" "TO" .CANCEL .MARK LTERM
  ( .TEST(PEXITT=PEXITF) .SAV .OUT(POPUP,#) .OUT(RETURN)
    / .EMPTY .OUT(JUMPA) ) ;
ASSGNST ::= AASSGN / BASSGN / SASSGN / LASSGN ;

AASSGN ::= LATCH(ALFTPT) .SAV(STO)
  $ ( "", ALFTPT .MARK .OUT(ST, #) .SAV )
  "=" EXP .OUT( # ) ;
BASSGN ::= LATCH(bLFTPT) .SAV(STO)
  $ ( "", bLFTPT .MARK .OUT(ST, #) .SAV )
  "=" BOOLEXP .OUT(# ) ;
SASSGN ::= SLFTPT .SAV(STO)
  $ ( "", SLFTPT .MARK .OUT(ST, #) .SAV )
  "=" SEXP .OUT(# ) ;
LASSIGN := •LATCH(LLFTPPT) •SAV(STO)  
$ \{ "\" LLFTPPT •MARK •OUT(SST*,#) •SAV \}  
\"=\" LTFRM •TEST(PEXITT=ZERO) •ERR(\"F: INDIRECT LABEL ASSIGNMENT\")  
•OUT(\#) ;

ALFTPPT := •ID •SEARCH("0")  
( •TSTTBL(00,01) / •TSTTBL(00,02)  
/ •TSTTBL(00,61) / •TSTTBL(00,62) ) •CANCEL  
•OUT(LDA,**) SUBPART ;

BLFTPPT := •ID •SEARCH("0")  
( •TSTTBL(00,03) / •TSTTBL(00,63) ) •CANCEL  
•OUT(LDA,**) SUBPART ;

LLFTPT := •ID •SEARCH("0")  
( •TSTTBL(00,05) / •TSTTBL(00,65) ) •CANCEL  
•OUT(LDA,**) SUBPART ;

SLFTP := •LATCH(SVARBLE) / LSUBSTR ;

LSUBSTR="SUBLSTR(" SVARBLE "," EXP  
( "," EXP / •EMPTY •OUT(LD,71,"0000") ) ")"  
•OUT(SUBSTR) ;

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

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

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

ILIST := IELMNT \$ ( "," IELMNT ) ;
IELMN T := IRPLIST
  / (. LATCH (SLFTPT) / . LATCH (ALFTPT) / . LATCH (BLFTPT) ) . OUT (GFT) ;
IRPLIST := "(" . MARK I LIST . SAV E RPLIST ;
ERPLIST := "DO" . ID UNDTST . NEWLAB /* "ERPLIST" IS SIMILAR TO "LOOP" */
  / OUT (LDA, **) . STKSYM
  / SET (IOSW = IO ITER) /* SET SWITCH FOR I TERPR T TO TEST */
ITERPR T
  / SET (IOSW = 00)
  / DO (POP) . OUT (LDA, **) . LD, **)
  / OUT (#)
  / DEFLAB (*1) "") ;
OUTPUT := "PUT" ( "STRING" . SAV (00) "(" SLFTPT ")"
  / . EMPTY . SAV (01) )
  / "EDIT" . CANCEL . NEWLAB . OUT (FMT, *1, *1)
  / (" OLIST ")" . OUT (PUT) . NEWLAB . OUT (JUMP, *1)
  / DEFLAB (*2) FMT LIST . DEFLAB (*1) ;
OLIST := OELMNT $ ( "*$ OELMNT ) ;
OELMNT := ORPLIST
  / (EXP / SSE / . LATCH (BVAR BLES ) ) . OUT (EDIT) ;
ORPLIST := "(" . MARK OLIST . SAV E RPLIST ;
FMT LIST := "(" . OUT (77) FMT ITEM $ ( "*$ FMT ITEM ")"
  / . OUT (77) ;
FMTITEM := CTRLFMT / DATAFMT ;
CTRLFMT := "X" . OUT (00) SPEC
  / "PAGE" . OUT (01)
  / "SKIP" . OUT (02) ( SPEC / . EMPTY . OUT ( "0001") )
  / "COL" . OUT (03) SPEC ;
DATAFMT := "A" . OUT (10) ( SPEC / . EMPTY . OUT ( "0000") )
/ "E" .OUT(11) SPEC
/ "I" .OUT(12) SPEC / "L" .OUT(13) SPEC ;
 SPEC :="(" .INUM .OUT(*) ")" ;

*ENC
****COMPILED PROGRAM SIZE = 7,362; METAX INSTRUCTION COUNT = 128,483****
****SYMTAB SEARCH COUNT = 2,308; SYMTAB COMPARE COUNT = 857,907****
****SYMBOL TABLE ENTRY COUNT = 846****
MOVE STACKT SYMBOL
SEARCH "O"
ENTL "$1000"
$006 CLM COMMEN BT $006 SET BEF CLM BHDBDY BEF TEST "END" BEF OUT STOP LB1 ENTELOC ENTL "O" "6" EnTA "1" DYNAMP ID BF $008 CHKSYM BT $009 BM "w: POSSI SET $009 B $007 $008 SET BF $007 $007 BEF BLKEXT "1" RESOLV $005 R BLKBDY MOVE BLNKH STMLAB
BF $061
ENTL "0"
ENTL "0"
ENTL "0"
ENTL "8"
BEF $059
SET BLK
ENTL MOVI EATCH "1"
MOVE SYMBOL STMLAB SYMBOL
MOVE DYNAMP SYMBOL
MOVE DYNAMP SYMBOL
MOVE LEVNO SYMBOL
MOVE LEVNO SYMBOL
MOVE STACKT SYMBOL
MOVE ONELEV SYMBOL
MOVE DYNAMP DYNAMP DYNAMP DYNAMP
A ONELING STACKT STACKT STACKT
MOV4 STACKT STACKT STACKT STACKT STACKT STACKT
SEARCH "1"
ENTL "1"
ENTL "5"
MOVE FOUR SYMBOL
M LEVNO SYMBOL
A FOUR SYMBOL
A SYMBOL DYNAMP
OUT ENTPRO
MOVE LEVNO SYMBOL
OUTSYM OUTSYM DYNAMP LBI EVAL "4"
"1"
MOVE ZERO ARGID TEST "(" BF CLM BEF $064
BEF ARGID BEF BEF BEF $067
BEF BEF BEF BT $065
BEF $065
SET TEST ")" BEF
BM "W: INCORRECT ARG
SET B $069
BF $069
BEF RHDDBY
TEST "END"
BEF OUT RETURN
LB1 ENTLOC
ENTL

"0"
"6"
ENTA "1"
DYNAMP "1"
BLKEXT PAGE
POP
MOVE SYMBOL
STACKT
POP
POP
MOVE SYMBOL
LEVNO
MOVE DYNAMB

ID "W: EXPECTED CLOS
SET BM $079
BT $079
ENTL "ENTRY"
BF $095
TEST "RETURNS(" BF $099
CLM ATTRIB BEF TEST ")"
BEF A OCTL20 TYPE
ENTA "0"
BF $098
SET BF $098
ENTL "0" "+"
BEF $095 R
CLLSTM ID BF $102
COMP SYMBOL CALLCO "2"
BEF ID BEF CANCEL SEARCH "0"
CLM CALLPR BEF $102 R
CALLPR MARK OUT LDA EVAL "5" "0"
OUT JUMPA
SAVE SET BF $104
MOVE SYMBOL STMLAB PUSHLB OUT LDA OUT "6"
OUT LD OUT IOCHK MOVE STMLAB SYMBOL
SEARCH "0" TSTTBL "0" "8"
BF $108 B $107 $108 TSTTBL "0" "9"
BF $109 B $107 $109 TSTTBL "0" "*"
BF $110 B $107 $110 TSTTBL "0"
<table>
<thead>
<tr>
<th>BF</th>
<th>$111</th>
</tr>
</thead>
<tbody>
<tr>
<td>BF</td>
<td>$107</td>
</tr>
<tr>
<td>$111</td>
<td>TSTTBL</td>
</tr>
<tr>
<td>BF</td>
<td>$107</td>
</tr>
<tr>
<td>BF</td>
<td>$106</td>
</tr>
<tr>
<td>OUT</td>
<td>LD</td>
</tr>
<tr>
<td>OUT</td>
<td>IOCHECK</td>
</tr>
<tr>
<td>B</td>
<td>$105</td>
</tr>
<tr>
<td>$106</td>
<td>TSTTBL</td>
</tr>
<tr>
<td>BF</td>
<td>$115</td>
</tr>
<tr>
<td>B</td>
<td>$114</td>
</tr>
<tr>
<td>$115</td>
<td>TSTTBL</td>
</tr>
<tr>
<td>BF</td>
<td>$116</td>
</tr>
<tr>
<td>B</td>
<td>$114</td>
</tr>
<tr>
<td>$116</td>
<td>TSTTBL</td>
</tr>
<tr>
<td>BF</td>
<td>$117</td>
</tr>
<tr>
<td>B</td>
<td>$114</td>
</tr>
<tr>
<td>$117</td>
<td>TSTTBL</td>
</tr>
<tr>
<td>BF</td>
<td>$118</td>
</tr>
<tr>
<td>B</td>
<td>$114</td>
</tr>
<tr>
<td>$118</td>
<td>TSTTBL</td>
</tr>
<tr>
<td>BF</td>
<td>$114</td>
</tr>
<tr>
<td>BF</td>
<td>$105</td>
</tr>
<tr>
<td>MARK</td>
<td>OUT</td>
</tr>
<tr>
<td>OUT</td>
<td>LD</td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>EVAL</td>
<td>&quot;4&quot;</td>
</tr>
<tr>
<td>DMA</td>
<td>LOA</td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>LB1</td>
<td>EVAL</td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>RESTOR</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>LD</td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;Z&quot;</td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;0004&quot;</td>
</tr>
<tr>
<td>OUT</td>
<td>ADD</td>
</tr>
<tr>
<td>OUT</td>
<td>STO</td>
</tr>
<tr>
<td>OUT</td>
<td>LD</td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>LB1</td>
<td>ENTLOC</td>
</tr>
<tr>
<td>$116</td>
<td>ENTL</td>
</tr>
<tr>
<td>$117</td>
<td>ENTL</td>
</tr>
<tr>
<td>$118</td>
<td>ENTL</td>
</tr>
<tr>
<td>BF</td>
<td>$118</td>
</tr>
<tr>
<td>B</td>
<td>$114</td>
</tr>
<tr>
<td>$115</td>
<td>TSTTBL</td>
</tr>
<tr>
<td>BF</td>
<td>$115</td>
</tr>
<tr>
<td>B</td>
<td>$114</td>
</tr>
<tr>
<td>$114</td>
<td>TSTTBL</td>
</tr>
<tr>
<td>BF</td>
<td>$105</td>
</tr>
<tr>
<td>BF</td>
<td>$114</td>
</tr>
<tr>
<td>MARK</td>
<td>OUT</td>
</tr>
<tr>
<td>OUT</td>
<td>LD</td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>EVAL</td>
<td>&quot;4&quot;</td>
</tr>
<tr>
<td>&quot;1&quot;</td>
<td>ENTLOC</td>
</tr>
<tr>
<td>$116</td>
<td>ENTL</td>
</tr>
<tr>
<td>$117</td>
<td>ENTL</td>
</tr>
<tr>
<td>$118</td>
<td>ENTL</td>
</tr>
<tr>
<td>BF</td>
<td>$118</td>
</tr>
<tr>
<td>B</td>
<td>$114</td>
</tr>
<tr>
<td>$115</td>
<td>TSTTBL</td>
</tr>
<tr>
<td>BF</td>
<td>$115</td>
</tr>
<tr>
<td>B</td>
<td>$114</td>
</tr>
<tr>
<td>$114</td>
<td>TSTTBL</td>
</tr>
<tr>
<td>BF</td>
<td>$105</td>
</tr>
<tr>
<td>BF</td>
<td>$114</td>
</tr>
<tr>
<td>MARK</td>
<td>OUT</td>
</tr>
<tr>
<td>OUT</td>
<td>LD</td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>EVAL</td>
<td>&quot;4&quot;</td>
</tr>
<tr>
<td>&quot;1&quot;</td>
<td>ENTLOC</td>
</tr>
</tbody>
</table>
### Listing

<table>
<thead>
<tr>
<th>Location</th>
<th>Instruction</th>
<th>Location</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>$104</td>
<td>ENTL &quot;0&quot;</td>
<td>TSTTB8A</td>
<td>&quot;7&quot;</td>
</tr>
<tr>
<td></td>
<td>ENTL &quot;6&quot;</td>
<td>PARMCN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>BT $133</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>FLAG PARMCN</td>
<td>BM &quot;W: INCORRECT PAR</td>
<td></td>
</tr>
<tr>
<td>MOVE</td>
<td>SYMBOL</td>
<td>SET B $126</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PARM CN SYMBOL</td>
<td>SET BF $126</td>
<td></td>
</tr>
<tr>
<td>STKSYM</td>
<td>ZERO PARM CN</td>
<td>BEF BF $126</td>
<td></td>
</tr>
<tr>
<td>MOVE</td>
<td>STMLAB SYMBOL</td>
<td>POP MOVE SYMBOL PARM CN</td>
<td></td>
</tr>
<tr>
<td>STKSYM</td>
<td>CLM PARM PR</td>
<td>$122 R PARM PR TEST &quot;(&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$124</td>
<td>SET BF $123</td>
<td>$137 TEST ,.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BF $122</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>POP BF $123</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEARCH &quot;0&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TSTTB6L &quot;0&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;9&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BF $129</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B $128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$129</td>
<td>TSTTB6L &quot;0&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;9&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BF $130</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B $128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$130</td>
<td>TSTTB6L &quot;0&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;9&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BF $131</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B $128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$131</td>
<td>TSTTB6L &quot;0&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;8&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BF $128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$128</td>
<td>BF $127</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$141 R PARMID LATCH ALFTPT</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$146</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>$145</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$147</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>$145</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$148</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>$145</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$149</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>$145</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>$152</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>$152</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$153</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$153</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$154</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td>$153</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TSTTB</td>
<td>$157</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$156</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TSTTB</td>
<td>$157</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$156</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPROC</td>
<td>$157</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LATCH**

BLFPT

**LATCH**

SLFPT

**LATCH**

LLFPT

**LATCH**

PROCHK

**PROCHK**

ID

**SEARCH**

"0"

**SEARCH**

"8"

**SEARCH**

"+"

**GPROC**

**MARK**

**PUSHLB**

**SEARCH**

"1"

**ENTA**

"1"

**DYNAMB**

**ENTL**

"0"

**EIGHT**

**DYNAMB**

**OUT**

LDA

**EVAL**

"5"

**EVAL**

"0"

**CLM**

EXP

**CLM**

SSE

**CLM**

LTERM

**CLM**

BPRMRY

**OUT**

STO

**OUT**

LDA

**OUT**

"1"
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"

DYNAMB
R $166
IPROC MARK
OUT LD
OUT "1"
EVAL "4"
"1"

SAVE MARK
OUT LD
OUT "/"
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"
ENTL  "0"
"1"

OUT  LDA
EVAL  "5"
"0"
A  FOUR
DYNAMB

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

RESTOR  OUT  LD
OUT  "2"
OUT  "0004"
OUT  ADD
OUT  STO
OUT  LD
OUT  "1"
LB1  ENTLLOC
ENTL  "0"
"6"
ENTL  "0"
"1"

OUT  "0000"
OUT  STO

$168  R
DECLAR  TEST  "DECLARE"
BF  $171
MOVI  ELATCH

CLM  DECL1
BT  $172
BM  "F"
SCAN  ";
SET

$172  TEST  ";
BF  $175
CLM  DECL1
BT  $175
BM  "F"
SCAN  ";
SET

$175  BT  $172
SET
BEF
TEST  ";
BEF

$171  R
DECL1  CLM  IDELMN
BF  $178
B  $177

$178  CLM  IDGROU
BF  $177

$177  R
IDGROU  TEST  "(
BF  $181
CLM  LISTPT
BEF

$181  R
LISTPT  ID
BF  $183
STKSYM
CLM  ARRYPT
BF  $185
MOVE  DIMCNT
SYMBOL
<table>
<thead>
<tr>
<th>Instruction</th>
<th>$219</th>
<th>$218</th>
<th>$217</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDELMN</td>
<td>BF</td>
<td>BF</td>
<td>BF</td>
</tr>
<tr>
<td>STKSYM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td>ARRAYT</td>
<td>ATTRIB</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$219</td>
<td>$219</td>
<td>$217</td>
</tr>
<tr>
<td>CLM</td>
<td>BF</td>
<td>BF</td>
<td>BF</td>
</tr>
<tr>
<td>BEF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POP</td>
<td>MOVE</td>
<td>SYMBOL</td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td>SYMMAV</td>
<td>ARRAYSE</td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td>$218</td>
<td>$218</td>
<td>$217</td>
</tr>
<tr>
<td>E</td>
<td>BF</td>
<td>BF</td>
<td>BF</td>
</tr>
<tr>
<td>$219</td>
<td>CLM</td>
<td>ATTRIB</td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td>$218</td>
<td>$218</td>
<td>$217</td>
</tr>
<tr>
<td>POP</td>
<td>SEARCH</td>
<td>&quot;1&quot;</td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td>IDSEM</td>
<td>IDSEM</td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>LENGTH</td>
<td>DYNAMB</td>
<td></td>
</tr>
<tr>
<td>$222</td>
<td>R</td>
<td>$222</td>
<td></td>
</tr>
<tr>
<td>$218</td>
<td>BF</td>
<td>BF</td>
<td>BF</td>
</tr>
<tr>
<td>$217 Bez</td>
<td>MOVE</td>
<td>SYMBOL</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>ALLOC</td>
<td>BF</td>
<td></td>
</tr>
<tr>
<td>EVAL</td>
<td>&quot;5&quot;</td>
<td>&quot;0&quot;</td>
<td></td>
</tr>
<tr>
<td>MOVE</td>
<td>BLNK8</td>
<td>$226</td>
<td></td>
</tr>
<tr>
<td>SYMBOL</td>
<td>BF</td>
<td>BF</td>
<td>BF</td>
</tr>
<tr>
<td>$225</td>
<td>ENT</td>
<td>ENT</td>
<td>ENT</td>
</tr>
<tr>
<td>SEARCH</td>
<td>&quot;1&quot;</td>
<td>$225</td>
<td></td>
</tr>
<tr>
<td>$224</td>
<td>BEF</td>
<td>BF</td>
<td>BF</td>
</tr>
<tr>
<td>$224</td>
<td>R</td>
<td>$224</td>
<td></td>
</tr>
<tr>
<td>$223</td>
<td>DYNAMB</td>
<td>DYNAMB</td>
<td></td>
</tr>
<tr>
<td>$222</td>
<td>BF</td>
<td>BF</td>
<td>BF</td>
</tr>
<tr>
<td>$222</td>
<td>BF</td>
<td>BF</td>
<td>BF</td>
</tr>
<tr>
<td>$222</td>
<td>BF</td>
<td>BF</td>
<td>BF</td>
</tr>
</tbody>
</table>

Note: The above table represents a sequence of assembly language instructions with corresponding addresses and values. Each instruction typically consists of a mnemonic (e.g., IDELMN), followed by operands and possibly addresses. This type of code is used for programming and debugging in assembly language environments.
<table>
<thead>
<tr>
<th>$246 R</th>
<th>WHILE</th>
<th>&quot;WHILE&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>BF</td>
<td>TEST</td>
<td>&quot;$248</td>
</tr>
<tr>
<td>CLM</td>
<td>WHLPT</td>
<td>BEF</td>
</tr>
<tr>
<td>TEST</td>
<td>&quot;$;&quot;</td>
<td>BEF</td>
</tr>
<tr>
<td>$255</td>
<td>TEST</td>
<td>&quot;$;&quot;</td>
</tr>
<tr>
<td>$248 R</td>
<td>WHLPT</td>
<td>&quot;{&quot;</td>
</tr>
<tr>
<td>BF</td>
<td>$250</td>
<td>BEF</td>
</tr>
<tr>
<td>CLM</td>
<td>BOOLEAN</td>
<td>$251</td>
</tr>
<tr>
<td>BT</td>
<td>$251</td>
<td>SET</td>
</tr>
<tr>
<td>BM</td>
<td>&quot;F: LOGICAL&quot;</td>
<td>$257</td>
</tr>
<tr>
<td>SCAN</td>
<td>&quot;$;&quot;</td>
<td>CLM</td>
</tr>
<tr>
<td>R</td>
<td>$251</td>
<td>TAIL</td>
</tr>
<tr>
<td>OUT</td>
<td>JUMPF</td>
<td>BEF</td>
</tr>
<tr>
<td>LB1</td>
<td>EVAL</td>
<td>POP</td>
</tr>
<tr>
<td>$250</td>
<td>BF</td>
<td>OUT</td>
</tr>
<tr>
<td>LOOP</td>
<td>&quot;$253</td>
<td>RESTOR</td>
</tr>
<tr>
<td>BF</td>
<td>$253</td>
<td>LB1</td>
</tr>
<tr>
<td>ENTLOC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CLM IOCHK
BEF
MARK OUT ADD
SAVE BEF MARK LATCH LOOPRM BF $271 RF $270 $270 BT $273 
$271 POP SAVE PUSHLB LB1 ENLOC ENL ENT
ENTL $273 RESTOR B $265
$266 SET BF $265 OUT STCKC OUT "4"
ENT A EIGHT DYNAMB OUT LD OUT "2"
OUT LDA $265 BEF OUT SST OUT JUMP LB1 EVAL "5"
"0"
"0"
"1"
"0"
"6"
"6"
""6"
""6"
"0"
"0"
"0"
"7"
"7"
"0"
"7"
"0"
""6"
""6"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
"0"
OUT JUMPT
LB2
EVAL "4"
  "1"
RESTOR SET B $276 SET BF $275
$275 R LOOPRM ID BF $281 OUT LD EVAL "5"
  "0"
  B $280 INUM BF $282 OUT LD OUT "Z"
OUTSYM E $280 FNUM BF $280 OUT LD OUT "z"
OUTSYM E $279 TEST "+"
  BF $287 B $286
$287 TEST "-"
  BF $288 B $286
$288 TEST "("
<table>
<thead>
<tr>
<th>$302</th>
<th>SET</th>
<th>$301</th>
<th>$305</th>
<th>CLM</th>
<th>BTERM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BF</td>
<td>$301</td>
<td></td>
<td>BF</td>
<td>$304</td>
<td></td>
</tr>
<tr>
<td>$301</td>
<td>BEF</td>
<td>$307</td>
<td>TEST</td>
<td>&quot;$OR&quot;</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>LATCH</td>
<td>BF</td>
<td>BTERM</td>
<td>CLM</td>
<td>BFACTO</td>
</tr>
<tr>
<td>$300</td>
<td>GOL</td>
<td>BF</td>
<td>$310</td>
<td>CLM</td>
<td>BFACTO</td>
</tr>
<tr>
<td>BOOLEX</td>
<td>IFCLSE</td>
<td>BF</td>
<td></td>
<td>BEF</td>
<td>OR</td>
</tr>
<tr>
<td></td>
<td>PUSHLB</td>
<td>BF</td>
<td></td>
<td>OUT</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>JUMPF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LB1</td>
<td>EVAL</td>
<td>&quot;4&quot;</td>
<td></td>
<td>SET</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;1&quot;</td>
<td></td>
<td>BEF</td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td>BTERM</td>
<td>$310</td>
<td>R</td>
<td>BFACTO</td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td>PUSHLB</td>
<td>BF</td>
<td>TEST</td>
<td>&quot;$AND&quot;</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>JUMP</td>
<td>BF</td>
<td>CLM</td>
<td>BSCNDR</td>
<td></td>
</tr>
<tr>
<td>LB1</td>
<td>EVAL</td>
<td>&quot;4&quot;</td>
<td></td>
<td>BEF</td>
<td>AND</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;1&quot;</td>
<td></td>
<td>OUT</td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td>&quot;ELSE&quot;</td>
<td>SET</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td>LB2</td>
<td>$315</td>
<td>R</td>
<td>BPRMRY</td>
<td></td>
</tr>
<tr>
<td>ENTL</td>
<td></td>
<td></td>
<td>BSCNDR</td>
<td>CLM</td>
<td></td>
</tr>
<tr>
<td>ENTL</td>
<td>&quot;0&quot;</td>
<td></td>
<td>BF</td>
<td>$320</td>
<td>$319</td>
</tr>
<tr>
<td></td>
<td>&quot;6&quot;</td>
<td></td>
<td>B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td>BOOLEX</td>
<td>$320</td>
<td>TEST</td>
<td>&quot;$NOT&quot;</td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td>LB1</td>
<td>BF</td>
<td>CLM</td>
<td>BPRMRY</td>
<td></td>
</tr>
<tr>
<td>ENTL</td>
<td></td>
<td>BEF</td>
<td></td>
<td></td>
<td>NOT</td>
</tr>
<tr>
<td>ENTL</td>
<td>&quot;0&quot;</td>
<td></td>
<td>OUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;6&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$307</td>
<td>BF</td>
<td>$305</td>
<td>$319</td>
<td>R</td>
<td>BVALUE</td>
</tr>
<tr>
<td>B</td>
<td>$304</td>
<td></td>
<td></td>
<td>BPRMRY</td>
<td>CLM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BVALUE</td>
</tr>
</tbody>
</table>
CANCEL
TEST "(" OUT "2"
BF $346 SAVE $343
OUT LDA $351 TEST ">="
EVAL BF $352
"5" BF $352
"0" OUT "5"
CLM $351
BEF $348
TEST ")" TEST ">"
BEF BF $353
OUT INDXR MARK OUT "4"
OUTSYM $348
B $348
SET SAVE $348
BF $345 TEST "\#"
OUT LD BF $348
EVAL MARK OUT "6"

$345 B $348
BEF R
$341 $348 CLM TERM
RELOP SAL CLM TERM
TEST ">=" BF $358
BF $357
MARK OUT "1"
BEF $349
OUT CLM TERM
SAVE BF $358
B $348
TEST "<=" OUT NEG
BF $357
MARK BF $359
OUT "3"
SAVE CLM TERM
B $350
TEST "<" BF $357
MARK BF $356
OUT "4"
SAVE B $351
TEST "\+"
BF $363
$363
CLM TERM
BEF OUT ADD
B $362
TEST "-"
BF $362 CLM TERM
BEF OUT SUB
BT $361

$362
BT $361

$356
R
TERM CLM PRIMRY
BF $366
TEST "*"
BF $369 CLM PRIMRY
BEF OUT MUL
B $368
TEST "/
BF $368 CLM PRIMRY
BEF OUT DIV
B $367
SET

$366
R
PRIMRY TEST "(" BF $372 CLM EXP
BEF TEST ")" BF $371
E $371

$372
CLM CONST
BEF BF $373
B $371

$373
LATCH VARIABLE
BF $374
B $371

$374
LATCH AFUNCT
BF $371

$371
R
AFUNCT ID
BF $377
SEARCH "0"
TSTTBL "0"
"9"

$379
TSTTBL "0"
"*
"A"

$380
TSTTBL "0"
"B"

$381
TSTTBL "0"
"B"

$378
BEF CANCEL
CLM CALLPR
BEF

$377
R
CONST INUM
BF $384
OUT LD
OUT "Z"
OUTSYM
```
B $383  OUT  INDEXR
FNUM $383  OUTSYM  $393
BF $383  SET  BF $393
OUT LD  EVAL  "5"
OUTSYM "e"
OUT  $394  BF $393
B $394  SET  LB1  JUMP
VARIABLE ID  $387  BF $397
BF $387  R  "0"
SEARCH "0"  EXP  LATCH IFCLSE
TSTTBL "0"  $397  BF
"1"
B $390  PUSHLB  "4"
BF $388  OUT  "1"
"2"
B $388  JUMPF
TSTTBL "0"  "4"
"/"
BF $391  EVAL  "5"
B $388  "1"
TSTTBL "0"  SUBLIS  BEF
"S"
BF $388  TEST "ELSE"
BEF
"5"
CANCEL  "4"
BEF  "1"
TEST  "0"
"(
BF $394  LB1  LB1
OUT LDA  EVAL  EVAL
EVAL  "5"
"0"
"0"
CLM SUBLIS  CLM  CLM
BEF  EXP  BEF  BEF
TEST  "6"
")"
BEF
```
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>466</td>
<td>BF</td>
</tr>
<tr>
<td>UNCOND</td>
<td>BM</td>
</tr>
<tr>
<td>466</td>
<td>CLM</td>
</tr>
<tr>
<td>469</td>
<td>LATCH</td>
</tr>
<tr>
<td>470</td>
<td>LATCH</td>
</tr>
<tr>
<td>471</td>
<td>CLM</td>
</tr>
<tr>
<td>472</td>
<td>CLM</td>
</tr>
<tr>
<td>473</td>
<td>CLM</td>
</tr>
<tr>
<td>467</td>
<td>R</td>
</tr>
<tr>
<td>476</td>
<td>R</td>
</tr>
<tr>
<td>482</td>
<td>CLM</td>
</tr>
</tbody>
</table>

**Note:** The above text appears to be a memory dump from a computer system, possibly representing memory addresses and values.
<table>
<thead>
<tr>
<th>BF</th>
<th>$492</th>
<th>SAVL</th>
<th>TEST</th>
<th>&quot;,&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>$491</td>
<td>BF</td>
<td>CLM</td>
<td>$505</td>
</tr>
<tr>
<td>CLM</td>
<td>$492</td>
<td>BF</td>
<td>BLFTPT</td>
<td>BEF</td>
</tr>
<tr>
<td>BF</td>
<td>$493</td>
<td>MARK</td>
<td>OUT</td>
<td>SST</td>
</tr>
<tr>
<td>B</td>
<td>$491</td>
<td>RESTOR</td>
<td>SAVE</td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td>$494</td>
<td>BF</td>
<td>$491</td>
<td>$505</td>
</tr>
<tr>
<td>BF</td>
<td>$497</td>
<td>BT</td>
<td>$503</td>
<td></td>
</tr>
<tr>
<td>AASSGN</td>
<td>LATCH</td>
<td>ALFTPT</td>
<td>$497</td>
<td>SET</td>
</tr>
<tr>
<td>BF</td>
<td>$500</td>
<td>BF</td>
<td>BEF</td>
<td>&quot;=&quot;</td>
</tr>
<tr>
<td>MARK</td>
<td>OUT</td>
<td>STO</td>
<td>CLM</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>SAVE</td>
<td>TEST</td>
<td>&quot;,&quot;</td>
<td>BF</td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td>ALFTPT</td>
<td>$500</td>
<td>BF</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$500</td>
<td>CLM</td>
<td>SLFTPT</td>
<td>$507</td>
</tr>
<tr>
<td>MARK</td>
<td>OUT</td>
<td>STO</td>
<td>BF</td>
<td></td>
</tr>
<tr>
<td>RESTOR</td>
<td>SST</td>
<td>SAVE</td>
<td>TEST</td>
<td>&quot;,&quot;</td>
</tr>
<tr>
<td>SAVE</td>
<td>BT</td>
<td>$498</td>
<td>SET</td>
<td></td>
</tr>
<tr>
<td>HEF</td>
<td>TEST</td>
<td>&quot;,=&quot;</td>
<td>BEF</td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td>EXP</td>
<td>BF</td>
<td>CLM</td>
<td>SST</td>
</tr>
<tr>
<td>BF</td>
<td>RESTOR</td>
<td>$502</td>
<td>BF</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$502</td>
<td>SAVL</td>
<td>TEST</td>
<td>&quot;,&quot;</td>
</tr>
<tr>
<td>BASSGN</td>
<td>LATCH</td>
<td>BLFTPT</td>
<td>$502</td>
<td>SET</td>
</tr>
<tr>
<td>BF</td>
<td>$502</td>
<td>BF</td>
<td>BEF</td>
<td>&quot;=&quot;</td>
</tr>
<tr>
<td>MARK</td>
<td>OUT</td>
<td>STO</td>
<td>CLM</td>
<td>SEXP</td>
</tr>
<tr>
<td>OUT</td>
<td>STO</td>
<td>BEF</td>
<td>CLM</td>
<td></td>
</tr>
</tbody>
</table>

168
B Ef RESTOR
$507
R LASSGN
B Ef LATCH
MARK
OUT
SAVE
$513
TST $512
STO
B Ef CLM
MARK
OUT
SST
RESTOR
SAVE
$515
B Ef SET
TST $513
B Ef CLM
TERM
B Ef COMP
PEXIT
ZERO
"2"
B Ef $516
F: INDIR
$518
EF TSTTBL "0"
B Ef $520
SST
B Ef $521
TSTTBL "0"
B Ef $519
"2"
B Ef $521
TSTTBL "0"
B Ef $519
"2"
B Ef $522
TSTTBL "0"
B Ef $519
"2"
B Ef $519
CANCE L
OUT
LDA
"5"
B Ef $519
SUBPAR
B Ef $516
SUBPAR
B Ef $518
R
B Ef $525
SEARCH "0"
B Ef $521
TSTTBL "0"
B Ef $527
SEARCH "0"
B Ef $526
"3"
B Ef $526
"4"
B Ef $519
"5"
B Ef $519
"0"
B Ef $519
"0"
B Ef $519
"0"
B Ef $519
"0"
BEF $525
R
LLFTPT ID
BF $530
SEARCH "0"
TSTTBL "0"
"5"
BF $532
B $531
$532 TSTTBL "0"
"y"
BF $531
CANCELED $531
BEF OUT LDA EVAL "5"
"0"
BEF CLM SUBPAR BEF TEST

$530 R
SLFTPT LATCH SVARBL BEF $535
B $534
BF $534
BEF LATCH INPUT
BF $546
BEF MARK OUT SYM
BEF OUTPUT
BEF GET
BEF EXP $540
BEF SET $539
BEF OUT LD $539
BEF OUT "Z"
BEF OUT "0000"
BEF TEST "y"
BEF OUT SUBSTR $539
BEF TEST "(" $538
BEF SUBPAR TEST "(
BEF CLM SUBLIS
BEF TEST "y"
BEF OUT INDXA
BEF SET $542
BEF R $542
BEF INPUT
BEF $546
BEF B $545
BEF LATCH OUTPUT
BEF $545
BEF INPUT
BEF TEST "GET"
BEF $549
BEF TEST "STRING"
BEF $551
BEF MARK
BEF OUT "y"
SAVE TEST "(" BEF CLM SLFTPT BEF TEST ")" BEF SET $550 BF MARK OUT "Z" SAVE BEF TEST "EDIT" BEF CANCEL PUSHLB OUT FMT LB1 EVAL "4" "1" RESTOR TEST "(" BEF CLM ILIST BEF TEST ")" BEF PUSHLB OUT JUMP LB1 EVAL "4" "1" LB2 ENTLOC ENTL "0" "6" CLM FMTLB TEST BEF LB1 ENTLOC ENTL "0" "6" CLM IELMNT BF $554 TEST "," BF TEST "EDIT" BF CLM IELMNT BF SET BF $555 ILIST CLM IELMNT BF $554 R IELMNT CLM IRPLIS BF $559 B $558 LATCH SLFTPT BF $562 B $561 LATCH ALFTPT BF $563 B $561 LATCH BLFTPT BF $561 R GET OUT $558 R GET
BF $566
MARK
CLM ILIST
BEF SAVL
CLM ERPLIS
BEF R
TEST "DO"
BF $568
ID
BEF CLM UNDTST
BEF PUSHLB
OUT LDA
EVAL "5"
"0"
STKSYM
MOVE IOITER
IOSw
CLM ITERP
BEF MOV1 IOSw "0"
POP OUT LDA
EVAL "5"
"0"
OUT LD
EVAL "5"
"0"
RESTOR LB1
ENTLUC
ENTL "0"
"6"
TEST ")"
BEF $568
OUTPUT TEST "PUT"
BF $570
TEST "STRING"
BF $572
MARK OUT "0"
SAVE TEST "(
BEF CLM SLFTPT
BEF TEST ")"
BEF B $571
SET $572
BF $571
MARK OUT "1"
SAVL
BEF TEST "EDIT"
BEF CANCEL
PUSHLB
OUT FMT
LB1 EVAL "4"
"1"
RESTOR
<table>
<thead>
<tr>
<th>Command</th>
<th>Argument</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEST</td>
<td>&quot;(&quot;</td>
<td>BF $580</td>
</tr>
<tr>
<td>BEF</td>
<td></td>
<td>b $579</td>
</tr>
<tr>
<td>CLM</td>
<td>OLIST</td>
<td>CLM $580</td>
</tr>
<tr>
<td>BEF</td>
<td></td>
<td>EXP</td>
</tr>
<tr>
<td>TEST</td>
<td>&quot;)&quot;</td>
<td>BF $583</td>
</tr>
<tr>
<td>BEF</td>
<td></td>
<td>b $582</td>
</tr>
<tr>
<td>OUT</td>
<td>PUT</td>
<td>BF $583</td>
</tr>
<tr>
<td>PUSHLB</td>
<td></td>
<td>b $582</td>
</tr>
<tr>
<td>OUT</td>
<td>JUMP</td>
<td>BF $584</td>
</tr>
<tr>
<td>LATCH</td>
<td>VARBL</td>
<td>B $582</td>
</tr>
<tr>
<td>LB1</td>
<td></td>
<td>BF $579</td>
</tr>
<tr>
<td>EVAL</td>
<td></td>
<td>OUT EDIT</td>
</tr>
<tr>
<td>&quot;4&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;1&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LB2</td>
<td>ORPLIS</td>
<td>TEST &quot;(&quot;</td>
</tr>
<tr>
<td>ENLLOC</td>
<td></td>
<td>BF $587</td>
</tr>
<tr>
<td>ENTL</td>
<td></td>
<td>MARK</td>
</tr>
<tr>
<td>&quot;0&quot;</td>
<td>CLM</td>
<td>CLM $587</td>
</tr>
<tr>
<td>&quot;6&quot;</td>
<td>OLIST</td>
<td>OLIST</td>
</tr>
<tr>
<td>CLM</td>
<td>FMTLIS</td>
<td>BEF</td>
</tr>
<tr>
<td>BEF</td>
<td></td>
<td>SAVE</td>
</tr>
<tr>
<td>LB1</td>
<td>ENLLOC</td>
<td>BEF</td>
</tr>
<tr>
<td>ENTL</td>
<td></td>
<td>ERPLIS</td>
</tr>
<tr>
<td>&quot;0&quot;</td>
<td>FMTLIS</td>
<td>TEST &quot;(&quot;</td>
</tr>
<tr>
<td>&quot;6&quot;</td>
<td></td>
<td>BF $589</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OUT &quot;c&quot;</td>
</tr>
<tr>
<td>OLIST</td>
<td>CLM</td>
<td>FMTITE</td>
</tr>
<tr>
<td>BEF</td>
<td>$575</td>
<td></td>
</tr>
<tr>
<td>$570</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td>&quot;=&quot;</td>
<td>BF $578</td>
</tr>
<tr>
<td>BEF</td>
<td></td>
<td>CLM</td>
</tr>
<tr>
<td>OELMNT</td>
<td></td>
<td>FMTITE</td>
</tr>
<tr>
<td>$576</td>
<td>TEST</td>
<td>BF $592</td>
</tr>
<tr>
<td>BT</td>
<td></td>
<td>CLM</td>
</tr>
<tr>
<td>SET</td>
<td></td>
<td>FMTITE</td>
</tr>
<tr>
<td>BEF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$578</td>
<td>BT</td>
<td>$592</td>
</tr>
<tr>
<td>$575</td>
<td>R</td>
<td>SET</td>
</tr>
<tr>
<td>OELMNT</td>
<td>CLM</td>
<td>TEST &quot;=&quot;</td>
</tr>
<tr>
<td>$575</td>
<td>OELMNT</td>
<td>BEF</td>
</tr>
<tr>
<td>R</td>
<td>CLM</td>
<td>OUT &quot;c&quot;</td>
</tr>
<tr>
<td>ORPLIS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$589</td>
<td>R</td>
<td>FMTITE</td>
</tr>
<tr>
<td>-------</td>
<td>-----</td>
<td>--------</td>
</tr>
<tr>
<td>$594</td>
<td>CLM</td>
<td>DATAFM</td>
</tr>
<tr>
<td>$593</td>
<td>R</td>
<td>TEST</td>
</tr>
<tr>
<td>$597</td>
<td>TEST</td>
<td>&quot;PAGE&quot;</td>
</tr>
<tr>
<td>$598</td>
<td>TEST</td>
<td>&quot;SKIP&quot;</td>
</tr>
<tr>
<td>$601</td>
<td>SET</td>
<td>BF</td>
</tr>
<tr>
<td>$600</td>
<td>BEF</td>
<td>BF</td>
</tr>
<tr>
<td>$599</td>
<td>TEST</td>
<td>&quot;COL&quot;</td>
</tr>
<tr>
<td>$596</td>
<td>R</td>
<td>DATAFM</td>
</tr>
</tbody>
</table>

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.

**Assemble in Four Char Addressing Mode**

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

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

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

INDEX REGISTER LOCATION DEFINITIONS AND USAGE DESCRIPTIONS

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

BEGIN PROGRAM INITIALIZATION
0075C*
0076C*****************************************************************************
0077C START EQU *
0078C CAM 60 SET FOUR CHAR ADDRESSING FOR EXECUTE
0079C SW STKEND-2 WORD MARK FOR MOVING AND TESTING
0080C MCW STKSTR,IR2 INITIALIZE STACK POINTER
0081C SW IR2-2 SHORTEN ARITHMETIC
0082C SI IR2 ITEM MARK FOR RIGHT MOVE
0083C MCW LODLOC,IR1 INITIALIZE INSTRUCTION COUNTER
0084C SW IR1-2 SHORTEN INDEX ARITHMETIC
0085C SI IR1 ACCOMMODATE RIGHT MOVE
0086C BS IR3 ZERO PROGRAM COUNTER
0087C SW IR3-2 SHORTEN ARITHMETIC
0088C SI IR3 ACCOMMODATE RIGHT MOVE
0089C SI IR15 ACCOMMODATE RIGHT MOVE
0090C MCW GENLOC,IR5 INITIALIZE CODE GENERATION LOCATION
0091C SW IR7-1 SHORTEN FETCH ARITHMETIC
0092C MCW SYMSTR,NEWSYM INITIALIZE NEXT LOCATION IN SYMBOL TABLE
0093C BS =1BC4,SYMEND INITIALIZE
0094C MCW SYMEND,IR14
0095C LCA =1C77*1+X14 BLOCK
0096C LCA NEWSYM*4*X14 LIST
0097C MCW SYMEND,CRBLKT INIT CURRENT BLOCK LIST POINTER
0098C MCW NEWSYM,IR15 INIT SYMTAB POINTER
0099C MCW SYMEND,SVSYME SAVE INITIAL SYMTAB END AS START OF BLOCK
01000* LIST
01010 MCW ::OUTPUT+132 CLEAR PRINT
01020 MCW OUTPUT+132 BUFFER
01030 MCW =1C21 CARRIAGE CONTROL
01040 SI INPUT*80 RESTORE LOST ITEM MARK ON INPUT BUFFER
01050* SKIP TO TOP OF PAGE AND INIT INPUT BUFFER
01060L :SKIP PRINT,57,
01070L :GET READ,
01080L :PUT PRINT,INPUT-1,
01090 MCW +INPUT+IR6
01100 B FIRST
01110*****************************************************************************
0112C*  INSTRUCTION FETCHING  *
0113C*  *  *
0114C*  *  *
01150********************************************************************************************
01160  FETCH  BA =1B3,IR1
01170  FETCH BA =1B1,INSTCT  INSTRUCTION COUNT
01180  FIRST BS IR7-1  CLEAR SECOND CHAR
01190  MRSD  0+X1,IR7  INSERT OP CODE
01200  SAR IR1  BUMP SEQUENCE COUNTER
01210  BA IR7  MULTIPLY
01220  BA IR7  BY 4
01230  MCw TVEC+3+X7,IR14
01240  B 0+X14
01250  TVEC EQU *
01260  REP 10
01270  DSA ERROR
01280  DSA MOVE
01290  DSA ADD
01300  DSA MULT
01310  DSA SETF
01320  DSA BEF
01330  DSA EXITI
01340  DSA RESOLV
01350  DSA BRANCH
01360  DSA BRNCHT
01370  DSA BRNCHF
01380  DSA BM
01390  DSA SET
01400  DSA SCANI
01410  DSA ERROR
01420  DSA MOVLI
01430  DSA ERASE
01440  DSA COMP
01450  DSA SRCHP
01460  DSA TSTTLB
01470  DSA TSTTBA
01480  DSA LATCH
**BRANCH, BRANCH TRUE, AND BRANCH FALSE PRIMITIVES**

```
01490   DSA  CANCEL
01500   DSA  MARK
01510   DSA  SAVE
01520   DSA  CLM
01530   DSA  RETURN
01540   DSA  ERROR
01550   DSA  RESTOR
01560   DSA  PUSHLB
01570   DSA  POP
01580   DSA  LB1
01590   DSA  LB2
01600   DSA  SIKSYM
01610   DSA  CHKSYM
01620   DSA  Swap
01630   DSA  ENTA
01640   DSA  ENTL
01650   REP  3
01660   DSA  ERROR
01670   DSA  OUT
01680   DSA  OUTSYM
01690   DSA  TEST
01700   DSA  ID
01710   DSA  ONUM
01720   DSA  STRTST
01730   DSA  EVAL
01740   DSA  ENTLOC
01750   DSA  ERROR
01760   DSA  INUM
01770   DSA  ENUM
01780   DSA  BLKENT
01790   DSA  BLKEXT
01800   DSA  ERROR
```

181D***************************************************************************
1823* *
1830* BRANCH, BRANCH TRUE, AND BRANCH FALSE PRIMITIVES *
1840* *
1850*****************************************************************************
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 MRSD 5+X2,ELATCH RESTORE ERROR LATCH UPON RETURN
02290 B CANCEL RETURN CANCELS ANY BACKUP LATCH
02300***************************************************************************
02310* PUSHLB GENERATES A NEW INTERNAL LABEL AND PUSHES IT ON THE
02320* CONTROL STACK.
02330* PUSHLB A :1:LABEL
02340* MRIDI LABEL-4+X2
02350* BA =1B9,IR2
02360* B FETCH
02370 DCW :6:
02380 RLABEL DCW :000: ITEM MARK RIGHT
02390***************************************************************************
02400 POP POPS THE CONTROL STACK RESTORING THE VALUE TO SYMBOL IF THE
02410* TOP OF THE STACK IS MARKED AS A SYMBOL.
02420* POP BS =1B9,IR2
02430* BCE POPSYM+0+X2,S BRANCH IF STACK TOP IS SYMBOL
02440* B FETCH
02450* POPSYM MCW 8+X2,SYMBOL+7 STACK SYMBOL TO SYMBOL AREA
02460* B FETCH
02470***************************************************************************
02480 LB1 SEARCHES THE CONTROL STACK TO FIND THE FIRST LABEL SYMBOL
02490* WHICH IS THEN MOVED TO SYMBOL. THE STACK IS NOT AFFECTED.
02500* LB1 LCA IR2,IR14
02510***************************************************************************
ONUM TESTS THE INPUT STRING FOR A VALID OCTAL NUMBER SETTING THE
TRUE-FALSE INDICATOR AND CONVERTING THE NUMBER TO BINARY IN SYMBOL
IF TRUE.

ONUM NEXT
HS SYMBOL+32 CLEAR TO ZEROES
MCW :F:,TF
LCA =3B0,IR13
HS IR14
ONMTST MRSD 0+X6,IR14
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.
**SEARCH MODE**

04450 ENTLOC Mw :0;SRCHTP

04460 B SEARCH

04470 MRID IR3-2,ADDP-2+X15 MOVE IN ADDRESS

04480 Sw ADDR=2+X15 MARK RELOCATABLE

04490 MCw :0;LEVEL+X15 STATIC STORAGE INDICATOR

04500 B FETCH

04510***************************************************************************

04520* ENTL AND ENTA ARE PRIMITIVES FOR INTERING LITERAL AND ADDRESSED

04530* VALUES, RESPECTIVELY, INTO THE SYMBOL TABLE. INDEX REGISTER 15

04540* MUST POINT TO THE PROPER SYMBOL TABLE ENTRY PRIOR TO EXECUTION.

04550* A SIX BIT LITERAL NUMBER FOLLOWS EACH OP CODE SPECIFYING THE

04560* RELATIVE POSITION WITHIN THE TABLE ENTRY TO BE ALTERED.

04570***************************************************************************

04600 ENTA EQU *

04610 ENTL EQU *

04620 BS IR14 COMPUTE LEFTMOST

04630 MRSL 0+X1,IR14 ADDRESS

04640 SAR IR1 OF

04650 BA IR15,IR14 RECEIVING FIELD

04660 CW 0+X14 CLEAR POSSIBLE WORD MARK

04670 BCE ENLIT,0-2+X15 TEST FOR LITERAL ENTRY

04680 MRID 0+X1,IR13-2 ADDRESS TO IR13

04690 SAR IR1 UPDATE INSTRUCTION COUNTER

04700 CW 0+X13 AVOID INADVERTENT RELOCATION MARKER

04710 MRIDw 0+X13,0+X14 ENTER, CLEAR ANY WORD MARKS

04720 B FETCH

04730 ENLIT MRIDw 0+X1,0+X14 ENTER LITERAL

04740 SAR IR1 UPDATE INSTRUCTION COUNTER

04750 B FETCH

04760***************************************************************************

04770 SEARCH IS A SUBROUTINE FOR SEARCHING A BLOCK STRUCTURED SYMBOL

04780 TABLE. NOTE THE SEARCH TYPE PARAMETER (SRCHTP) WHICH MAY

04790 BE USED TO CONTROL THE SEARCH MODE. SEARCH MAY BE CALLED

04800 BY PRIMITIVES EVAL, ENTLOC, SRCHP, OR BLKEXT.
* 05520 155AL EQU
  * 05540 155AL EQU

********************************************
  *
  * INDEX REGISTER IS MUST POINT TO THE PROPER
  * TABLE ENTRY
  * RELATIVE POSITION WITHIN THE TABLE ELEMENT TO BE TESTED.
  * A SIX BIT LITERAL NUMBER FILLS EACH OP CODE SPECIFYING THE
  * LITERAL VALUES. REPEAT THIS VITAL, AGAINST SYMBOL TABLE VALUES.
  * S1 AND S11 ARE PRIMITIVES FOR TESTING ADDRESS AND
  *
  ******************

B FETCH
B SEARCH
I R
S1 IS CHanged
S1C 0+X+1, SRC1CHG

05490 SC1CHP MR3D 0+X1, SRC1CHG

******************************************************************************
*
*
THE OP CODE WHICH CONTROLS THE SEARCH MODE.
SUBROUTINE SEARCH. NOTE THE ONE CHARACTER PARAMETER FOLLOWING.
SC1CHP IS A SYMBOL TABLE SEARCH PRIMITIVE WHICH CALLS U1PON
*
*
******************************************************************************

05320 E3
95320 65 1R13,1R14
5310 5C 5131, IR14
5300 NCM 5V5W, IR14
5290 setup of block list
5280 multiply by block list entry size
5270 insert surrounding block number
5260 CLEAR
5250 65 1R13
5240 63 1R13
5230 05240 WEPB X3E+1+X4+77 CHECK FOR LAST BLOCK
5220 05250 C (SC1CHS+1)
5210 05260 F:TF
5200 NCM: 0
5290 05230 5M AL = 1B32-2+X15
5220 55220 5M SW 5220 05220 5M AL = 2+X15
5210 05210 5M SW 5200 05200 5M AL = 2+X15
5190 05190 5M SW 5200 05200 5M SW 5210 05210 5M SW 5220 05220 5M SW 5230 05230 5M
<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>05560</td>
<td>BS</td>
<td>IR14</td>
</tr>
<tr>
<td>05570</td>
<td>MRSD</td>
<td>0×X1, IR14</td>
</tr>
<tr>
<td>05580</td>
<td>SAR</td>
<td>IR1</td>
</tr>
<tr>
<td>05590</td>
<td>BA</td>
<td>IR15, IR14</td>
</tr>
<tr>
<td>05600</td>
<td>BCE</td>
<td>TSTLT, 0-2×X1×13</td>
</tr>
<tr>
<td>05610</td>
<td>MRID</td>
<td>0×X1, IR13-2</td>
</tr>
<tr>
<td>05620</td>
<td>SAR</td>
<td>IR1</td>
</tr>
<tr>
<td>05630</td>
<td>Sw</td>
<td>0×X13</td>
</tr>
<tr>
<td>05640</td>
<td>MRIN</td>
<td>0×X13, 0×X14</td>
</tr>
<tr>
<td>05650</td>
<td>SAR</td>
<td>IR13</td>
</tr>
<tr>
<td>05660</td>
<td>SBR</td>
<td>IR14</td>
</tr>
<tr>
<td>05670</td>
<td>C</td>
<td>0-1×X14, 0-1×X1</td>
</tr>
<tr>
<td>05680</td>
<td>BE</td>
<td>SET</td>
</tr>
<tr>
<td>05690</td>
<td>B</td>
<td>SETF</td>
</tr>
<tr>
<td>05700</td>
<td>TSTLT</td>
<td>Sw 0×X1</td>
</tr>
<tr>
<td>05710</td>
<td>MRIN</td>
<td>0×X1, 0×X14</td>
</tr>
<tr>
<td>05720</td>
<td>SAR</td>
<td>IR1</td>
</tr>
<tr>
<td>05730</td>
<td>SBR</td>
<td>IR14</td>
</tr>
<tr>
<td>05740</td>
<td>C</td>
<td>0-1×X14, 0-1×X1</td>
</tr>
<tr>
<td>05750</td>
<td>BE</td>
<td>SET</td>
</tr>
<tr>
<td>05760</td>
<td>B</td>
<td>SETF</td>
</tr>
</tbody>
</table>

**THE Compare Primitive** compares the 2nd addressed operand to the first. The six bit character following the operand addresses is used as the variant of the conditional branch instruction following the comparison.

**symbols**

- BS: Branch to Subroutine
- MRSD: Move Right Shift Data
- SAR: Shift Right
- BA: Branch to Address
- BCE: Branch on Compare Equal
- MRID: Move Right Immediate Data
- TSTLT: Test for Literal Test
- SBR: Set Bit Register
- MRIN: Move Right Immediate
- BE: Branch to End

**Code Explanation**

1. **Compute Leftmost Address**
2. **Table Field**
3. **To Be Tested**
4. **Test for Literal Test**
5. **Update Instruction Counter**
6. **Word Mark to Stop Compare**
7. **Position to Right End**
8. **Set**
9. **Index Registers**
10. **Word Mark to Stop Compare**
11. **Position to Right End**
12. **Set**
13. **Index Registers**
14. **Word Mark to Stop Compare**
**SCAN FOR ITEM MARK**

06150

**FOLLOWING THE OP CODE**

06260

**SCAN I/T THE INPUT STRING FOR THE SPECIFIED LITERAL STRING**

06370

**NEXT INSTRUCTION**

06420

**TURN OFF BACKUP SWITCH**

06510

**CANCEL MCW**

06620

**F:BACKUP**

06710

**CANCEL TURNS OFF ANY BACK UP LATCH**

06810

**BACKUP INITIALLY TURNED OFF**

06910

**CALL THE SPECIFIED ROUTINE**

07010

**SET BACKUP INDICATOR**

07110

**SAVE OUTPUT POINTERS**

07210

**SAVE INPUT POINTERS**

07310

**AVOID CARD BOUNDARY PROBLEMS**

07410

**RETURN TO THE CALLING PROCEDURE**

07510

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

07610

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

07710

**INDICATOR**

07810

**SET**

07910

**SET TF**

08010

**SET BF**

08110

**SET CF**

08210

**SET OF**

08310

**POSITION A AND B ADDRESS REGISTERS**

08410
06300  SAR  IR1
06310  SBR  IR6
06320  SCNCMP  C  0-1X6,0-1X1
06330  BE  SCANT
06340  BI  NEXT,0+X6
06350  BA  =1B1,IR6
06360  B  SCNCMP
06370  SCANT  MCW :T:,TF
06380  B  FETCH
06390 ****************************
06400  MOVLIT MOVES THE LITERAL CHARACTER STRING FOLLOWING THE ADDRESS
06410  (WHICH FOLLOWS THE OP CODE) TO THE ADDRESSED LOCATION.
06420 ****************************
06430  MOVLIT MRIU 0+X1,IR13-2 ADDRESS OF RECEIVING CHAR FIELD
06440  SAR  IR14
06450  MRIU 0+X14,0+X13 MOVE LITERAL DATA
06460  SAR  IR1
06470  SW  0+X14 UPDATE LOCATION COUNTER
06480  B  FETCH
06490 ****************************
06500  INUM CALLS INM FOR AN ATTEMPTED RECOGNITION OF AN INTEGER NUMBER.
06510 ****************************
06520  INUM  B  INM
06530  B  FETCH
06540 ****************************
06550  SUBROUTINE INM TESTS THE INPUT STRING FOR AN INTEGER NUMBER SETTING
06560  TRUE-FALSE CODE AND CONVERTING THE NUMBER TO BINARY IN SYMBOL
06570  IF TRUE.
06580 ****************************
06590  INM  SBR  INMR TN+4
06600  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  TEST FOR POSSIBLE FLTNG PT NM
0679C  BCE  NOINT,0+X6,  TEST FOR INTEGER DIGIT
0680C  BIO  MVNUM,IDTAB+X14  HAVE WE FOUND AN INTEGER
0681C  BCE  NOINT,TF,F  YES, GO CONVERT TO BINARY
0682C  B  CONVRT
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  DA  =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,CVBFLD  MOVE IN DECIMAL INTEGER
0694C  SST  ISGN,CVBFLD,60  SET SIGN IN CONVERSION FIELD
0695C  DTB  CVBFLD,00  BINARY MANTISSA IN FR 0
0696C  TAM  CVBFLD,00  STORE IT
0697C  C  CVBFLD-6,=2C7777  TEST NUMBER TOO LARGE FOR 24 BITS
0698C  BE  INMOK
0699C  C  CVBFLD-6,=2C0000
0700C  BL  INMERR
0701C  INMOK  EQU  *
0702C  MCW  CVBFLD-2*S SYMBOL+3  SAVE 24 BITS
0703C  SI  SYMBOL+3
ERASE ERASES THE SPECIFIED NUMBER OF CHARACTERS FROM THE CODE STRING.

ERASE BS
MRSB SYMBOL, IR13
BS IR13, IR3
MCW IR5, IR14
ERTST BCE FETCH, IR13, 00
MRSBL IR13-1, 0 x14
SBR IR14
BS =1B1, IR13
MCW IR5, IR13
BS IR5, IR3
MCW SAVOUT, IR5
ERTST
B

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

BEF EQU *
BCE FETCH, TF, T
BCE DEFMES, BCKUP, F
MCW SAVIN, IR6
BS SAVOUT, IR5
MCW IR5, IR13
BS IR5, IR3
MCW SAVOUT, IR5
07410 MCW IR5,IR14 SET UP RETURN FROM CODE ERASURE
07420 MCW *ERSRTN,ERTST+4 ERASE CODE
07430 B ERTST RESTORE INSTRUCTION IN ERASE ROUTINE
07440 ERST RTN MCW *FETCH,ERTST+4 BACKUP CANCELS AND RETURNS
07450 B RETURN TEST PREVIOUS PENDING MESSAGE
07460 BCE ERSRTN+OUTPUT+20,F TEST PREVIOUS PENDING MESSAGE
07470 B ERMPRP
07480 MCW =9AF: SYNTAX,OUTPUT+28 DEFAULT MESSAGE
07490 B ERRHD FINISH TESTING AND MESSAGE
07500 ERMES B ERMPRP NO BACKUP SO CONTINUE WITH ERROR MESSAGE
07510 MRID 0+X1,OUTPUT+20
07520 SAR IR1
07530 ERRPRT 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 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 MRIN 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 ERMPRP 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,ELATCH,F IF NO LATCH RETURN
07740 B ERRPRT ELSE PRINT AND CONTINUE
07750 ERMPRP SBR PPRRTN+4
07760 MCW : OUTPUT+132 CLEAR PRINT LINE
07770 MCW OUTPUT+132 CHAINED MOVE
07780       SI  INPUT+80   RESTORE LOST ITEM MARK ON INPUT BUFFER
07790       MCl  IR6+IR14  COMPUTE ERROR
07800       BS  +INPUT,IR14 LOCATION
07810       MCW  *;OUTPUT+1*X14  MARK IT
07820       MCW  :;OUTPUT  CARRIAGE CONTROL
07830       :PUT  PRINT,OUTPUT.  CARRIAGE CONTROL
07840       MCW  :;OUTPUT+1*X14  CLEAR ERROR MARK
07850       PRPRTN  B  *
07860       EXIT  EQU  ERRFLG
07870       OVFLw  EQU  *
07880       :PUT  PRINT,OVFMES,
07890       ERRFLG  MCw  :F:,CMPLCD  FATAL ERROR FLAG
07900       B  Exit
07910       OVFMESDCw  1SYMBOL TABLE OVERFLOW, JOB ABORTED:
07920       L  DCw  =1C45  RECORD MARK
07930 ****************************
07940  *
07950  * ERROR IS EXECUTED IF AN ATTEMPT IS MADE TO INTERPRET AN INVALID *
07960  * OP CODE. THE JOB IS ABORTED.
07970  *
07980 ****************************
07990       ERROR  EQU  *
08000       :PUT  PRINT,OPCDMS,
08010       H
08020       B  Exit
08030       OPCODEDCw  1INVALID OP CODE, JO6 ABORTED:
08040       L  DCw  =1C45  RECORD MARK
08050 ****************************
08060  *
08070  * THE ENUM PRIMITIVE EXAMINES THE INPUT STRING FOR A FLOATING POINT *
08080  * NUMBER SETTING THE TRUE-FALSE INDICATOR AND CONVERTING THE NUMBER *
08090  * TO BINARY IN SYMBOL IF TRUE.
08100  *
08110 ****************************
08120       ENUM  MCw  *DHOLD+IR12  SET POINTER TO DECIMAL HOLD FIELD
08130       BS  SCALE  CLEAR SCALE EXPONENT FIELD
08140       B  NEXT
CALL DECIMAL NUMBER RECOGNIZER
IF DNUM.TURE THEN GO SAVE INPUT
AVOID POSSIBLE LOGICAL CONSTANT
DETERMINE FRACTION FOLLOWING
IF NONE THEN FETCH NEXT INSTRUCTION
CLEAR HOLD AREA
MOVE DECIMAL CHARACTERS
SAVE POINTER FOR MOVING FRACTION
DETERMINE FRACTION FOLLOWING
IF NONE MOVE DECIMAL FIELD FOR CONVERSION
LOOK FOR DECIMAL FRACTION
IF NONE MOVE DECIMAL FIELD FOR CONVERSION
CONCATENATE FRACTION WITH INTEGER PART
SCALE EXPONENT ADJUSTMENT
ADJUST IT
RESTORE X12
CLEAR SYMBOL AREA
TEST IF FIELD TOO LONG
WARNING MESSAGE
PRINT,OUTPUT
SET UP DECIMAL FIELD
FOR DECIMAL TO BINARY CONVERSION
LOOK FOR EXPONENT
TEST FOR EXPONENT
NO EXPONENT, GO CONVERT
TEST MINUS SIGN
TEST PLUS SIGN
SET ADDITION OP CODE
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
BINARY 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 CALLER
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 NEGATIVE EXPONENT FOR SCALF FACTOR
SET EXPONENT SIGN FLAG
TEST SCALE EXPONENT FOR VALID RANGE
SET UP

EXCEPTION OUT OF RANGE, OUTPUT + 25, ERROR MESSAGE

FATAL COMPLETION CODE

PRINT IT

IF EXPONENT ZERO CONVERSION FINISHED

CLEAR INDEX TO ZERO

INSERT LOW 4 BITS FOR INDEXING

LEFT 3 BITS TO INDEX CVTTAB

CONVERSION FACTOR TO FR1

CLEAR EIGHT CHAR FLOATING POINT FIELD

3 CHAR FIELD TO 8 CHAR FLOATING POINT FIELD

LOAD IT TO FR3

SHIFT RIGHT 4 BITS

STORE IT

CLEAR

INSERT LOW 4 BITS TO INDEX

SHIFT LEFT 3 BITS TO INDEX CVTTAB

CONVERSION FACTOR TO FR2

COMPUTE INTERMEDIATE FACTOR

SAVE LOW ORDER FOR DOUBLE PRECISION

NEXT 4 BITS

STORE IT

CLEAR

MAX 10 BITS FOR SCALE EXPONENT

SHIFT FOR INDEX

CONVERSION FACTOR TO FR3

LOW ORDER FACTOR

HIGH ORDER FACTOR

SAVE LOW ORDER

ACCUMULATE LOW ORDER FACTORS

TEXT EXPONENT SIGN

MULTIPLY BY HIGH ORDER SCALE FACTOR

SAVE LOW ORDER

LOW ORDER SCALE FACTOR

ACCUMULATE LOW ORDER FACTORS

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
09640 CTAB16 DCw F1E0
09700 DCw F1E96
09710 DCw F1E112
09720 DCw F1E128
09730 DCw F1E144
09740 DCw F1E160
09750 DCw F1E176
09760 DCw F1E192
09770 DCw F1E208
09780 DCw F1E224
09790 DCw F1E240
09800 CTB256 DCw F1E0
09810 DCw F1E256
09820 DCw F1E512
09830 **************************************************************************
09840 * THE MARK PRIMITIVE PUSHES THE ADDRESS OF THE NEXT OUTPUT STRING *
09850 * LOCATION ON THE CONTROL STACK FOR LATER USE BY THE SAVE PRIMITIVE. *
09860 * *
09870 **************************************************************************
09880 MARK BA =1B9,IR2 BUMP STACK POINTER
09890 MCw IR5,0-5+X2 SAVE OUTPUT CODE ADDRESS
09900 MCw :M:,0-9+X2 MARK STACK ELEMENT TYPE
09910 B STKOVF CHECK STACK OVERFLOW
09920 **************************************************************************
09930 * SAVE PUSHES THE CODE GENERATED SINCE THE LAST MARK OPERATION *
09940 * ON THE VARIABLE LENGTH CODE STACK AND RESETS THE OUTPUT LOCATION *
09950 * BACK TO THE MARKED LOCATION. *
09960 *
1000C  SAVE  BCE  DOSAVE,0-9+X2,M  TREAT AS NO-OP IF STACK TOP
1001C  B  FETCH  IS NOT A CODE MARK
1002C  DOSAVE  MCw  0-5+X2,IR14  MARKED ADDRESS TO IR14
1003C  MCw  STKEND,IR13  STACK END VALUE TO IR13
1004C  BS  =1B1,IR5  ADJUST CODE POINTER TO LAST OUTPUT CHAR
1005C  SVTEST  C  IR14,IR5  TEST MOVE  NOTE POSSIBLE NULL
1006C  BL  SVEXIT  COMPLETION  SAVE OPERATION
1007C  MLSUR  0+X5,0+X13  MOVE AND
1008C  SAR  IR5  ADJUST
1009C  SBR  IR13  POINTERS
1010C  CW  1+X5  CLEAR ANY POSSIBLE WORD MARK
1011C  BS  =1B1,IR3
1012C  B  SVTEST
1013C  SVEXIT  LCA  STKEND,0+X13  NEXT ELEMENT POINTER
1014C  LCA  =1C77,0-3+X13  MARKER FOR NO-OP TEST
1015C  SBR  STKEND  NEW SYSTEM STACK END
1016C  MCw  IR14,IR5  RESTORE OUTPUT MARKER
1017C  BS  =1B9,IR2  POP CODE MARKER OFF STACK
1018C  B  STKOVF  CHECK STACK OVERFLOW
1019C  STKOVF  C  IR2,STKEND  TEST STACK
1020C  BH  FETCH  OVERFLOW
1021C  :PUT  PRINT,STKMES,
1022C  H  HALT INTERRUPT, DUMP REQUEST
1023C  STKMESDCW  ;1 STACK OVERFLOW:
1024C  L  DCw  =1C45;  RECORD MARK
1025C  **************************************************************************
1026C  *
1027C  *  THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE  *
1028C  *  CONTROL STACK.                                             *
1029C  *
1030C  **************************************************************************
1031C  STKSYM  MCw  SYMBOL+7*8+X2  MOVE SYMBOL TO STACK
1032C  MCw  ;S:*0+X2  MARK STACK ELEMENT AS SYMBOL
1033C  BA  =1B9,IR2  BUMP STACK POINTER
1034C  B  STKOVF  CHECK FOR STACK OVERFLOW
1035C  **************************************************************************
1036C  *
THE RESTOR PRIMITIVE RESTORES THE TOP OF THE VARIABLE LENGTH CODE STACK TO THE OUTPUT STRING.

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

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

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

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

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

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

**BS** =1B9,IR2 ADJUST STACK POINTER
8+X2,SYMBOL+7 TEST AGAINST SYMBOL VALUE
BE SET IF EQUAL SET TRUE
B SETF ELSE SET FALSE
THE ADD AND MULT PRIMITIVES COMPRIS THE BINARY ARITHMETIC CAPABILITIES (TWO ADDRESS) OF THE META9 PSEUDO-MACHINE.

ADD

SW 0+X11
SW 0+X10
MRIN 0+X10,0
SAR IR10
MRIN 0+X11,0
SAR IR11
BA 0-1+X10,0-1+X11
B FETCH

MULT

SW 0+X10
MRIN 0+X10,0
SAR IR10
BS MFLD1
MCW 0-1+X10,MFLD1
Sw 0+X11
MRIN 0+X11,0
SAR IR11
BS MFLD2
MCW 0-1+X11,MFLD2
BIM MFLD1,MFLD2
MCW MFLD2,0-1+X11
B FETCH

MFLD1 DCW =4B0
MFLD2 DCW =4B0
GTOPRA SBR GTRTN
MRID 0+X1,IR10-2
SAR IR1
MRID 0+X1,IR11-2
SAR IR1
THE MOVE PRIMITIVE MOVES THE FIRST ADDRESSED FIELD TO THE SECOND.

MOVE B GTOPRA
MRIDI 0+X10,0+X11
B FETCH

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

BLKCNT DCW :0:
PRVBLK DCw =1C00
CRBLKT DCW =3
BLKENT BA =1B1,BlKCNT
BS =1B4,SYMEND
MCw SYMEND,IR13
LCA PRVBLK,1+X13
MCw BLKCNT,PRVBLK
BA =1B20,NEWSYM
LCA NEWSYM,4+X13
MCw SYMEND,CRBLKT
B FETCH

THE BLOCK EXIT PRIMITIVE RESTORES CRBLKT POINTER AND PRVBLK NUMBER FOR THE SURROUNDING BLOCK. THE SYMBOL TABLE ENTRIES FOR THE TERMINATING BLOCK ARE SCANNED FOR UNRESOLVED SYMBOLS. UNRESOLVED ENTRIES ARE ADDED TO THE SURROUNDING BLOCK IF NOT FOUND IN THAT PORTION OF THE TABLE. APPROPRIATE LINKING PARAMETERS ARE SET FOR THE RESOLVE PRIMITIVE TO USE. DIABOLICAL LABEL REFERENCES IN A BLOCK STRUCTURE ARE RESOLVABLE.
BLKSAV DCw = 3
BLKPRM DCw = 1
BLKEXT MCw CRBLKT, BLKSAV
         SAVE FOR UNRESOLVED SEARCH
MRSd  0+X1, BLKPRM
SAR   IR1
BS    IR13
MCw   CRBLKT, IR14
MCW   I+X14, IR13
MCW   I+X14, PRVBLK
MCW   4+X12, IR12
BIM   = 4B4, IR13
MCW   SVSYME, IR14
MCW   IR13, IR14
MCW   IR14, CRBLKT
MCW   BLKSAV, IR12
MCW   4+X12, IR12
BCE   FETCH, BLKPRM, 01
       TEST NO LABEL LINK UP
MCW   CHAIN+X12, :000:
       CHECK FOR END OF
Be    FETCH
       BLOCK TABLE ENTRIES
MCW   CHAIN+X12, IR12
       NOTE 1ST TIME JUMP OVER DUMMY ENTRY
BCE   CHKPRV, DTYPEx+X12, 00
       CHECK FOR UNDEFINED SYMBOLS
B     CHKUND
       LOOK FOR MORE
MCW   NAME+X12, SYMBOL+7
       SET NAME TO USE SEARCH SUBROUTINE
MCW   :1::SRCHTP
       BLOCK ONLY SEARCH MODE
B     SEARCH
BCE   ADDSYM, TF, F
       IF FALSE SYMBOL ADDED TO SURROUNDING
BCE   PRVUN, 0+X15, 00
       IF FOUND BUT STILL UNDEFINED SET MARKERS
SI    0+X15
MLIDw  DIMCNT+X15, DIMCNT+X12
       FOUND, SET VALUES FOR RESOLVE
CI    0+X15
B     CHKUND
ADDsym EQU *
PRVUN Sw  IR15-2
MCW   IR12, I9
       SAVE CURRENT ENTRY POINTER
MCW   CHAIN+X12, IR12
       SAVE NEXT ENTRY POINTER
RESOLVE IS A TERMINAL PRIMITIVE WHICH RESOLVES FORWARD REFERENCES AND DETECTS ANY UNDEFINED ADDRESSES. THE OBJECT TEXT IS SCANNED FOR WORD MARKS TO FIND RELOCATABLE ADDRESSES. THE LEFTMOST BIT OF THE ADDRESS MARKS UNRESOLVED ADDRESSES.

RESOLVE BCE EXIT,CMPLCD,F EXIT IF FATAL COMPILATION TO THIS POINT
BS CVBF LD CLEAR
MCW IR3,CVBFLD-2 MOVE PROGRAM SIZE
TMA CVBF LD,00 LOAD TO FRO
BTD CVBF LD,00 CONVERT TO DECIMAL AND STORE
LCA EwORD,PSIZE EDIT CONTROL WORD
MCE CVBF LD,PSIZE MOVE AND EDIT
CW P SIZE-8 CLEAR WORD MARK
BS CVBF LD CLEAR
MCW INST CT,CVBFLD-2 MOVE INSTRUCTION COUNT
TMA CVBF LD,00 LOAD TO FRO
BT D CVBF LD,00 CONVERT TO DECIMAL AND STORE
LCA EwORD,ICOUNT EDIT CONTROL WORD
MCE CVBF LD,ICOUNT MOVE AND EDIT
CW ICOUNT-8 CLEAR WORD MARK
BP UT PRINT,EXITMS, PRINT IT
BS CVBF LD
MCW SRCH CT,CVBFLD-2 MOVE SEARCH COUNT
TMA CVBF LD,00
BT D CVBF LD,00
LCA EwORD,SC OUNT
MCE CVBF LD,SCOUNT
BS CVBF LD
UNRESOLVED ADDRESS IS POINTER TO SYMBOL TABLE. MOVE IT TO IR14.

REMOVE UNRESOLVED MARKER

TEST FOR VALID SYMBOL

ADDRESS

CHAIN TO SURROUNDING BLOCK

GO TEST FOR ADDITIONAL CHAINING

SET LEVEL AND DISPLACEMENT

TEST POSSIBLE LABEL VARIABLE - LABEL CONSTANT RESOLUTION

SCAN

SCAN
LBVTST BCE LBV, DTYPE, X14, 05  TEST LABEL VARIABLE IN SYMTAB
B SCAN  ELSE CONTINUE
LBV BNP SCAN, 0-4, X15  MAKE SURE OPERAND PRECEDES ADDRESS
MRSD :5:*0-3, X15  CHANGE DATA TYPE TO LABEL VARIABLE
B SCAN  AND CONTINUE
STERR EQU *
:PUT PRINT, CMPLMS,
H EXIT
CMPLMSDCW :ACOMPILER ERROR DISCOVERED DURING RESOLVE:
L DCW =1C45
NTDEFN MCW NAME, X14, PSYM
L :PUT PRINT, NDFMES,
MCW :F:, CMPLCD  SET COMPLETION CODE
SCAN
NDFMESDCW :UNDEFINED SYMBOL :
PSYM DCW :
L DCW =1C45
TABCNTDCW :A ****SYMBOL TABLE ENTRY COUNT :=
ECOUNT DC =9
DC :****:
L DCW =1C45
EXITMSDCW :2 ****COMPILED PROGRAM SIZE :=
PSIZE DC =9
DC ;: METAX INSTRUCTION COUNT :=
ICOUNT DC =9
DC ;: ****:
L DCW =1C45
SRCHCT DCW =4B0
CMPCNT DCW =4B0
TABMESDCW :B ****SYMTAB SEARCH COUNT :=
SCOUNT DC =9
DC ;; SYMTAB COMPARE COUNT :=
ICOUNT DC =9
DC ;: ****:
L DCW =1C45
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 NXXTRTN+4
ENDTST BI GETCRD*0+x6
BLKTST BCE NBLNK*0+x6,15
NXXTRTN B *
NBLNK BA =1B1,IR6
B 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,
B ERRFLG
EOFMES:DCW :UNEXPECTED 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

EXIT IF POST LIST NOT REQUESTED

PROGRAM COUNTER

START OF GENERATED CODE

CLEAR
PRINT LINE
CARRIAGE CONTROL
RESTORE LOST ITEM MARK ON INPUT BUFFER
CLEAR CONVERSION FIELD
LOAD TO FRO
CONVERT TO DECIMAL
REMOVE SIGN BITS
DETERMINE SYMBOLIC LABEL, IF ANY
JUMP OVER LITERAL DESIGNATOR
SET UP
LITERAL OPERAND
FOR PRINTING
KILL FIRST TIME BRANCH
13700  BA =1B1,IR13  BUMP PROGRAM COUNT
13710  BA =1B1,IR15  AND CODE POINTER
13720  BCE FMTCD<E+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 =4B6,IR12 MULT BY TABLE ENTRY SIZE
13780  MCW OPTAB+4+X12,OUTPUT+21 OP CODE TO PRINT
13790  BCE PLSTPR+OPTAB+5+X12,00 TEST FOR NO OPERANDS
13800  BCE TWOOP+OPTAB+5+X12,10 TEST TWO OPERANDS
13810  LITERL EQU LITRL
13820  BCE LITERL+OPTAB+5+X12,01 TEST SINGLE CHARACTER LITERAL
13830  BCE ADDR+4+OPTAB+5+X12,04 TEST FOUR CHAR ADDRESS
13840  BCE TWOOP+OPTAB+5+X12,10 TEST TWO OPERANDS
13850  B LITERL
13860  TWOOP B ADFV
13870  MCW IR14,IR11
13880  B ADFV
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 ADFV GET ADDRESS OF FIRST SYMBOL
13950  MCW IR14,IR11 SAVE IT
13960  B ADFV 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 MCW ::OUTPUT+132 PRINT
14010  MCW ::OUTPUT+132 CLEAR
14020  MCW OUTPUT+132
14030  MCW =1C21,OUTPUT CARRIAGE CONTROL
14040L :PUT MCW PRINT+DPVMES, DOPE VECTOR MESSAGE
14050  BS IR12 CLEAR
14060  MRSD 0+X15,IR12 INSERT DIM COUNT
1407C  BIM =4B10FIR12  MULT BY SIZE OF BOUND PAIR CODE
1408C  BA =IR3,IR12  SIZE OF LENGHT AND DIM COUNT FIELDS
1409C  BA IR12,IR13  BUMP PROG COUNTER
1410C  BA IR15,IR12  END OF DOPE VECTOR
1411C  SW 0-1X12  MARK FOR MOVE
1412C  MCw ";OUTPUT+15  MOVE IT
1413C  MRRD 0*X15,OUTPUT+16  MOVE IT
1414C  SAR IR15
1415C  SBR IR11
1416C  CW 0-1+X12
1417C  MCw ":;0*X11
1418C  B PLSTPR  GO PRINT
1419C  DPVMESDCw :A ***DOPE VECTOR CODE***:
1420C  L DCw =1C45
1421C  LDA MCw :LDA ;OUTPUT+21
1422C  SST 0*X15,LITCHR,70  LEFT THREE BITS
1423C  BCE LITOPR,LITCHR,70  TEST FOR LITERAL
1424C  SST 0*X15,O TYPE,07  SAVE TYPE
1425C  B ADDFV  GET OPERAND SYMBOL
1426C  BCE LNGCDE,O TYPE,04  TEST FOR STRING TYPE
1427C  B PLSTPR  GO PRINT
1428C  LNGCDE MCw ":;OUTPUT+32
1429C  MRSD 0*X15,OUTPUT+33  MOVE IN
1430C  EXM  LENGTH CODE
1431C  SAR IR15
1432C  MCw ":;OUTPUT+35
1433C  BA =1B2,IR13
1434C  B PLSTPR
1435C  O TYPE DCw :0:
1436C  ADDR4 B ADDFR
1437C  BCE IOTYPE,0-5*X15,50  CHECK I/O SETUP
1438C  B PLSTPR
1439C  IOTYPE MCw ":;OUTPUT+32  I/O TYPE FOR PRINTING
1440C  MRSD 0*X15,OUTPUT+33
1441C  SAR IR15
1442C  BA =1B1,IR13
1443C  MCw ":;OUTPUT+34
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>14440</td>
<td>B</td>
<td>PLSTPR</td>
<td>SET RETURN ADDRESS</td>
</tr>
<tr>
<td>14450</td>
<td>ADD5</td>
<td>B</td>
<td>MOVE ADDRESS</td>
</tr>
<tr>
<td>14460</td>
<td>B</td>
<td>PLSTPR</td>
<td>BUMP CODE COUNTER</td>
</tr>
<tr>
<td>14470</td>
<td>ADDFR</td>
<td>SBR</td>
<td>TABLE START</td>
</tr>
<tr>
<td>14480</td>
<td>MRID</td>
<td>0+X15,ADD4CN-3</td>
<td></td>
</tr>
<tr>
<td>14490</td>
<td>SAR</td>
<td>IR15</td>
<td></td>
</tr>
<tr>
<td>14500</td>
<td>BA</td>
<td>=1B4,IR13</td>
<td></td>
</tr>
<tr>
<td>14510</td>
<td>MCw</td>
<td>SYMSTR,IR14</td>
<td></td>
</tr>
<tr>
<td>14520</td>
<td>A4COMP</td>
<td>4+X14,ADD4CN</td>
<td></td>
</tr>
<tr>
<td>14530</td>
<td>BE</td>
<td>ADFRFD</td>
<td></td>
</tr>
<tr>
<td>14540</td>
<td>BA</td>
<td>=1B20,IR14</td>
<td></td>
</tr>
<tr>
<td>14550</td>
<td>C</td>
<td>IR14,NEWSYM</td>
<td></td>
</tr>
<tr>
<td>14560</td>
<td>BEH</td>
<td>A4COMP</td>
<td></td>
</tr>
<tr>
<td>14570</td>
<td>MISSAD</td>
<td>MCw :*****:,OUTPUT+35</td>
<td></td>
</tr>
<tr>
<td>14580</td>
<td>B</td>
<td>LITOPR</td>
<td></td>
</tr>
<tr>
<td>14590</td>
<td>ADFRFD</td>
<td>MCw NAME+X14,OUTPUT+30</td>
<td></td>
</tr>
<tr>
<td>14600</td>
<td>AFRRTN</td>
<td>B *</td>
<td></td>
</tr>
<tr>
<td>14610</td>
<td>ADD4CN</td>
<td>DCw =4</td>
<td></td>
</tr>
<tr>
<td>14620</td>
<td>ADDFV</td>
<td>SBR AFRRTN+4</td>
<td></td>
</tr>
<tr>
<td>14630</td>
<td>MRSD</td>
<td>0+X15,ADD5CN-4</td>
<td></td>
</tr>
<tr>
<td>14640</td>
<td>EXM</td>
<td></td>
<td>GET ALL FIVE CHARACTERS</td>
</tr>
<tr>
<td>14650</td>
<td>EXM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14660</td>
<td>EXM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14670</td>
<td>EXM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14680</td>
<td>SAR</td>
<td>IR15</td>
<td></td>
</tr>
<tr>
<td>14690</td>
<td>BA</td>
<td>=1B5,IR13</td>
<td></td>
</tr>
<tr>
<td>14700</td>
<td>MCw</td>
<td>SYMSTR,IR14</td>
<td></td>
</tr>
<tr>
<td>14710</td>
<td>A5COMP</td>
<td>SI 4+X14</td>
<td></td>
</tr>
<tr>
<td>14720</td>
<td>MRID</td>
<td>0+X14,AD5CN-4</td>
<td></td>
</tr>
<tr>
<td>14730</td>
<td>C</td>
<td>ADD5CN+AD5CN1-4</td>
<td></td>
</tr>
<tr>
<td>14740</td>
<td>BE</td>
<td>ADFVF D</td>
<td></td>
</tr>
<tr>
<td>14750</td>
<td>BA</td>
<td>=1B20,IR14</td>
<td></td>
</tr>
<tr>
<td>14760</td>
<td>C</td>
<td>IR14,NEWSYM</td>
<td></td>
</tr>
<tr>
<td>14770</td>
<td>BEH</td>
<td>A5COMP</td>
<td></td>
</tr>
<tr>
<td>14780</td>
<td>B5</td>
<td>=1B5,IR15</td>
<td></td>
</tr>
<tr>
<td>14790</td>
<td>B5</td>
<td>=1B5,IR13</td>
<td></td>
</tr>
<tr>
<td>14800</td>
<td>B</td>
<td>MISSAD</td>
<td></td>
</tr>
</tbody>
</table>
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 REL OCT BNP NOSYM,2+X14 IGNORE NON-RELOCATABLE SYMBOLS
14880 C 4+X14,IR13 TEST
14890 BE SYMFND EQUALITY
14900 NOSYM BA =IB20,IR14 NEXT
14910 C SYMEND(IR14) TEST
14920 BH OPCODE TABLE END
14930 B RELOCT TEST NEXT
14940 SYMFND MCW NAME+X14,OUTPUT+15
14950 R OPCODE
14960 PEXIT MCW :END PROGRAM: OUTPUT+32
14970L :PUT PRINT,
14980 H EXIT
14990 TLITRL SST 0+X15,LITCHR,70 LEFT THREE BITS OF TYPE CHAR
15000 BCE LITOPR,LITCHR,70 TEST FOR LITERAL
15010 B ADDR5 MUST BE FIVE CHAR ADDR
15020 LITCHR DCW :0:
15030 FMTCDE EQU *
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 =IB1,IR13
15110 BCE FMTDNE,0+X15,77
15120 B NFMTCH
15130 FMTDNE MCW :"",0+X12 FINISH
15140 BA =IB1,IR13 BUMP
15150 BA =IB1,IR15 COUNTERS
15160 B PLSTPR
15170 FMTMESDCW :A ***FORMAT CODE***:
OP CODE OPERAND TABLE FOR POSTLISTING

OP CODE 10
OP CODE 11
OP CODE 12

OP CODE 20
OP CODE 21
OP CODE 22
OP CODE 23

OP CODE 25
OP CODE 26
OP CODE 27
OP CODE 30
OP CODE 31
OP CODE 32
OP CODE 33
OP CODE 34
OP CODE 35
OP CODE 36

OP CODE 40
OP CODE 41
OP CODE 42
OP CODE 43
OP CODE 44

OP CODE 50
OP CODE 51
OP CODE 52
OP CODE 53
15550\hspace{1cm} DCw :ERROR0: OP CODE 60
15560\hspace{1cm} DCw :OR 0: OP CODE 61
15570\hspace{1cm} DCw :AND 0: OP CODE 62
15580\hspace{1cm} DCw :NOT 0: OP CODE 62
15590\hspace{1cm} REP 3
15600\hspace{1cm} DCw :ERROR0: OP CODE 66
15610\hspace{1cm} DCw :INDXR1: OP CODE 67
15620\hspace{1cm} DCw :INDXAI: OP CODE 70
15630\hspace{1cm} DCw :CAT 0: OP CODE 71
15640\hspace{1cm} DCw :ERROR0: OP CODE 76
15650\hspace{1cm} REP 4
15660\hspace{1cm} DCw :ERROR0: OP CODE 76
15670\hspace{1cm} DCw :STOP 0: \text{LAST}
15680\hspace{1cm} DCw :ERROR0: \text{LAST}
15690\hspace{1cm} LITORG* END START

SYMBOL DEFINITION - CARD REFERENCE INDEX

A4COMP 14520; A5COMP 14710; ADDCN1 14840; ADD4CN 14610; ADD5CN 14830;
ADD 10810; ADDFR 14470; ADDFV 14620; ADDR4 14360; ADD5 14450;
ADDR 04950; ADDRSL 12540; ADDSYM 11810; ADFRFD 14590; ADFVFD 14810;
AFRTN 14600; AFRVRN 14820; ALLOC 13930; BCKUP 06130; BEF 07330;
BLKCNT 11260; BLKENT 11290; BLKEXT 11520; BLKPRM 11510; BLKSAV 11500;
BLKTST 13180; BM 07610; BRANCH 01860; BRNCHF 01900; BRNCHT 01880;
CANCEL 06190; CHAIN 04920; CHKPRV 11710; CHKSYM 10600; CHKUND 11660;
CHNADD 12520; CHNTST 12490; CLM 02080; CMPCNT 12880; CMPLCD 00420;
CMPLMS 12680; COMP 05850; COMPT 05970; CONUM 03760; CONVRT 06920;
CRBLKT 11280; CTABI6 09640; CTB256 09800; CVBFLD 07050; CVTTAB 09480;
DECMVE 08340; DEFMES 07460; DHOLO 09450; DIMCNT 04930; DMVE 08200;
DNMRTN 08710; DNMTST 08670; DNUM 06630; DOENT 05130; DORES 10440;
DORRTN 00400; ENDSTST 13170; ENTA 04600; ENTL 04610; ENTLIT 04730;
ENTLOC 04450; ENUM 08120; EOFMES 13310; ERASE 07140; ERMES 07500;
TABMES 12890; TCOUNT 12920; TEST 03080; TESTLB 02600; TESTT 03170;
TF 00710; TLITRL 14990; TSTAN 03340; TSTLB1 02730; TSTLB2 02760;
TSTLIT 05700; TSTSL 08870; TSTTBA 05540; TSTTEL 05550; TVEC 01250;
Twoop 13860; UUCHK 11690; UPDATE 08540;

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

//
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

STRNGS: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 STRNG;
INPUT / OUTPUT (INCLUDING STRING I/O)

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

DO GROUPS

DOGRP:BEGIN; DECLARE (I,J,K,M(-2:10,10)) FIXED;
PUT EDIT ("BEGIN DO GROUPS") (SKIP(3),A);
DO I=0 TO 4;
DO CASE 4-1;
PUT EDIT ("CASE 0") (A);
PUT EDIT ("CASE 1") (A);
PUT EDIT ("CASE 2") (A);
PUT EDIT ("CASE 3") (A);
PUT EDIT ("CASE 4") (A);
END CASE;
END;
DO I=10 TO -2 BY -1;
M(I,5)=0;
DO J=(3*2) - 5 TO 10 WHILE(J<5);
M(I+J)=J;
END;
PUT EDIT ("I = " "M(I,1) = " "M(I,5) = " "A*I(5)) ;
END;
PUT EDIT ("EXIT DO GROUPS") (SKIP,A);
END DOGRP;

/*
 ARITHMETIC
 */

ARITH:BEG;
DECLARE (X,Y,Z) FLOAT; (A,B,C) FIXED;
PUT EDIT ("ENTER ARITHMETIC BLOCK") (SKIP(5),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:BEG;
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 RPRLC;

/* 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..1)=TEST THEN DO; I=I+1; RETURN(.T.); END;
  RETURN(.F.);
END LITERAL;
PRIMARY:PROCEDURE;
  IF LITERAL("(") THEN
    DO;
      IF .NOT. EXP1 THEN RETURN(.F.);
      IF . 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(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
*/

glob: begin;
P: PROCEDURE(X,Y);  
  DECLARE X ENTRY, Y FIXED;  
  DECLARE I FIXED;  
  BEGIN;  
    Q: PROCEDURE(Z);  
      DECLARE Z ENTRY;  
      DECLARE F(1:10) FIXED;  
      F(1)=13;  
      CALL Z((F(1)+Y));  
    END Q;  
    CALL Q(X);  
  END;  
END P;  
R: PROCEDURE;  
  DECLARE (I,G(1:10)) 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; META X 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 STKTP $STKTO,$STKTI
00026 $003 FMT $004,"1"
00032 LDA "BEGIN STRINGS"
00047 EDIT
00048 PUT
00049 JUMP $005
***FORMAT CODE***
00054 $004 "2000380000"
00060 $005 LDA S,"OL"
00074 LDA "THIS IS A STRING."
00093 STO
00094 FMT $006,"1"
00109 LDA S,"OL"
00109 EDIT
00113 PUT
00119 JUMP $007
***FORMAT CODE***
00115 $006 "80000"
00122 $007 LDA T,"OL"
00130 LD "0001"
00136 LD "0004"
00142 SBSTR
00143 LDA "THIS"
00149 STO
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."
00210  STM
00219  FMT $008 "1"
00225  LDA T "0L"
00230  EDIT
00235  PUT
00235  JUMP $009
**FORMAT CODE***
00240  $008 "8000J"
00247  $009 LDA S "0L"
00255  LD "0001"
00261  LD "0005"
00267  SBSTR
00263  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 "0L"
00333  LD "0001"
00339  LD "0004"
00345  SBSTR
00346  LDA T "0L"
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
***FORM\$T CODE***
00411 $014 "80000"
00418 $013 FMT $016 ,"1"
00424 LDA "EXIT STRINGS"
00430 EDIT
00434 PUT
00440 JUMP $017

***DOPE VECTOR CODE***
104Z0001Z0005
00492 JUMP $018
00497 $018 FMT $019 ,"1"
00503 LDA "BEGIN I/O BLOCK"
00520 EDIT
00521 PUT
00522 JUMP $020

***FORM\$T CODE***
00527 $019 "2000380000"
00539 $020 FMT $021 ,"Z"
00545 LDA S ,"0D"
00550 GET
00554 JUMP $022

***FORM\$T CODE***
00559 $021 "80006"
00565 $022 FMT $023 ,"Z"
00572 LDA A
00575 GET
00579 LDA B
00585 GET
00586 LDA C
00592 GET
00593 JUMP $024

***FORM\$T CODE***
00598 $023 "30001:0005:0005:0005"
00620 $024  LDA T ,"OD"
00628  FMT $025 ,"0"
00634   LD A
00640   EDIT
00646   LD B
00647   EDIT
00648   LD C
00654   EDIT
00655   PUT
00656  JUMP $026
***FORMAT CODE***
00661 $025  "00C5"
00663 $026  FMT $027 ,"1"
00674   LDA S ,"OD"
00682   EDIT
00683   LDA T ,"OD"
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  "20001800003000D800003000D0005*0005*0005*0005"
00761 $029  FMT $029 ,"Z"
00767   LDA S ,"OD"
00775   GET
00776   LDA T ,"OD"
00784   GET
00785  JUMP $030
***FORMAT CODE***
00790 $029  "200018000D8000D"
00807 $030  LDA S ,"OD"
00815  FMT $031 ,"Y"
00827  LDA  X
0082B  GET
0082F  LDA  Y
00833  GET
00835  JUMP  $032

***FORMAT CODE***
00840  $031  "9000*"
00847  $032  LDA  T  "0D"
00855  FMT  $033  "Y"
00861  LDA  Z
00867  GET
00868  JUMP  $034

***FORMAT CODE***
00873  $033  "9000*"
00880  $034  FMT  $035  "1"
00885  LD  X
00892  EDIT
00893  LD  Y
00899  EDIT
00900  LD  Z
00906  EDIT
00907  LDA  S  "0D"
00915  EDIT
00916  LDA  T  "0D"
00924  EDIT
00925  PUT
00926  JUMP  $036

***FORMAT CODE***
00931  $035  "200029000D9000D9000D200018000D8000D"
00968  $036  LDA  A
00974  LD  "0001"
00980  SST
00981  $038  LD  "0005"
00987  STCKC  "4"
00989  JUMPT  $037
00994  LDA  M
01000  LD  A
<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>01006</td>
<td>INDXA &quot;1&quot;</td>
<td></td>
</tr>
<tr>
<td>01008</td>
<td>LD A</td>
<td></td>
</tr>
<tr>
<td>01014</td>
<td>STO</td>
<td></td>
</tr>
<tr>
<td>01015</td>
<td>LDA A</td>
<td></td>
</tr>
<tr>
<td>01021</td>
<td>LD A</td>
<td></td>
</tr>
<tr>
<td>01022</td>
<td>LD &quot;0001&quot;</td>
<td></td>
</tr>
<tr>
<td>01033</td>
<td>ADD</td>
<td></td>
</tr>
<tr>
<td>01034</td>
<td>SST</td>
<td></td>
</tr>
<tr>
<td>01035</td>
<td>JUMP $038</td>
<td></td>
</tr>
<tr>
<td>01040</td>
<td>LDA A</td>
<td>$037</td>
</tr>
<tr>
<td>01046</td>
<td>FMT $039</td>
<td></td>
</tr>
<tr>
<td>01052</td>
<td>LD &quot;0001&quot;</td>
<td></td>
</tr>
<tr>
<td>01058</td>
<td>SST</td>
<td></td>
</tr>
<tr>
<td>01059</td>
<td>LD &quot;0005&quot;</td>
<td>$041</td>
</tr>
<tr>
<td>01065</td>
<td>STCKC &quot;4&quot;</td>
<td></td>
</tr>
<tr>
<td>01067</td>
<td>JUMPT $040</td>
<td></td>
</tr>
<tr>
<td>01072</td>
<td>LDA M</td>
<td></td>
</tr>
<tr>
<td>01073</td>
<td>LD A</td>
<td></td>
</tr>
<tr>
<td>01084</td>
<td>INDXR &quot;1&quot;</td>
<td></td>
</tr>
<tr>
<td>01085</td>
<td>EDIT</td>
<td></td>
</tr>
<tr>
<td>01087</td>
<td>LDA A</td>
<td></td>
</tr>
<tr>
<td>01093</td>
<td>LD A</td>
<td></td>
</tr>
<tr>
<td>01099</td>
<td>LD &quot;0001&quot;</td>
<td></td>
</tr>
<tr>
<td>01105</td>
<td>ADD</td>
<td></td>
</tr>
<tr>
<td>01106</td>
<td>SST</td>
<td></td>
</tr>
<tr>
<td>01107</td>
<td>JUMP $041</td>
<td></td>
</tr>
<tr>
<td>01112</td>
<td>PUT</td>
<td>$040</td>
</tr>
<tr>
<td>01113</td>
<td>JUMP $042</td>
<td></td>
</tr>
<tr>
<td>01118</td>
<td>LDA A</td>
<td>$039</td>
</tr>
<tr>
<td>01125</td>
<td>FMT $043</td>
<td></td>
</tr>
<tr>
<td>01131</td>
<td>LDA &quot;EXIT I/O BLOCK&quot;</td>
<td>$042</td>
</tr>
<tr>
<td>01147</td>
<td>EDIT</td>
<td></td>
</tr>
<tr>
<td>01148</td>
<td>PUT</td>
<td></td>
</tr>
<tr>
<td>01149</td>
<td>JUMP $044</td>
<td></td>
</tr>
<tr>
<td>01154</td>
<td>LDA A</td>
<td>$043</td>
</tr>
<tr>
<td>01160</td>
<td>FMT $044</td>
<td></td>
</tr>
<tr>
<td>01166</td>
<td>PUT</td>
<td>$045</td>
</tr>
</tbody>
</table>

***FORMAT CODE***
01166 $044 STKTP $STKTO $STKTL
01177' ALLOC $STKTL $M

***DOPE VECTOR CODE***
"204ZccqZ000*Z0001Z0001"

01211 JUMP $045
01216 $045 FMT $046 "1"
01222 LDA "BEGIN DO GROUPS"
01230' EDIT
01240' PUT
01244 JUMP $047

***FORMAT CODE***
01246 $046 "2000380000"
01258 $047 LDA I
01264 LD "0000"
01270 SST
01271 $049 LD "0004"
01277 STCKC "4"
01279 JUMPT $048
01285 LD "0004"
01290 LD I
01295 SUB
01297 LD "0005"
01303 MULT
01304 LD $051
01310 ADD
01311 JUMPA
01312 $053 FMT $054 "1"
01318 LDA "CASE 0"
01326 EDIT
01327 PUT
01328 JUMP $055

***FORMAT CODE***
01333 $054 "80000"
01340 $055 JUMP $052
01345 $056 FMT $057 "1"
01351 LDA "CASE 1"
01359 EDIT
01360   PUT
01361   JUMP $058

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

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

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

***FORMAT CODE***
01465 $066 "80000"
01472 $067   JUMP $052
01477 $068   LD $050
01483 $050   JUMP $053
01488   JUMP $056
01493   JUMP $059
01498   JUMP $062
01503   JUMP $065
01508 $052   LDA I
01514   LD I
01520 LD "0001"
01526 ADD
01527 SST
01528 JUMP $049
01535 LDA I
01536 LD "0004"
01545 SST
01546 JUMP $069
01552 STCKC "2"
01554 JUMPT $069
01559 LDA M
01565 LD I
01571 LD "0005"
01577 INDXA "2"
01579 LD "0000"
01585 STO
01586 LDA A
01592 LD "0003"
01598 LD "0002"
01604 MULT
01605 LD "0005"
01611 SUB
01612 SST
01613 STCKC "4"
01619 JUMPT $071
01625 LD A
01631 LD "0005"
01637 STCKC "2"
01643 JUMPF $071
01649 LDA M
01655 LD I
01657 LD A
01663 INDXA "2"
01665 LD A
01671 STO
01672 LDA A
01678     LD A
01684     LD "0001"
01690     ADD
01691     SST
01692     JUMP $072
01697 $071  FMT $073 ,"1"
01703     LDA "I = "
01705     EDIT
01710     LD I
01716     EDIT
01717     LDA ", M(I,1) = "
01720     EDIT
01721     LDA M
01727     LD I
01732     LD "0001"
01742     INDXR "2"
01747     EDIT
01752     LDA ", M(I,5) = "
01757     EDIT
01762     LDA M
01768     LD I
01773     LD "0005"
01778     INDXR "2"
01783     EDIT
01788     JUMP $074

***FORMAT CODE***
01793 $073  "8000000005"
01805 1074     LDA I
01810     LD I
01817     LD "0001"
01823     SUB
01828     SST
01833 $069  JUMP $070
01839 $069  FMT $075 ,"1"
01845     LDA "EXIT DO GROUPS"
01851     EDIT
01853 PUT
01854 JUMP $076
***FORMAT CODE***
01859 $075 "2000160000"
01871 $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
01992 LDA Y
01998 LD X
02004 DIV
02005 STO
02006 LDA Z
02012 LD A
02018 LD "0005"
02024 STCKC "2"
02026 JUMPF $082
02031 LD A
02037 JUMP $083
02042 $082 LD "00000000"
02052 $083 STO
02053 $084 FMT *084 "1"
02059 LD X
02065 EDIT
02066 LD Y
02072 EDIT
02073 LD Z
02079 EDIT
02086 PUT
02081 JUMP $085

***FORMAT CODE***
02086 $084 "9000D"
02095 $085 FMT $086 "1"
0209b LD A
02105 EDIT
02106 PUT
02107 JUMP $087

***FORMAT CODE***
02112 $086 "0005"
02119 $087 LDA A
02125 LD A
02131 LD "0001"
02137 ADD
02138 SST
02139 JUMP $081
02144 $080 FMT $088 "1"
02150 LDA "EXIT ARITHMETIC BLOCK"
02173 EDIT
02174 PUT
02175 JUMP $089

***FORMAT CODE***
02180 $088 "2000180000"
02192 $089 STkTP $STkTO $STkT1
02203 JUMP $090
02208 NFACT ENTPR "2"
02210 DYNAM $091
02215 JUMP $092
02220 $092  LD  I
02226  LD "0000"
02232  STCKC "1"
02234  JUMPF $093
02239  LD "0001"
02245  SWAP
02246  RETRN
02247 $093  LDA $094
02250  LD $ACTIVE
02250  LD $ACTIVE
02265  FLAG
02266  LDA $095
02272  LDA $095
02278  LD I
02284  LD "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  JUMPA $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 $STKT0,$STKT1
02550    JUMP $106
02555    OUT ENTPK "2"
02557    DYNAM $095
02562 JUMP $108
02567 LDA T "1+
02575 LD J
02581 LD "0001"
02587 SBS T
02588 LDA OUTCHAR "01"
02596 STO
02597 LDA J
02603 LD J
02605 LD "0001"
02615 ADD
02616 STO
02617 RETR N
02618 NEXT ENTRP "2"
02620 DYNAM I
02625 JUMP $110
02630 LDA INPUT "1+
02638 LD I
02644 LD "0001"
02650 SBS T
02651 LDA " "
02654 COMPC "1"
02656 JUMPF $112
02661 LDA I
02667 LD I
02673 LD "0001"
02679 ADD
02680 STO
02681 LD I
02687 LD "001+
02693 STCKC "4"
02695 JUMPF $113
02700 LD "F"
02703 SWAP
02706 RETR N
02705 LDA INPUT "1+
02713 LD I
02719   LD     "0001"
02725   SBSTk
02726   LDA     "$1"
02729   COMPC     "$1"
02731   JUMPF     $114
02736   LDA     "$F"
02739   SWAP
02740   RETRN
02741 $114   JUMP     $110
02746 $112   LD     "$T"
02749   SWAP
02750   RETRN
02751   RETRN
02752 NUMBF R   ENTPR     "2"
02754   DYNAM     I
02759   JUMP     $116
02764 $116   LDA     $117
02770   LDA     $ACTIVE
02776   LDA     $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     "$1+
02815   LD     I
02821   LD     "$0001"
02827   SBSTR
02828   LDA     "$9"
02831   COMPC     "$3"
02833   LDA     INPUT     "$1+
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+
02989  LD  I
02995  LD  "0001"
03001  SBSTR
03002  LDA  "A"
03005  COMPC  "5"
03007  LDA  INPUT  ,"1+
03015  LD  I
03021  LD  "0001"
03027  SBSTR
03028  LDA  "Z"
03031  COMPC  "3"
03033  AND
03034  JUMPF  $124
03039  LDA  CHAR  ,"01"
03047  LDA  INPUT  ,"1+
03055  LD  I
03061  LD  "0001"
03067  SBSTR
03068  STO
03069  LDA  I
03075  LD  I
03081  LD  "0001"
03087  ADD
03088  STO
03092  SWAP
03093  RETRN
03094  $124  LD  "F"
03097  SWAP
03098  RETRN
03100  RETRN
03100  LITERAL
03102  ENTRP  "2"
03107  DYNAM  $095
03112  JUMP  $126
03112  LDA  $127
03392  SWAP
03393  RETRN
03394  $134  LDA  $140
03400  LD  $ACTIVE
03406  LD  $ACTIVE
03412  FLAG
03413  LD  $STKT2
03419  LDA  NUMBER
03425  JUMPA
03426  $140  JUMPF  $141
03431  LDA  $142
03437  LD  $ACTIVE
03443  LD  $ACTIVE
03449  FLAG
03450  LDA  CHAR  "01"
03456  LD  $STKT2
03462  LDA  OUT
03470  JUMPA
03471  $142  LD  "T"
03477  SWAP
03478  RETRN
03484  $141  LDA  $143
03488  LD  $ACTIVE
03494  LD  $ACTIVE
03495  FLAG
03496  LD  $STKT2
03502  LDA  ID
03508  JUMPA
03514  $143  JUMPF  $144
03531  LDA  $145
03538  LD  $ACTIVE
03544  LD  $ACTIVE
03550  FLAG
03551  LDA  CHAR  "01"
03558  LD  $STKT2
03564  LDA  OUT
03572  JUMPA
03553  $145  LD    "T"
03556  SWAP
03557  RETRN
03558  $144  LD    "F"
03561  SWAP
03562  RETRN
03563  RETRN
03564  TERM    ENTPR "2"
03566  DYNAM $146
03567  JUMP $147
03576  $147  LDA    $148
03582  LD    $ACTIVE
03588  LD    $ACTIVE
03594  FLAG
03595  LD    $5STK2
03601  LDA    PRIMARY
03607  JUMPA
03608  $148  NOT
03609  JUMPF $149
03614  LD    "F"
03617  SWAP
03619  $149  LDA    $150
03625  LD    $ACTIVE
03631  LD    $ACTIVE
03637  FLAG
03638  LDA    $133
03644  LDA    $133
03650  LDA    "*"
03653  STO
03654  LD    $STK2
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 $SKT2
03843  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 $SKT2
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 "/"
03974  LD  $STKT2
03980  LDA  LITERAL
03986  JUMPA
03987  JUMPF $166
03992  LDA  $169
03998  LD  $ACTIVE
04004  LD  $ACTIVE
04010  FLAG
04017  LDA  $STKT2
04023  JUMPA
04026  $169
04029  JUMPF $170
04030  LD  "F"
04033  SWAP
04036  RETRN
04035  LDA  $171
04041  LD  $ACTIVE
04047  LD  $ACTIVE
04053  FLAG
04054  LDA  $138
04060  LDA  $138
04066  LDA  "m"
04069  STO
04070  LD  $STKT2
04076  LDA  OUT
04082  JUMPA
04083  $171
04086  LD  "T"
04087  SWAP
04088  RETRN
04083  $168
04094  LDA  $171
04099  LD  $ACTIVE
04100  LD  $ACTIVE
04106  FLAG
04107  LDA  $158
04113  LDA  $158
04119  LDA  "m"
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
04388 $188  LDA $189
04394  LD $ACTIVE
04400  LD $ACTIVE
04406  FLAG
04407  LDA $138
04413  LDA $138
04419  LDA "+"
04422  STO
04423  LD $STKT2
04429  LDA OUT
04435  JUMPA
04436 $189  LDA $183
04442  JUMPA
04443 $186  LDA $191
04449  LD $ACTIVE
04455  LD $ACTIVE
04461  FLAG
04462  LDA $158
04468  LDA $158
04474  LDA "-"
04477  STO
04478  LD $STKT2
04484  LDA LITERAL
04490  JUMPA
04491 $191  JUMPF $193
04495  LDA $194
04502  LD $ACTIVE
04508  LD $ACTIVE
04514  FLAG
04515  LD $STKT2
04521  LDA TERM
04527  JUMPA
04528 $194  NOT
04529  JUMPF $195
04534  LD "F"
04537  SWAP
04538  RETRN
04539 $195 LDA $196
04545 LD $ACTIVE
04551 LD $ACTIVE
04557 FLAG
04558 LDA $163
04564 LDA $163
04570 LDA ":-"
04576 STK
04584 LD $STKT2
04587 LDA $183
04588 JUMPA
04591 LDA $196
04592 JUMPA
04595 LD "$T"
04597 SWAP
04598 RETRN
04599 RETRN
04600 $106 FMT $198 "$1"
04600 LDA "ENTER POSTFIX"
04621 EDIT
04622 PUT
04623 JUMP $199

***FORMAT CODE***
04628 $198 "2000380000"
04640 $199 FMT $200 "$Z"
04640 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+
04720  EDIT
04721  PUT
04722  JUMP  $203
***FORMAT CODE***
04734  $202  "2000180000000028001+
04756  $203  LDA  T  "1+
04764  LDA  INPUT  "1+
04772  STO
04773  LDA  $204
04775  LD  $ACTIVE
04778  LD  $ACTIVE
04781  FLAG
04782  LD  $STKTI
04790  LDA  EXP1
04804  JUMPA
04805  $204  JUMPF  $205
04810  FMT  $206  "1"
04816  LDA  "POSTFIX EXPRESSION ="
04830  EDIT
04839  LDA  T  "1+
04845  LD  "0001"
04853  LD  J
04859  LD  "0001"
04865  SUB
04866  SBSTR
04867  LDA  ","
04870  CAT
04871  EDIT
04872  PUT
04873  JUMP  $207
***FORMAT CODE***
04873  $206  "20001800000000280000"
04900  $207  JUMP  $208
04905  $205  FMT  $209  "1"
04911  LDA  "*****ERROR*****"
04926 EDIT
04927 PUT
04928 JUMP $209

***FORMAT CODE***
04932 $209 "2000160000"
04945 $208 FMT $211 "1"
04951 LDA "EXIT POSTFIX"
04965 EDIT
04966 PUT
04967 JUMP $212

***FORMAT CODE***
04972 $211 "2000180000"
04984 $212 STKP $STK0,$STK1
04995 JUMP $213
05000 P ENTPR "2"
05002 DYNAM $163
05007 JUMP $215
05012 $215 STKP $STK2,$STK3
05023 JUMP $216
05028 W ENTPR "3"
05030 DYNAM $217
05035 ALLOC $STK4,$F

***DOPE VECTOR CODE***
"104200012000"
05059 JUMP $219
05064 $218 LDA F
05070 LD "0001"
05075 INDXA "1"
05078 LD "000"
05084 STO
05085 LDA $219
05091 LD $ACTIVE
05097 LDA $220
05103 LD "300+" ****
05109 LD "0004"
05115 ADD
05116 STO
05117  LD  "0000"  *****
05123  FLAG
05124  LDA  $221
05130  LDA  $221
05136  LDA  F
05142  LD  "0001"
05148  INDXR  "1"
05150  LD  Y
05156  ADD
05157  STO
05158  LD  $STKT4
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  $STKT3
05249  LDA  Q
05255  JUMPA
05256  $222  RETRN
05257  R  ENTRP  "2"
05259  DYNAM  $226
05264  ALLOC  $STKT2  *G
<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>05288</td>
<td>JUMP $227</td>
</tr>
<tr>
<td>05293</td>
<td>$227 STkT $STkT2 $STkT3</td>
</tr>
<tr>
<td>05304</td>
<td>JUMP $228</td>
</tr>
<tr>
<td>05309</td>
<td>U ENTPR &quot;3&quot;</td>
</tr>
<tr>
<td>05311</td>
<td>DYNAM F</td>
</tr>
<tr>
<td>05316</td>
<td>JUMP $230</td>
</tr>
<tr>
<td>05321</td>
<td>$230 LDA G</td>
</tr>
<tr>
<td>05327</td>
<td>LD I</td>
</tr>
<tr>
<td>05333</td>
<td>INDXA &quot;1&quot;</td>
</tr>
<tr>
<td>05335</td>
<td>LD W</td>
</tr>
<tr>
<td>05341</td>
<td>STO</td>
</tr>
<tr>
<td>05342</td>
<td>RETRN</td>
</tr>
<tr>
<td>05343</td>
<td>$228 LDA I</td>
</tr>
<tr>
<td>05349</td>
<td>LD &quot;0001&quot;</td>
</tr>
<tr>
<td>05355</td>
<td>SST</td>
</tr>
<tr>
<td>05356</td>
<td>$232 LD &quot;000*&quot;</td>
</tr>
<tr>
<td>05362</td>
<td>STCKC &quot;4&quot;</td>
</tr>
<tr>
<td>05364</td>
<td>JUMPT $231</td>
</tr>
<tr>
<td>05369</td>
<td>LDA G</td>
</tr>
<tr>
<td>05375</td>
<td>LD I</td>
</tr>
<tr>
<td>05381</td>
<td>INDXA &quot;1&quot;</td>
</tr>
<tr>
<td>05383</td>
<td>LD &quot;000G&quot;</td>
</tr>
<tr>
<td>05389</td>
<td>STO</td>
</tr>
<tr>
<td>05390</td>
<td>LDA $233</td>
</tr>
<tr>
<td>05396</td>
<td>LD $ACTIVE</td>
</tr>
<tr>
<td>05402</td>
<td>LD $ACTIVE</td>
</tr>
<tr>
<td>05408</td>
<td>FLAG</td>
</tr>
<tr>
<td>05409</td>
<td>LDA $224</td>
</tr>
<tr>
<td>05415</td>
<td>LDA $224</td>
</tr>
<tr>
<td>05421</td>
<td>LDA &quot;018%&quot;</td>
</tr>
<tr>
<td>05427</td>
<td>STO</td>
</tr>
<tr>
<td>05428</td>
<td>LDA $235</td>
</tr>
<tr>
<td>05434</td>
<td>LD $ACTIVE</td>
</tr>
<tr>
<td>05440</td>
<td>STO</td>
</tr>
<tr>
<td>05441</td>
<td>LDA $236</td>
</tr>
</tbody>
</table>
05447:    LDA $236
05453:    LD "0001"
05459:    STO
05460:    LD $STKT3
05466:    LDA P
05472:    JUMPA
05473:    $233 LDA I
05479:    LD I
05485:    LD "0001"
05491:    ADD
05492:    SST
05493:    JUMP $232
05493:    $231 FMT $237 "1"
05504:    LDA "GLOBAL DISPLAY TEST"
05525:    EDIT
05526:    PUT
05527:    JUMP $238

***FORMAT CODE***
05532:    $237 "2000360000"
05544:    $238 FMT $239 "1"
05550:    LDA I
05556:    LD "0001"
05562:    SST
05563:    $241 LD "0001"
05569:    STCKC "4"
05571:    JUMPT $240
05576:    LDA G
05582:    LD I
05588:    INDXR "1"
05590:    EDIT
05591:    LDA I
05597:    LD I
05603:    LD "0001"
05609:    ADD
05610:    SST
05611:    JUMP $241
05616:    $240 PUT
0561:7 JUMP $242

***FORMAT CODE***
0562:2 $239 "0007"
0562:9 $242 RETRN
0563:0 $213 LDA $243
0563:6 LD $ACTIVE
0564:2 LD $ACTIVE
0564:3 FLAG
0564:9 LD $STKT1
0565:5 LDA R
0566:1 JUMPA
0566:2 $243 FMT $244 ;"1"
0566:8 LDA "EXIT GLOBAL TEST"
0568:6 EDIT
0568:7 PUT
0568:8 JUMP $245

***FORMAT CODE***
0569:3 $244 "2000180000"
0570:5 $245 STKP $STKT0 ,S$STKT1
0571:6 ALLOC $STKT1 ,Z

***DOPE VECTOR CODE***
"104Z00012Z0003"
0574:0 JUMP $246
0574:5 LBL ENTPK "2"
0574:7 DYNAM $095
0575:2 JUMP $248
0575:8 $248 POPUP
0575:8 LDA LABEL
0576:4 LD "0003"
0577:0 INDXR "1"
0577:2 RETRN
0577:3 RETRN
0577:4 $246 FMT $249 ;"1"
0578:0 LDA "ENTER LABEL"
0579:3 EDIT
0579:4 PUT
0579:5 JUMP $250
```assembly
***FORMAT CODE***
05800  $249  "2000380000"
05812  1250  LDA  Y
05818  LD  "011"
05824  STD
05825  STKTP  STKT1,STKT2
05836  JUMP  $251
05841  $251  LDA  I
05847  LD  "0001"
05853  STO
05854  LD  "012"
05860  JUMPA
05861  LDA  B
05867  LD  I
05873  LD  "0002"
05879  DIV
05880  STO
05881  LBL1  FMT  $252  "1"
05887  LDA  "LABEL TEST"
05899  EDIT
05900  LD  I
05906  EDIT
05907  PUT
05903  JUMP  $253
***FORMAT CODE***
05913  $252  "2000180000:0005"
05930  $253  LDA  A
05935  LD  I
05942  LD  "0001"
05948  ADD
05949  STO
05950  LD  Y
05956  JUMPA
05957  LBL2  FMT  $254  "1"
05963  LDA  "INCORRECT LABEL TEST"
05985  EDIT
05986  PUT
```
JUMP $255

***FORMAT CODE***

$254 "80000"

FMT $256 "1"

LDA "INCOMPLETE LABEL TEST"

EDIT

PUT

JUMP LBL2

***FORMAT CODE***

LBL2 FMT $258 "1"

LDA "LABEL TEST"

EDIT

LD A

EDIT

PUT

JUMP $259

***FORMAT CODE***

$258 "2000180000*0005"

LDA Z

LD "0001"

INDX A "1"

LD "01-J"

STO

LDA Z

LD "0002"

INDX A "1"

LD "01-J"

STO

LDA Z

LD "0003"

INDX A "1"

LD "01-J"

STO

LDA BADLAB

LD $ACTIVE

LD $ACTIVE
06171  FLAG
06172  LDA  Z
06173  LD  $STKTI
06184  LDA  LBL
06190  JUMPA
06191  BADLAB  FMT  $261  "1"
06197  LDA  "INCORRECT LABEL RETURN"
06221  EDIT
06222  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"
06305  EDIT
06306  PUT
06307  JUMP  $266
   ***FORMAT CODE***
06312  $265  "2000180000"
06324  $266  STOP
06325  END  PROGRAM
BEGIN STRINGS
THIS IS A STRING.
THIS IS A CONCATENATED SUBSTRING.
STRING COMPARE 1 WORKS
STRING COMPARE 2 WORKS
EXIT STRINGS

BEGIN I/O BLOCK
123  -12  20  123  -12  20  123  -12  20
5.12340  .4300000000E+01  -.1120000000E-007
  5.1234  4.3E+14  -11.2E-9
  1  2  3  4  5
EXIT I/O BLOCK

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

ENTER ARITHMETIC BLOCK

<table>
<thead>
<tr>
<th>I</th>
<th>M(I,1)</th>
<th>M(I,5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.330</td>
<td>1.330</td>
</tr>
<tr>
<td>2</td>
<td>1.330</td>
<td>2.0</td>
</tr>
<tr>
<td>3</td>
<td>1.330</td>
<td>3.0</td>
</tr>
<tr>
<td>4</td>
<td>1.330</td>
<td>4.0</td>
</tr>
<tr>
<td>5</td>
<td>1.330</td>
<td>5.0</td>
</tr>
<tr>
<td>6</td>
<td>1.330</td>
<td>6.0</td>
</tr>
<tr>
<td>7</td>
<td>1.330</td>
<td>7.0</td>
</tr>
<tr>
<td>8</td>
<td>1.330</td>
<td>8.0</td>
</tr>
<tr>
<td>9</td>
<td>1.330</td>
<td>9.0</td>
</tr>
<tr>
<td>10</td>
<td>1.330</td>
<td>10.0</td>
</tr>
</tbody>
</table>

EXIT ARITHMETIC BLOCK
ENTER RPRCC
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
EXIT GLOBAL TEST

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

//
DECLARE INPUT CHAR(80), (CRDCTN, I, SYMCNT) FIXED, SYMBOL(500) CHAR(6), (CRDREF(500), DECNT) CHAR(5);

I, CRDCTN=0;
SYMCTN=1;
GETCRD: GET EDIT(INPUT) (A(80));
IF SUBSTR(INPUT, 1, 4) = "****" THEN GO TO SORT;
CRDCTN=CRDCTN+10;
PUT STRING(DECNT) EDIT (CRDCTN) (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;
LBL60: J=J+1;
IF J < 1 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 PLXCRPL
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{IP,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{I,J*3+I/14},I);

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

*****ERROR**** F: SYNTAX
END;
END ERROR;
FATAL ERROR(S) ENCOUNTERED, JOB ABORTED
PLXiNT INTERPRETIVELY EXECUTES OBJECT PROGRAMS PRODUCED BY THE PLEX COMPILER FOR THE PLEX LANGUAGE.
COMMUNICATION AREA FIELD LOCATION DEFINITIONS

INSTCT EQU 209
STKSTR EQU 223
STKEND EQU 227
LODLOC EQU 219
DYNSTR EQU 231
DYNEND EQU 235

INDEX REGISTER LOCATION DEFINITIONS AND USAGE DESCRIPTIONS

IR1 EQU 4
IR2 EQU 8
IR3 EQU 12
IR4 EQU 16
IR5 EQU 20
IR6 EQU 24
IR7 EQU 28
IR8 EQU 32
IR9 EQU 36
IR10 EQU 40
IR11 EQU 44
IR12 EQU 48
IR13 EQU 52
IR14 EQU 56
IR15 EQU 60

MACRO CALLS TO SOURCE LIBRARY TO ESTABLISH CONVERSION SUBROUTINES

L F8/HD 4,BD, FLOATING BINARY TO FLOATING DECIMAL
L FD/FB 4,DB, FLOATING DECIMAL TO FLOATING BINARY
00750L MUL/D 4,MLT, DOUBLE PRECISION FLOATING BINARY MULTIPLY
00760L DIV/D 4,DIV, DOUBLE PRECISION FLOATING BINARY DIVIDE
00770L INT 4,ITG, FLOATING BINARY TO INTEGER BINARY (TRUNCATED)
00780L IFIX EQU INT RESOLVE CONVERSION NAME MISMATCH
00790L***************************************************************************
00800L* PROGRAM INITIALIZATION:
00810L* SET REGISTERS:
00820L* INITIALIZE DYNAMIC STORAGE DISPLAY:
00830L* START EQU *
00840L***************************************************************************
00850L START EQU *
00860L CAM 60 SET FOUR CHAR ADDRESSING FOR EXECUTION
00870L SW STKEND-2 WORD MARK FOR MOVING AND TESTING
00880L MCw STKSTR,IR2 STACK POINTER
00890L SW IR2-2
00900L SI IR2 ITEM MARK FOR RIGHT MOVE
00910L MCw LODLOC,IR3 INITIALIZE INSTRUCTION COUNTER
00920L SI IR3 ACCOMODATE RIGHT MOVE
00930L LCA =4B0,IR5 PROPERLY
00940L LCA =4B0,IR4 PUNCTUATE
00950L SI IR4,IR5 INDEX REGISTERS IR4,IR5
00960L BS IR7 CLEAR OP CODE REGISTER
00970L SW IR7-1 SHORTEN FETCH ARITHMETIC
00980L SI IR7 ACCOMODATE RIGHT MOVE
00990L LCA =4B0,IR14 INITIALIZE ACTIVE STORAGE POINTER
01000L MCw DYNSTR,IR14 ACCOMMODATE RIGHT MOVE
01010L SI IR14
01020L SW DYNEND-2 WORD MARK FOR COMPARISON
01030L MLWDR IR14,3+X14 INIT DYNAMIC STOR STACK TOP POINTER
01040L MLWDR IR14,7+X14 LEVEL ONE STORAGE POINTER
01050L MCw : ,OUTPUT+132 CLEAR PRINT BUFFER
01060L MCw OUTPUT+132
01070L MCw =1C21,OUTPUT CARRIAGE CONTROL
01080L SI INPUT+80 RESTORE LOST ITEM MARK ON INPUT BUFFER
01090L :SKIP PRINT+57, SKIP TO TOP OF PAGE
01100L LCA +PRNTBF,IR8 OUTPUT BUFFER POINTER
01110L B GETIPT INITIALIZE INPUT BUFFER
01490  00ADD   OP CODE 40
01500  00MULT  OP CODE 41
01510  00SUB   OP CODE 42
01520  00DIV   OP CODE 43
01530  00NEG   OP CODE 44
01540  REP 3  00ERROR
01550  00ERROR OP CODE 50
01560  00FMT   OP CODE 51
01570  00GET   OP CODE 52
01580  00PUT   OP CODE 53
01590  00EDIT  OP CODE 54
01600  REP 4  00ERROR
01610  00ERROR OP CODE 60
01620  00OR    OP CODE 61
01630  00AND   OP CODE 62
01640  00NOT   OP CODE 63
01650  REP 3  00ERROR
01660  00ERROR OP CODE 64
01670  00INDXR OP CODE 66
01680  00INDXA OP CODE 67
01690  00CAT   OP CODE 70
01700  00STR   OP CODE 71
01710  REP 4  00ERROR
01720  00ERROR OP CODE 76
01730  00STOP  OP CODE 76
01740  00ERROR LAST
01750***************************************************************************
01760*                        ADDRESS COMPUTATIONS ARE PERFORMED BY THE ADCOMP *
01770*                        SUBROUTINE. ONE OR TWO ADDRESSES ARE COMPUTED *
01780*                        DEPENDING ON ADDRESSING PARAMETER IN TVEC WHICH IS *
01790*                        ADDRESSED BY THE OP CODE REGISTER IR7. RETURN *
01800*                        ADDRESS IN IR13. *
01810***************************************************************************
01820 ADCOMP SST 0+X3,STYPE,70 SAVE STORAGE TYPE
01830 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 SHIFT LEFT TWO
01910 BA IR9 BITS FOR DISPLAY ADDRESSING
01920 BA IR9+X3,IR9 DISPLAY ADDRESS OF STORAGE LEVEL ADDR
01930 MRID 0+X9,IR4-3 LEVEL BASE ADDRESS
01940 BA 4+X3,IR4 ADD DISPLACEMENT
01950 BS IP4-3 CLEAR HIGH BITS
01960 BA =1B5,IR3 clear high bits
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 IR9+X3,IR9
02050 MRID 0+X9,IR5-3
02060 BA 4+X3,IR5-3
02070 BS IR5-3
02080 BA =1B5,IR3
02090 B 0+X13 RETURN
02100 LITADD MRID IR3-2,IR4-2 LITERAL
02110 BA =1B1,IR4 ADDRESS
02120 MRIN 1+X3,0
02130 SAR IR3 SEQUENCE COUNTER
02140 B 0+X13 RETURN
02150 PRCADD MRID 0+X4,IR4-3 DO IT TWICE FOR REMOTE PROCEDURE
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: SEQUENCE COUNTER
02220 DTYPE DCw :0:
DYNAM - ALLOCATE FIXED DYNAMIC STORAGE (PROCEDURE ENTRY)

DYNAM BA 3+X3,3+X14 INCREMENT STACKTOP
C 3+X14,DYNEND CHECK FOR DYNAMIC STORAGE
BL DYNOFL OVERFLW
BA =1B4,IR3 INSTRUCTION COUNTER
B FETCH

STKTP - INITIALIZE NEW STACKTOP (BLOCK ENTRY)
STKTP MRIDR 0+X4,0+X5
B FETCH

ALLOC - SET DOPE VECTOR FOR INDICATED ARRAY
IN DYNAMIC STORAGE, ALLOCATE DYNAMIC STORAGE FOR THE ARRAY ITSELF.
NOTE THAT ADCOMP IS CALLED TO COMPUTE THE ADDRESSES
OF LOWER AND UPPER BOUND SUBSCRIPT LIMITS.

ALLOC MRSD 0+X3,IR10 INSERT SUBSCRIPT COUNT
SAR IR3
MLWD IR4,IR11 SAVE LOCATION OF STACK TOP VALUE
MLWD IR5,IR12 SAVE DYNAMIC DOPE VECTOR ADDRESS
MRIDI 0+X3,LNGTH-1 MOVE IN ELEMENT LENGTH SIZE
SAR IR3
MCW LNGTH,SIZE INIT ARRAY SIZE COUNTER
BS CONPRT CLEAR
MCW =2C0120,IR7 SET OP CODE REGISTER TO ONE ADDRESS CODE
MCW +ALCRT1,IR13 RETURN ADDRESS FROM ADCOMP
B ADCOMP CALL ADCOMP
02600 ALCRT1 MCW +ALCRT2,IR13
RETURN ADDRESS FROM SECOND CALL
02610 MCW IR4,IR5
SAVE FIRST ADDRESS
02620 B ADCOMP
CALL ADCOMP
02630 ALCRT2 MRIDR 0+X4*0+X12
MOVE UPPER BOUND TO DYNAMIC D.V.
02640 SBR IR12
NEXT MULTIPLIER LOCATION
02650 MCW IR5,IR4
RESTORE FIRST ADDRESS
02660 SW 0-4+X12
PUNCTUATION TO STOP ARITHMETIC
02670 BS 3+X4+0-1+X12
SUBTRACT LOWER BOUND
02680 BA =1B1+0-1+X12
FINISH MULTIPLIER
02690 BIM 0-1+X12+SIZE
UPDATE ARRAY SIZE COMPUTATION
02700 EIM 0-1+X12+CONPRT
UPDATE CONSTANT PART COMPUTATION
02710 BA 3+X4+CONPRT
ADD IN LOWER BOUND
02720 BS =1B1+IR10
02730 BCE 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,CONPRT
LENGTH FACTOR
02790 BS CONPRT,3+X12
FINISH CONSTANT PART
02800 BA SIZE,3+X11
BUMP STACK TOP LOCATION BY ARRAY SIZE
02810 C 3+X11,DYNEND
CHECK FOR DYNAMIC STORAGE ALLOC OVERFLOW
02820 BL DYNFL
02830 E FETCH
02840 LNGTH DCW =4B0
02850 SIZE DCW =4B0
02860 CONPRT DCW =4B0
02870***************************************************************************
02880***************************************************************************
02890* LDA - LOAD ADDRESS TO THE STACK, INCLUDING THE
02900* DATATYPE CODE. IF A CHARACTER STRING ALSO LOAD
02910* THE MAX STRING LENGTH.
02920* *
02930***************************************************************************
02940 LDA MLWDR IR4+4+X2
ADDRESS TO STACK
02950 LCA DTYPE,0+X2
DATA TYPE
02960 BCE LDLNG,DTYPE,04
IF A CHARACTER STRING LOAD LENGTH
02970 UPSTCK BA =1B9,IR2  STACK POINTER
02980 B FETCH ELSE FETCH NEXT INSTRUCTION
02990 LDLNG BCE LITLNG,STYPE,70 IF A LITERAL COMPUTE LENGTH
03000 MRIDR 0+X3,5+X2 ELSE EXTRACT FROM INSTRUCTION
03010 SAR IR3 BUMP SEQUENCE COUNTER
03020 B UPSTCK
03030 LITLNG BS IR5
03040 MRIN 0+X4,0
03050 SAR IR5
03060 BS IR4
03070 MRIDR IR5-1,5+X2
03080 B UPSTCK
03090***************************************************************************
03100* *
03110* LD - LOAD AN OPERAND TO THE STACK, INCLUDING DATA TYPE.
03120* *
03130***************************************************************************
03140 LD MRIDR 0+X4,1+X2 LOAD DATA
03150 BCE CHKTYP,DTYPE,07 IF UNDETERMINED TYPE CHECK IT
03160 LCA DTYPE,0+X2 SET TYPE
03170 B UPSTCK
03180 CHKTYP BI 3+X4,SETINT INTEGER
03190 BI 7+X4,SETFLT FLOAT
03200 B TYPERR
03210 SETINT LCA =1B1,0+X2
03220 B UPSTCK
03230 SETFLT LCA =1B2,0+X2
03240 B UPSTCK
03250***************************************************************************
03260* *
03270* STO -STORES THE ITEM AT THE TOP OF THE STACK AT THE ADDRESS
03280* NEXT TO THE TOP. NOTE SPECIAL HANDLING OF CHARACTER
03290* STRINGS. POP TWO STACK ELEMENTS UPON COMPLETION.
03300* *
03310* SST -SAME AS STO EXCEPT ADDRESS IS DISCARDED BUT THE
03320* STACK TOP IS RETAINED.
03330*
0334C***************************************************************************
0335C
0336C
0337C
0338C
0339C
0340C
0341C
0342C
0343C
0344C
0345C
0346C
0347C
0348C
0349C
0350C
0351C
0352C
0353C
0354C
0355C
0356C
0357C
0358C
0359C
0360C
0361C
0362C
0363C
0364C
0365C
0366C
0367C
0368C
0369C
0370C

STO
EUU *

BCE
CNVRTN,0-18+X2,04
BYPASS IF STRING OR SUBSTRING

BCE
CNVRTN,0-18+X2,34

C
0-9*X2,0-18+X2
IF TYPES NOT EQUAL

BNE
SCNV
DO CONVERSION

CNVRTN
MCW 0-14+X2,IR4
RECEIVING ADDRESS

BCE
CHRSTR,0-9+X2,04
SPECIAL HANDLING IF CHARACTER STRING

BCE
CSBSTR,0-9+X2,34
CHECK FOR SENDING SUBSTRING

B5
=1B18,IR2
POP STACK TWO ELEMENTS

MCW
+ENDPRG,ENDADR
RESTORE STRING WORKING STORAGE POINTER

BCHKE
SAVTOP
PUNCTUATION FOR MOVE

SAVTOP
MLRDR 0-9*X2
DISCARD ADDRESS

CI
0-9*X2,0-10+X2
CLEAR ITEM MARKS

BS
=1B9,IR2
AND SAVE STACK TOP ON TOP

B
FETCH

SSTCKH
BCE
PRMOVE,0-9+X2,34
CHECK FOR SAVE AND STORE (SST)

PNCRTN
BCE
SAVTOP,0-1+X3,23
POP STACK TWO ELEMENTS

BS
=1B18,IR2
POP STACK TWO ELEMENTS

MCW
+ENDPRG,ENDADR
RESTORE STRING WORKING STORAGE POINTER

B
FETCH

SAVTOP
0-9+X2
PUNCTUATION FOR MOVE

SCNV
BCE
CNVRTN,0-18+X2,07
UNIVERSAL TYPE O.K.

BCE
FIXFLT,0-9+X2,01
CHECK FOR SAVING RESULT

TMA
0-1+X2,00
GO CONVERT

B
IFIX
CONVERSION OVERLOW INSTRUCTION

R
B
CNVERR

MCW
IR15,0-5+X2
INTEGER RESULT

SI
0-5+X2
TO

LCA
=1B1,0-9+X2
STACK

B
CNVRTN

MLWDI =4C00000027,0-1+X2
NORMALIZED LOAD

AMA
0-1+X2,70
CLEAR EXTRANEOUS PUNCTUATION

MLWDI =8B0,0-1+X2
FLOATING

TAM
0-1+X2,00
RESULT

SI
0-1+X2
TO STACK

LCA
=1B2,0-9+X2
TO STACK

B
CNVRTN
03710 CHKSTR BS IR5
03720 MCW =1C10,IR5
03730 BCE CHRTRN,0-18+X2,07
03740 MRID 0-13+X2,IR5-1
03750 CHRTRN MCW 0-5+X2,IR9
03760 SW 0+X9
03770 BS IR10
03780 MRLN 0+X9,0+X4
03790 SBR IR10
03800 EA IR4,IR5
03810 C IR5,IR10
03820 BH TMOVE
03830 BCE SUBMV,0-18+X2,34
03840 MRIDR 0+X9,0+X4
03850 SUBMV MRL 0+X9,0+X4
03860 B SSTCHK
03870 B SSTCHK
03880 TMOVE BS IR4,IR5
03890 BA IR5,IR9
03900 BA IR4,IR5
03910 MLWDR 0-1+X9+0-1+X5
03920 B SSTCHK
03930 CSHSTR EQU *
03940 MCW 0-5+X2,IR15
03950 BS IR9
03960 MRL 0-4+X2,IR9-1
03970 BA IR9,IR15
03980 EI NOSET,0-1+X15
03990 SI 0-1+X15
04000 B CHRSTR
04010 NOSET SST =1B0+0-9+X2,70
04020 B CHRSTR
04030 PRMOVE CI 0-1+X15
04040 MCW 0-5+X2,IR9
04050 CW 0+X9
04060 B PNCRTN
04070***************************************************************************
**04080**
*FLAG- MARK STACK FOR THE LIMIT OF PARAMETER ADDRESSES*  
*PRIOR TO A PROCEDURE CALL*

**04130**
FLAG LCA =1C77,0+X2

**04140**
B UPSTCK

**04150**
FLAG LCA =1C77,0+X2

**04160**

**04170**
ENTPRO - ESTABLISH DYNAMIC STORAGE DISPLAY, STORE PARAMETER ADDRESSES IN NEW DYNAMIC STORAGE AREA AND THEN PASS CONTROL TO THE CALLLD PROCEDURE.

**04220**
ENTPRO MRID 0-8+X2,IR14-3 NEW DISPLAY AND STORAGE AREA

**04230**
BS =1B9,IR2

**04240**
BS IR13

**04250**
MRSD 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
04450 BS =1818,IR2 RETURN ADDRESS
04460 MRLDR IR14-3¢X14 1ST DISPLAY ENTRY (STACK TOP VALUE)
04470 SBR IR12 NEXT DISPLAY LOCATION
04480 MRSR 0-1¢X3,IR13
04490 DISPLY MRLDR 0+X9,0¢X12 MOVE GLOBALLY ACTIVE STORAGE LEVEL
04500 SAR IR9 ADDRESSES TO CURRENT DISPLAY
04510 SBR IR12
04520 BS =181,IR13
04530 BCE CURLEV,IR13,01 CURRENT DISPLAY BASE ADDRESS
04540 B DISPLY
04550 CURLEV MRLDR IR14-3¢X12 CURRENT DISPLAY RASE ADDRESS
04560 B FETCH
04570 PRMCPNT DCw =1BO
04580***************************************************************************
04590* *
04600* SWAP - SWAP THE TOP TWO ELEMENTS OF THE STACK *
04610* *
04620***************************************************************************
04630 SWAP SI 0-9¢X2,0-18¢X2
04640 MLRDR 0-1¢X2,HOLD
04650 MLRDR 0-10¢X2,0-1¢X2
04660 MLRDR HOLD,0-10¢X2
04670 CI 0-9¢X2,0-18¢X2
04680 B FETCH
04690 HOLD DCw =9
04700***************************************************************************
04710* *
04720* RETURN - RESTORE DISPLAY ADDRESS FOR CALLING PROCEDURE *
04730* AND SET INSTRUCTION COUNTER WITH RETURN ADDRESS *
04740* *
04750***************************************************************************
04760 RETRN MRLD 0-8¢X2,IR3-3 RETURN ADDRESS TO INST COUNTER
04770 MRLD 0-4¢X2,IR14-3 DISPLAY AT POINT OF CALL
04780 BS =189,IR2
04790 B FETCH
04800***************************************************************************
04820*  JUMPA - JUMP TO THE ADDRESS IN THE STACK; POP THE STACK
04830*  POPUP - POP THE STACK
04840*  
04850*  JUMPA MRID 0-8*X2+IR3-3  SET INSTRUCTION COUNTER
04860*  POPUP BS =1B9*IR2
04870*  B  FETCH
04880*  
04890*  JUMP - JUMP TO THE ADDRESS FOLLOWING THE OP CODE
04900*  
04910*  JUMPT - JUMP IF TOP OF STACK IS TRUE; POP THE STACK
04920*  JUMPF - JUMP IF TOP OF STACK IS FALSE; POP THE STACK
04930*  
04940*  
04950*  JUMP MRID 0*X3+IR3-3
04960*  B  FETCH
04970*  JUMPT BCE JUMPP,0-8*X2,T
04980*  BA =1B4*IR3
04990*  B  POPUP
05000*  JUMPP MRID 0*X3+IR3-3
05010*  B  POPUP
05020*  JUMPF BCE JUMPP,0-8*X2,F
05030*  BA =1B4*IR3
05040*  B  POPUP
05050*  
05060*  STCKC - TESTS 2ND STACK ITEM AGAINST 1ST ACCORDING TO THE LITERAL
05070*  CHARACTER FOLLOWING THE OP CODE.
05080*  RESULTS IN A BOOLEAN VALUE ON TOP OF THE STACK.
05090*  
05100*  
05110*  
05120*  STCKC BCE CHK2ND+0-9*X2,02
05130*  MLWDI =4C00000027,0-1*X2  CONVERT TO FLOATING INTEGER
05140*  CHK2ND BCE SETCND+0-18*X2,02
05150*  MLWDI =4C00000027,0-10*X2  CONVERT TO FLOATING INTEGER
05160*  SETCND MRSD 0*X3,COND-1  SET CONDITION TO BE TESTED
05170*  SAR IR3  BUMP INSTRUCTION COUNTER
05190  AMA  0-10+x2,70  LOAD
05200  SMA  0-1+x2,00  SUBTRACT
05210  COND  FBA  SETT,00  BRANCH ON FLOATING REGISTER COND
05220  SETF  MCW  :F*,0-17+x2  PUNCTUATION FOR POSSIBLE STORE
05230  SI  0-17+x2  SET STACK
05240  MCW  =1C03,0-18+x2  SET STACK
05250  B  POPUP
05260  SETT  MCW  :T*,0-17+x2  SET STACK
05270  SI  0-17+x2  PUNCTUATION FOR POSSIBLE STORE
05280  MCW  =1C03,0-18+x2
05290  B  POPUP
05300  ************************************************************
05310  COMPC-COMPARE THE CHARACTER STRING WHOSE ADDRESS IS 2ND IN THE
05320  STACK AGAINST THE STRING WHOSE ADDRESS IS 1ST. IF THE
05330  STRINGS ARE OF UNEQUAL LENGTH THE SHORTER IS MOVED AND
05340  Padded on the right with blanks.
05350  THE COMPARISON CONDITION IS CONTAINED IN THE CHARACTER
05360  FOLLOWING THE OP CODE.
05370  *
05380  *
05390  *****************************************************
05400  COMPC  BS  IR9  CLEAR
05410  BS  IR10
05420  BS  IR15
05430  MCW  0-14+x2,IR13  LEFT ADDRESSES OF STRINGS
05440  BCE  SUBMV1,0-18+x2,34  IF SUBSTRING PUT IN WORKING STORE
05450  MVRT1  MCW  0-5+x2,IR12  TO INDEX REGISTERS
05460  BCE  SUBMV2,0-9+x2,34  IF SUBSTRING PUT IN WORKING STORE
05470  MVRT2  MRIN  0+x13,0  DETERMINE
05480  SAR  IR9  STRING
05490  MRIN  0+x12,0  LENGTHS
05500  SAR  IR10
05510  BS  IR13,IR9
05520  BS  IR12,IR10
05530  SW  IR9=2
05540  C  IR10,IR9  TEST
05550  CW  IR9=2
05560  BL  MOVNXT
05570  BH  MOVTOP
05580  TSTRNG BS  IR15  SET PROPER
05590  MRSD  0+x3,IR15  CONDITION
05600  SAR  IR3
05610  MRSD  CNDTBL+X15,CNDTST  CODE
05620  Sw  0+x13  WORK MARK FOR COMPARE
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  CNDTSBCT  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  ENDADR+IR13
05750  MCw  ENDADR+IR11
05760  BA  IR10,IR11  RIGHT END OF PADDED FIELD
05770  MRBLNK MRSDR  6LNK,0+x15  PAD
05780  SBR  IR15
05790  C  IR11,IR15  WITH
05800  BBH  SETPNC
05810  B  MRBLNK  BLANKS
05820  SETPNC SI  0-1+x11
05830  B  TSTRNG
05840  MOVTOP MRIDR  0+x12,(ENDADR-3)  RIGHT END OF PADDED FIELD
05850  SBR  IR15
05860  CI  0-1+x15
05870  MCw  ENDADR+IR12
05880  MCw  ENDADR+IR11
05890  BA  IR9,IR11
05900  B  MRBLNK
05910  SUBMV1 MRID  0-13+x2,IR10-1  GET LENGTH
05920  SW  0+x13  WORD MARK FOR MOVE
05930  SAR  IR9                REMEMBER WORD MARK
05940  BA  IR10+IR13           RIGHT END (PLUS ONE)
05950  BA  ENADR+IR10         
05960  MLwDR  0-1+X13+0-1+X10  
05970  SI  0-1+X10            
05980  MCw  ENADR+IR13        ADDRESS IN WORKING STORE
05990  MCw  IR10+ENADR        NEXT WORKING STORE LOCATION
06000  CW  0+1+X9             REMOVE WORD MARK
06010  B  MVRT1
06020  ENADR DSA ENDPRE
06030  SUBMV2 BS  IR10
06040  MRID  0-4+X2*IR10-1
06050  Sw  0+X12
06060  SAR  IR9
06070  BA  IR10+IR12
06080  BA  ENADR+IR10
06090  MLwDR  0-1+X12+0-1+X10
06100  SI  0-1+X10            
06110  MCw  ENADR+IR12
06120  MCw  IR10+ENADR
06130  CW  0+1+X9
06140  B  MVRT2
06150  CNDTBLDCw =8C404241434464547
06160  BLNK DC : :
06170***************************************************************************
06180*  ADD - REPLACE THE TOP TWO NUMBERS ON THE STACK WITH THEIR SUM  *
06190*  SUB - REPLACE THE TOP TWO NUMBERS ON THE STACK WITH THEIR        *
06200*  DIFFERENCE (2ND - 1ST)                                             *
06210*  MULT - REPLACE THE TOP TWO NUMBERS ON THE STACK WITH THEIR PRODUCT *
06220*  DIV - REPLACE THE TOP TWO NUMBERS ON THE STACK WITH THEIR          *
06230*  RATIO ( 2ND / 1ST )                                               *
06240*  NEG - UNARY MINUS OPERATION ON THE ELEMENT ON THE TOP OF THE STACK *
06250*  NOTE THE TYPE CONVERSIONS WHICH MAY TAKE PLACE.                   *
06260*                                                                       *
06270***************************************************************************
06300 BCE INTADD,ARI,01 STACK TOP TO FRO
06310 TMA 0-1+X2,00 ADD NEXT
06320 AMA 0-10+X2,00
06330 SETBCK FBI FERR,06 BRANCH ON ANY ERROR
06340 BS =1B9,IR2 REDUCE STACK
06350 MLWDR =8BO+0-1+X2 CLEAR EXTRANEOUS PUNCTUATION
06360 SI 0-1+X2 PROPER PUNCTUATION
06370 TAM 0-1+X2,00 STORE RESULT
06380 LCA =1B2,0-9+X2 DATA TYPE
06390 B FETCH
06400 INTADD Sw 0-17+X2 WORD MARK TO STOP ADDITION
06410 BA 0-5+X2,0-14+X2 BINARY ADD
06420 BS =1B9,IR2 REDUCE STACK
06430 B FETCH
06440 SUB B CNVCHK
06450 BCE INTSUB,ARI,01 IF INTEGERS BRANCH
06460 TMA 0-10+X2,00 FLOATING
06470 SMA 0-1+X2,00 POINT
06480 B SETBCK
06490 INTSUB Sw 0-17+X2 WORD MARK TO STOP ARITHMETIC
06500 BS 0-5+X2,0-14+X2 INTEGER SUBTRACTION
06510 BS =1B9,IR2
06520 B FETCH
06530 MULT B CNVCHK
06540 BCE INTMLT,ARI,01 IF INTEGERS BRANCH
06550 TMA 0-1+X2,00 FLOATING
06560 MAM 0-10+X2,00 POINT
06570 B SETBCK
06580 INTMLT BIM 0-5+X2,0-14+X2
06590 BS =1B9,IR2
06600 B FETCH
06610 DIV MRSR 0-9+X2,ARI SUM OF TYPES FOR LATER TESTING
06620 BA 0-18+X2,ARI
06630 BCE NUMER,0-9+X2,02 CONVERT TO UNNORMALIZED FLTNG PT
06640 MLWDI =4C00000027,0-1+X2
06650 NUMER BCE DODIV,0-18+X2,02 CONVERT TO UNNORMALIZED FLTNG PT
06660 MLWDI =4C00000027,0-10+X2
06670 DODIV AMA 0-1+X2,71 NORMALIZED LOAD TO FR1
06680 AMA 0-10+X2,70 NORMALIZED LOAD TO FRO
06690 DAA 10 RESULT IN FRO
06700 BCE INTDIV,ARI,02 CHECK FOR INTEGER RESULT
06710 B SETBCK
06720 INTDIV B IFIX GO CONVERT
06730 R B CNVERR CONVERSION OVERFLOW INSTRUCTION
06740 MLwD IR15,0-14+X2 PUT RESULT IN STACK
06750 BS =189,IR2
06760 B FETCH
06770 NEG BCE INTNEG,0-9+X2,01 IF INTEGER THEN BRANCH
06780 SMA 0-1+X2,70 ELSE SUBTRACT FROM NORMAL ZERO
06790 TAM 0-1+X2:00 AND STORE
06800 B FETCH
06810 INTNEG LCA =480,IR15 CLEAR REGISTER
06820 BS 0-5+X2,IR15 SUBTRACT
06830 MLwD IR15,0-5+X2 PUT BACK
06840 B FETCH
06850 ARI DCw ;0 SET RETURN
06860 CNVCHK SB R CNVRTR+4 SET RETURN
06870 MRSD 0-9+X2,ARI SET
06880 SST 0-18+X2,ARI,02 INDICATOR
06890 BCE (CNVRTR+1),ARI,01 IF ALL INTEGER RETURN
06900 BCE (CNVRTR+1),ARI,02 IF ALL FLING PT RETURN
06910 BCE CNVTOP,0-9+X2,01 DO
06920 MLwD =4C00000027,0-10+X2 NECESSARY
06930 B (CNVRTR+1) CONVERSION
06940 CNVTOP MLwD =4C00000027,0-1+X2
06950 CNVRTR B *
06960* **************************************************
06970* OR REPLACE TOP TWO ELEMENTS WITH LOGICAL SUM
06980* AND REPLACE TOP TWO ELEMENTS WITH LOGICAL PRODUCT
06990* NOT-LOGICAL NEGATION OF TOP ELEMENT
07000* **************************************************
07010*
07020* **************************************************
07030 OR BS =1B9,IR2
0704C BCE BSTT,1+X2,T
ALREADY PROPERLY TRUE OR FALSE
0705C B FETCH
0706C AND BS =IR9,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* INDXA-Compute address according to dope vector (address
0718C* INSTACK) and indices on the stack. Resulting address
0719C* ON TOP OF THE Stack.
0720C* INDXR-SAME AS INDXA EXCEPT THAT ADDRESSED ITEM IS LOADED
0721C* TO THE STACK.
0722C***************************************************************************
0723C***************************************************************************
0724C***************************************************************************
0725C INDXA EQU *
0726C INDXR EQU *
0727C BS IR13
CLEAR
0728C MRSD 0+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 =IR9,IR2
0735C BS =IR1,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 MRSD 0+X3,IR13
SUBSCRIPT COUNT
07410 SAR IR3
07420 MRI 1*X2,IR15-3 FIRST SUBSCRIPT
07430 MORIDX BS =1B1,IR13 REDUCE SUBSCRIPT COUNT
07440 BCE SUBDNE,IR13,00 IF NO MORE THEN FINISH
07450 BA =1B4,IR10 POINT TO NEXT MULTIPLIER
07460 BA =1B9,IR2 NEXT SUBSCRIPT
07470 BIM 3*X10,4+X2 SUBSCRIPT * MULTIPLIER
07480 BA 4+X2,IR15 ACCUMULATED 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 MLw IR15,0-5+X2 ADDRESS TO STACK
07560 BCE FETCH,0-2+X3,67 IF INDXA THEN DONE
07570 MRIDR 0+X15,0-8+X2 ELSE LOAD RESULT
07580 B FETCH
07590***********************************************************************
07600 CAT -CONCATENATE TWO STRINGS WHOSE ADDRESSES ARE IN THE TOP
07610 CAT - TWO POSITIONS OF THE STACK. THE RESULTING STRING IS PLACED
07620 IN WORKING STRING STORAGE AND THE TWO ADDRESSES ARE REPLACED
07630 BY THE ADDRESS IN WORKING STORAGE.
07640***********************************************************************
07650***********************************************************************
07660 CAT MCw 0-14+X2,IR13 ADDRESS TO REGISTER
07670 BCE CTSUBL,0-18+X2,34 CHECK FOR SUBSTRING
07680 MRIDR 0+X13,(ENDAOR-3) CHECK FOR SUBSTRING
07690 SBR IR12
07700 CI 0-1*X12 CLEAR ITEM MARK
07710 CAT2 MCw 0-5+X2,IR13 CHECK FOR SUBSTRING
07720 BCE CTSUB2,0-9+X2,34 CHECK FOR SUBSTRING
07730 MRIDR 0+X13,0+X12 NEXT LOCATION IN WORKING STORAGE
07740 SBR IR13 CLEAR POSSIBLE WORD MARK
07750 CAT3 CW 0+X12 ADDRESS OF RESULTING STRING
07760 MCw ENDA,0-14+X2
NEXT LOCATION IN WORKING STORAGE

POP STACK

MOVE

SUBSTRING

TO

WORKING

STORAGE

REMOVE SUB STRING MARK

CAT2

CATENATE

SUBSTRING

TO

WORKING

NEXT LOCATION IN WORKING STORAGE

NEXT LOCATION IN WORKING STORAGE

SUBSTR - PL/I SUBSTRING OPERATION

COMPUTE

STARTING

POINT OF STRING

IF NO LENGTH GIVEN

COMPUTE IT

ELSE STORE LENGTH

MARK AS SUBSTRING ELEMENT

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

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

FMT MRID 0+X3,FMTCD1-3 SET FORMAT CODE ADDRESS
SAR IR3 BUMP SEQUENCE COUNTER
BA =1B1,FMTCD1 JUMP OVER BEGINNING MARKER
MCW FMTCD1,FMTCD2 SAVE FOR REPETITION
SST 0+X3,INOROT,70 INPUT OR OUTPUT CODE
SST 0+X3,DEVTYP,07 STRING OR UNIT RECORD
BA =1B1,IR3 BUMP SEQUENCE COUNTER
BCE STRSET,DEVTYP,00 IF STRING INITIALIZE
BCE FETCH,INOROT,70 ELSE INITIALIZE FOR UNIT RECORD
MCW *PRNTBF,IR8 OUTPUT PRINT BUFFER
MCW *PRNTBF+132,STREND
BA FETCH
FMTCD1 DCW =4
FMTCD2 DCW =4
INOROT DCW :0:
DEVTYP DCW :0:
STRADD DCW =4
LNGSTR DCW =4B0
STREND DCW =4
STRSET MRID 0-8+X2,STRADD-3 STRING ADDRESS
MRID 0-4+X2,LNG5TR-1 LENGTH
MCW STRADD,STREND STRING ADDRESS
BA LNGSTR,STREND STRING END ADDR (PLUS ONE)
08520  BS   =1B9,IR2    POP STACK
08530  BCE FETCH,INOROT,70  IF INPUT STRING ALL DONE
08540  MCW STRADD,IR8    ELSE SET OUTPUT POINTER
08550  MCW STRADD,IR9
08560  CLRMOR MRSBR BLNK,0+X9  CLEAR OUTPUT
08570  SBR  IR9    STRING
08580  C   IR9,STREN 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 PUNCTUATION
08660*  AND RESET BUFFER POINTER TO PRINTER BUFFER.
08670*  *
08680*  *
08690***************************************************************************

08700  PUT BCE STRPUT,DEVTYP,00
08710P  :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 FMTRST,(FMTCD1-3),77  CHECK IF RESET IS REQUIRED
08880  BBE DATFMT,(FMTCD1-3),70  CHECK FOR DATA FORMAT
ELSE IT MUST BE CONTROL FORMAT

SPACE THE REQUIRED NUMBER OF COLUMNS

SET THE BUFFER

THE PROPER CHARACTER

ONE ORIGIN INDEX FOR BUFFER POINTER

IF STRING THEN ERROR

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

ERRLOC:

BUFF:**** BUFFER OVERFLOW****:

BUFOFL:

PRINT, BFOFL:

B: ERRLOC:

A: BFOFL DCW: ****BUFFER OVERFLOW****:

DCW =1C45

IRFMA =1B1 FORM

MR ID (FMTCD1-3), IR10-3

SAR FMTCD1

IR8, IR10

C IR10, STREND

BUF0FL

C

CER

BUFOFL

CNVFLD

CONVERT

0-5+X2

MARK FOR MOVE

IFMNEG, 0-8+X2, 40

CHECK FOR MINUS SIGN

IFMCNV TMA CNVFLD, 00

INTEGER

IFMNEG 0-8+X2, 40

ZERO SUPPRESSION SYMBOL

CNVFLD, 00

TO DECIMAL

MCW 0-5+X2, CNVFLD-2

BINARY

MCE CNVFLD, 0-1+X10

MOVE AND EDIT DECIMAL INTEGER

MCW IR10, IR8

NEXT BUFFER LOCATION

BS =1B9, IR2

POP STACK

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN, 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT

BBE IFMSGN 1+X2, 40

CHECK FOR NEGATIVE SIGN PLACEMENT
10000 C IR8,STREND
10010 BL BUFOFL
10020 BCE FOUT:0-8*X2,F
10030 MCw :T:0-1*8
10040 B FETCH
10050 FOUT MCw :F:0-1*X8
10060 B FETCH
10070 EFORM BA =1B1,FMTCD1
10080 MRID (FMTCD1-3),IR10-3
10090 SAR FMTCD1
10100 C IR10,STREND
10110 BL BUFOFL
10120 TMA 0=1*X2,00
10130 BS =1B9,IR2
10140 B FB/FD
10150 R DSA DECRES
10160 MLWD IR8,IR5
10170 BBE ROUND,DECRES-3,20
10180 SST :0:DECRES-3,60
10190 MCw :0:*X5
10200 ROUND LCA :0:DECRES-14
10210 Cw DECRES-13
10220 A 5:*DECRES-3
10230 BCE MARK,DECRES-14,00
10240 MCw DECRES-4,DECRES-3
10250 A 1:*DECRES
10260 MARK Sw DECRES-13
10270 SI 19*X5
10280 BSN EFRM,DECRES
10290 SST :0:DECRES-60
10300 C DECRES:009:
10310 BL EFRM
10320 MCw DECRES,IR1
10330 BA IR1,IR5
10340 Sf DECRES-4
10350 BCE MVPT,DECRES,00
10360 MCw DECRES-14+X1,0+X5
10370  MVPT  MCW  ::1+X5  MOVE IN .
10380  MRID  DECRES-13+X1.2+X5  MOVE DIGITS TO THE RIGHT OF .
10390  SBR  IR5  
10400  NZTST  BBE  NXTFLD,0-1+X5,77  REPLACE
10410  MCW  ::0+X5  ORDER ZEROES
10420  SBR  IR5  WITH ELNKS
10430  B  NZTST  
10440  EFRM  MCW  ::1+X5  MOVE IN DECIMAL PT.
10450  MCW  DECRES-4,11+X5  MOVE TEN FRACTION DIGITS
10460  MCW  E::12+X5  MOVE EXPONENT SYMBOL
10470  MCW  ::::13+X5  EXPONENT
10480  BBE  LDIGIT·DECRES,40  SIGN
10490  MCW  ::::13+X5  LOGIC
10500  LDIGIT  SST  DECRES,16+X5,17  LOW DIGIT BUT NO SIGN
10510  MCW  MOVE TWO HIGH DIGITS OF EXP
10520  B  NXTFLD  
10530  DCW  :0:  
10540  DCW  =11
10550  DECRES  DCW  =3
10560  NXTFLD  MRIN  0+X5,0
10570  SAR  IR5  
10580  CI  0-1+X5  REMOVE ITEM MARK
10590  BA  IR10,IRB  NEXT OUTPUT BUFFER LOCATION
10600  B  FETCH  

*****************************************************************************
10620*  GET - PROCESS INPUT EITHER FROM CARD BUFFER OR STRING
10630*   ACCORDING TO INFORMATION SET BY FMT.
10640*   *
10650*

*****************************************************************************
10660 GET  BCE  GETFM,DEVTYP,00  IF STRING INPUT BYPASS EOF TEST
10670 BCE  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
10740 B FMTE RR IF NONE OF THE ABOVE THEN ERROR
10750 IFMRST MCW FMTCD2,FMTCD1 RESET FORMAT POINTER
10760 B GET
10770 NXTIFM LA =IB1,FMTCD1 NEXT FORMAT CODE
10780 B GET
10790 ISPCE BA =IB4,FMTCD1 POINT TO SPACE COUNT
10800 MCW (FMTCD1-3),IR9 MOVE IT TO INDEX REG.
10810 SPCCNT C IR9,:000: TEST FINISH
10820 BE NXTIFM
10830 BCE STRSPC,DEVTyp,00 TEST FOR STRING INPUT
10840 EA =IB1,IR6
10850 C +INPUT+79,IR6 TEST FOR
10860 BH GETIPT NEW RECORD REQUIRED
10870 BS =IB1,IR9 DECREMENT SPACE COUNT
10880 B SPCCNT
10890 STRSPC BA IR9,STRADD SPACE
10900 C STREND,STRADD AND
10910 BEH BUFOFL TEST FOR STRING OVER FLOW
10920 B NXTIFM
10930 ICOL BCE STRCOL,DEVTyp,00 TEST FOR STRING
10940 MCW +INPUT,IR6
10950 BA =IB4,FMTCD1 POINT TO COL INDICATOR
10960 BA (FMTCD1-3),IR6
10970 BS =IB1,IR6 ONE ORIGIN INDEX FOR COLUMN POINTER
10980 C +INPUT+79,IR6
10990 BH BUFOFL CHECK FOR ERROR
11000 B NXTIFM
11010 STRCOL MCW STREND,STRADD ESTABLISH
11020 BS LNGSTR,STRADD ORIGINAL ADDRESS
11030 BA =IB5,FMTCD1 POINT TO COL INDICATOR
11040 BA (FMTCD1-3),STRADD
11050 BS =IB1,STRADD ONE ORIGIN COL INDICATOR
11060 C STREND,STRADD TEST
11070 BEH BUFOFL FOR ERROR
11080 B NXTIFM
11090 ISKP BCE FMTE RR,DEVTyp,00 IF STRING INPUT THEN ERROR
11100 BA =IB1,FMTCD1
SAR FMTCD1

ISKPTS BCE GET,IR10,00

B GETIPT

BS =IB1,IR10

B ISKPTS

GETIPT SBR GETRTN+4

BCE ENDFMS,ENDF,T

:GET READ,

MCw +INPUT,IR6

C INPUT+3:1EOF:

BNE GETRTN

MCw :T:;ENDF

GETRTN B *

ENDF DCw :F:

ENDFMS EQU *

:PUT PRINT,EOFMES,

B ERRLOC

EOFMSEDCW :A ****END OF FILE ON INPUT UNIT****:

DCw =1C45

IDTFMT EQU *

BCE AFRMI,(FMTCD1-3),10

BCE EFRMI,(FMTCD1-3),11

BCE IFRMI,(FMTCD1-3),12

BCE LFRTMI,(FMTCD1-3),13

B FMTERR

BA =IB1,FMTCD1

MRID (FMTCD1-3),IR10-3

SAR FMTCD1

BBE AOK*0-9+X2.04

B CNVERR

AOK SW 0-4+X2

C IR10,0-5+X2

BL FMTERR

MRID 0-8+X2,IR9-3

BA IR10,IR9

AMORE SW 0+X6
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
I OK
IMORE
LENDT
IDECMV

DFLD

I SON
5T1SGN
I5TRNG
ISTRUP

BS
IR9
MCw : : : I5GN
BS CNVFLD
BCE ISTRNG, DEVTYP, 00
BCE ISTRNG, 0 + X6, -
BCE ISTRNG, 0 + X6, +
SST 0 + X6, DFLD + X9, 17
BA =1B1, IR9
IENDT BA =1B1, IR6
C IR6, INPEND
BL GETIPT
C IR10, IR9
BL IMORE
IDECMV MCw DFLD - 1 + X9, CNVFLD
SST ISGN, CNVFLD, 60
DTB CNVFLD, 00
TAM CNVFLD, 00
MRID 0 - 8 + X2, IR9 - 3
SI CNVFLD = 2
MRID CNVFLD - 5, 0 + X9
CI CNVFLD - 2
B POPUP
Dfld DCw =30
ISGN DCw : :
STISGN MRSU 0 + X6 + ISGN
BS =1B1, IR10
E IENDT
ISTRNG SST (STRADD - 3) + DFLD + X9, 17
BCE ISTSGN,(STRADD - 3),+
BCE ISTSGN,(STRADD - 3),-
ISTRUP EA =1B1, IR9
BA =1B1, STRADD
C IR10, IR9
EH IDECMV
C STREND, STRADD
BEH BUF0FL
B ISTRNG
ISTSGN MRSD (STRADD-3)*ISGN
ISTRUP
EFRMI BA =1B1,FMTCDI
JUMPOVER CODE TYPE
MRID (FMTCDI-3),IR10-3 FIELD LENGTH
SAR FMTCD1 NEXT FORMAT CODE LOCATION
BCE EOK*0-9*X2,02 CHECK FOR TYPE MISMATCH
CNVERR
EOK BS DEXP INITIALIZE DECIMAL EXPONENT
IR9
BCE ESTRNG,DEVTYP,00 CHECK FOR STRING INPUT
IR11
EMORE MRSD 0*X6*EFHLD*X9
IR6
SAR IR9
C IR6,INPEND CHECK FOR
GETIPT NEW RECORD REQUIREMENT
BL IR10,IR9 DETERMINE
EMORE FINISH
IR11
ECNV GO CONVERT
EFHLD DFLD
ESTRNG STRADD,IR9
IR10,IR9
IR9,STREND
BUFDFL
MCW 0-1*X9*EFHLD=1*X10
IR10,STRADD NEXT STRING INPUT
BA
ECNV EQU *
BFRACT+10 CLEAR DECIMAL MANTISSA
DFRACF+10 CLEAR DECIMAL EXPONENT FIELD
MCW ++:FSGN CLEAR CHAR COUNTER
MCW ++:ESGN
MCW 0-1+X9,EFHLD+1+X10
MCW
ECKBLK BCE ECNT1,EFHLD*X9,15 CHECK FOR BLANKS
IR10,STRADD
BCE ECNT1,EFHLD*X9,15 CLEAR MANTISSA COUNTER
BCE FSGNST,EFHLD*X9,, CHECK
BCE FSGNST,EFHLD*X9,, SIGNS
BCE FRACT,EFHLD*X9,, CHECK BEGINNING OF FRACTION
DECIMAL FRACTION POINTER
INCREMENT EXPONENT (DECIMAL)
INPUT CHAR COUNTER
IF MORE
THEN GO CHECK IT
ELSE CONVERT FD/FB
INPUT CHAR COUNTER
IF NO MORE
THEN CONVERT FD/FB
ELSE CHECK EXPONENT
IF BLANK THEN LOOK FOR EXPONENT
ELSE MOVE DIGIT
GO LOOK FOR MORE
BUMP CHAR POINTER
THEN CONVERT FD/FB
THEN CONVER FD/FB
DEFAULT SIGN
CHECK FOR
EXPONENT SIGN
GET EXPONENT
SET SIGN
SET SIGN
SET FRACTION SIGN FOR CONVERSION
ACCUMULATE DEC EXP FOR CONVERSION
12960 B FD/FB CALL CONVERSION ROUTINE
12970 R DSA DEXP ADDRESS OF RIGHT END OF 14 CHAR
12980 B FD/FB FLOATING DECIMAL FIELD
12990 R B CNVERR ERROR INSTRUCTION
13000 MRID 0-8×X2, IR10-3 ADDRESS OF RESULT
13010 MLWDI =880×0-1+X2 CLEAR ANY EXTRANEOUS PUNCTUATION
13020 TAM 7×X10,00 STORE RESULT
13030 SI 7×X10 ITEM MARK RIGHT
13040 B POPUP POP STACK AND FETCH NEXT INSTR
13050 DFRAC TDC w =11
13060 DEXP DC w =3
13070 DECEXP DC w =3
13080 ESGN DC w =1
13090 FSGN DC w =1
13100 LFRMI BA =1B1,FMTCD1 JUMP OVER CODE TYPE
13110 MR (FMTCD1-3), IR10-3 FIELD LENGTH
13120 SAR FMTCD1 NEXT FORMAT CODE LOCATION
13130 BCE LOK, 0-9×X2, 03 CHECK FOR TYPE MISMATCH
13140 B CNVERR
13150 LOK BS IR9
13160 BCE LSTR, DREVTyp, 00 CHECK FOR STRING INPUT
13170 LRECP FA IR10, IR6
13180 C IR6×INPUT+80 CHECK
13190 BL LNwREC NEW RECORD REQUIREMENT
13200 MRID 0-8×X2, IR9-3 MOVE ADDRESS
13210 MRSDR 0-1×X6, 0×X9
13220 BE GETIPT
13230 B POPUP
13240 LNwREC BS +INPUT+80, IR6 DETERMINE EXCESS CHAR
13250 MCw IR6×IR10 NEW LENGTH
13260 B GETIPT
13270 B LRECP
13280 LSTR MCw STRADD, IR9
13290 BA IR10, IR9
13300 C IR9×STREND
13310 BL BUF0FL CHECK OVERFLOW
13320 MRID 0-8×X2, IR10-3 ADDRESS OF RECEIVING FIELD
MOVE INPUT
NEXT STRING INPUT
POP STACK AND FETCH
STOP - PRINT INSTRUCTION COUNT MESSAGE AND EXIT.

STOP BS CNVFLD
MCW INSTCT,CNVFLD-2 BINARY COUNT TO CONVERSION FIELD
TMA CNVFLD,00 CONVERT IT
BTD CNVFLD,00 TO DECIMAL
LCA EWORD,PRTCNT EDIT WORD
MCE CNVFLD,PRTCNT MOVE IT TO PRINT FIELD (EDITED)

:PUT PRINT,CNTMES,
B (164) EXIT

CMTMESDCW :A ****INSTRUCTION COUNT = :
PRTCNT DC =9
DC :****:
L DCw =1C45

ERROR MESSAGES

ERROR EQU *
:PUT PRINT,BADOP,
:ERRLOC
BADOP DCw :A ****ILLEGAL OP CODE***:
L DCw =1C45
DYNOFL EQU *
:PUT PRINT,STROFL,
:ERRLOC
STROFLDCW :A ****DYNAMIC STORAGE EXHAUSTED****:
L DCw =1C45
CNVERR EQU *
: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  FERME5DCW :A  ****FLTNG PT OVFLW OR ZERO DIVIDE****:
13770  L   DCW  =1C45
13780  IDXERR EQU *
13790  L   :PUT PRINT,IDXMES,
13800  B   ERRLOC
13810  IDXMESDCW :A  ****INDEXED ADDRESS BEYOND DYNAMIC STORAGE****:
13820  L   DCW  =1C45
13830  TYPERR EQU *
13840  L   :PUT PRINT,TYPMES,
13850  B   ERRLOC
13860  TYPMESDCW :A  ****INCONSISTENT DATA TYPE ERROR****:
13870  L   DCW  =1C45
13880  FMTERR EQU *
13890  L   :PUT PRINT,FMTMES,
13900  B   ERRLOC
13910  FMTMESDCW :A  ****FORMAT CODE ERROR****:
13920  L   DCW  =1C45
13930  ERRLOC BS LUDLOC,IR3   FIND RELATIVE LOCATION
13940  MCW IR3*CNVFLD-2
13950  TMA CNVFLD*00   CONVERT
13960  BTD CNVFLD*00   TO
13970  LCA EWORD,ELOC   DECIMAL
13980  MCE CNVFLD,ELOC   FOR PRINTING
13990  L   :PUT PRINT,LOCMES,
14000  BCT HALT,01   IF SENSE SWITCH ONE THEN DUMP REQUEST
14010  B   STOP   ELSE EXIT
14020  HALT H   DUMP REQUEST
14030  EWORD DCw  : , 0  :
14040  LOCMESDCW :A  ****ERROR AT RELATIVE LOCATION :
14050  ELOC DC  =9
14060  DC  :****:

14070 L     DCw  =1C45
14080 CNVFLD DCw  =1180
14090 LITORG*  
14100 ENDPRG EQU * 
14110 END   START

SYMBOL DEFINITION - CARD REFERENCE INDEX

ADCQV  01640 
ADD  06290 
AFIN  11560 
AFORM  09390 
AFRMA  09450 
AFRMI  11560 
AGK  06850 
ALCRTL  02600 
ALCR2  02630 
ALLOC  02490 
ALN1  11410 ;
AND  07060  
AOK  11420  
ARI  06850  
ASPLIT  11600  
ASTRNG  03710 ;
BADOP  13610  
BFOFL  09660  
BGSTR  09050  
BLNK  06160  
BSTF  07130 ;
BSTT  07100  
BUFOFL  09630  
CAT2  07720  
CAT3  07760  
CAT  07670 ;
CFDFB  12930  
CH<2ND  05150  
CHKBFF  09450  
CHKTYP  03180  
CHRSTR  03710 ;
CHRTPN  03750  
CLRMUNR  08560  
CMPADD  07380  
CNDTBL  06150  
CNDTST  05680 ;
CNTMES  13490  
CNVCHK  06860  
CNVERR  13680  
CNVFLD  14080  
CNVRTN  03410 ;
CNVRTR  06950  
CNVTUP  06940  
COLMN  08990  
COLSET  09010  
COMPC  05400  
CONPRT  02860  
CSBSTR  03930  
CTSUBL  07810  
CTSUB2  07920 ;
CURLEV  04550  
CVEMS  13710  
DATFMT  09330  
DECEXP  13070  
DECRE5  10550 ;
DEVTYP  08440  
DEXP  13060  
DFLD  12070  
DFRAC  13050  
DISPLAY  04490 ;
DYNAM  02280  
DYNEND  00460  
DYNFL  13630  
DYNSTR  00450  
ECKBLK  12550 ;
ECNT1  12630  
ECNV  12480  
EDIT  08870  
EFHLD  12410  
EFORM  10070  
EFRMI  10440  
EFRMI  12240  
ELOC  14050  
EMORE  12330  
ENDADR  06020 ;
ENDF  11250  
ENDFMS  11260  
ENDPRG  14100  
ENTPRO  04220  
EOFMES  11290 ;
EOK  12290  
ERRLOC  13930  
ERROR  13580  
ESGN  13080  
ESGNST  12870 ;
ESTRNG  12420  
EWORD  14030  
EXPNUM  12890  
EXP  12770  
EXP  12820 ;
FERMES  13760  
FERR  13730  
FETCH  01170  
FIXFLT  03640  
FLAG  04130 ;
FLGCHK  04310  
FMT  08290  
FMTCD1  08410  
FMTCD2  08420  
FMTERP  13880 ;
FMTMES  13910  
FMSTR  09130  
FNDLNG  08140  
FOUT  10050  
FRACT  12690 ;
FSGN  13090  
FSTGNT  12670  
GET  10670  
GETFM  10690  
GETIPT  11170 ;
GETRTN  11240  
GTALN  09560  
GTSBLN  09610  
HALT  14020  
HOLD  04690 ;
ICOL  10930  
IDECMV  11980  
IDTFMT  11310  
IDXERR  13780  
IDXMES  13810 ;
IENDT  11930  
IFIX  00780  
IFMCNV  09780  
IFMNEG  09870  
IFMRST  10750 ;
***INSTRUCTION COUNT = 296,018***
IDENTIFICATION:

PROGRAM-ID: MTXMCP03.

AUTHOR: J. R. VAN DOREN.

SOURCE LANGUAGE: EASYCODER.

SOURCE COMPUTER: H-1200

OBJECT COMPUTER: H-1200

PURPOSE:

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

INDEX REGISTER LOCATION DEFINITIONS

IR1 EQU 4
IR2 EQU 8
IR3 EQU 12
IR4 EQU 16
IR5 EQU 20
IR6 EQU 24
IR7 EQU 28
IR13 EQU 52
IR14 EQU 56
00380 IR15 EQU 60
00390*---------------------------------------------------------------*
00400* OCTAL ADDRESS DEFINITIONS OF PERTINENT SYMBOLS IN THE RESIDENT *
00410* INPUT/OUTPUT ROUTINE*                                      *
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* 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 OBJECT PROGRAMS)
00620* SYMF2 EQU 235 CONTAINS SYMBOL TABLE LIMIT ADDRESS
00640* (LIMIT OF DYNAMIC STORAGE FOR PLEX OBJECT PROGRAMS)
00650* CMPLCD EQU 236 COMPLETION CODE FIELD SET BY COMPILERS
00670 DSKLOD EQU 237 DISK LOADING OPTION FIELD
00680 EXCPPG EQU 238 GO OPTION FIELD
00690 PSTILST EQU 239 POST LISTING OPTION FIELD
00700 PRNGME 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*---------------------------------------------------------------*
<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00750</td>
<td>MTXPRG DCW</td>
<td>LOCATION FOR RESIDENT METAX PROGRAM</td>
</tr>
<tr>
<td>00760</td>
<td>MTXCRA DCW</td>
<td>LOCATION FOR RESIDENT CONTROL RECORD ANALYZ</td>
</tr>
<tr>
<td>00770</td>
<td>START CAM</td>
<td>SET FOUR CHAR ADDRESSING MODE</td>
</tr>
<tr>
<td>00780</td>
<td>GET READ,</td>
<td>GET NAMES OF RESIDENT METAX PROGRAM</td>
</tr>
<tr>
<td>00790</td>
<td>MCW INPUT+15,SAVNME</td>
<td>SAVE FOR LATER USE</td>
</tr>
<tr>
<td>00800</td>
<td>Sw SAVNME-7</td>
<td>SET LOCATION FOR CONTROL RECORD ANALYZPRZ</td>
</tr>
<tr>
<td>00810</td>
<td>LCA MTXCRALODFLD</td>
<td>NAME TO METAX COMMUNICATIONS FIELD</td>
</tr>
<tr>
<td>00820</td>
<td>Sw INTNME-7</td>
<td>SEGMENT AND</td>
</tr>
<tr>
<td>00830</td>
<td>Sw PRGNAME-7</td>
<td>NAME OF DISK TO MEMORY LOAD PROGRAM</td>
</tr>
<tr>
<td>00840</td>
<td>MCW INPUT+7,PPGNME</td>
<td>SET UP RETURN FOR RETURN START</td>
</tr>
<tr>
<td>00850</td>
<td>MCW LDRSEG+75</td>
<td>FETCH AND EXECUTE</td>
</tr>
<tr>
<td>00860</td>
<td>MCW +RTNST+167</td>
<td>SET UP RETURN START</td>
</tr>
<tr>
<td>00870</td>
<td>B (168)</td>
<td>EXIT POINT</td>
</tr>
<tr>
<td>00880</td>
<td>RTNST SBR</td>
<td>RETURN TO LOADER</td>
</tr>
<tr>
<td>00890</td>
<td>LDEXIT MCW</td>
<td>SET NAME OF RESIDENT METAX PROGRAM</td>
</tr>
<tr>
<td>00900</td>
<td>LCA =4B0,INSTCT</td>
<td>SET LOCATION FOR RESIDENT METAX PROGRAM</td>
</tr>
<tr>
<td>00910</td>
<td>B (LDRTN+1)</td>
<td>EXIT POINT</td>
</tr>
<tr>
<td>00920</td>
<td>LDR TN B</td>
<td>FETCH RESIDENT METAX PROGRAM</td>
</tr>
<tr>
<td>00930</td>
<td>CRACLL LCA</td>
<td>ZERO INSTRUCTION COUNT</td>
</tr>
<tr>
<td>00940</td>
<td>LCA =4B0,INSTCT</td>
<td>COMMUNICATIONS</td>
</tr>
<tr>
<td>00950</td>
<td>LCA +ENDPRG+1,STCKF1</td>
<td>FIELDS FOR EXECUTING</td>
</tr>
<tr>
<td>00960</td>
<td>LCA +ENDPRG+999,STCKF2</td>
<td>CONTROL RECORD ANALYZER</td>
</tr>
<tr>
<td>00970</td>
<td>MCW MTXCRALODFLD</td>
<td>INTERPRETER SEGMENT AND NAME</td>
</tr>
<tr>
<td>01000</td>
<td>BS PRGNAME</td>
<td>FETCH AND EXECUTE</td>
</tr>
<tr>
<td>01010</td>
<td>BS INTNME</td>
<td>ZERO INSTRUCTION COUNT</td>
</tr>
<tr>
<td>01020</td>
<td>MCW INTSEG+75</td>
<td>INITIALIZE COMPLETION CODE</td>
</tr>
<tr>
<td>01030</td>
<td>MCW +INTRT1+167</td>
<td>SET UP FOR MEMORY CLEAR</td>
</tr>
<tr>
<td>01040</td>
<td>B (168)</td>
<td>FETCH AND EXECUTE</td>
</tr>
<tr>
<td>01050</td>
<td>INTRT1 LCA</td>
<td>MEMORY</td>
</tr>
<tr>
<td>01060</td>
<td>=4B0,INSTCT</td>
<td>CLEAR</td>
</tr>
<tr>
<td>01070</td>
<td>MCW :0:*CMPLCD</td>
<td>MEMORY</td>
</tr>
<tr>
<td>01080</td>
<td>LCA =3C117776,IR15</td>
<td>SET UP FOR MEMORY CLEAR</td>
</tr>
<tr>
<td>01090</td>
<td>MRSGR =4B0,ENDPPG+1</td>
<td>MEMORY</td>
</tr>
<tr>
<td>01100</td>
<td>SBR IR14</td>
<td>CLEAR</td>
</tr>
<tr>
<td>01110</td>
<td>SI 0+X15</td>
<td>CLEAR</td>
</tr>
</tbody>
</table>
01120  Sw  0+X15
01130  B   CLEAR
01140  MCw  LODFLD,LODSAV
01150  C   PRGNME,SAVNME
01160  BNE MFETCH
01170  MCw  MTPRGR,LODFLD
01180  GETINT MCw  INTTREE,75
01190  MCw  
01200  MCw  +INTRT2,167
01210  B   (168)
01220  INTRT2 BCE  FATAL,CMPLCD,F
01230  BCE  LTODSK,DSKLD,Y
01240  BCE  EOFTST,EXCPG,N
01250  GO   EQU *
01260  MCw  LODSAV+LODFLD
01270  SI  1+X5
01280  Sw  iw+X5
01290  MLwD  GENFLD,IR15
01300  MLwD  LODFLD+IR14
01310  MORPRG MRwDR 0+X15,0+X14
01320  SAR  IR15
01330  SBR  IR14
01340  MRLD  0+X15,0+X14
01350  SAR  IR15
01360  SBR  IR14
01370  BA  LODFLD,0-1+X14
01380  CW  0-3+X14
01390  CW  0-1+X14
01400  BCE  BLKCNT,0-4+X14,00
01410  ENDTST C   IR15,IR5
01420  BH  MORPRG
01430  MLwD  =3C117777,IR15
01440  MRSRR  =4BO,0+X14
01450  SBR  IR14
01460  SI  0+X15
01470  B   CLEAR
01480  MCw  SYMF1,IR14

OPERATION

DO IT

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

COMPILLED PROGRAM LOCATION
LOADING LOCATION

MOVE REMAINDER OF ADDRESS
RELOCATE THE ADDRESSES

TEST POSSIBLE BLOCK PSEUDO OP CODE

PREPARE FOR CLEARING REMAINING MEMORY

CLEAR
SYMBOL (OR DYNAMIC STORAGE FOR PLFX)

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

CORE TO DISK UPDATE PROGRAM NAME
METAX SYSTEM LOADER PROGRAM NAME
MTXCRA INTERPRETER NAME

INITIAL METAX PROGRAM NAMES SAVE LOC
LODFLD SAVE LOCATION
MEMORY CLEAR SUBROUTINE

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

SYMBOL DEFINITION - CARD REFERENCE INDEX

BLKCNT 01570; CLEAR 01690; CLRRTN 01760; CMPLCD 00660; CRACLL 00960;
DSKLOAD 00670; ENDPRIA 02030; ENDTST 01410; EOFTST 01810; EXCPPG 00680;
EXIT 01770; FATAL 01850; FTLMES 01880; FTRTST 02010; FTPTST 01990;
GENFLD 00560; GETINT 01180; GO 01250; INPUT 00470; INSTRCT 00730;
INTNAME 00710; INTRT1 01060; INTRT2 01220; INTSEG 01660; IR13 00360;
**:INSTRUCTION COUNT = 47,601**
XIII. APPENDIX E

Some of the pertinent hardware and software characteristics of the host computer system, an H-1200, are presented below. Comments about machine dependent characteristics of the METAX system are also included.

Basically the host system is a variable word length two address computer. An eight bit character consisting of six data bits and two punctuation bits is the unit of addressable storage. Normally only the data bits participate directly in data manipulation operations, the punctuation bits being used to delimit the respective fields. Punctuation may participate in data moving instructions, however.

The two punctuation bits are referenced as a word mark and an item mark. For the most part the H-1200 instruction set expects delimiting word marks on the left of a data field with addresses being given on the right. A specific exception to the punctuation requirements occurs with the floating point hardware option in that floating point instructions do not utilize these bits in any way since all operands are fixed length. However, no boundary alignments are required which simplifies certain translation or interpreter factors.

Floating point instructions also represent a departure from the two address scheme in that floating point registers are used in a one address fashion.
The internal data representations for the respective pseudo-machines correspond to the host computer with the exceptions that addressing is always on the left and item marks are used as right end delimiters. The respective interpreters make the necessary adjustments for addressing and may insert word marks on the left during execution. However, word marks are never generated for data fields during translation or loading.

The only explicit use of word marks in object code is to mark the left hand character of an 18 bit address field as a relocatable address or pseudo-address. During loading by either the control program or MTXLD the 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 NETAX processors.

The addressing structure of the H-1200 is binary. Address modification may be effected with either indirect addressing or indexing. There are three addressing modes
based on the amount of storage to be addressed and the number of index registers to be used. The mode used in all the assembler programs in the METAX system is the four character or 24 bit mode which allows a 19 bit address and a five bit address modifier. The latter is used to specify one of fifteen index registers or indirect addressing.

The index registers are resident in main storage and are thus manipulated with standard storage-to-storage arithmetic and data moving instructions. Assembly control statements are used to equate the symbols IR1, IR2,...,IR15 to the proper addresses for purposes of symbolic reference. Thus

\[ BA = 1B1, IR13 \]

specifies that a one character binary constant of one is to be added (in binary) to index register 13.

The specification of indexing is exemplified by

\[ MCW TVEC+3+X7, IR14 \]

which specifies that the first operand is to be moved to index register 14. The address computation TVEC+3 is effected at translation time while the indexing via index register seven (specified by +x7) takes place at execution time.

There are two address registers, the A and B address registers, which are referenced frequently for updating index registers. Thus

\[ SAR IR1 \]
SBR IR10

specify that the A and B registers are to be stored in index registers one and ten, respectively. Such instructions are used frequently in the interpreters immediately following an extended move instruction as discussed below.

The B register may also be used for subroutine linkage.

Specific forms of the generic EXM (extended move) instruction are used extensively for data and punctuation moving and for scanning purposes. With this instruction one may establish three categories of options. The first is the direction of the move, left or right. This is important because the A and B registers will be set one position beyond the last character position processed for the first and second operands, respectively. On completion of an EXM instruction SAR and SBR may be used to store the contents of the address registers.

The second category is the terminating condition which may be a single character move or any one of three combinations of punctuation bits. The third category specifies which combination of data and/or punctuation bits to move, if any.

Then

MRIDI 0+X6,SYMBOL

specifies the data and item mark bits of the first operand.
are to be moved from left to right to SYMBOL with the move
terminated by the first item mark in the sending field while
MRIN 0×14,0×13
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 op­
erands. 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 respec­
tive 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 pub­
lifications (27,28) for more information on the assembler lan­
guage 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).