An experimental compiler-compiler system

James Robert Van Doren

Iowa State University

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

Part of the Computer Sciences Commons

Recommended Citation
VAN DOREN, James Robert, 1939-
AN EXPERIMENTAL COMPILER-COMPILER SYSTEM.

Iowa State University, Ph.D., 1971
Computer Science

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

by

James Robert Van Doren

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

Major Subject: Computer Science

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Of Science and Technology
Ames, Iowa

1971
PLEASE NOTE:

Some Pages have indistinct print. Filmed as received.

UNIVERSITY MICROFILMS
## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>II. REVIEW OF THE LITERATURE</td>
<td>4</td>
</tr>
<tr>
<td>III. THE METAX METALANGUAGES</td>
<td>14</td>
</tr>
<tr>
<td>A. Developmental Rationale</td>
<td>14</td>
</tr>
<tr>
<td>B. Bootstrapping</td>
<td>16</td>
</tr>
<tr>
<td>C. The METAX9 Metalanguage</td>
<td>18</td>
</tr>
<tr>
<td>1. Compiler definition commands</td>
<td>18</td>
</tr>
<tr>
<td>2. Elementary syntactical commands</td>
<td>20</td>
</tr>
<tr>
<td>3. Metasyntactic elements</td>
<td>26</td>
</tr>
<tr>
<td>4. Semantic commands</td>
<td>27</td>
</tr>
<tr>
<td>5. Output operators</td>
<td>35</td>
</tr>
<tr>
<td>6. Error processing</td>
<td>37</td>
</tr>
<tr>
<td>D. The METAX9 Pseudo-Machine</td>
<td>41</td>
</tr>
<tr>
<td>1. Primitive operations</td>
<td>41</td>
</tr>
<tr>
<td>2. Control stack</td>
<td>46</td>
</tr>
<tr>
<td>3. Symbol table structure</td>
<td>47</td>
</tr>
<tr>
<td>4. Addressing structure</td>
<td>51</td>
</tr>
<tr>
<td>5. Block list</td>
<td>51</td>
</tr>
<tr>
<td>6. General comments</td>
<td>52</td>
</tr>
<tr>
<td>E. METAX8 and METAX9 Pseudo-Machine Differences</td>
<td>52</td>
</tr>
<tr>
<td>F. Chronological Development</td>
<td>54</td>
</tr>
<tr>
<td>IV. THE METAX SYSTEM ORGANIZATION</td>
<td>57</td>
</tr>
<tr>
<td>A. Control Program</td>
<td>57</td>
</tr>
<tr>
<td>B. Control Record Analyzer</td>
<td>59</td>
</tr>
<tr>
<td>C. Communications Region</td>
<td>60</td>
</tr>
<tr>
<td>D. Main Storage Usage</td>
<td>62</td>
</tr>
<tr>
<td>V. PLEX: THE LANGUAGE AND ITS TRANSLATOR</td>
<td>64</td>
</tr>
<tr>
<td>A. The PLEX Language</td>
<td>64</td>
</tr>
</tbody>
</table>
I. INTRODUCTION

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Thus the representation of a subscript list may be written as

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

\(<\text{SUBLIST}>\) ::= \(<\text{SUBLIST}>\), \(<\text{EXP}>\) | \(<\text{EXP}>\);

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 ""\(\mid\)"" as a syntax specification which cannot be accomplished in BNF because such a symbol in unquoted form would represent a metasymbol and not a syntax specification of the language being described.

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

```
.SYNTAX PROGRAM
PROGRAM := ".SYNTAX" .ID $ ST ".END";
ST := .ID ":=" .ID EX1 ";";
EX1 := EX2 $( "\mid" 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 implementa-
tion of META5 at UCLA as well as some of the history of de-
velopment 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 operat-
ing 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 to be available from System De-
velopment Corporation which has begun pursuit of profit making goals.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

A. Developmental Rationale

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

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

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

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

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

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

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

B. Bootstrapping

The initial version of the compiler writing language or metalanguage was based to a degree on META II (54) except that symbol table facilities and revised code generation and internal label facilities were incorporated at the outset. This version, called METAX0, was first manually translated to a symbolic form of the code for the pseudo-machine or translation machine and then manually assembled into absolute octal form and punched into card form. The interpreter for the pseudo-code was then written, a simple card loader prepared and an elementary control program written to appropriately control the loading and execution sequence.
The hand compiled version of 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 metacompile time.

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

1. Compiler definition commands

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

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

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

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

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

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

Example:    PUSHLB .OEQU 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 syntactical commands below are the symbolic representations of the pseudo-machine operation codes which may possibly result in the object version of a procedure utilizing the specified command.

"XYZ" (TEST):

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

PQR (CLM):

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

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

.LATCH(...) (LATCH):

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

.EMPTY (SET):

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

**.ONUM (ONUM):**

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

**.INUM (INUM):**

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

**.FNUM (FNUM):**

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).

.Strings (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 sec-
ond argument. If equality results the true-false flag is set
ture, 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 charac-
ter 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(DYNAMB>DYNAMP)

.STKCHK (CHKSYM):

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

3. Metasyntactic elements

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

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

\[
\text{SUBLIST} := \text{EXP } \# (",\ " \text{EXP})
\]
as opposed to

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

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

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

\[
\text{ENDING} := \text{"END" (.ID/.EMPTY);}
\]

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

4. Semantic commands

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

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

Example: \ .\OUT(DYNAM,*1)

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

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

With .DO it is possible to specify code sequences which may not be generated automatically by translation of other commands. For example, the object code of either METAX9 or METAX8 may contain binary addition or binary multiplication operation codes but the source languages provide no means other than .DO for specifying these operations. Thus .DO(A,ONE,LEVNO) and .DO(M,LEVNO,DYNAMP) specify these respective operations.

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

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

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

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

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

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

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

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

.SET(...) (MOVE,MOV1):

This command is in effect an assignment command allowing the assignment of a metavariable or literal value to a metavariable. Anomalous behavior may result if the receiving field is shorter than the sending field.
Example: .SET(DYNAMP=DYNAMB)

.BLKENT (BLKENT):

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

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

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

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

.CAT(...) (CAT):

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

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

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

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

Example: .ENTERL(00,01)

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

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

Example: .ENTERA(07,DIMCNT)

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

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

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

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

.STKSYM (STKSYM):

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

.SCAN(...) (SCAN):

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

Example: .SCAN("*/")

.RETURN (R):

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

5. Output operators

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

*1 (LB1,EVAL):

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

*2 (LB2,EVAL):

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

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

Example:    **(04,01)

* (OUTSYM):

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

# (RESTORE):

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

Identifier, octal number, string (OUT):

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

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

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

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

6. Error processing

.CANCEL (CANCEL):

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

.ERRLATCH (MOVI):

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

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

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

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

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

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

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

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

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

D. The METAX9 Pseudo-Machine

1. Primitive operations

There are 47 operation codes or primitives available on the METAX9 pseudo-machine. Only selected primitives are reviewed here due to the pertinent discussions with the cor-
responding source language elements and also due to the ex-
tensive 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 ap-
    propriate number of elements. The error latch and instruc-
    tion counter are restored and the backup latch is reset.

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

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

SETP:
Set the true-false code false.

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

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

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

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

EXIT (EXITI):

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

RESOLVE (RESOLV):

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The main storage area for the control stack is delimited by the contents of two fields in the communications region, this being discussed in more detail in the next chapter. However, the amount of space available may vary during the operation of the META9 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 (MTXINTO4) 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 METAX8 interpreter, which is not presented here, the code generated for various pseudo-machines (BASIC,
METAX8, METAX9) and the symbol table structure is sufficiently simple that it is possible to give a reasonably meaningful postlisting without the interpreter having any knowledge of the code being compiled. It is dependent, however, on the type codes given for operation codes which may be generated (.TYPEG and .TYPEB) by the compiled code versus operation codes that may be executed (.TYPEB and .TYPEO), this being required because of the use of literal operands in code generation.

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

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

F. Chronological Development

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

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

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

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

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

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

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

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

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

A. Control Program

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

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

MTXMCP03 then processes an input record which determines the names of the control record analyzer and a METAX translator both of which are to remain resident in main storage. 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 in­
terpreter 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 pro­
gram 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.

SYMF1 (229-231)* and SYMF2 (233-235)*:
The beginning and ending locations for symbol table space (dynamic storage space) are kept in these two fields and utilized by the respective interpreters.

CMPLCD (236):
Translators are expected to set the completion code to record the status of a translation.

DSKLOD (237)*:
If a library update (STORE=YES) subsequent to the next translation is requested it is recorded here.

EXCPPG (238)*:
A request for execution (GO=YES) of the object program from the resulting translation is set in this field.

PSTLST (239)*:
A postlisting request (POSTLIST=YES), honored by the respective interpreter, may be recorded in this field.

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

PRGNME (283-290)*:

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

INTNME (291-298)*:

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

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

D. Main Storage Usage

Recall that a 56K memory segment is utilized by the METAX system. The memory segments listed below are given with inclusive storage locations given in decimal. An asterisk means the respective area is under control of fields set by MTXCRD 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 (6000-9999)
dynamic storage region.
METAX program execution region. *(10,000-32,767)
Code generation region. *(32,768-40,959)
Resident METAX program region (40,960-45,055)
Transient program region. (45,056-57,343)

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

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

A. The PLEX Language

Five data types may be declared for identifiers including FLOAT, FIXED, CHARACTER, LABEL and LOGICAL, the scope of the identifiers being determined by the block and/or procedure structure of the program. Binary precision of arithmetic variables is (35) and (23,0) for the FLOAT and FIXED attributes respectively, these not being adjustable by declaration but being implementation defined characteristics. All variables have a storage class attribute similar to the PL/I AUTOMATIC attribute or the standard ALGOL assumption. The LABEL attribute may be used only for label variables and not for resolving references to label constants as in XPL.
Character string variables by an undeclarable assumption have a PL/I-like VARYING attribute with a maximum length value required in the declaration of the string. Arrays may be declared with an implementation defined limit of 63 dimensions but with default or explicit lower bounds and explicit upper bounds all of which may be integer constants or variables but with no expressions, the amount of storage allocated for an array depending upon run-time evaluation of the array bounds.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

B. The PLEX Pseudo-Machine

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

Run-time storage administration is concerned with the proper management of available dynamic storage. With respect to the METAX system the space available to PLEX object programs for dynamic storage is delimited by the same communications area fields as used for specifying the symbol table space at compilation time. The scheme outlined here owes much to previous publications on ALGOL translators (13, 21, 40, 47), being rather similar, but not identical, to one outlined in detail by Gries (20). One consequence of the method is that an exit from a block requires no special action for releasing dynamic storage, thus implying that a jump outside the block (but within the containing procedure) requires no special handling. A negative consequence is that explicit source program controlled allocation and release of dynamic storage in a random manner is not possible. Thus a fuller 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 procedure 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 storage 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 information to be available to establish the display, the formal-actual parameter correspondence and the global display address 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 information 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 number 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 ENTPRO 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 instruction (immediately following the address) unless it is a literal string in which case the length is computed by scanning for a delimiting item mark on the right.

LD:

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

STO:

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

SST (save and store):

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

**JUMPA:**

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

**JUMP:**

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

**JUMPT:**

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

**JUMPF:**

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

**STCKC:**

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

**COMPC:**

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

SWAP:

Exchange the top two items on the stack.

POPUP:

Pop one element off the stack.

ADD, MULT, SUB, DIV, NEG:

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

FMT:

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

GET:

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

EDIT:

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

PUT:

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

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

OR, AND, NOT:

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

INDXR,INDXA:

The two indexing primitives expect the requisite number of arithmetic indexing values on the stack with the right most index on top. A single character literal value 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 significant dependence on the host machine structure for the implementation of variable length character strings.

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

C. The PLEX Translator

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

The discussion which follows next comprises additional comments about some, but not all, of the procedures in a sequential fashion including additional information about certain restrictions mentioned above. A careful preliminary or parallel perusal of 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 esthetically pleasing from a formal syntactical point of view but they may be rather effective.
As PLXCPL is presently written, an actual parameter such as \((I-1)\) must be enclosed in parentheses as shown. An example may be found in the recursive factorial computation in program TEST.

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

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

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

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

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

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

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

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

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

Certainly the translator contains errors of commission or omission as extensive testing on it has not been undertaken. The speed of the translator is relatively good, processing input at essentially card reader speed, 400 cards per minute, except perhaps in cases of multiple statements per card. It is true, however, that certain improvements in speed could be made primarily because an initial identifier in a statement may 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 METAX9 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 METAX 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 inves-
tigation into the implementation of additional PL/I-like fea-
tures within the basic framework presented suggests itself.

Even though it has been shown that a great deal of proc-
essing can be effected in one pass, one pass translators pre-
sent 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 in-
termediate 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 chang-
es 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 va-
rity as well as symbol table structures may be of consider-
able 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 HTXCRA, 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


38. McKeeman, W. M., Hornung, J. J. and Wortman, D. B. 
A Compiler Generator. Englewood Cliffs, N.J., 


1960.

42. Naur, P., ed. Revised report on the algorithmic 
language ALGOL 60. Communications of the ACM 6, 

43. O'Neil, J. T., Jr. META-PI: an on-line interactive 
compiler-compiler. Fall Joint Computer Conference 

44. O'Neil, J. T., Jr. An interactive language design 
Pennsylvania, Library, University of Pennsylvania. 
1970.

45. Oppenheim, D. K. and Haggerty, D. P. META5: a tool to 
manipulate strings of data. ACM National Conference 

46. Presser, L. The structure, specification and eval­
uation of translators and translator writing systems. 
Unpublished Ph.D. dissertation. Los Angeles, 
California, Library, University of California at Los 
Angeles. 1968.


48. Reynolds, J. C. An introduction to the COGENT 
programming system. ACM National Conference 

49. Ritland, Steven R. Implementation of a general purpose 
simulation language using a syntax directed translator. 
State University of Science and Technology. 1968.


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 META8
INTERPRETER=MTXINTO3
STORE=YES
GO=NO

//
**IDENTIFICATION:**

---

PROGRAM-ID: METAX9.

AUTHOR: J.R. VAN DOREN.

SOURCE LANGUAGE: METAX.

OBJECT LANGUAGE: METAX PSEUDO-MACHINE CODE.

OBJECT INTERPRETER: MTXINT03.


**PURPOSE:**

---

METAX9 IS A REVISION OF THE METAX8 METALANGUAGE AND ASSOCIATED COMPILER-COMPILER AND IS INTENDED FOR USE IN IMPLEMENTING THE PLEX COMPILER.

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

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

BEM .EQU 16 .TYPEO; MOVSYM .EQU 27 .TYPEO; DELETE .EQU 32 .TYPEO;
TYPST .EQU 34 .TYPEO; ENTYYP .EQU 35 .TYPEO; ENTFR .EQU 72 .TYPEO;
DECNUM .EQU 75 .TYPEO;
/* OPERATION CODES COMMON TO METAX8 AND METAX9 OBJECT PROGRAMS. */

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

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

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

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

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

MTX009 := .EMPTY PROGHD .ERR("\w: INVALID OR MISSING PROGRAM NAME",SET)
PRGbDY .DO(REVOLVE) ;

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

PRCGHD := "\d 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(*))

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

PRGURY :=$ ST "END" .ERR("F: UNRECOGNIZABLE STATEMENT") ;
ST := ID ( ":=" .DO(ENTLOC) MTXEXP .OUT(R) / .DO(SEARCH) )

/* 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

/* 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" COMPRISSE SEMANTIC CHECKING RATHER THAN PHRASE STRUCTURE SYNTAX. THESE CONSTRUCTS ARE INCLUDED HERE DUE TO CONDITIONAL JUMPING AND ERROR CODE GENERATION.

```
ELMSTX := .ID .OUT(CLA,**) / .STRING .OUT(TEST**) / "ID" .OUT(ID) / "STRING" .OUT(STRTST) / .ONUM" .OUT(INUM) / .INUM" .OUT(INUM) / .EMPTY" .OUT(SET)
```

OBSERVE THAT THE NEXT ALTERNATIVE PERMITS FACTORING OF METALANGUAGE EXPRESSIONS.

```
/ "(" MTXEXP ")" *ERR("mt: EXPECTED")" .SET)
```

THE NEXT ALTERNATIVE PERMITS ITERATIVE EXPRESSIONS.

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

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

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

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

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

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

OUT1 := "*1" OUT(LH1, EVAL, 04, 01) / "*2" OUT(LH2, EVAL, 04, 01)
    / "**" OUT(EVAL) ( "" ONUM OUT(*)
        ( "", ONUM OUT(*) / EMPTY OUT(00) ) "")
    / EMPTY OUT(05, 00) )
    / "*" OUT(OUTSYM) / "#" OUT(RESTORE)
    / ID OUT(OUT,\*) / ONUM OUT(OUT,\*)
    / STRING OUT(OUT,\*) ;

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

**COMPiled PROGRAM SIZE = 2653; METAX INSTRUCTION COUNT = 33,888****
SYMTAB SEARCH COUNT = 895; SYMTAB COMPARE COUNT = 143,068****
SYMBOL TABLE ENTRY COUNT = 346****
X. APPENDIX B
CONTROL RECORD
FUNCTION MTX009.
INTERPRETER=MTXINT03.
GO=NO.
START SYMTAB AT 5096, END SYMTAB AT 15096.
EXECUTE AT 15097.
POSTLIST=YES.
STORE=YES.

//
.PROG PLXCPL;

/***************************************************************************/
/* IDENTIFICATION: */
/* --------------------- */
/* PROGRAM-ID: PLXCPL. */
/* AUTHOR: J. R. VAN DOREN. */
/* SOURCE LANGUAGE: METAX9. */
/* OBJECT LANGUAGE: METAX9 PSEUDO-MACHINE CODE. */
/* OBJECT INTERPRETER: MTXINT04. */
/* */
/* PURPOSE: */
/* --------- */
/* PLXCPL IS THE COMPILER FOR THE PLEX LANGUAGE. */
/* */
/***************************************************************************/

DECLARE DYNAMP "1008", DYNAMB "0000", STACKTP "$STK TO ", ONELEV 01000000,
LEVNO 01, ONE LNG "00000100", DOPFIX 0008;

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, PARMCNT 00, OCTEN 10, OCTL60 60,
OCTL20 20;
/* VARIABLES USED FOR LABEL PROCESSING (PRIMARILY FOR PROCEDURE EXITS) */
.DECLARE PEXITF 77, PEXITT 00;
/* VARIABLES USED FOR TESTING I/O ITERATION LOOPS */
.DECLARE IOITER 77, IO5W 00;
/* OTHER VARIABLES AND VALUES FOR GENERAL USE */
.DECLARE ONE 01, FOUR "0004", EIGHT "0008", BLNK8 "", ZERO 00;

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

* / OPERATION CODES FOR METAX9 OBJECT PROGRAMS. */

MOVE .OEQU 12 .TYPEO; A .OEQU 13 .TYPEO; M .OEQU 14 .TYPEO;
BEF .OEQU 16 .TYPEO; EXIT .OEQU 17 .TYPEO; RESOLVE .OEQU 20 .TYPEO;
R .OEQU 21 .TYPEO; BT .OEQU 22 .TYPEO; BF .OEQU 23 .TYPEO;
BM .OEQU 24 .TYPEO; SET .OEQU 25 .TYPEO; SETF .OEQU 15 .TYPEO;
SCAN .OEQU 26 .TYPEO; MOVI .OEQU 30 .TYPEO; SEARCH .OEQU 33 .TYPEO;
COMP .OEQU 32 .TYPEO; TSTTBLL .OEQU 34 .TYPEO; TSTTBA .OEQU 35 .TYPEO;
LATCH .OEQU 36 .TYPEO; CANCEL .OEQU 37 .TYPEO; CLM .OEQU 42 .TYPEO;
/* OPERATION CODES FOR PLEX OBJECT PROGRAMS */

DYNAM .OEUQ 10 .TYPEG; STKTOP .OEUQ 11 .TYPEG; ALLOC .OEUQ 12 .TYPEG;
LDA .OEUQ 20 .TYPEG; LD .OEUQ 21 .TYPEG; STC .OEUQ 22 .TYPEG;
SST .OEUQ 30 .TYPEG; JUMP .OEUQ 31 .TYPEG; JUMPT .OEUQ 32 .TYPEG;
ADD .OEUQ 40 .TYPEG; MULT .OEUQ 41 .TYPEG; COMPC .OEUQ 42 .TYPEG;
DIV .OEUQ 43 .TYPEG; NEG .OEUQ 44 .TYPEG; FMT .OEUQ 45 .TYPEG;
GET .OEUQ 50 .TYPEG; PUT .OEUQ 51 .TYPEG; EDIT .OEUQ 52 .TYPEG;
OR .OEUQ 60 .TYPEG; AND .OEUQ 61 .TYPEG; NOT .OEUQ 62 .TYPEG;
INDXR .OEUQ 65 .TYPEG; INDXA .OEUQ 66 .TYPEG; CAT .OEUQ 67 .TYPEG;
SUBSTR .OEUQ 70 .TYPEG; JUMPA .OEUQ 71 .TYPEG; STOP .OEUQ 72 .TYPEG;
ENTPRO .OEUQ 73 .TYPEG; RETURN .OEUQ 74 .TYPEG; FLAG .OEUQ 75 .TYPEG;
SWP .OEUQ 76 .TYPEG; POPUP .OEUQ 77 .TYPEG; SAVE .OEUQ 80 .TYPEG;

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

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

/* COMMENCE THE RECURSIVE PROCEDURES COMPRISING THE COMPILER PROPER. */
PLXCPL := .EMPTY .ERRLATCH PROGRAM .ERR("F: COMPILER ABORT - BAD PROGRAM",EXIT);
PROGRAM:= .ERRLATCH .NEWLAB .ID ":" .STKSYM /* SAVE NAME FOR END CHECK */
.SET(PRNGM='SYMBOL')  /* SET PROG NAME FOR STORE OPTION */
.SET(INTFN='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(DYNMB=DYNAMP)  /* DYNAMIC STORAGE AT RUN TIME */
.SET(SYM=SYMBOL)  /* INITIALIZE BLOCK DYNAMIC STORAGE */
.SET(NTM='PLXINT00')  /* COUNTER FOR LEVEL AND DISPACEMENT */

"PROCEDURE" "MAIN" .BLKENT";  /* REQUIRED SYNTAX; START BLOCK LIST */
.OUT(DYNMB,*1)  /* PROCEDURE DYNAMIC STORAGE CODE */
.SET(SYMBOL=STACKTP)  /* TYPE AND INIT STACKTOP LOCATION */
.SEARCH .ENTERL(00,0501000000)  /* LABEL AND VALUE */

$ COMMENT  /* ADMIT HEADER COMMENTS */
.BHDBDY  /* PROCESS BLOCK HEAD AND BODY */
.END  /* POSITION SYMTAB POINTER */
.DEFLAB(*1)  /* MAXIMUM SIZE OF LEVEL 1 STORAGE */
.ENTERL(01,DYNAMP)  /* FOR THIS PROCEDURE */

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

BLKEDY := $( .SET(STMLAB=BLMK8) STMENT ) ;

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

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

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

LABEL := .SET(STMLAB=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:\n::="BEGIN" ";} * BLKENT ERRLATCH /* SET BLOCK LIST AND ERROR LATCH */\n  * STACK(STMLAB, DYNAMB, STACKTP) /* STACK VARIABLES TO PREPARE FOR */\n  * OUT(STKTOP,**)) /* CODE TO SET BLOCK STACK TOP */\n  * DO(A, ONELNG, STACKTP) /* CREATE NEW STACK TOP LABEL */\n  * SET(SYMB0L=STACKTP) /* SET FOR SYMTAB PROCESSING */\n  * SEARCH("1") * ENTRA(01, DYNAMB) /* SYMTAB ENTRY */\n  * ENTERL(00, 05) /* NEW STACK TOP ADDRESS AS 2ND OPER */\n  * OUT(**) /* INCREMENT DYNAMIC STORAGE COUNTER */\n  * DO(A, FOUJR, DYNAMB) /* PROCESS BLOCK HEAD AND BODY */\n  * UNSTACK(DYNAMB, STACKTP) /* RESTORE PREVIOUS STACK TOP SYMBOL */\n  * "END" * BLKEXT /* RESTORE SYMTAB TO PREVIOUS LEVEL */\n( * ID * STKCHK * ERR("W: POSSIBLE BLOCK CLOSING ERROR",SET)\n  * EMPTY) ;

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

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

PROCDEF := * ID ":" "PROCEDURE" * CANCEL * NEWLAB /* LABEL FOR STORAGE COUNT */\n  * SEARCH("1") * DO(ENTLOC) /* LOCATION AND PROCEDURE TYPE */\n  * TSTTBL(00,10) * DO(BM,"W: DUP PROC DCL",SET)\n  / TSTTBL(00,00) * ENTERL(00,10) / * DO(SET) )\n  * BLKENT * ERRLATCH\n  * SET(STMLAB=SYMB0L) /* SAVE PROCEDURE ID FOR ENDCHECK */\n  * STKSYM /* SAVE PREVIOUS PROCEDURE INFO */\n  * STACK(DYNAMP, DYNAMB, LEVNO, STACKTP) /* SET UP */\n  * DO(A, ONELNG, LEVNO) /* DYNAMIC STORAGE */
*DO(M,LEVNO,DYNAMP) /* COUNTERS */
*SET(DYNA\(b\)=DYNAMP) /* FOR THIS PROCEDURE */

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

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

*DO(M,LEVNO,SYMBOL,A,FOUR,SYMBOL,A,S\(\)YMBOL,DYNAM\(b\))
*OUT(ENTPRO) /* PROCEDURE ENTRY */
*SET(SYMBOL=LEVNO) .OUT(*) /* DYNAMIC STORAGE LEVEL NUMBER */
*OUT(DYNA\(m\),*1) /* CODE TO ALLOCATE DYNAMIC STORAGE */

*SET(ARGCNT=ZERO)
( "(" ARGID $ ( "," ARGID ) ")" / .EMPTY ) ;; /* DUMMY ARGUMENTS */
*SET(SYMBOL=STMTAB) /* ENTER ARGUMENT COUNT */
*SEARCH .ENTFRA(07,ARGCNT)
*SET(ARGDEC=ZERO)
( TARGCNT /* TEST FOR ARGUMENT COUNT */
/ .EMPTY ) /* ATTRIBUTES FOR DUMMY ARGUMENTS */
( ."DECLARE" ARGDEC $ ( "," ARGDEC ) ";" / .EMPTY )
*TEST(ARGCNT=ARGDEC) *ERR("w: INCORRECT ARG DCL COUNT",SET)
/ .EMPTY ) /* BYPASS IF NO ARGS */
BHDEHY /* PROCESS THE REST OF THIS PROCEDURE */
"END" .OUT(RETURN)
*DEFLAB(*1) /* POSITION SYMTAB */
*ENTERA(01,DYNAMP) /* MAXSIZE OF FIXED DYNAMIC STORAGE */
/* FOR THIS PROCEDURE */
*BLKEXT("1") /* PREVENT UNRESOLVED LABEL LINKAGE */
/ .EMPTY; /* OUTSIDE THIS PROCEDURE */

*UNSTACK(DYNA\(m\),DYNAM\(b\),LEVNO,STACKTP) /* RESTORE FOR PREV PROC */
*ID .ERR("w: EXPECTED CLOSING PROC NAME",SET)
*STKCHK .ERR("w: POSS PROC CLOSING ERR",SET) ";;" ;

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

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

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

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

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

CALLPRT := SAV(LDA,**,JUMPA) .EMPTY /* SAVE CODE TO JUMP TO PROC */
   .SET(STMAB=SYMBOL) /* SAVE PROC NAME FOR CHECKING */
   .NEWLAB .OUT(LDA+06+1)
   .OUT(LD+0100000065) /* CODE TO LOAD RETURN ADDRESS */
   .SET(SYMBOL=STMAB) .SEARCH
   ( ( TSTTBL(00,10)
     / TSTTBL(00,11)
     / TSTTBL(00,12)
     / TSTTBL(00,13)
     / TSTTBL(00,14) )
   .OUT(LD+0100000065) /* CODE TO LOAD CURRENT DISPLAY ADDR */
   / ( 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,*1,*0004,ADD)
OUT(STO)
OUT(LD,01) DEFLAB(*1)
ENTERL(00,01)
OUT("0000")
DO(POP)
DO(SLT)
ERR("F: INVALID PROC CALL")
CLEPRM
SET(SYMBOL=STACKTP)
OUT(LD,**)
OUT(#)
DEFLAB(*1)

/* CODE TO LOAD ADDR FOR STOPING
COMPUTED GLOBAL DISPLAY ADDR */
/* COMPUTE ADDR OF GLOBAL DISPLAY ADDR */
/* CODE TO STORE ADDR OF GLOBAL
DISPLAY ADDR IN NEXT INSTR */
/* CODE TO LOAD GLOBAL DISPLAY ADDR */
/* POP *1 LABEL */
/* SET TF CODE FOR COMPILER TESTING */
/* CODE TO LOAD ADDR FOR STOPING
COMPUTED GLOBAL DISPLAY ADDR */
/* CURRENT STACK TOP SYMBOL */
/* CODE TO LOAD STACK TOP VALUE */
/* RESTORE PROC JUMP CODE */
/* 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(PARMPRT)
SEARCH("0")
( TSTTBL(00,11) / TSTTBL(00,12) / TSTTBL(00,13) / TSTTBL(00,10))
TSTTBA(07,PARMCNT)
ERR("W: INCORRECT PARM CNT",SET)
/ *EMPTY ) UNSTACK(PARMCNT);

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

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

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

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

PARMEXP:= NEWLAB DO(LB1) SEARCH("1") ENTERA(01,DYNAMB)
ENTERL(00,01) DO(A,EIGHT,DYNAMB) OUT(LDA,**,LDA,**)
( EXP / SSE / LTERM / dPRMRY )
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, **) /* CODE TO LOAD ADDR FOR STORING GLOBAL DISPLAY */

• DO(A*FOUR,DYNAMB) /* CODE TO LOAD ADDR FOR STORING COMPUTED ADDR OF PASSFD GLOBAL DISPLAY ADDRESS */

• NEWLAB /* CODE TO STORE IN NEXT INSTR */

• OUT(LDA,*01, *1) /* CODE TO LOAD GLOBAL DISPLAY ADDR */

• OUT(?) /* CODE TO STORE IN CURR DYNAMIC STOR */

• OUT(LD, 71, "0004", ADD) /* CODE TO LOAD ADDR FOR STORING COMPUTED ADDR OF PASSFD GLOBAL DISPLAY ADDRESS */

• OUT(STU) /* CODE TO LOAD ADDR OF PROC ADDR CODE */

• OUT(LD,*01) /* CODE TO STORE IN NEXT INSTR */

• DEFLAB(*1) /* CODE TO LOAD ADDR OF PROCE ADDR CODE */

• ENTERL(0001) /* CODE TO LOAD ADDR OF PROCE ADDR CODE */

• OUT("0000") /* CODE TO STORE IN CURR DYNAMIC STOR */

• OUT(STU) ; /* CODE TO STORE IN CURR DYNAMIC STOR */

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

$( "", DECL1 ERR("F", SCAN, ";", SET) ) "" ;

DECL1 := IDELMNT / IDGROUP;

IDGFOUP := "" LISTPT ;

LISTPT := ID STKSYM: / "" LISTPT /* STACK IDENTIFIER FOR LATER USE */

• STACK(DIMCNT) /* CHECK FOR ARRAY ATTRIBUTES */

• IDLIST /* RECURSIVELY SEARCH FOR MORE */

• UNSTACK(SYMSAV, DIMCNT) /* PERFORM ARRAY SEMANTICS */

• SYMSEM /* RECURSIVE SEARCH */

• IDLIST /* PERFORM IDENTIFIER SEMANTICS */

• DO(P) SEARCH("1") /* BUMP DYNAMIC STORAGE COUNTER */

• ALENGTH, DYNAMB) ; /* BUMP DYNAMIC STORAGE COUNTER */

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

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

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

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

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

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

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

ARRYSEM := .SET(SYMBOL=STACKTP)
*OUT(ALLOC,***)  /* OP CODE AND STACK TOP ADDRESS */
*SET(SYMBOL=BLNK8)  /* FOLLOWED BY SDV */
*SET(SYMBOL=SYM5AV)/* CLEAR SYMBOL TO BLANKS */
*SEARCH("I")  /* POSITION SYMTAB POINTER */
IDSLM:
*ENTERA(07,DIMCNT)  /* ENTER DIMENSION COUNT */
*OUT(**)  /* ENTER TYPE, LENGTH AND ADDRESS */
*OUT(**(01,07))  /* DIMCNT ALSO TO SDV */
*OUT(**(02,05))  /* LENGTH TO SDV */
*OUT(?)  /* FINISH SDV WITH BOUND PAIR CODE */
DO(A,D0PF1X,DYNAMB)  /* ADD FIXED DOPE VECTOR SIZE */
SET(SYMBOL=FOUR)
DO(M,DIMCNT,SYMBOL)
DO(A,SYM60L,DYNAMB)  /* ADJUST FOR MULTIPLIER STORAGE */
DO(SET) ;

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

DOGROUP :=*ID *TEST(SYMBOL=DOSYM) *ERRLATCH
( ";;" TAIL / CASE / DOWHILE / LOOP )
*ERR("F: INVALID DO GROUP SYNTAX",CLM,TAIL)
TAI :=BLKB0D Y 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",SCA N,";",CLM,TAIL,R) ";;
*OUT(LD,7100000005,MULT)  /* CODE TO MULT BY TV ELEMENT SIZE */
*DO(LBI) *SEARCH("I") *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) STMENT /* LABEL AND COMPILE STATEMENT */
OUT(JUMP,*2) /* CODE TO JUMP OUT OF CASE GROUP */
CAT(JUMP,*1) /* CATEGORIZE CODE TO TRANSFER VECTOR */
DO(POP) /* POP EXTRANEOUS LABEL */
OUT(LD,06) /* DUMMY FOR POSTLISTING */
DEFLAB(*2) ENTERL(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(*) /* OUTPUT TV TO CODE STRING */
ENDING DEFLAB(*1) /* DEFINE CASE GROUP EXIT ADDRESS */

DO WHILE:="WHILE" *NEWLAB DEFLAB(*1) *NEWLAB WHLPT ":;" TAIL
OUT(JUMP,*2) DEFLAB(*1)

WHILE :="WHILE" WHLPT ":;" ;

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

LOOP := ID UDNTST /* LOOP EXIT LABEL */
NEWLAB /* CODE TO LOAD ADDRESS FOR INIT VALUE*/
OUT(LDA,**) /* SAVE IDENTIFIER FOR LATER USE */
STKSYM ITPRPT
OUT(JUMPT,*1) /* LOOP EXIT CODE */
(WHILE "/;" ) ERR("M: EXPECTED ";",SET)
TAIL
DO(POP) /* RETRIVE LOOP INDEX */
OUT(LDA,**,LD,**) /* CODE TO PREPARE FOR INCREMENT */
OUT(#) /* AND TEST */
OUT(*) /* RESTORE END OF LOOP CODE FROM */
DEFLAB(*1) /* CODE STACK */
DEFLAB(*1) ; /* DEFINE LOOP EXIT ADDRESS */
**ITERPRT:** "=" • ERR LATCH EXP

* ERR("F: INVALID INITIAL INDEX") • OUT(SST)

"TO" /* INITIAL VALUE */

• NEWLAB • DEFLAB(*1) /* ITERATION LABEL AND POSSIBLE VALUE */

• MARK /* POSSIBLE CODE REORDERING POINT */

(*LATCH(LOCPRM)) /* COMPARE SIMPLE PRIMARY IF ANY */

/ • NEWLAB • DEFLAB(*1) /* LABEL FOR TEMPORTARY 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) /* COMPILILE 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) /* COMPILE LOOP INDEX TESTING CODE */

("=" • OUT(02) • IOCHK • SAV(SUB)) /* AND STACK PROPER OP CODE FOR */

/ • EMPTY • OUT(04) • IOCHK • SAV(ADD)) /* INCREMENTING LOOP INDEX */

• MARK /* POSSIBLE CODE REORDERING POINT */

(*LATCH(LOCPRM)) /* COMPARE SIMPLE PRIMARY IF ANY */

/ • DO(FSP) • SAV /* DISCARD CODE MARK, SAVE TEST CODE */

• NEWLAB • DEFLAB(*1) /* LABEL FOR TEMPORARY IN DYNAMIC */

• ENTERA(01 • DYNAM) /* 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)

• 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 */ /* */
EMPTIOUT(STKC,04) /* DEFAULT TEST */ /* */
ICCHK MARk
OUT(LL+71,0001,ADD) /* AND INCREMENT CODE */ /* */
OUT(SST,JUMP,*1) /* FINAL LOOP END CODE */ /* */
SAV ; /* SAVE ALL LOOP END CODE */ /* */
/* RESTORE Iclist CODE IF I/O ITERATION */

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

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

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

DUP'TST := .SEARCH("1") .EMPTY .TSTTBL(00,00) .ERR("W: MULTIPLE DECLARATION") ;

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

BOOLEXP := ( .LATCH(IFCLOSE) .NEWLAB .OUT(JUMPF,*1) 
BTERM .NEWLAB .OUT(JUMPF,*1) . "ELSE" .DEFLAB(*2) BOOLEXP 
. DEFLAB(*1) ) / BTERM ;

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

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

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

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

BFUNCT :=.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(00,63) ) .CANCEL
( "(" OUT(LDA,**) SUBLIST ")" .OUT(INDXR*)
/ .EMPTY .OUT(LD,**)) ;

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

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

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

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

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

EXP :=LATCH(IFCLOSE) .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(JUMP\*1) \\
\*DEFLAB(*2) SEXP \*DEFLAB(*1) \\
/ SSE ;

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

STEPM := SUBSTRG / \*STRING \*OUT(LDA,74,*) / \*LATCH(SVARBLE) / \*LATCH(SFUNCT) ;

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

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

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

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

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

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

CONST := LATCH(IFCLOSE) .NEWLAB .OUT(JUMPF,***)
   BSCSTM ( "ELSE" .NEWLAB .OUT(JUMP,***)
      .DEFLAB(*) STMNT .DEFLAE(****)
      / .EMPTY .DEFLAB(*) ) ;
IFCLOSE := ID .TEST(SYMBOL=IFSY1) .CANCEL BOOLEXP "THEN" ;

IDENT := ID .SEARCH("0") .TSTTBL(00,00) .DO(BM,"W: UNDECLARED VARIABLE") ;

UNCOND := LATCH(GOTOST) / INOUT / LATCH(CELLSTM) / LATCH(RTNSTM)
   / STPSTM / ASSGNST / IDENT ;

STPSTM := "STOP" .OUT(STOP) ;

RTNSTM := "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 := ASSGN / HASSGN / SASSGN / LASSGN ;

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

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

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

ALFPT := ID .SEARCH("O")
( TSTTBL(00,01) / TSTTBL(00,02)
 / TSTTBL(00,061) / TSTTBL(00,062) ) .CANCEL
.OUT(LDA,***) SUBPART ;

BLFTPT := ID .SEARCH("O")
( TSTTBL(00,03) / TSTTBL(00,063) ) .CANCEL
.OUT(LDA,***) SUBPART ;

LLFTPT := ID .SEARCH("O")
( TSTTBL(00,05) / TSTTBL(00,065) ) .CANCEL
.OUT(LDA,***) SUBPART ;

SLFTPT := LATCH(SVARBLE) / LSUBSTR ;

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

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

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

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

ILIST := IELEMNT $ ( "," IELEMNT ) ;
IELMNT := IRPLIST
   /* (LIST (SLFTPT) / (LATCH (ALFTPT) / (LATCH (BLFTPT)) OUT (GFT) )

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

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

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

OLIST := OELMNT $ ( "*" OELMNT )

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

ORPLIST := "(" .MARK OLIST .SAVE ERPLIST ";

FMTLIST := "(" .OUT (77) FMTITEM $ ( "*" FMTITEM ")") .OUT (77)

FMTITEM := CTRLFMT / DATAFMT

CTRLFMT := "X" .OUT (00) SPEC
   / "PAGE" .OUT (01)
   / "SKIP" .OUT (02) ( SPEC / .EMPTY .OUT ( "0001") )
   / "COL" .OUT (03) SPEC

DATAFMT := "A" .OUT (10) ( SPEC / .EMPTY .OUT ( "0000") )
/ "E" .OUT(11) SPEC
/ "I" .OUT(12) SPEC / "L" .OUT(13) SPEC ;

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

*ENC

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

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

****SYMBOL TABLE ENTRY COUNT = 846****
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Address</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVE SYMBOL</td>
<td>$006</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEARCH &quot;O&quot;</td>
<td>$005</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENTL &quot;$1000&quot;</td>
<td>$006</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLM COMMEN BT $006</td>
<td>$006</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SET BF $006</td>
<td>$008</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNF BHDDBY BEF</td>
<td>$006</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEST &quot;END&quot; BEF</td>
<td>$008</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUT STOP LB1</td>
<td>$006</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENTL &quot;$O&quot; &quot;6&quot;</td>
<td>$007</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENTA &quot;$1&quot;</td>
<td>$008</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DYNAMP</td>
<td>$007</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID BF $008</td>
<td>$009</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHKSYM BT $009</td>
<td>$009</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BM &quot;W: POSSI SET</td>
<td>$009</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$009 B $007</td>
<td>$008</td>
<td></td>
<td>SCAN DOSYM</td>
</tr>
<tr>
<td>$007 BEF</td>
<td>$007</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSCSTM LATCH</td>
<td>$007</td>
<td>$017 R</td>
<td></td>
</tr>
<tr>
<td>RESOLV</td>
<td>$005 R</td>
<td>$028</td>
<td>LATCH BLOCK</td>
</tr>
<tr>
<td>BLKBDY MOVE BLNKH STMLAB</td>
<td>$005</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLKBDY MOVE</td>
<td>$005</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>Instruction</td>
<td>Address</td>
<td>Instruction</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>$029</td>
<td>CLM UNCOND</td>
<td>$039</td>
<td>BF $39</td>
</tr>
<tr>
<td></td>
<td>BF $30</td>
<td></td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>BF $27</td>
<td></td>
<td>TEST &quot;BEGIN&quot;</td>
</tr>
<tr>
<td>$030</td>
<td>SET BF $27</td>
<td></td>
<td>BF $043</td>
</tr>
<tr>
<td></td>
<td>BF $26</td>
<td></td>
<td>TEST &quot;;&quot;</td>
</tr>
<tr>
<td>$027</td>
<td>TEST BF &quot;;&quot;</td>
<td></td>
<td>BEF BLOCK</td>
</tr>
<tr>
<td>$026</td>
<td>R</td>
<td></td>
<td>BLKENT MOVI</td>
</tr>
<tr>
<td></td>
<td>COMMENT TEST</td>
<td></td>
<td>ELATCH &quot;1&quot;</td>
</tr>
<tr>
<td></td>
<td>BF $27</td>
<td></td>
<td>MOVE STMLAB</td>
</tr>
<tr>
<td></td>
<td>SCAN DOSYM</td>
<td></td>
<td>SYMBOL</td>
</tr>
<tr>
<td>$033</td>
<td>SET ID BF $33</td>
<td>STKSYM</td>
<td>MOVE DYNAMB</td>
</tr>
<tr>
<td>LABEL</td>
<td>BF $35</td>
<td></td>
<td>SYMBOL</td>
</tr>
<tr>
<td></td>
<td>TEST BF &quot;:&quot;</td>
<td></td>
<td>STKSYM MOVE</td>
</tr>
<tr>
<td></td>
<td>BEF MOVL BF</td>
<td></td>
<td>STACKT SYMBOL</td>
</tr>
<tr>
<td></td>
<td>SYMBOL STMLAB</td>
<td>OUT</td>
<td>EVAL &quot;5&quot;</td>
</tr>
<tr>
<td></td>
<td>SEARCH ENTLOC</td>
<td></td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td></td>
<td>ENTL &quot;0&quot;</td>
<td></td>
<td>&quot;6&quot;</td>
</tr>
<tr>
<td>$035</td>
<td>R LATCH</td>
<td>$037</td>
<td>A ONELNG</td>
</tr>
<tr>
<td>ENDSTM</td>
<td>ENDSTM BF</td>
<td></td>
<td>STACKT MOVE</td>
</tr>
<tr>
<td></td>
<td>SETF BF $36</td>
<td></td>
<td>STACKT SYMBOL</td>
</tr>
<tr>
<td></td>
<td>B $36</td>
<td></td>
<td>SEARCH &quot;1&quot;</td>
</tr>
<tr>
<td>$037</td>
<td>SET BF $36</td>
<td></td>
<td>ENTA &quot;1&quot;</td>
</tr>
<tr>
<td></td>
<td>BF $36</td>
<td></td>
<td>DYNAMB ENTL</td>
</tr>
<tr>
<td></td>
<td>R TEST BF &quot;END&quot;</td>
<td>EVAL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BEF SETF BF</td>
<td></td>
<td>&quot;5&quot;</td>
</tr>
<tr>
<td></td>
<td>B $39</td>
<td></td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td>$040</td>
<td>SET CLM $39</td>
<td></td>
<td>A FOUR</td>
</tr>
<tr>
<td></td>
<td>$039</td>
<td></td>
<td>DYNAMB CLM</td>
</tr>
<tr>
<td></td>
<td>BF $040</td>
<td></td>
<td>BHDBDY</td>
</tr>
<tr>
<td>BEF</td>
<td>BLKHD</td>
<td>CLM</td>
<td>DECLAR</td>
</tr>
<tr>
<td>-----</td>
<td>-------</td>
<td>-----</td>
<td>--------</td>
</tr>
<tr>
<td>POP</td>
<td>BT</td>
<td>SET</td>
<td>PLKHD</td>
</tr>
<tr>
<td>MOVE SYMBOL STACKT</td>
<td>SET</td>
<td>BF</td>
<td>$054</td>
</tr>
<tr>
<td>POP</td>
<td>BF</td>
<td>PUSHLB</td>
<td>OUT</td>
</tr>
<tr>
<td>MOVE SYMBOL DYNAMB</td>
<td>LB1</td>
<td>EVAL</td>
<td>&quot;4&quot;</td>
</tr>
<tr>
<td>TEST</td>
<td>&quot;END&quot;</td>
<td>&quot;1&quot;</td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td>BLKEXT</td>
<td>&quot;0&quot;</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>$046</td>
<td>LATCH</td>
<td>PROCDE</td>
</tr>
<tr>
<td>BF</td>
<td>$045</td>
<td>BT</td>
<td>$056</td>
</tr>
<tr>
<td>CHKSYM</td>
<td>$044</td>
<td>SET</td>
<td></td>
</tr>
<tr>
<td>BT</td>
<td>BF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BM</td>
<td>&quot;w: POSSI&quot;</td>
<td>BEF</td>
<td></td>
</tr>
<tr>
<td>SET</td>
<td>BLKHD</td>
<td>B</td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td>$046</td>
<td>$045</td>
<td>$054</td>
<td>R</td>
</tr>
<tr>
<td>$044</td>
<td>SET</td>
<td>ProcDE</td>
<td>ID</td>
</tr>
<tr>
<td>BEF</td>
<td>BF</td>
<td>$044</td>
<td></td>
</tr>
<tr>
<td>$043</td>
<td>$054</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>&quot;0&quot;</td>
<td>ProcDE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$044</td>
<td>BF</td>
<td>$058</td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td>TEST</td>
<td>&quot;:&quot;</td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td>TEST</td>
<td>&quot;PROCEDURE&quot;</td>
<td></td>
</tr>
<tr>
<td>BLKHD</td>
<td>BEF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>CANCEL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$049</td>
<td>PUSHLB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td>SEARCH</td>
<td>&quot;1&quot;</td>
<td></td>
</tr>
<tr>
<td>BLKBDY</td>
<td>ENTLOC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td>TSTTLB</td>
<td>&quot;0&quot;</td>
<td></td>
</tr>
<tr>
<td>COMP</td>
<td>&quot;1&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;w: POSSI&quot;</td>
<td>&quot;8&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>&quot;w: DUP PROC DCL&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVE DYNAMB DYNAMP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$051</td>
<td>&quot;0&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$051</td>
<td>SET</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>BF</td>
<td>$060</td>
<td></td>
</tr>
<tr>
<td>$049</td>
<td>$050</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;w: POSSI&quot;</td>
<td>BM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>&quot;w: POSSI&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$049</td>
<td>$050</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;w: POSSI&quot;</td>
<td>SET</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$050</td>
<td>BF</td>
<td>$059</td>
<td></td>
</tr>
<tr>
<td>REF</td>
<td>B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$050</td>
<td>SET</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>BF</td>
<td>$060</td>
<td></td>
</tr>
<tr>
<td>$049</td>
<td>$059</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;w: POSSI&quot;</td>
<td>B</td>
<td>&quot;$059&quot;</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>&quot;0&quot;</td>
<td>SEARCH</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>-----</td>
<td>-------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>ENTL</td>
<td>&quot;0&quot;</td>
<td>ENTA</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;8&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>$059</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SET</td>
<td>$061</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td>BLKLNT</td>
<td>MOVE</td>
<td>FOUR</td>
</tr>
<tr>
<td></td>
<td>MOVI</td>
<td>SYMBOL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EATCH</td>
<td>SYMBOL</td>
<td>SYMBOL</td>
</tr>
<tr>
<td>STKSYM</td>
<td>MOVE</td>
<td>DYNAMP</td>
<td>DYNAMB</td>
</tr>
<tr>
<td></td>
<td>SYMBOL</td>
<td>SYMBOL</td>
<td></td>
</tr>
<tr>
<td>STKSYM</td>
<td>MOVE</td>
<td>DYNAMB</td>
<td>DYNAMB</td>
</tr>
<tr>
<td></td>
<td>SYMBOL</td>
<td>SYMBOL</td>
<td></td>
</tr>
<tr>
<td>STKSYM</td>
<td>MOVE</td>
<td>LEVNO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SYMBOL</td>
<td>SYMBOL</td>
<td></td>
</tr>
<tr>
<td>STKSYM</td>
<td>MOVE</td>
<td>STACKT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SYMBOL</td>
<td>SYMBOL</td>
<td></td>
</tr>
<tr>
<td>STKSYM</td>
<td>A</td>
<td>ONE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LEVNO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVE</td>
<td>ONELEV</td>
<td>DYNAMP</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>LEVNO</td>
<td>DYNAMP</td>
<td></td>
</tr>
<tr>
<td>MOV</td>
<td>DYNAMP</td>
<td>DYNAMB</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>ONEE</td>
<td>STACKT</td>
<td></td>
</tr>
<tr>
<td>MOV</td>
<td>STACKT</td>
<td>SYMBOL</td>
<td></td>
</tr>
</tbody>
</table>
BM
SET
$080
TEST
";"
BEF
$058
R
TARGCN
COMP
ARGCNT
ZERO
"2"
BF
SETF
R
$082
$081
SET
BF
$081
$081
R
ARGID
ID
BF
$085
SEARCH
"1"
ENTA
"1"
DYNAMB
"0"
"<"
A
ONE
ARGCNT
A
FOUR
DYNAMB
SET
R
$085
ARGDEC
ID
BF
$087
SEARCH
"1"
MOVI
ELATCH
"1"
TSTTBL
"0"
"<"
BT
$088
BM
"F: NON-E
$088
CLM
ARGARY
BEF
CLM
ARGATR
BEF
A
ONE
ADECNT
$087
R
ARGARY
TEST
"("
BF
$090
MOVE
ZERO
DYNAMB
"*
TEST
BEF
A
ONE
DIMCNT
$091
TEST
",
BF
$093
TEST
"*
BF
A
ONE
DIMCNT
$093
BT
SET
BEF
TEST
")"
BEF
ENTA
"7"
DIMCNT
B
$089
$090
SET
BF
$089
R
ARGATR
CLM
ATTRIB
BF
$096
ENTA
"5"
LENGTH
A
OCTL60
ENTAB
"0"

TYPE
$095

B
TEST
"ENTRY"
BF
$095
TEST
"RETURNS("#)
BF
$099
CLM
ATTRIB
BEF
TEST
")
BEF
A
OCTL20
ENTAB
"0"
TYPE
$098
B
SET
BF
$098
ENTL
"0"
"4"

$098
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"
LB1
EVAL
"4"
"1"

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>ENT1</th>
<th>&quot;0&quot;</th>
<th>TSTTBA</th>
<th>&quot;7&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>$104</td>
<td>R</td>
<td>OUT</td>
<td>&quot;6&quot;</td>
</tr>
<tr>
<td>CLLPRM</td>
<td>FLAG</td>
<td>PARMCN</td>
<td>BM</td>
</tr>
<tr>
<td>MOVE</td>
<td>SYMBOL</td>
<td></td>
<td>&quot;W: INCORRECT PAR</td>
</tr>
<tr>
<td>STKSYM</td>
<td>ZERO</td>
<td>PARMCN</td>
<td>SET</td>
</tr>
<tr>
<td>MOVE</td>
<td>SYMBOL</td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>STKSYM</td>
<td>CLM</td>
<td>PARMPR</td>
<td>BF</td>
</tr>
<tr>
<td></td>
<td>BF</td>
<td>$124</td>
<td>$133</td>
</tr>
<tr>
<td>$124</td>
<td>SET</td>
<td>BF</td>
<td>$127</td>
</tr>
<tr>
<td></td>
<td>BF</td>
<td>$123</td>
<td>$126</td>
</tr>
<tr>
<td></td>
<td>POP</td>
<td>SEARCH &quot;0&quot;</td>
<td>BF $122</td>
</tr>
<tr>
<td></td>
<td>BF</td>
<td>$122</td>
<td>$126</td>
</tr>
<tr>
<td>$123</td>
<td>BF</td>
<td>$123</td>
<td>$126</td>
</tr>
<tr>
<td></td>
<td>POP</td>
<td>TSTTBL &quot;0&quot;</td>
<td>BF $122</td>
</tr>
<tr>
<td></td>
<td>BF</td>
<td>$129</td>
<td>$126</td>
</tr>
<tr>
<td>$129</td>
<td>TSTTBL</td>
<td>&quot;9&quot;</td>
<td>BEF</td>
</tr>
<tr>
<td></td>
<td>BF</td>
<td>$128</td>
<td>$126</td>
</tr>
<tr>
<td></td>
<td>TSTTBL</td>
<td>&quot;*&quot;</td>
<td>BEF</td>
</tr>
<tr>
<td></td>
<td>BF</td>
<td>$130</td>
<td>$126</td>
</tr>
<tr>
<td>$130</td>
<td>TSTTBL</td>
<td>&quot;*&quot;</td>
<td>BEF</td>
</tr>
<tr>
<td></td>
<td>BF</td>
<td>$131</td>
<td>$126</td>
</tr>
<tr>
<td></td>
<td>$131 TSTTBL</td>
<td>&quot;8&quot;</td>
<td>BEF</td>
</tr>
<tr>
<td></td>
<td>BF</td>
<td>$128</td>
<td>$126</td>
</tr>
<tr>
<td></td>
<td>BF</td>
<td>$127</td>
<td>$126</td>
</tr>
<tr>
<td>$128</td>
<td>BF</td>
<td>$128</td>
<td>$126</td>
</tr>
<tr>
<td>Line</td>
<td>Instructions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>--------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$146</td>
<td>BF $146</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B $145</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LATCH BLFTPT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$147</td>
<td>BF $147</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B $145</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LATCH SLFTPT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$148</td>
<td>BF $148</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B $145</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LATCH LLFTPT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$149</td>
<td>BF $149</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B $145</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LATCH PROCHK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$145</td>
<td>R $145</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PROCHK ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BF $152</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TEST &quot;(&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BF $154</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SETF $154</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BEF $153</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SET $153</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BF $153</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$153</td>
<td>BEF $153</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEARCH &quot;0&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TSTTbL &quot;0&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;A&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BF $157</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CANCEL CLM GPROC BEF</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B $156</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TSTTbL &quot;0&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;+&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BF $156</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CANCEL CLM GPROC</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OUT &quot;1&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>R $156</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PUSHlb PARMEX</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LBl SEARCH &quot;1&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ENTA &quot;1&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DYNAMb</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ENTL &quot;0&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;7&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A EIGHT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DYNAMb OUT LDA</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EVAL &quot;5&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;0&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OUT LDA EVAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;5&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;0&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLM EXP</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BF $162</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>E $161</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$162 CLM SSE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BF $163</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B $161</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$163 CLM LTERM</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BF $164</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B $161</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$164 CLM BPRMRY</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BF $161</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$161 BF $160</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OUT STO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$160 R MARK</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OUT LDA</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OUT &quot;$1&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ENTL "0"
"1"
OUT LDA
EVAL "5"
"0"
A FOUR DYNAMB
PUSHLB OUT LDA
OUT "1"
LB1 EVAL "4"
"1"
RESTOR OUT LD
OUT "2"
OUT "0004"
OUT ADD
OUT STO
OUT LD
OUT "1"
LB1 ENTL OC
ENTL "0"
"6"
ENTL "0"
"1"
OUT "0000"
OUT STO
$168 R DECLAR TEST "DECLARE"
BF $171 MOVI ELATCH
CLM "1"
BT DECL1
BM "F"
SCAN ";"
SET
$172 TEST ";"
BF $175 CLM DECL1
BT $175 BM "F"
SCAN ";"
SET
$175 BT $172 SET
TEST ";"
BF $178 DECL1 CLM IDELMN
BF $178 B $177
CLM IDGROU
BF $177 R $177
IDGROU TEST "("
BF $181 CLM LISTPT
BEF
$181 R LISTPT ID
BF $183 STKSYM
CLM ARRYPT
BF $185 MOVE DIMCNT
SYMBOL
MOVI LENGTH
LEVNO
"3"
B $190
$198 TEST "LABEL"
BF $190
MOVI LENGTH
LEVNO
MOVI TYPE
"5"

$190 R ARRARY
TEST "(" BF $201
MOVI DIMCNT
"0"
CLM BDPRLS
BEF TEST ")"
BEF

$201 R BDPRLS MARK
CLM BNDPAI BF $203
SAVE TEST ","
BF $206
MARK
RETOR CLM BNDPAI BF $203
SAVE BT $204
SET BEF

$203 R BNDPAI A ONE $212

DIMCNT
CLM INTBND
BF $208
TEST ";"
BF $210
RETOR
CLM INTBND
BF
RETOR
B $209
SET BF $204
OUT "Z0001"
RETOR

$210 BEF
INTBND ID BF $213
SEARCH "0"
TSTTAL "0"
"1"
ET $214
BM "W: INVALID ARRAY"

$214 MARK EVAL "5"
"0"

SAVE B $212
INUM BF $212
MARK
OUT
"Z"
OUTSYM
SAVE R
IDELMN ID
BF $217
STKSYM CLM BF $219
ATTRIB CLM BF $218
POP MOVE SYMBOF CLM E $218
SEARCH "1" CLM ATTRIB BF $219
BEF CLR
BEF $217
E $218
SEARCH "1"

BF $222
EVAL "5"
EVAL "0"
EVAL "1"
EVAL "7"
EVAL "2"
EVAL "5"

RESTOR A DOPFIX
MOV E FOUR SYMBOF
M SYMBOL
A SYMBOL
DYNAMB

SET R
ENT A "0"
ENT TYPE
ENT "5"
ENT LENGTH
ENT LENGTH
ENT FUNCT
ENT "2"

BF $226
R $225
B $225

SET $226
BF $225
ENT A "1"
DYNAMB

ENT $225
BF $224
R $224

ENT $224
BF $229
<table>
<thead>
<tr>
<th>COMP</th>
<th>SYMBOL</th>
<th>DOSYM</th>
<th>&quot;2&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOV I</td>
<td>ELATCH</td>
<td>&quot;1&quot;</td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td>&quot;;&quot;</td>
<td></td>
<td>$241</td>
</tr>
<tr>
<td>BF</td>
<td>$231</td>
<td>CLM</td>
<td>TAIL</td>
</tr>
<tr>
<td>BEF</td>
<td>$230</td>
<td>CLM</td>
<td>CASE</td>
</tr>
<tr>
<td>BF</td>
<td>$232</td>
<td>CLM</td>
<td>LOOP</td>
</tr>
<tr>
<td>B</td>
<td>$230</td>
<td>CLM</td>
<td>TAIL</td>
</tr>
<tr>
<td>$231</td>
<td>CLM</td>
<td>CASE</td>
<td></td>
</tr>
<tr>
<td>$232</td>
<td>CLM</td>
<td>DOWHIL</td>
<td></td>
</tr>
<tr>
<td>$233</td>
<td>CLM</td>
<td>LOOP</td>
<td></td>
</tr>
<tr>
<td>$230</td>
<td>BT</td>
<td>$229</td>
<td></td>
</tr>
<tr>
<td>BM</td>
<td>&quot;F: INVALID&quot;</td>
<td>CLM</td>
<td>TAIL</td>
</tr>
<tr>
<td>$229</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TAIL</td>
<td>CLM</td>
<td>BLKB0DY</td>
<td>LBI</td>
</tr>
<tr>
<td>BF</td>
<td>$237</td>
<td>CLM</td>
<td>ENTLOC</td>
</tr>
<tr>
<td>BT</td>
<td>$237</td>
<td>BM</td>
<td>&quot;W: INVALID&quot;</td>
</tr>
<tr>
<td>SCAN</td>
<td>&quot;; &quot;</td>
<td>SET</td>
<td></td>
</tr>
<tr>
<td>$237</td>
<td>P</td>
<td>CASE</td>
<td>TEST</td>
</tr>
<tr>
<td>CASE</td>
<td>TEST</td>
<td>&quot;CASE&quot;</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$240</td>
<td>PUSHLB</td>
<td></td>
</tr>
<tr>
<td>MOV I</td>
<td>ELATCH</td>
<td>&quot;1&quot;</td>
<td></td>
</tr>
<tr>
<td>PUSHLB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td>EXP</td>
<td>BT</td>
<td>$241</td>
</tr>
<tr>
<td>BM</td>
<td>&quot;F: BAD CASE EXP&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCAN</td>
<td>&quot;; &quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td>TAIL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>BEF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>LD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;Z0005&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>MULT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LBI</td>
<td>SEARCH</td>
<td>&quot;1&quot;</td>
<td></td>
</tr>
<tr>
<td>ENTL</td>
<td>&quot;0&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>LD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EVAL</td>
<td>&quot;5&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;4&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JUMPA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUSHLB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LB2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EVAL</td>
<td>&quot;4&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JUMP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;1&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SAVE SWAP pop POP $242 PUShLB L62 LBl EVAL ENTLOC "4" ENTL "1" STMNT bF S244 "0" OUT JUMP "6" LB2 RESTOR EVAL CLM ENDING "4" BEF "1" LB1 MARK ENTLOC RESTOR ENTL "0" OUT JUMP "0" LBl "6" EVAL 5.240 R "4" DOWHIL TEST "WHILE" "1" BF $246 SAVE PUSMLB POP LBl LB2 SET "6" BT $242 ENTLOC ENTL "1" CLM STMNT ENDING "0" "6" "4" "1" LB2 RESTOR ENTLOC ENTL "0" "4" "1" LBl EVAL
ITEPR
$253
TEST =
BF $259
MOVI ELATCH
1
CLM EXP
BT $260
BM "F: INVALID"
OUT SST
TEST "TO"
BEF
PUSHLB
LB1
ENTLOC
ENTL
"0"
"6"
MARK
LATCH LOOPRM
BEF $262
B $264
$261
BT $264
TEST "BY"
BF $266
OUT STCKC
TEST "=
BF $268
OUT "2"
CLM IOCHK
B $267
ENTL "0"
"6"
ENTA "1"
DYNAMB
ENTL "0"
"7"
A EIGHT $268
DYNAMB
OUT LDA

LB1
EVAL
"5"
"0"
CLM EXP
BF $261
OUT STO
MARK
LB2
ENTLOC
ENTL
"0"
"6"
OUT LD
LB1
EVAL
"5"
"0"
SWAP
POP
BT $264
BM "F: INVALID INDEX"

$262
PUSHLB
LB1
ENTLOC
ENTL
"0"
"6"
ENTL
"1"
DYNAMB
ENTL "0"
"7"
A EIGHT $268
DYNAMB
OUT LDA

OUT "4"
CLM I Checklist
BEF OUT MARK ADD
SAVE BEF MARK

LATCH LOOPRM BF $271
B $270 $270 $270 BT $273

POP SAVE PUSHLB LB1 ENL LOC
ENT LB1 ENT LB1 ENT LB1 CB $266 SET
"0" "5"
"6" "0"
"7" "F: INVALID INCRE"

"0"
"6"
"7"
"4"
"Z"
"0001"
ADD

A EIGHT DYNAM LB1 EVAL
OUT LDA $265 BEF
LB1 OUT SST
EVAL OUT JUMP
LB1

CLM EXP BF $270 OUT STO $270
SAVE LB2 $259 R
ENT LOC IO CHK COMP IO I TER
ENT LB1 LB2 "2"
ENT LB1 LB2 "1"

156
<table>
<thead>
<tr>
<th>Location</th>
<th>Instruction</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$302</td>
<td>SET</td>
<td>$301</td>
</tr>
<tr>
<td>$301</td>
<td>BEF</td>
<td>$301</td>
</tr>
<tr>
<td>$300</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>$307</td>
<td>LATCH</td>
<td>IFCLS</td>
</tr>
<tr>
<td>$307</td>
<td>BF</td>
<td>$307</td>
</tr>
<tr>
<td></td>
<td>PUSHLB</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>JUMPF</td>
<td></td>
</tr>
<tr>
<td>LB1</td>
<td>EVAL</td>
<td>&quot;4&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>CLM</td>
<td>BTERM</td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUSHLB</td>
<td>OUT</td>
<td>JUMP</td>
</tr>
<tr>
<td>LB1</td>
<td>EVAL</td>
<td>&quot;4&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>TEST</td>
<td>&quot;ELSE&quot;</td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td>LB2</td>
<td></td>
</tr>
<tr>
<td>ENTL</td>
<td>&quot;0&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;6&quot;</td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td>BOOLEAN</td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td>LB1</td>
<td></td>
</tr>
<tr>
<td>ENTL</td>
<td>&quot;0&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;6&quot;</td>
<td></td>
</tr>
<tr>
<td>$307</td>
<td>BF</td>
<td>$305</td>
</tr>
<tr>
<td>$305</td>
<td>CLM</td>
<td>BTERM</td>
</tr>
<tr>
<td>$304</td>
<td>BF</td>
<td>$304</td>
</tr>
<tr>
<td>$304</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BTERM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLM</td>
<td>BFACTO</td>
</tr>
<tr>
<td>BEF</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$311</td>
<td>TEST</td>
<td>&quot; OR &quot;</td>
</tr>
<tr>
<td>BF</td>
<td></td>
<td>$310</td>
</tr>
<tr>
<td></td>
<td>CLM</td>
<td>BFACTO</td>
</tr>
<tr>
<td>BEF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$313</td>
<td>OUT</td>
<td>OR</td>
</tr>
<tr>
<td>$313</td>
<td>BT</td>
<td>$311</td>
</tr>
<tr>
<td>$310</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BFACTO</td>
<td></td>
</tr>
<tr>
<td>$315</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BSCNDR</td>
<td></td>
</tr>
<tr>
<td>$316</td>
<td>TEST</td>
<td>&quot; AND &quot;</td>
</tr>
<tr>
<td>BF</td>
<td></td>
<td>$318</td>
</tr>
<tr>
<td></td>
<td>CLM</td>
<td>BSCNDR</td>
</tr>
<tr>
<td>BEF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$318</td>
<td>OUT</td>
<td>AND</td>
</tr>
<tr>
<td>$318</td>
<td>BT</td>
<td>$316</td>
</tr>
<tr>
<td>$315</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BSCNDR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLM</td>
<td>BPRMRY</td>
</tr>
<tr>
<td>$320</td>
<td>TEST</td>
<td>&quot; NOT &quot;</td>
</tr>
<tr>
<td>BF</td>
<td></td>
<td>$319</td>
</tr>
<tr>
<td></td>
<td>CLM</td>
<td>BPRMRY</td>
</tr>
<tr>
<td>BEF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$319</td>
<td>OUT</td>
<td>NOT</td>
</tr>
<tr>
<td>$319</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BPRMRY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLM</td>
<td>BVALUE</td>
</tr>
<tr>
<td>$323</td>
<td>BF</td>
<td>$322</td>
</tr>
<tr>
<td>$323</td>
<td>B</td>
<td>$323</td>
</tr>
<tr>
<td>Line</td>
<td>LATCH</td>
<td>BVARBL</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>$323</td>
<td>BF</td>
<td>$324</td>
</tr>
<tr>
<td>$324</td>
<td>LATCH</td>
<td>BFNCT</td>
</tr>
<tr>
<td></td>
<td>$322</td>
<td></td>
</tr>
<tr>
<td>$325</td>
<td>CLM</td>
<td>RELATN</td>
</tr>
<tr>
<td>$326</td>
<td>TEST</td>
<td>&quot;(&quot;</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>$322</td>
</tr>
<tr>
<td></td>
<td>TEST</td>
<td>&quot;)&quot;</td>
</tr>
<tr>
<td></td>
<td>EM</td>
<td>&quot;w: EXPE&quot;</td>
</tr>
<tr>
<td>$322</td>
<td>R</td>
<td>RELATN</td>
</tr>
<tr>
<td></td>
<td>$330</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLM</td>
<td>RELOP</td>
</tr>
<tr>
<td></td>
<td>CLM</td>
<td>SAE</td>
</tr>
<tr>
<td></td>
<td>BEF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLM</td>
<td>STCKC</td>
</tr>
<tr>
<td></td>
<td>$329</td>
<td></td>
</tr>
<tr>
<td>$330</td>
<td>CLM</td>
<td>SSE</td>
</tr>
<tr>
<td></td>
<td>BF</td>
<td>$329</td>
</tr>
<tr>
<td></td>
<td>BF</td>
<td>RELOP</td>
</tr>
<tr>
<td></td>
<td>CLM</td>
<td>SSE</td>
</tr>
<tr>
<td></td>
<td>BEF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RESTOR</td>
<td>$343</td>
</tr>
<tr>
<td></td>
<td>$342</td>
<td>B</td>
</tr>
<tr>
<td>$325</td>
<td>R</td>
<td>BFNCT</td>
</tr>
<tr>
<td></td>
<td>$342</td>
<td>B</td>
</tr>
<tr>
<td>CANCEL</td>
<td>TEST</td>
<td>&quot;(&quot;</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>TEST</td>
<td>BF</td>
<td>$346</td>
</tr>
<tr>
<td>BF</td>
<td>OUT</td>
<td>LDA</td>
</tr>
<tr>
<td>OUT</td>
<td>EVAL</td>
<td>$351</td>
</tr>
<tr>
<td>EVAL</td>
<td>&quot;5&quot;</td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td>&quot;5&quot;</td>
<td>&quot;0&quot;</td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td>SUBL15</td>
<td></td>
</tr>
<tr>
<td>BEF</td>
<td>TEST</td>
<td>&quot;$)&quot;</td>
</tr>
<tr>
<td>TEST</td>
<td>BF</td>
<td>$346</td>
</tr>
<tr>
<td>BF</td>
<td>OUT</td>
<td>LD</td>
</tr>
<tr>
<td>OUT</td>
<td>SET</td>
<td>$345</td>
</tr>
<tr>
<td>OUTSYM</td>
<td>SET</td>
<td>$345</td>
</tr>
<tr>
<td>B</td>
<td>SET</td>
<td>$345</td>
</tr>
<tr>
<td></td>
<td>BEF</td>
<td>$345</td>
</tr>
<tr>
<td>$346</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$346</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>$345</td>
<td>$345</td>
<td>$345</td>
</tr>
<tr>
<td>Line</td>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>1</td>
<td>CLM</td>
<td>TERM</td>
</tr>
<tr>
<td>2</td>
<td>BEF</td>
<td>ADD</td>
</tr>
<tr>
<td>3</td>
<td>OUT</td>
<td>$362</td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>&quot;$&quot;</td>
</tr>
<tr>
<td>5</td>
<td>TEST</td>
<td>$362</td>
</tr>
<tr>
<td>6</td>
<td>BF</td>
<td>TERM</td>
</tr>
<tr>
<td>7</td>
<td>BEF</td>
<td>ADD</td>
</tr>
<tr>
<td>8</td>
<td>OUT</td>
<td>$361</td>
</tr>
<tr>
<td>9</td>
<td>BT</td>
<td>$361</td>
</tr>
<tr>
<td>10</td>
<td>SET</td>
<td>BF</td>
</tr>
<tr>
<td>11</td>
<td>BEF</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>R</td>
<td>TERM</td>
</tr>
<tr>
<td>13</td>
<td>CLM</td>
<td>PRIMRY</td>
</tr>
<tr>
<td>14</td>
<td>BF</td>
<td>$366</td>
</tr>
<tr>
<td>15</td>
<td>TEST</td>
<td>&quot;$&quot;</td>
</tr>
<tr>
<td>16</td>
<td>$367</td>
<td>BF</td>
</tr>
<tr>
<td>17</td>
<td>$369</td>
<td>PRIMRY</td>
</tr>
<tr>
<td>18</td>
<td>BEF</td>
<td>MULT</td>
</tr>
<tr>
<td>19</td>
<td>B</td>
<td>$368</td>
</tr>
<tr>
<td>20</td>
<td>TEST</td>
<td>&quot;$&quot;</td>
</tr>
<tr>
<td>21</td>
<td>$369</td>
<td>BF</td>
</tr>
<tr>
<td>22</td>
<td>$368</td>
<td>PRIMRY</td>
</tr>
<tr>
<td>23</td>
<td>BEF</td>
<td>DIV</td>
</tr>
<tr>
<td>24</td>
<td>BT</td>
<td>$367</td>
</tr>
<tr>
<td>25</td>
<td>SET</td>
<td>BF</td>
</tr>
<tr>
<td>26</td>
<td>BEF</td>
<td>CANCEL</td>
</tr>
<tr>
<td>27</td>
<td>R</td>
<td>PRIMRY</td>
</tr>
<tr>
<td>28</td>
<td>TEST</td>
<td>&quot;)&quot;</td>
</tr>
<tr>
<td>29</td>
<td>$366</td>
<td>BF</td>
</tr>
<tr>
<td>30</td>
<td>$372</td>
<td>EXP</td>
</tr>
<tr>
<td>31</td>
<td>BF</td>
<td>$373</td>
</tr>
<tr>
<td>32</td>
<td>R</td>
<td>INUM</td>
</tr>
<tr>
<td>33</td>
<td>CLM</td>
<td>CALLPR</td>
</tr>
<tr>
<td>34</td>
<td>BEF</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>R</td>
<td>OUT</td>
</tr>
<tr>
<td>36</td>
<td>&quot;Z&quot;</td>
<td>OUTSYM</td>
</tr>
<tr>
<td>37</td>
<td>OUT</td>
<td>$384</td>
</tr>
<tr>
<td>38</td>
<td>OUT</td>
<td>LD</td>
</tr>
<tr>
<td>39</td>
<td>OUT</td>
<td>$371</td>
</tr>
<tr>
<td>Address</td>
<td>Instruction</td>
<td>Value</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------</td>
<td>-------</td>
</tr>
<tr>
<td>$384</td>
<td>BT</td>
<td>$383</td>
</tr>
<tr>
<td>$384</td>
<td>FNUM</td>
<td>$383</td>
</tr>
<tr>
<td>$384</td>
<td>BF</td>
<td>$383</td>
</tr>
<tr>
<td>$384</td>
<td>OUT</td>
<td>LD</td>
</tr>
<tr>
<td>$384</td>
<td>OUTSYM</td>
<td>&quot;e&quot;</td>
</tr>
<tr>
<td>$383</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>$383</td>
<td>VARELE</td>
<td>ID</td>
</tr>
<tr>
<td>$383</td>
<td>BF</td>
<td>$387</td>
</tr>
<tr>
<td>$383</td>
<td>SEARCH</td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td>$383</td>
<td>TSTTBL</td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td>$383</td>
<td>TSTTBL</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>$383</td>
<td>TSTTBL</td>
<td>&quot;2&quot;</td>
</tr>
<tr>
<td>$383</td>
<td>TSTTBL</td>
<td>&quot;3&quot;</td>
</tr>
<tr>
<td>$388</td>
<td>TSTTBL</td>
<td>&quot;4&quot;</td>
</tr>
<tr>
<td>$388</td>
<td>TSTTBL</td>
<td>&quot;5&quot;</td>
</tr>
<tr>
<td>$388</td>
<td>TSTTBL</td>
<td>&quot;6&quot;</td>
</tr>
<tr>
<td>$388</td>
<td>CANCEL</td>
<td>TEST</td>
</tr>
<tr>
<td>$388</td>
<td>TEST</td>
<td>&quot;(&quot;</td>
</tr>
<tr>
<td>$388</td>
<td>BF</td>
<td>$394</td>
</tr>
<tr>
<td>$388</td>
<td>OUT</td>
<td>LDA</td>
</tr>
<tr>
<td>$388</td>
<td>EVAL</td>
<td>&quot;5&quot;</td>
</tr>
<tr>
<td>$388</td>
<td>CLM</td>
<td>SUBLIS</td>
</tr>
<tr>
<td>$388</td>
<td>BEF</td>
<td>TEST</td>
</tr>
<tr>
<td>$388</td>
<td>TEST</td>
<td>&quot;)&quot;</td>
</tr>
<tr>
<td>$388</td>
<td>BEF</td>
<td></td>
</tr>
<tr>
<td>$393</td>
<td>OUTSYM</td>
<td>$393</td>
</tr>
<tr>
<td>$393</td>
<td>Set</td>
<td>BF</td>
</tr>
<tr>
<td>$393</td>
<td>OUT</td>
<td>LD</td>
</tr>
<tr>
<td>$393</td>
<td>EVAL</td>
<td>&quot;5&quot;</td>
</tr>
<tr>
<td>$393</td>
<td>&quot;0&quot;</td>
<td></td>
</tr>
<tr>
<td>$397</td>
<td>LATCH</td>
<td>IFCLSE</td>
</tr>
<tr>
<td>$397</td>
<td>BF</td>
<td>$397</td>
</tr>
<tr>
<td>$397</td>
<td>PUSHLB</td>
<td></td>
</tr>
<tr>
<td>$397</td>
<td>OUT</td>
<td>JUMP</td>
</tr>
<tr>
<td>$397</td>
<td>EVAL</td>
<td></td>
</tr>
<tr>
<td>$397</td>
<td>&quot;4&quot;</td>
<td></td>
</tr>
<tr>
<td>$397</td>
<td>&quot;1&quot;</td>
<td></td>
</tr>
<tr>
<td>$397</td>
<td>SAE</td>
<td></td>
</tr>
<tr>
<td>$397</td>
<td>BEF</td>
<td>TEST</td>
</tr>
<tr>
<td>$397</td>
<td>&quot;ELSE&quot;</td>
<td></td>
</tr>
<tr>
<td>$397</td>
<td>BEF</td>
<td>PUSHLB</td>
</tr>
<tr>
<td>$397</td>
<td>OUT</td>
<td>JUMP</td>
</tr>
<tr>
<td>$397</td>
<td>EVAL</td>
<td></td>
</tr>
<tr>
<td>$397</td>
<td>&quot;0&quot;</td>
<td></td>
</tr>
<tr>
<td>$397</td>
<td>&quot;6&quot;</td>
<td></td>
</tr>
<tr>
<td>$397</td>
<td>&quot;7&quot;</td>
<td></td>
</tr>
<tr>
<td>$397</td>
<td>&quot;8&quot;</td>
<td></td>
</tr>
<tr>
<td>$397</td>
<td>&quot;9&quot;</td>
<td></td>
</tr>
<tr>
<td>$397</td>
<td>&quot;A&quot;</td>
<td></td>
</tr>
<tr>
<td>$397</td>
<td>&quot;B&quot;</td>
<td></td>
</tr>
<tr>
<td>$397</td>
<td>&quot;C&quot;</td>
<td></td>
</tr>
<tr>
<td>$397</td>
<td>&quot;D&quot;</td>
<td></td>
</tr>
<tr>
<td>$397</td>
<td>&quot;E&quot;</td>
<td></td>
</tr>
<tr>
<td>$397</td>
<td>&quot;F&quot;</td>
<td></td>
</tr>
</tbody>
</table>

Note: The text appears to be a snippet of machine code or assembly language, possibly for a computer or microcontroller. The instructions and values suggest some form of arithmetic or logical operation.
<table>
<thead>
<tr>
<th>$415</th>
<th>B</th>
<th>$414</th>
<th>BF</th>
<th>$425</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSTTbL</td>
<td>&quot;0&quot;</td>
<td>TSTTbL</td>
<td>&quot;0&quot;</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$414</td>
<td>BF</td>
<td>$424</td>
<td></td>
</tr>
<tr>
<td>$414</td>
<td>BEF</td>
<td>CANCEL</td>
<td>BEF</td>
<td>CANCEL</td>
</tr>
<tr>
<td>CLM</td>
<td>CALLPR</td>
<td>EVAL</td>
<td>LDA</td>
<td></td>
</tr>
<tr>
<td>$413</td>
<td>R</td>
<td>SASTRN</td>
<td>TEST</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>BF</td>
<td>$418</td>
<td>BF</td>
<td>$428</td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td>SEXP</td>
<td>CLM</td>
<td>SUBLIS</td>
<td></td>
</tr>
<tr>
<td>$420</td>
<td>SET</td>
<td>SET</td>
<td>BEF</td>
<td>$427</td>
</tr>
<tr>
<td>BF</td>
<td>$419</td>
<td>BF</td>
<td>$427</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>LD</td>
<td>BF</td>
<td>$427</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;Z&quot;</td>
<td>BEF</td>
<td>$423</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>&quot;0000&quot;</td>
<td>SUBLIS</td>
<td>MOVE</td>
<td>ONE</td>
</tr>
<tr>
<td>$419</td>
<td>BEF</td>
<td>TEST</td>
<td>STKSYM</td>
<td></td>
</tr>
<tr>
<td>BYTE</td>
<td>EXP</td>
<td>BUS</td>
<td>EXP</td>
<td></td>
</tr>
<tr>
<td>$418</td>
<td>R</td>
<td>SVARBL</td>
<td>ID</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>$423</td>
<td>BF</td>
<td>$434</td>
<td></td>
</tr>
<tr>
<td>SEARCH</td>
<td>&quot;0&quot;</td>
<td>BEF</td>
<td>ONE</td>
<td></td>
</tr>
<tr>
<td>TSTTbL</td>
<td>&quot;0&quot;</td>
<td>A</td>
<td>DIMCNT</td>
<td></td>
</tr>
<tr>
<td>TSTTbL</td>
<td>&quot;4&quot;</td>
<td>DIMCNT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$434</td>
<td>BT</td>
<td>$432</td>
<td>TEST</td>
<td>&quot;)&quot;</td>
</tr>
<tr>
<td></td>
<td>SET</td>
<td></td>
<td>BEF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BEF</td>
<td></td>
<td>OUT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>POP</td>
<td></td>
<td>OUTSYM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEARCH</td>
<td></td>
<td>B</td>
<td>$442</td>
</tr>
<tr>
<td></td>
<td>&quot;0&quot;</td>
<td></td>
<td>$443</td>
<td>SET</td>
</tr>
<tr>
<td></td>
<td>TSTTD</td>
<td></td>
<td>BF</td>
<td>$442</td>
</tr>
<tr>
<td></td>
<td>&quot;7&quot;</td>
<td>TSTTA</td>
<td></td>
<td>OUT</td>
</tr>
<tr>
<td></td>
<td>DIMCNT</td>
<td></td>
<td>OUTL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;□&quot;</td>
<td></td>
<td>&quot;4&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BL</td>
<td></td>
<td>EVAL</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;$\wedge$&quot;</td>
<td></td>
<td>$442</td>
<td>BF</td>
</tr>
<tr>
<td>$435</td>
<td>MOVE</td>
<td>$435</td>
<td>$438</td>
<td>BEF</td>
</tr>
<tr>
<td></td>
<td>DIMCNT</td>
<td></td>
<td>$437</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>SYMBOL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$432</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$442</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$438</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LBLCON</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;1&quot;</td>
<td></td>
<td>BF</td>
<td>$449</td>
</tr>
<tr>
<td></td>
<td>BF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;$5$&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;0&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;0&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$439</td>
<td>CLM</td>
<td>$439</td>
<td>$440</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>LBLVAR</td>
<td></td>
<td>BF</td>
<td>$449</td>
</tr>
<tr>
<td></td>
<td>&quot;$5$&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;$0$&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;$\sqrt{\nu}$&quot;</td>
<td></td>
<td>BF</td>
<td>$453</td>
</tr>
<tr>
<td></td>
<td>&quot;$V$&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;$453$&quot;</td>
<td></td>
<td>BF</td>
<td>$452</td>
</tr>
<tr>
<td></td>
<td>&quot;$452$&quot;</td>
<td></td>
<td>BF</td>
<td>$451</td>
</tr>
<tr>
<td>$440</td>
<td>TEST</td>
<td>$440</td>
<td>$452</td>
<td>TEST</td>
</tr>
<tr>
<td></td>
<td>&quot;${}$&quot;</td>
<td></td>
<td>&quot;$1^\circ$&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;$1^\circ$&quot;</td>
<td></td>
<td>BF</td>
<td>$452</td>
</tr>
<tr>
<td></td>
<td>&quot;$452$&quot;</td>
<td></td>
<td>BF</td>
<td>$456</td>
</tr>
<tr>
<td></td>
<td>&quot;$456$&quot;</td>
<td></td>
<td>OUT</td>
<td>LDA</td>
</tr>
<tr>
<td></td>
<td>$444$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$446$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EXP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$453$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PEXITT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;0&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;$452$&quot;</td>
<td></td>
<td>BF</td>
<td>$452</td>
</tr>
<tr>
<td></td>
<td>&quot;$452$&quot;</td>
<td></td>
<td>BF</td>
<td>$451</td>
</tr>
<tr>
<td></td>
<td>&quot;$451$&quot;</td>
<td></td>
<td>TEST</td>
<td>&quot;${}$&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;$451$&quot;</td>
<td></td>
<td>BF</td>
<td>$456</td>
</tr>
<tr>
<td></td>
<td>&quot;$456$&quot;</td>
<td></td>
<td>OUT</td>
<td>LDA</td>
</tr>
</tbody>
</table>
EVAL
  "5"
  "0"
CLM
  SUBLIS
BEF
  TEST "1"
BEF
  OUT INDXR
OUTSYM
  B $455
$456
  SET
BEF
  OUT $455
EVAL
  "5"
  "0"
$455
  BEF
$451
  R
CONDST
  LATCH IFCLSE
BEF
  $459
PUSHLB
  OUT JUMPF
LB1
  EVAL
    "4"
    "1"
CLM
  BSCSTM
BEF
  TEST "ELSE"
BEF
  PUSHLB
OUT
  OUT $461
LB1
  JUMP
EVAL
  "4"
  "1"
LB2
ENTLOC
  "0"
ENTL
  "6"
CLM
  STMENT
BEF
  LB1
ENTLOC
  "0"
ENTL
  "6"
B $460
$460
  SET
BEF
  LB1
ENTLOC
  ENTLOC
ENTL
  "0"
$461
  BEF
$459
  R
IFCLSE
  ID
BEF
  $464
COMP SYMBOL IFSYM
  "2"
BEF
  CANCEL
CLM BOOLEAN
BEF
  TEST "THEN"
BEF
  SEARCH "0"
TSTBL "0"
B E F  "0"  B F  $483
B M  "n: UNDEC"  $483  C L M  B P R A R Y
R  $466  L A T C H  G O T O S T  $481  B F  T E S T  "\)"
B F  $466  B F  $467  T E S T  "\)"
B F  $466  C L M  I N O U T  O U T  S W P
B F  $469  B F  $479
B F  $467  $479  B E F
B F  $467  $479  B E F
B F  $470  O U T  R E T U R N
B F  $467  R  $478  R
B F  $471  G O T O S T  T E S T  "G O"
B F  $472  T E S T  "T O"
B F  $472  C A N C E L
B F  $467  M A R K
B F  $473  C L M  L T E R M
B E  $467  B E F
B F  $467  C O M P  P E X I T T
B F  $467  P E X I T F
B F  $467  T E S T  "2"
B F  $476  S A V E
B F  $476  O U T  P O P U P
B F  $476  R E S T O R
B F  $478  O U T  R E T U R N
B F  $478  B
B F  $480  S E T
B F  $480  O U T  J U M P A
B F  $482  B E F
B F  $482  B E F
B F  $482  B E F
B F  $482  B E F
B F  $487  R
B F  $487  R
B F  $487  R
B F  $487  R
B F  $487  R
BF $492
   CLM $491
   BF $493
   B $491
SAVL $503
   TEST "$",
   BF $505
   CLM BLFTPT
   BEF
$492
   CLM $493
   BF $494
   B $491
   CLM $494
   BF $491
   CLM LASSGN
   BEF
   BF $493
   CLM SASSGN
   MARK
   OUT $503
   SST
   RESTOR
$493
   CLM $491
   BF $491
   R AASSGN
   LATCH ALFTPT
   BF $497
   MARK
   OUT STO
   SAVE
   TEST "$",
   BF $500
   CLM ALFTPT
   BEF
   MARK
   OUT SST
   RESTOR
   SAVE
$494
   TEST "$",
   BF $502
   CLM ALFTPT
   BEF
   MARK
   OUT STO
   RESTOR
   SAVE
$500
   BT $498
   SET
   HEF
   TEST "$",
   BF $508
   CLM SASSGN
   CLM SLFTPT
   BF $507
   Mark
   OUT STO
   RESTOR
   SAVE
$507
   BT $502
   SET
   BEF
   TEST "$",
   BF $510
   CLM SASSGN
   CLM SLFTPT
   BF $508
   Mark
   OUT SST
   RESTOR
   SAVE
$510
   BT $497
   LATCH BLFTPT
   BF $502
   MARK
   OUT STO
   CLM SEXP
| $525 | BEF   | $530 | BF | $540 | SET  |
|      | R     | LLFTPT | ID | BF | $531 | TSTTL | "0" |
|      | BF    | SEARCH | "0" | BF | $532 | TSTTL | "3" |
|      | BEF   | $531 | B  | $539 | TEST | ")"  |
|      | BEF   | CANCEL | OUT | LDA | $538 | SUBPAR | TEST  |
|      | BEF   | $531 | CLM | SUBPAR | $540 | SET | $539 |
|      | BEF   | $531 | BF | $534 | OUT | SUBSTR |
|      | BEF   | $534 | CLM | SVARBL | $543 | SET  |
|      | BF    | $535 | B  | $542 | BF | $542 |
|      | CLM   | $534 | BF | $546 | LATCH | OUTPUT |
|      | BF    | $534 | CLM | SVARBL | $545 | BF | $545 |
|      | TEST  | "SUBSTR("
|      | BF    | $538 | BF | $549 | TEST | "GET"
|      | CLM   | SVARBL | EXP |
|      | TEST  | ")"  |
|      | BEF   | $540 | BF | $551 | MARK |
|      | TEST  | ")"  |
|      | BEF   | $540 | BF | OUT | "Y"  |
SAVE TEST "(" BEF

CLM SLFTPT BEF

TEST ")" BEF

$550 SET BF

MARK OUT "Z" SAVE BF

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
$589  R
  FMTITE  CLM  CTRLFM
       BF  $594
       B  $593
  $594  CLM  DATAFM
       BF  $593
  $593  R
  CTRLFM  TEST  "X"
       BF  $597
       OUT  "O"
       CLM  SPEC
       BEF  B  $596
  $597  TEST  "PAGE"
       BF  $598
       OUT  "1"
       B  $596
  $598  TEST  "SKIP"
       BF  $599
       OUT  "2"
       CLM  SPEC
       BEF  B  $596
  $601  SET  "0001"
       BF  $600
  $600  BEF  B  $596
  $599  TEST  "COL"
       BF  $596
       OUT  "3"
       CLM  SPEC
       BEF  B  $596
  $596  R
  DATAFM  TEST  "A"
       BF  $605
       OUT  "8"
  $607  CLM  SPEC
       BF  $606
       B  $606
  $608  B
  $609  TEST  "E"
       BF  $607
       OUT  "0000"
       CLM  SPEC
       BEF  B  $604
  $604  B
  $605  TEST  "I"
       BF  $609
       OUT  "9"
       CLM  SPEC
       BEF  B  $604
  $604  B
  $606  BEF  B  $604
  $609  TEST  "I"
       BF  $610
       OUT  "0"
       CLM  SPEC
       BEF  B  $604
  $604  B
  $607  SET  "0000"
       BF  $606
       OUT  "0000"
       CLM  SPEC
       BEF  B  $604
  $604  B
  $610  TEST  "L"
       BF  $604
       OUT  "=
       CLM  SPEC
       BEF  B  $604
  $604  B
  $601  SET  "0001"
       BF  $600
  $600  BEF  B  $604
  $613  SPEC  TEST  ")"
       BF  $613
       INUM  B
       BEF  OUTSYM  B
       TEST  "("
00010  PROG  MTXINT
00020  SEG  04
00030  **********************************************
00040  *
00050  IDENTIFICATION:
00060  ********
00070  *
00080  PROGRAM-ID: MTXINT04.
00090  AUTHOR: J. R. VAN DOREN.
00100  SOURCE LANGUAGE: EASYCODER.
00110  SOURCE COMPUTER: H-1200.
00120  OBJECT COMPUTER: H-1200.
00130  *
00140  PURPOSE:
00150  ------
00160  *
00170  MTXINT04 INTERPRETIVELY EXECUTES OBJECT PROGRAMS PRODUCED BY
00180  THE METAX9 COMPILER-COMPILER.
00190  *
00200  **********************************************
00210  ADMODE4  ASSEMBLE IN FOUR CHAR ADDRESSING MODE
00220  ORG  45056  EXECUTION LOCATION
00230  **********************************************
00240  *
00250  OCTAL ADDRESS DEFINITIONS OF PERTINENT SYMBOLS IN THE RESIDENT
00260  INPUT/OUTPUT ROUTINE.
00270  *
00280  **********************************************
00290  #RDWR  CEQU =4C00000754
00300  READ  CEQU =4C00005430
00310  INPUT  CEQU =4C00006144
00320  OUTPUT  CEQU =4C00006265
00330  PRINT  CEQU =4C00005647
00340  #SKP  CEQU =4C00000756
00350  **********************************************
00360  *
00370  COMMUNICATION AREA FIELD LOCATION DEFINITIONS
00380*
00390***************************************************************************
00400 ELATCH EQU 242 BACKUP ERROR LATCH
00410 SYMBOL EQU 243 CURRENT SYMBOL VALUE FIELD
00420 CMPLCD EQU 236 COMPLETION CODE FIELD
00430 PSTLST EQU 239 POST LISTING OPTION FIELD
00440 INSTCT EQU 209 INSTRUCTION COUNT FIELD
00450 GENLOC EQU 215 BEGINNING CODE GENERATION POINTER
00460 LODLOC EQU 219 BEGINNING LOCATION POINTER FOR INTERP.
00470 SYMSTR EQU 231 POINTER TO BEGINNING OF SYMBOL TABLE
00480 SYMEND EQU 235 UPPER BOUND OF SYMBOL TABLE AREA
00490 STKSTR EQU 223 POINTER TO START OF CONTROL STACK AREA
00500 STKEND EQU 227 UPPER LIMIT OF CONTROL STACK AREA
00510***************************************************************************
00520*
00530* INDEX REGISTER LOCATION DEFINITIONS AND USAGE DESCRIPTIONS*
00540*  *
00550***************************************************************************
00560 IR1 EQU 4 INSTRUCTION COUNTER FOR PROGRAM BEING INTERPRETED
00570 IR2 EQU 8 SYSTEM PUSHDOWN STACK POINTER
00580 IR3 EQU 12 PROGRAM COUNTER FOR PROGRAM BEING COMPILED
00590 IR4 EQU 16 WORK REGISTER
00600 IR5 EQU 20 POINTER TO NEXT OUTPUT CODE LOCATION
00610 IR6 EQU 24 POINTER TO NEXT CHARACTER IN INPUT STRING
00620 IR7 EQU 28 USED BY INSTRUCTION FETCH
00630 IR8 EQU 32 WORK REGISTER
00640 IR9 EQU 36 WORK REGISTER
00650 IR10 EQU 40 WORK REGISTER
00660 IR11 EQU 44 WORK REGISTER
00670 IR12 EQU 48 WORK REGISTER
00680 IR13 EQU 52 WORK REGISTER
00690 IR14 EQU 56 WORK REGISTER
00700 IR15 EQU 60 POINTER TO SYMTAB ENTRY FOUND BY LAST SEARCH
00710 TF DCW :F:
00720***************************************************************************
00730*  *
00740* BEGIN PROGRAM INITIALIZATION  *
START EQU *
CAM 60 SET FOUR CHAR ADDRESSING FOR EXECUTE
SW STKEND-2 WORD MARK FOR MOVING AND TESTING
MCw STKSTR,IR2 INITIALIZE STACK POINTER
SW IR2-2 SHORTEN ARITHMETIC
SI IR2 ITEM MARK FOR RIGHT MOVE
MCw LODLOC,IR1 INITIALIZE INSTRUCTION COUNTER
SW IR1-2 SHORTEN INDEX ARITHMETIC
SI IR1 ACCOMMODATE RIGHT MOVE
BS IR3 ZERO PROGRAM COUNTER
SW IR3-2 SHORTEN ARITHMETIC
SI IR3 ACCOMMODATE RIGHT MOVE
SI IR15 ACCOMMODATE RIGHT MOVE
MCw GENLOC,IR5 INITIALIZE CODE GENERATION LOCATION
SW IR7-1 SHORTEN FETCH ARITHMETIC
MCw SYMSTR,NEWSYM INITIALIZE NEXT LOCATION IN SYMBOL TABLE
BS =1B4,SYMEND INITIALIZE
MCw SYMEND,IR14
LCA =1C77*1+X14 BLOCK
LCA NEWSYM+4*X14 LIST
MCw SYMEND,CRLKLT INIT CURRENT BLOCK LIST POINTER
MCw NEWSYM,IR15 INIT SYMTAB POINTER
MCw SYMEND,SVSYME SAVE INITIAL SYMTAB END AS START OF BLOCK
LCA NEWSYM4*14 LIST
MCw :OUTPUT+132 CLEAR PRINT
MCw OUTPUT+132 BUFFER
MCw =1C21 CARRIAGE CONTROL
SI INPUT+80 RESTORE LOST ITEM MARK ON INPUT BUFFER
SKIP TO TOP OF PAGE AND INIT INPUT BUFFER
0112C* INSTRUCTION FETCHING *
0113C* ******************************** ***********************************
01150 FETCH3 BA =1B3,IR1
01170 FETCH BA =1B1,INSTCT INSTRUCTION COUNT
01180 FIRST BS IR7-1 CLEAR SECOND CHAR
01190 MRSO O+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 O*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 TSTTBL
01470 DSA TSTTBA
01480 DSA LATCH
01490  DSA  CANCEL
01500  DSA  MARK
01510  DSA  SAVE
01520  DSA  CLM
01530  DSA  RETURN
01540  DSA  ERROR
01550  DSA  RESTOR
01560  DSA  PUSHLB
01570  DSA  POP
01580  DSA  LB1
01590  DSA  LB2
01600  DSA  S1KSYM
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  ENTLQC
01750  DSA  ERROR
01760  DSA  INUM
01770  DSA  ENUM
01780  DSA  BlKEnt
01790  DSA  B1KEXT
01800  DSA  ERROR
01810***************************************************************************
01820* ***************************************************************************
01830*  BRANCH, BRANCH TRUE, AND BRANCH FALSE PRIMITIVES  *
01840*  ***************************************************************************
01850***************************************************************************
01860 BRANCH MRID 0:X1,2
01870 B FETCH
01880 BRNCHT BCE FETCH3,TF,F
01890 B BRANCH
01900 BRNCHF BCE BRANCH,TF,F
01910 B FLTCH3
01920*************************************************************************
01930* THE SET AND SETF PRIMITIVES SET THE TRUF-FALSE INDICATOR.*
01940**************************************************************************
01960*************************************************************************
01970 SETF MCW :F,TF
01980 B FETCH
01990 SET MCW :T,TF
02000 E FETCH
0201C*************************************************************************
0202C**************************************************************************
0203C* THE CLM PRIMITVE (CALL META PROCEDURE) STACKS THE RETURN ADDRESS*
0204C* AND ERROR LATCH CODES, RESETS THE ERROR LATCH AND SETS THE*
0205C* INSTRUCTION COUNTER TO THE BEGINNING OF THE CALLED PROCEDURE.*
0206C**************************************************************************
02070*************************************************************************
02080 CLM EQU *
02090 MRID 0+X1,IR1-2
02100 SAR 4+X2 SAVE RETURN
02110 MCW :00:1+X2
02120 MRSU ELATCH,5+X2 STACK ERROR LATCH WITH RETURN ADDRESS
02130 MCW :F:ELATCH RESET ERROR LATCH FOR CALLED PROCEDURE
02140 BA =189,IR2 BUMP STACK POINTER
02150 B STKOVF CHECK FOR POSSIBLE OVERFLOW
02160*************************************************************************
02170**************************************************************************
02180* THE RETURN PROCEDURE POPS THE CONTROL STACK UNTIL A RETURN ADDRESS*
02190* IS FOUND WHICH IS SENT TO THE INSTRUCTION COUNTER. THE ERROR LATCH*
02200* PREVIOUSLY STACKED WITH THE RETURN ADDRESS IS RESTORED.*
02210**************************************************************************
02220*************************************************************************
**SEARCH AND POP**

**RETURN BS**

**BCE DORET+0+X2,00**

**UNTIL RETURN**

**RETURN ADDRESS IS FOUND**

**DORET MCW 4+X2,IR1**

**RETURN ADDRESS TO LOCATION COUNTER**

**SI IR1**

**RESTORE ITEM MARK**

**MRSU 5+X2,ELATCH**

**RESTORE ERROR LATCH UPON RETURN**

**B CANCEL**

**RETURN CANCELS ANY BACKUP LATCH**

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

**CONTROL STACK.**

**PUSHLR A LABEL**

**MRIDI LABEL-4•0+X2**

**BA =1B9,IR2**

**FETCH**

**DCw :6$:**

**ITEM MARK RIGHT**

**POP POPS THE CONTROL STACK RESTORING THE VALUE TO SYMBOL IF THE**

**TOP OF THE STACK IS MARKED AS A SYMBOL.**

**POP BS =1B9,IR2**

**BCE POPSYM,0+X2,S**

**BRANCH IF STACK TOP IS SYMBOL**

**B FETCH**

**Popsym MCw 8+X2,SYMBOL+7**

**STACK SYMBOL TO SYMBOL AREA**

**B FETCH**

**LB1 SEARCHES THE CONTROL STACK TO FIND THE FIRST LABEL SYMBOL**

**LCA IR2•IR14**

**WHICH IS THEN MOVED TO SYMBOL. THE STACK IS NOT AFFECTED.**

**LB1**

**LCA IR2•IR14**
0334C TSTAN BA =1B1,IR14
0335C MRSD 0+X14,IR13
0336C BI TSTAN,IDTAB+X13 TEST ALPHANUMERIC
0337C SI 0-1+X14 SAVE ID
0338C MRID 0+X6+SYMBOL IN
0339C SAR IR6 SYMBOL
0340C CI 0-1+X6 AREA
0341C SI SYMBOL+7
0342C MCW :T:,TF
0343C B FETCH
0344C DCW ;0:
0345C IDTAB EQU *
0346C REP 10
03470 L DC : : 0-9
03480 DC : : 
03490 REP 9
03500 L DCW : : A-I
03510 DC : :
03520 REP 9
03530 L DCW : : J-K
03540 DC : :
03550 REP 8
03560 L DCW : : S-Z
03570 DC : :
03580********************************************************************'^******
03590* *
03600* ONUM TESTS THE INPUT STRING FOR A VALID OCTAL NUMBER SETTING THE *
03610* TRUE-FALSE INDICATOR AND CONVERTING THE NUMBER TO BINARY IN SYMBOL *
03620* IF TRUE.*
03630*
03640********************************************************************
03650 ONUM B NEXT
03660 HS SYMBOL+32 CLEAR TO ZEROES
03670 MCW :F:,TF
03680 LCA =3B0,IR13
03690 BS IR14
03700 ONMTST MRSD 0+X6,IR14
MOVE STRING TO SYMBOL

IF TRUE.

SETTING THE TRUE-FALSE INDICATOR AND MOVING THE STRING TO SYMBOL

STRINST EXAMINES THE INPUT STRING FOR A CHARACTER STRING LITERAL

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

03940

03930

03920

03910

03900

03890

03880

03870

03860

03850

03840

03830

03820

03810

03800

03790

03780

03770

03760

03750

03740

03730

03720

03710

******************************************************************************
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 ENELOC PRIMITIVE CAUSES A FULL SEARCH OF THE SYMBOL TABLE AND THEN CAUSES THE PROGRAM COUNTER TO BE ENTERED AS A VALUE FOR THE ADDRESS OF THE SPECIFIED SYMBOL. LEVEL 0 (STATIC) IS ASSIGNED FOR THE STORAGE LEVEL.
ENTLOC MCw :0:,SRCHTP
SEARCH MODE

MOVE IN ADDRESS
MARK RELOCATABLE
STATIC STORAGE INDICATOR
B FETCH

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

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

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

SEARCH IS A SUBROUTINE FOR SEARCHING A BLOCK STRUCTURED SYMBOL TABLE. NOTE THE SEARCH TYPE PARAMETER (SRCHTP) WHICH MAY BE USED TO CONTROL THE SEARCH MODE. SEARCH MAY BE CALLED BY PRIMITIVES EVAL, ENTLOC, SRCHP, OR BLKEXT.
05560 BS IR14  COMPUTE LEFTMOST ADDRESS
05570 MRSB 0+X1,IR14  OF
05580 SAR IR1  TABLE FIELD
05590 BA IR15,IR14  TO BE TESTED
05600 BCE TSLIT,0-2+X1,34  TEST FOR LITERAL TEST
05610 MRID 0+X1,IR13-2  ADDRESS TO IR13
05620 SAR IR1  UPDATE INSTRUCTION COUNTER
05630 Sw 0+X13  WORD MARK TO STOP COMPARE
05640 MRIN 0+X13,0+X14  POSITION TO RIGHT END
05650 SAR IR13  SET
05660 SBR IR14  INDEX REGISTERS
05670 C 0-1+X14,0-1+X13
05680 BE SET
05690 B SETF
05700 TSLIT Sw 0+X1  WORD MARK TO STOP COMPARE
05710 MRIN 0+X1,0+X14  POSITION TO RIGHT END
05720 SAR IR1  SET
05730 SBR IR14  INDEX REGISTERS
05740 C 0-1+X14,0-1+X1
05750 BE SET
05760 B SETF
05770* .......................................................... ......................
05780* THE COMPARE PRIMITIVE COMPARES THE 2ND ADDRESSED OPERAND TO
05790* THE FIRST. THE SIX BIT CHARACTER FOLLOWING THE OPERAND ADDRESSES
05800* IS USED AS THE VARIANT OF THE CONDITIONAL BRANCH INSTRUCTION.
05810* FOLLOWING THE COMPARISON.
05820* 
05830* 
05840* .......................................................... ......................
05850 EQU *
05860 MRID 0+X1,IR13-2  FIRST ADDRESS TO IR13
05870 SAR IR1
05880 MRID 0+X1,IR14-2  SECOND ADDRESS TO IR14
05890 SAR IR1
05900 SST 0+X1,COMPT,07  INSERT CONDITIONAL BRANCH CODE
05910 BA =IR1,IR1
05920 Sw 0+X14  WORD MARK TO STOP COMPARE
MOVLIT MOVES THE LITERAL CHARACTER STRING FOLLOWING THE ADDRESS (WHICH FOLLOWS THE OP CODE) TO THE ADDRESSED LOCATION.

MOVLIT MRIU 0+X1,IR13-2 ADDRESS OF RECEIVING CHAR FIFLD
SAR IR14
MOVLIT 0+X14,0+X13 MOVE LITERAL DATA
SAR IR1 UPDATE LOCATION COUNTER
SW 0+X14
B FETCH

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

INUM INM
B FETCH

SUBROUTINE INMTESTS THE INPUT STRING FOR AN INTEGER NUMBER SETTING TRUE-FALSE CODE AND CONVERTING THE NUMBER TO BINARY IN SYMBOL IF TRUE.
06670  MCW  IR6,IR10  SAVE INPUT POINTER
06680  MCW  :F:,TF
06690  LCA  =380,IR13
06700  BS  IR14
06710  BCE  STISGN,0*X6,-  TEST FOR MINUS SIGN
06720  BCE  STISGN,0*X6,+  TEST FOR PLUS SIGN
06730  MCW  ::;ISGN  MUST BE POSITIVE
06740  B  INMST5
06750  ISGN  DCW  =1
06760  STISGN  MRSD  0*X6,ISGN  SAVE SIGN
06770  SAR  IR6  UPDATE INPUT POINTER
06780  INMST  MRSD  0*X6,IR14  TEST FOR POSSIBLE FLTNG PT NM
06790  BCE  NOINT,0*X6,0,  TEST FOR INTEGER DIGIT
06800  BIO  MVNUM,;IDTAB+X14  HAVE WE FOUND AN INTEGER
06810  BCE  NOINT,TF,F  YES, GO CONVERT TO BINARY
06820  B  CONVRT
06830  NOINT  MCW  :F:,TF  SET TRUE-FALSE INDICATOR
06840  MCW  IR10,IR6  RESTORE INPUT POINTER
06850  B  INMRTN  RETURN
06860  MVNUM  MRSDI  0*X6,SYMBOL+X13  TEST TOO MANY DIGITS
06870  SAR  IR6
06880  DA  =1Bl,IR13
06890  BCE  INMERR,IR13,10  WE HAVE PART OF AN INTEGER
06900  MCW  :T:,TF  GO LOOK FOR MORE
06910  B  INMST  CLEAR HOLD FIELD
06920  CONVRT  BS  CVRFLD  MOVE IN DECIMAL INTEGER
06930  MCW  SYMBOL-1*X13,;CVBFLD  SET SIGN IN CONVERSION FIELD
06940  SST  ISGN,;CVBFLD,60  BINARY MANTISSA IN FR 0
06950  DTB  CVBFLD+00  STORE IT
06960  TAM  CVBFLD+00
06970  C  CVBFLD-6;=2C7777  TEST NUMBER TOO LARGE FOR 24 BITS
06980  BE  INMOK
06990  C  CVBFLD-6;=2C0000
07000  BL  INMERR
07010  INMOK  EQU  *
07020  MCW  CVBFLD-2;SYMBOL+3  SAVE 24 BITS
07030  SI  SYMBOL+3
ERASE ERASES THE SPECIFIED NUMBER OF CHARACTERS FROM THE CODE STRING.

ERASE BS
MRSL SYMBOL,IR13 MOVE ERASE COUNT TO INDEX REG
BS IR13,IR3 ADJUST PROGRAM COUNTER
BS IR3,IR5 AND OUTPUT POINTERS
MCW IR5,IR14
ERTST BCE FETCH,IR13,00 TEST ERASE LOOP FINISHED
MRSLR IR13-1,0+X14 ERASE A CHARACTER
SBR IR14 NEXT CHARACTER TO ERASE
BS =1B1,IR13 DECREMENT LOOP COUNT
ERTST

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

BEF EQU *
BCE FETCH,TF,T IF TRUE CONTINUE
BCE DEFMES,BCKUP,F IF NO BACKUP BYPASS BACKUP MECHANISM
MCW SAVIN,IR6 RESTORE INPUT POINTER
BS SAVOUT,IR5 COMPUTE ERASE
MCW IR5,IR13 COUNT
BS IR5,IR3 ADJUST PROGRAM COUNTER
MCW SAVOUT,IR5 RESTORE OUTPUT POINTER
07410  MCW  IR5,IR14
07420  MCW  +ERSRTN,ERTST+4  SET UP RETURN FROM CODE ERASURF.
07430  B  ERTST  ERASE CODE
07440  ERSRTN MCW  +FETCH,ERTST+4  RESTORE INSTRUCTION IN ERASE ROUTINE
07450  B  RETURN  BACKUP CANCELS AND RETURNS
07460  DEFINE BCE  ERRRTN,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  O+X1,OUTPUT+20
07520  SAR  IR1
07530  ERRRTN EQU *
07540  MCW  :****ERROR****:OUTPUT+16
07550  L :PUT PRINT,OUTPUT,
07560  MCW  :
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,O+X1,F  IF FATAL CONTINUE TESTING
07640  B  ERMES  ELSE PRINT MESSAGE AND CONTINUE
07650  ERPASS MRIN  O+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  O+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 PRPRTN+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    MCw   IR6*IR14    COMPUTE ERROR
07800    BS   +INPUT,IR14    LOCATION
07810    MCw   :;OUTPUT+1*X14    MARK IT
07820    MCw   :1;OUTPUT    CARRIAGE CONTROL
07830L    :PUT   PRINT,OUTPUT,
07840    MCw   :;OUTPUT+1*X14    CLEAR ERROR MARK
07850    PRPRTN B    *
07860    EXIT EQU ERRFLG
07870    OVFLW EQU *
07880L    :PUT   PRINT,OVFMES,
07890    ERRFLG MCw   :F:*CMPLCD    FATAL ERROR FLAG
07900    B    Exit
07910    OVFMESDCw    :1SYMBOL TABLE OVERFLOW, JOB ABORTED:
07920    DCw    =1C45    RECORD MARK
07930*************************************************************************
07940*************************************************************************
07950*************************************************************************
07960*************************************************************************
07970*************************************************************************
07980*************************************************************************
07990    ERROR EQU *
08000L    :PUT   PRINT,OPCDMS,
08010    H
08020    B    Exit
08030    OPCDMSEDCw    :1INVALID OP CODE, JOB ABORTED:
08040    DCw    =1C45    RECORD MARK
08050*************************************************************************
08060*************************************************************************
08070*************************************************************************
08080*************************************************************************
08090*************************************************************************
08100*************************************************************************
08110*************************************************************************
08120    ENUM MCw   +DHOLD,IR12    SET POINTER TO DECIMAL HOLD FIELD
08130    BS    SCALE    CLEAR SCALE EXPONENT FIELD
08140    B    NEXT
CALL DECIMAL NUMBER RECOGNIZER

IF DNUM = TRUE THEN GO SAVE INPUT

AVOID POSSIBLE LOGICAL CONSTANT

DETERMINE FRACTION FOLLOWING

IF NONE THEN FETCH NEXT INSTRUCTION

CLEAR HOLD AREA

MOVE DECIMAL CHARACTERS

SAVE POINTER FOR MOVING FRACTION

DETERMINE FRACTION FOLLOWING

IF NONE MOVE DECIMAL FIELD FOR CONVERSION

UPDATE INPUT POINTER

IF RECORD END GO GET MORE

LOOK FOR DECIMAL FRACTION

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

TEST PLUS SIGN

TEST MINUS 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 EXPONENT SIGN FOR VALID RANGE
CONVERT NEGATIVE
SCALE EXPONENT
TO POSITIVE
SET EXPONENT SIGN FLAG
TEST SCALE EXPONENT FOR VALID RANGE
SET UP

ERROR MESSAGE

OUTPUT+25

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 FOR INDEXING

SHIFT LEFT 3 BITS TO INDEX CVTTab

CONVERSION FACTOR TO FR2

CLEAR 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
THE MARK PRIMITIVE PUSHES THE ADDRESS OF THE NEXT OUTPUT STRING LOCATION ON THE CONTROL STACK FOR LATER USE BY THE SAVE PRIMITIVE.

MARK BA =1B9,IR2 BUMP STACK POINTER
MCw IR5,0-5+X2 SAVE OUTPUT CODE ADDRESS
MCw :M:,0-9+X2 MARK STACK ELEMENT TYPE
B STKOVF CHECK STACK OVERFLOW

SAVE PUSHES THE CODE GENERATED SINCE THE LAST MARK OPERATION ON THE VARIABLE LENGTH CODE STACK AND RESETS THE OUTPUT LOCATION BACK TO THE MARKED LOCATION.

***************************************************************************
***************************************************************************
THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.

THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.

THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.

THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.

THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.

THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.

THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.

THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.

THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.

THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.

THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.

THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.

THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.

THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.

THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.

THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.

THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.

THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.

THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.

THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.

THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.

THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.

THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.

THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.

THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.

THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE CONTROL STACK.
10370* THE RESTOR PRIMITIVE RESTORES THE TOP OF THE VARIABLE LENGTH CODE
10380* STACK TO THE OUTPUT STRING.
10390*
10400***************************************************************************
10410 RESTOR LCA STKEND,IR14
10420 BCE DORES,1+X14,77 TREAT AS NO-OP IF NULL
10430 B FETCH STK
10440 DORES MCw 4+X14,STKEND RESTORE PREVIOUS STK END POINTER
10450 MCw 4+X14,IR13 ALSO TO IR13 FOR LOOP TEST
10460 BA =1B5,IR14 POINT TO CODE TO MOVE
10470 RESTST C IR13,IR14 TEST MOVE NOTE POSSIBLE NULL RESTORE
10480 BH FETCH COMPLETION
10490 MRSDR 0+X14,0+X5 MOVE CODE
10500 SAR IR14 AND ADJUST
10510 SBR IR5 POINTERS
10520 BA =1B1,IR3
10530 B RESTST
10540***************************************************************************
10550* THE CHKSYM PRIMITIVE TESTS THE EIGHT CHARACTER FIELD ON TOP OF THE
10560* CONTROL STACK AGAINST SYMBOL SETTING THE TRUE-FALSE CODE.
10570*
10580***************************************************************************
10590 CHKS YM BS =1B9,IR2 ADJUST STACK POINTER
10600 C 8+X2,SYMBOL,7 TEST AGAINST SYMBOL VALUE
10610 BE SET IF EQUAL SET TRUE
10620 B SETF ELSE SET FALSE
10630***************************************************************************
10640* SWAP SWAPS THE TOP TWO ELEMENTS ON THE CONTROL STACK.
10650*
10660***************************************************************************
10670 SWAP MCw 0-1+X2,SWPTMP MOVE TOP TO TEMPORARY
10680 SI 0-18+X2 ITEM MARK FOR NEXT MOVE
10690 MLIU 0-10+X2,0-1+X2 SWAP
10700 MCw SWPTMP,0-10+X2
10710 MCw 0+X14,0+X5
10720 MCw 4+X14,STKEND
10730 B FETCH
THE ADD AND MULT PRIMITIVES COMPRIS THE BINARY ARITHMETIC CAPABILITIES (TWO ADDRESS) OF THE METAX9 PSEUDO-MACHINE.

ADD B GTOPRA GO GET OPERAND ADDRESSES
SW 0+X11 WORD MARK TO STOP ADDITION
SW 0+X10
MRIN 0+X10,0 FIND RIGHT POSITION
SAR IR10 SET INDEX
MRIN 0+X11,0
SAR IR11 REGISTERS
BA 0-1+X10,0-1+X11 BINARY ADD
B FETCH
MUL B GTOPRA GO GET OPERAND ADDRESSES
SW 0+X10 WORD MARK TO STOP MOVE
MRIN 0+X10,0 FIND
SAR IR10 RIGHT END
BS MFLD1 CLEAR RECEIVING FIELD
MCW 0-1+X10,MFLD1 MOVE MULTIPLIER
SW 0+X11 WORD MARK TO STOP MOVE
MRIN 0+X11,0 FIND
SAR IR11 RIGHT END
BS MFLD2 CLEAR RECEIVING FIELD
MCW 0-1+X11,MFLD2 MOVE MULTIPLICAND
BIM MFLD1,MFLD2 BINARY MULTIPLY REQUIRES 24 BITS
MCW MFLD2,0-1+X11 PUT BACK RESULT
B FETCH
MFLD1 DCW =4B0
MFLD2 DCW =4B0
GTOPRA SBR GTRTN SET RETURN
MRID 0+X1,IR10-2 FIRST OPERAND ADDRESS
SAR IR1 UPDATE LOCATION COUNTER
MRID 0+X1,IR11-2 SECOND OPERAND ADDRESS
SAR IR1 UPDATE LOCATION COUNTER
11110  GTRTN B   *             RETURN TO CALLER
11120***************************************************************************
11130* THE MOVE PRIMITIVE MOVES THE FIRST ADDRESSED FIELD TO THE SECOND. *
11140****************************************************************************
11150  MOVE B GTOPRA     GET ADDRESSES IN IR10 AND IR11
11160  MRIDI 0+X10,0+X11 MOVE DATA AND TERMINATING ITEM MARK
11170  B FETCH
11180***************************************************************************
11190* THE BLOCK ENTRY PRIMITIVE CONSTRUCTS A NEW TABLE ENTRY IN THE *
11200* BLOCK LIST AND MAINTAINS BLOCK COUNTERS AND POINTERS. *
11210***************************************************************************
11220  BLKCNT DCW :0:     BLOCK COUNTER
11230  PRVBLK DCW =1C00   PREVIOUS BLOCK NUMBER
11240  CRBLKT DCW =3      INITIAL SYMTAB SEARCH ENTRY
11250  BLKENT BA =1B1,BLKCNT BLOCK COUNT
11260  BS =1B4,SYMEND     REDUCE SYMBOL TABLE END LOCATION
11270  MCW SYMEND,IR13   TO ACCOMMODATE NEW BLOCK LIST ENTRY
11280  LCA PRVBLK,1+X13  SET SURROUNDING BLOCK NUMBER
11290  MCW BLKCNT,PRVBLK SET UP FOR NEXT BLOCK ENTRY
11300  BA =1B20,NEWSYM   SPACE FOR DUMMY ENTRY
11310  LCA NEWSYM,4+X13  STORE IN BLOCK LIST
11320  MCW SYMEND,CRBLKT SET FOR CURRENT SYMTAB SEARCH
11330  B FETCH
11340***************************************************************************
11350* THE BLOCK EXIT PRIMITIVE RESTORES CRBLKT POINTER AND PRVBLK *
11360* NUMBER FOR THE SURROUNDING BLOCK. THE SYMBOL TABLE ENTRIES FOR *
11370* THE TERMINATING BLOCK ARE SCANNED FOR UNRESOLVED SYMBOLS. *
11380* UNRESOLVED ENTRIES ARE ADDED TO THE SURROUNDING BLOCK IF NOT *
11390* FOUND IN THAT PORTION OF THE TABLE. APPROPRIATE LINKING *
11400* PARAMETERS ARE SET FOR THE RESOLVE PRIMITIVE TO USE. THUS *
11410* DIAEOLICAL LABEL REFERENCES IN A BLOCK STRUCTURE ARE RESOLVABLE. *
BLKSAV DCw = 3
BLKPRM DCw = 1
BLKEXT Cw CRBLKT·BLKSAV SAVE FOR UNRESOLVED SEARCH
MRSD Cw 0·X1·BLKPRM BLOCK EXIT PARAMETER
SAR Cw IR1
BS Cw IR13 CLEAR
MCw CRBLKT·IR14 INSERT PREVIOUS BLOCK NUMBER
MCw 1·X14·IR13 NEW SURROUNDING BLOCK NUMBER
MCw 1·X14·PRVBLK COMPUTE
MCw =484·IR13
MCw SVSYME·IR14 BLOCK LIST
BS Cw IR13·IR14 LOCATION FOR SURROUNDING BLOCK
MCw IR14·CRBLKT BLOCK LIST POINTER
MCw BLKSAV·IR12 TERMINATING BLOCK LIST POINTER
MCw 4·X12·IR12 SYMBOL TABLE POINTER
MCw FETCH·BLKPRM·01 TEST NO LABEL LINK UP
CHKUND Cw CHAIN·X12·:000: CHECK FOR END OF
BE FETCH BLOCK TABLE ENTRIES
MCw CHAIN·X12·IR12 NOTE 1ST TIME JUMP OVER DUMMY ENTRY
UDCHK BCE CHKPRV·DTYPE·X12·00 CHECK FOR UNDEFINED SYMBOLS
B CHKUND LOOK FOR MORE
CHKPRV MCw NAME·X12·SYMBOL·7 SET NAME TO USE SEARCH SUBROUTINE
MCw :1·SRCHTP BLOCK ONLY SEARCH MODE
B SEARCH
ECE ADDSYM·TF·F IF FALSE SYMBOL ADDED TO SURROUNDING
PRVUN·0·X15·00 IF FOUND BUT STILL UNDEFINED SET MARKERS
BCE PRVUN·0·X15 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
ADDw EQU *
PRVUN Sw IR15·2
MCw IR12·IR9 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
CLEAR IF FATAL COMPILATION TO THIS POINT

MOVE PROGRAM SIZE
LOAD TO FRO
CONVERT TO DECIMAL AND STORE
EDIT CONTROL WORD
MOVE AND EDIT
CLEAR WORD MARK
CLEAR
MOVE INSTRUCTION COUNT
LOAD TO FRO
CONVERT TO DECIMAL AND STORF
EDIT CONTROL WORD
MOVE AND EDIT
CLEAR WORD MARK
PRINT,EXITMS
PRINT IT
MOVE SEARCH COUNT
MOVE PROGRAM SIZE
LOAD TO FRO
CONVERT TO DECIMAL AND STORF
EDIT CONTROL WORD
MOVE AND EDIT
CLEAR WORD MARK
MOVE SEARCH COUNT
MOVE PROGRAM SIZE
LOAD TO FRO
CONVERT TO DECIMAL AND STORF
EDIT CONTROL WORD
MOVE AND EDIT
CLEAR WORD MARK
MOVE SEARCH COUNT
MOVE PROGRAM SIZE
LOAD TO FRO
CONVERT TO DECIMAL AND STORF
EDIT CONTROL WORD
MOVE AND EDIT
CLEAR WORD MARK
MOVE SEARCH COUNT
12220  MCW  CMPCNT, CVBFLD-2  MOVE COMPARISON COUNT
12230  TMA  CVBFLD, 00
12240  BTD  CVBFLD, 00
12250  LCA  EWORD, TCOUNT
12260  MCE  CVBFLD, TCOUNT
12270  :PUT  PRINT, TAMES,  
12280  BS  CVBFLD
12290  MCW  SYMCNT, CVBFLD-2  MOVE TABLE ENTRY COUNT
12300  TMA  CVBFLD, 00
12310  BTD  CVBFLD, 00
12320  LCA  EWORD, ECOUNT
12330  MCF  CVBFLD, ECOUNT  MOVE AND EDIT
12340  LCA  GENLOC, IR15  START OF COMPILED CODE
12350  SCAN MRWN 0+X15, 0  SCAN FOR WORD MARK
12360  SAR IR15  SAVE NEXT POSITION
12370  C IR15, IR5  DETERMINE
12380  BL PLIST  COMPLETION
12390  BBE GETADD 0+1+X15, 40  TEST UNRESOLVED ADDRESS
12400  SCAN B SCAN
12410  GETADD MRIDI 0+1+X15, IR14-2  UNRESOLVED ADDRESS IS POINTER TO SYMBOL
12420  *  MOVE IT TO IR14.
12430  HA =1C40, IR14-2  REMOVE UNRESOLVED MARKER
12440  C SYMSTR, IR14  TEST FOR
12450  BL STERR  VALID
12460  C SYMEND, IR14  SYMBOL
12470  BH STERR  ADDRESS
12480  CHNTST BCE CHNADD+ETYPE+X14, 77
12490  BBE NDEFN+ADDR-2+X14, 40  IF STILL UNDEFINED PRINT ERROR
12500  B ADDRL
12510  CHNADD MCW CHAIN+X14, IR14  CHAIN TO SURROUNDING BLOCK
12520  B CHNTST  GO TEST FOR ADDITIONAL CHAINING
12530  ADDRSL SI ADDR+X14
12540  MRIDR LEVEL+X14, 0-2+X15  SET LEVEL AND DISPLACEMENT
12550  BCE LBVTST, 0-3+X15, 76  TEST POSSIBLE LABEL VARIABLE - LABEL
12560  *  CONSTANT RESOLUTION
12570  B SCAN
12580  Scanner
12590 LBVTST BE LBV$DTYPEx14.05 TEST LABEL VARIABLE IN SYMTAB
12600 B SCAN ELSE CONTINUE
12610 LBV BNP SCAN$0-4x15 MAKE SURE OP CODF PRECEDES ADDR
12620 MRSD $5:x0-3x15 CHANGE DATA TYPE TO LABEL VARIABLE
12630 B SCAN AND CONTINUE
12640 STERR EQU *
12650L :PUT PRINT$CMPLM5,
12660 H
12670 B EXIT
12680 CMPLMSDCw :ACOMPIILER ERROR DISCOVERED DURING RESOLVE:
12690 L DCw =1C45
12700 NTDEFN MCw NAME$X14,PSYM
12710L :PUT PRINT,NDMFES5,
12720 MCw :F:,CMPLCD SET COMPLETION CODE
12730 B SCAN
12740 NDFMESDCw :1UNDEFINED SYMBOL :
12750 PSYM DCw :
12760 L DCw =1C45
12770 TABCNTDCw :A ****SYMBOL TABLE ENTRY COUNT =:
12780 ECOUNT DC =9
12790 DC :****:
12800 L DCw =1C45
12810 EXITMSDCw :2 ****COMPILED PROGRAM SIZE =:
12820 PSIZE DC =9
12830 DC :: METAX INSTRUCTION COUNT =:
12840 ICOUNT DC =9
12850 DC :****:
12860 L DCw =1C45
12870 SRCHCT DCw =4B0
12880 CMPCNT DCw =4B0
12890 TABMESCw :B ****SYMTAB SEARCH COUNT = :
12900 SCOUNT DC =9
12910 DC :: SYMTAB COMPARE COUNT = :
12920 TCOUNT DC =9
12930 DC :****:
12940 L DCw =1C45
12950 EWORD DCw :: 0 :
EXIT IS A TERMINAL POINT IN THE PROGRAM, CLEARING CERTAIN PUNCTUATION BEFORE EXITING.

EXIT EQU *

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

NEXT IS A SUBROUTINE WHICH SCANS THE INPUT STRING FOR THE NEXT NON-BLANK CHARACTER READING NEW RECORD(S) IF REQUIRED. IF AN END OF FILE IS SENSED A MESSAGE IS PRINTED AND THE PROGRAM EXITS.

NEXT SBR NxTRTN+4
ENDTST BI GETCRD+0+x6
BLKTST BCE NBLNK+0+x6,15
NXTRTN B *
NBLNK BA =181,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
.EOFMESDCW :UNEXPECTED END OF FILE, JOB ABORTED:
DCw =1C45 RECORD MARK
PLIST IS EXECUTED IF A POST LISTING IS REQUESTED.

EXIT IF POST LIST NOT REQUESTED

PROGRAM COUNTER

START OF GENERATED CODE

CLEAR

PRINT LINE

CARRIAGE CONTROL

RESTORE LOST ITEM MARK ON INPUT BUFFER

CLEAR CONVERSION FIELD

LOAD TO FRO

CONVERT TO DECIMAL

REMOVE SIGN BITS

DETERMINE SYMBOLIC LABEL, IF ANY

JUMP OVER LITERAL DESIGNATOR

SET UP

Determine symbolic label, if any.

Jump over literal designator.

Test completion.

Kill first time branch.
13700 BA $1B1,IR13 BUMP PROGRAM COUNT
13710 BA $1B1,IR15 AND CODE POINTER
13720 BCE FMTCD,0-1+X15,77 DETERMINE FORMAT CODE
13730 BCE ALLOC,0-1+X15,12 TEST ALLOC OP CODE
13740 BCE LDA,0-1+X15,20 TEST LOAD ADDRESS OP CODE
13750 BS IR12 CLEAR
13760 MRSD 0-1+X15,IR12 INSERT OP CODE
13770 BCE =486,IR12 MULT BY TABLE ENTRY SIZE
13780 MCW OPTAB+4+X12,OUTPUT+21 OP CODE TO PRINT
13790 BCE PLSTPR,OPTAB+5+X12,00 TEST FOR NO OPERANDS
13800 BBE TLITRL,OPTAB+5+X12,60 TEST POSSIBLE LITERAL
13810 LITERL EQU LITRL
13820 BCE LITERL,OPTAB+5+X12,01 TEST SINGLE CHARACTER LITERAL
13830 BCE ADDR4,OPTAB+5+X12,04 TEST FOUR CHAR ADDRESS
13840 BCE TWOOP,OPTAB+5+X12,10 TEST TWO OPERANDS
13850 B LITERL
13860 TWOOP B ADDFV
13870 MCW IR14,IR11
13880 B ADDFV
13890 MCW NAME+X11,OUTPUT+30
13900 MCW ::,OUTPUT+31
13910 MCW NAME+X14,OUTPUT+39
13920 B PLSTPR
13930 ALLOC MCW ::ALLOC::,OUTPUT+21 SET OP CODE
13940 B ADDFV GET ADDRESS OF FIRST SYMBOL
13950 MCW IR14,IR11 SAVE IT
13960 B ADDFV SECOND SYMBOL
13970 MCW NAME+X11,OUTPUT+30 FIRST SYMBOL TO PRINT
13980 MCW ::,OUTPUT+31
13990 MCW NAME+X14,OUTPUT+39 SECOND SYMBOL TO PRINT
14000L :PUT PRINT,1 PUT
14010 MCW ::,OUTPUT+132 CLEAR
14020 MCW OUTPUT+132
14030 MCW $1C21,OUTPUT CARRIAGE CONTROL
14040L :PUT PRINT,DPVMES, DOPE VECTOR MESSAGE
14050 BS IR12 CLEAR
14060 MRSD 0+X15,IR12 INSERT DIM COUNT
1407C  BIM  =4810,IR12  MULT BY SIZE OF BOUND PAIR CODE
14080  BA  =1B3,IR12  SIZE OF LENGTH AND DIM COUNT FIELDS
14090  BA  IR12,IR13  BUMP PROG COUNTER
14100  BA  IR15,IR12  END OF DOPE VECTOR
14110  SW  0-1,X12  MARK FOR MOVE
14120  MCRW  ":" ;OUTPUT+15  MOVE IT
14130  MRRD  0+X15,OUTPUT+16  MOVE IT
14140  SAR  IR15
14150  SBR  IR11
14160  CWS  0-1+X12
14170  MCW  ":" ;0+X11
14180  B  PLSTPR  GO PRINT
14190  DPVMESDCW  :A  ***DOPE VECTOR CODE***:
14200  LDCW  =1C45
14210  LDA  MCW  ":LDA ":OUTPUT+21
14220  SST  0+X15,LITCHR,70  LEFT THREE BITS
14230  BCE  LITOPR,LITCHR,70  TEST FOR LITERAL
14240  SST  0+X15,OYPE,07  SAVE TYPE
14250  B  ADDFV  GET OPERAND SYMBOL
14260  BCE  LNGCDE,OYPE,04  TEST FOR STRING TYPE
14270  B  PLSTPR  GO PRINT
14280  LNGCDE  MCW  ":" ;OUTPUT+32
14290  MRSD  0+X15,OUTPUT+33  MOVE IN
14300  EXM
14310  SAR  IR15
14320  MCW  ":" ;OUTPUT+35
14330  BA  =1B2,IR13
14340  B  PLSTPR
14350  OYPE  DCW  :0:
14360  ADDR4  B  ADDFR
14370  BCE  I0TYPE,0-5+X15,50  CHECK I/O SETUP
14380  B  PLSTPR
14390  I0TYPE  MCW  ":" ;OUTPUT+32  I/O TYPE FOR PRINTING
14400  MRSD  0+X15,OUTPUT+33
14410  SAR  IR15
14420  BA  =1B1,IR13
14430  MCW  ":" ;OUTPUT+34
14440  B  PLSTPR
14450  ADDR5 B  ADDFV
14460  B  PLSTPR
14470  ADDFR SBR  AFRRTN+4  SET RETURN ADDRESS
14480  MRID  0+X15,ADD4CN-3  MOVE ADDRESS
14490  SAR  IR15
14500  BA  =1B4,IR13  BUMP CODE COUNTER
14510  MCw  SYMSTR,IR14  TABLE START
14520  A4COMP C  4+X14,ADD4CN  TEST
14530  BE  ADFRFD
14540  BA  =1B20,IR14  NEXT
14550  C  IR14,NEWSYM  TEST
14560  BEH  A4COMP  TABLE END
14570  MISSAD MCw  ;*****:,OUTPUT+35  MISSING SYMBOL MARKER
14580  B  LITOPR
14590  ADFRFD MCw  NAME+X14,OUTPUT+30  MOVE SYMBOL
14600  AFRRTN B *  RETURN
14610  ADD4CN DCw  =4
14620  ADDFV SBR  AFRRTN+4
14630  MRSD  0+X15,ADD5CN-4  GET ALL FIVE CHARACTERS
14640  EXM
14650  EXM
14660  EXM
14670  EXM
14680  SAR  IR15
14690  BA  =1B5,IR13
14700  MCw  SYMSTR,IR14
14710  A5COMP SI  4+X14  ITEM MARK FOR MOVE
14720  MRID  0+X14,ADD5CN1-4  MOVE DATA, NO WORD MARKS
14730  C  ADD5CN+ADD5CN1  TEST EQUALITY
14740  BE  ADFVFD
14750  BA  =1B20,IR14
14760  C  IR14,NEWSYM  TEST
14770  BEH  A5COMP
14780  BS  =1B5,IR15
14790  BS  =1B5,IR13
14800  B  MISSAD
14810 ADFVFD MCW NAME=X14,OUTPUT+30
14820 AFVRTN B *
14830 ADD5CN DCW =5
14840 ADD5CN DCW =5
14850 SYMADD B OPCODE FIRST TIME ONLY
14860 MCW SYMSTR,IR14 TABLE START
14870 REL O CT BNP NOSYM*=X14 IGNORE NON-RELOCATABLE SYMBOLS
14880 C 4+X14,IR13 TEST
14890 BE SYMFND EQUALITY
14900 NOSYM BA =1B20,IR14 NEXT
14910 C SYMEND,IR14 TEST
14920 BH OPCODE TABLE END
14930 B REL O CT TEST NEXT
14940 SYMFND MCW NAME=X14,OUTPUT+15
14950 B 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,FMTME5, MESSAGE
15050 MCW "";OUTPUT+15 BUILD
15060 MCW +OUTPUT+16,IR12 FORMAT
15070 NFMTCH MRSD 0+X15,0+X12 LITERAL
15080 SAR IR15
15090 SBR IR12
15100 BA =1B1,IR13
15110 BCE FMTDNE+0+X15,77
15120 B NFMTCH
15130 FMTDNE MCW :"";0+X12 FINISH
15140 BA =1B1,IR13 BUMP
15150 BA =1B1,IR15 COUNTERS
15160 B PLSTPR
15170 FMTMESDCW :A ***FORMAT CODE***:
<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15180</td>
<td>L</td>
<td>DCw =1C45</td>
</tr>
</tbody>
</table>
| 15190   | OPTAB       | EQU *       | OP CODE OPERAND TABLE FOR POSTLISTING
| 15200   | REP 8       |             |
| 15210   | DCw :ERRORO:|             |
| 15220   | DCw :DYNAM4:|             |
| 15230   | DCw :STKTP8:|             |
| 15240   | DCw :ALLOC4:|             |
| 15250   | REP 5       |             |
| 15260   | DCw :ERRORO:|             |
| 15270   | DCw :LDA 5: |             |
| 15280   | DCw :LD 5:  |             |
| 15290   | DCw :STQ 0: |             |
| 15300   | DCw :SST 0: |             |
| 15310   | DCw :FLAG 0:|             |
| 15320   | DCw :ENTPR1:|             |
| 15330   | DCw :RETRNO:|            |
| 15340   | DCw :JUMPA0:|              |
| 15350   | DCw :JUMP 4:|              |
| 15360   | DCw :JUMPT4:|             |
| 15370   | DCw :JUMPF4:|             |
| 15380   | DCw :STCKC1:|              |
| 15390   | DCw :COMPC1:|              |
| 15400   | DCw :SWAP 0:|              |
| 15410   | DCw :POPUPO:|             |
| 15420   | DCw :ERRORO:|             |
| 15430   | DCw :ADD 0: |              |
| 15440   | DCw :MULT 0:|              |
| 15450   | DCw :SUB 0: |              |
| 15460   | DCw :DIV 0: |              |
| 15470   | DCw :NEG 0: |              |
| 15480   | REP 3       |              |
| 15490   | DCw :ERRORO:|             |
| 15500   | DCw :FMT 4: |              |
| 15510   | DCw :GET 0: |              |
| 15520   | DCw :PUT 0: |              |
| 15530   | DCw :EDIT 0:|              |
| 15540   | REP 4       |              |

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
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A4COMP</td>
<td>14520;</td>
<td>A5COMP</td>
<td>14710;</td>
</tr>
<tr>
<td>ADD</td>
<td>10810;</td>
<td>ADDFR</td>
<td>14470;</td>
</tr>
<tr>
<td>ADDR</td>
<td>04950;</td>
<td>ADDRS</td>
<td>12540;</td>
</tr>
<tr>
<td>AFRRTN</td>
<td>14600;</td>
<td>AFVR</td>
<td>14820;</td>
</tr>
<tr>
<td>BLKCN</td>
<td>11260;</td>
<td>BLK</td>
<td>11290;</td>
</tr>
<tr>
<td>BLKT</td>
<td>13180;</td>
<td>BM</td>
<td>07610;</td>
</tr>
<tr>
<td>CANCEL</td>
<td>06190;</td>
<td>CHAIN</td>
<td>04920;</td>
</tr>
<tr>
<td>CHNADD</td>
<td>12520;</td>
<td>CHN</td>
<td>12490;</td>
</tr>
<tr>
<td>CMPLMS</td>
<td>12680;</td>
<td>CMP</td>
<td>05850;</td>
</tr>
<tr>
<td>CRBLKT</td>
<td>11280;</td>
<td>CTABI6</td>
<td>09640;</td>
</tr>
<tr>
<td>DECMVE</td>
<td>08340;</td>
<td>DEFMES</td>
<td>07460;</td>
</tr>
<tr>
<td>DNMRTN</td>
<td>08710;</td>
<td>DNMT</td>
<td>08670;</td>
</tr>
<tr>
<td>DORET</td>
<td>02260;</td>
<td>DOAVE</td>
<td>10020;</td>
</tr>
<tr>
<td>ELATCH</td>
<td>00400;</td>
<td>ENDT</td>
<td>13170;</td>
</tr>
<tr>
<td>ENLCLC</td>
<td>04450;</td>
<td>ENUM</td>
<td>08120;</td>
</tr>
</tbody>
</table>

**SYMBOL DEFINITION - CARD REFERENCE INDEX**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>A4COMP</td>
<td>14520;</td>
</tr>
<tr>
<td>ADD</td>
<td>10810;</td>
</tr>
<tr>
<td>ADDR</td>
<td>04950;</td>
</tr>
<tr>
<td>AFRRTN</td>
<td>14600;</td>
</tr>
<tr>
<td>BLKCN</td>
<td>11260;</td>
</tr>
<tr>
<td>BLKT</td>
<td>13180;</td>
</tr>
<tr>
<td>CANCEL</td>
<td>06190;</td>
</tr>
<tr>
<td>CHNADD</td>
<td>12520;</td>
</tr>
<tr>
<td>CMPLMS</td>
<td>12680;</td>
</tr>
<tr>
<td>CRBLKT</td>
<td>11280;</td>
</tr>
<tr>
<td>DECMVE</td>
<td>08340;</td>
</tr>
<tr>
<td>DNMRTN</td>
<td>08710;</td>
</tr>
<tr>
<td>DORET</td>
<td>02260;</td>
</tr>
<tr>
<td>ELATCH</td>
<td>00400;</td>
</tr>
<tr>
<td>ENLCLC</td>
<td>04450;</td>
</tr>
</tbody>
</table>
TABLES 12890; TCOUNT 12920; TEST 03080; TESTLB 02600; TESTT 03170;
TF 00710; TLITRL 14990; TSTAN 03340; TSTLB1 02730; TSTLB2 02760;
TSTLIT 05700; TSTSCL 08870; TSTTBA 05540; TSTTEL 05550; TVEC 01250;
Twoop 13860; UUCHK 11690; UPDATE 08540;

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

//
TEST: PROCEDURE MAIN;
/* */
/* IDENTIFICATION: */
/* */
/* PROGRAM-ID: TEST */
/* AUTHOR: J. R. VAN DOREN */
/* SOURCE LANGUAGE: PLEX */
/* OBJECT LANGUAGE: PLEX PSEUDO-MACHINE CODE */
/* OBJECT INTERPRETER: PLXINT */
/* */
/* PURPOSE: */
/* */
/* TEST DEMONSTRATES MOST OF THE FEATURES OF THE PLEX LANGUAGE */
/* */
/****************************%**************************%*****************/

/* STRINGS AND SUBSTRINGS */

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

IOBLK:BEGIN;
DECLARE (A,B,C,M(5)) FIXED, (X,Y,Z) FLOAT, (5,T) CHAR(20);
PUT EDIT ("BEGIN I/O BLOCK") (SKIP(3),A);
GET EDIT (S) (A(15));
GET EDIT (A,B,C) (COL(5),I(5),I(5),I(5));
PUT STRING (T) EDIT (A,B,C) (I(5));
PUT EDIT (S,T,A,B,C) (SKIP,A(COL(20)),A,COL(40),I(5),I(5),I(5));
GET EDIT (S,T) (SKIP,A20, A(20));
GET STRING (5) EDIT (X,Y) (E(10));
GET STRING (T) EDIT (Z) (E(10));
PUT EDIT (X,Y,Z,S,T) (SKIP(2),T(20),E(20),E(20),SKIP,A(20),A(20));
DO A=1 TO 5;
  M(A)=A;
END;
PUT EDIT ((M(A) DO A=1 TO 5)) (I(5));
PUT EDIT ("EXIT I/O BLOCK") (SKIP,A);
END IOBLK;

DO GROUPS

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

/*
   ARITHMETIC
   */

ARITH:BEGIN; DECLARE (X,Y,Z) FLOAT, (A,B,C) FIXED;
PUT EDIT ("ENTER ARITHMETIC BLOCK") (SKIP,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) (I(5));
END;
PUT EDIT ("EXIT ARITHMETIC BLOCK") (SKIP,A);
END ARITH;

/*
   PROCEDURE CALLS AND RECURSION
   */

/* RECURSIVE FACTORIAL EXAMPLE */

RPROC:BEGIN; DECLARE NFACT RETURNS(FIXED);
NFACT: PROCEDURE (I); DECLARE I FIXED;
   IF I=0 THEN RETURN (1);
   RETURN (NFACT((I-1))*I);
END NFACT;
PUT EDIT ("ENTER RPROC") (SKIP(3),A);
PUT EDIT ("7 FACTORIAL =",NFACT(NFACT(3)+1)) (SKIP,A,I(10));
PUT EDIT ("EXIT RPROC") (SKIP,A);
END RPRGC;
/* TRANSLATION OF INFIX ARITHMETIC EXPRESSIONS TO POSTFIX
FORM USING RECURSIVE PROCEDURES */
POSTF:BEGIN;
DECLARE (LITERAL,NBR,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.);
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. EXP 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\$X(2),A(80));
OUTPUT=INPUT; /* CLEAR OUTPUT FIELD. */
IF EXP1 THEN PUT EDIT ("POSTFIX EXPRESSION =",SUBSTR(OUTPUT,1,J-1)/";")
(SKIP,A\$X(2),A);
ELSE PUT EDIT ("****ERROR****") (SKIP,A);
PUT EDIT ("EXIT POSTFIX") (SKIP,A);
END POSTF;

PROCEDURE PARAMETER EXAMPLE TO TEST GLOBAL DISPLAY

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

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

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

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

LABEL:BEGIN; DECLARE (Y,Z(3)) LABEL, (I,J,K) FIXED;
LBL:PROCEDURE(LABEL); DECLARE LABEL(*) LABEL;
GO TO LABEL(3);
END LBL;
PUT EDIT ("ENTER LABEL") (SKIP(3)•A);
Y=LBL2;
BEGIN;
  I=1;
  GO TO LBL1;
  K=I/2;
LBL1: PUT EDIT ("LABEL TEST" •I) (SKIP•A•I(5));
  J=I+1;
  GO TO Y;
LBL2: PUT EDIT("INCORRECT LABEL TEST") (A);
END;
LBL1: PUT EDIT("INCORRECT LABEL TEST") (A);
LBL2: PUT EDIT("LABEL TEST",J) (SKIP•A•I(5));
  Z(1)=BADLAB; *
  Z(2)=BADLAB;
  Z(3)=GOODLAB;
  CALL LBL(Z);
BADLAB: PUT EDIT("INCORRECT LABEL RETURN") (SKIP•A);
GOODLAB: PUT EDIT("CORRECT LABEL RETURN") (SKIP•A);
PUT EDIT("EXIT LABEL") (SKIP•A);
END LABEL;
END TEST;

****COMPILED PROGRAM SIZE = 6,325; 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 STK TP $STKT0 $STKT1
00026 $003 FMT $004 "1"
00032 LDA "BEGIN STRINGS"
00047 EDIT
00052 PUT
00053 JUMP $006

***FORMAT CODE***
00060 $004 "2000380000"
00065 $005 LDA S "OL"
00070 LDA "THIS IS A STRING."
00093 STO
00094 FMT $006 "1"
00100 LDA S "OL"
00103 EDIT
00109 PUT
00114 JUMP $007

***FORMAT CODE***
00116 $006 "80000"
00122 $007 LDA T "OL"
00130 LD "0001"
00136 LD "0004"
00142 SbSTR
00143 LDA "THIS"
00149 STO
00150 LDA T "OL"
00156 LD "0005"
00162 LD "0006"
00168 SbSTR
00171 LDA S "OL"
00179 LD "0005"
00185 LD "0006"
00191 SbSTR
00192 LDA "CONCATENATED SUBSTRING."
00217 CAT
00210:  STO
00219:  FMT   $008  ,"1"
00225:  LDA   T   ,"OL"
00233:  EDIT
00234:  PUT
00235:  JUMP  $009
     ***FORMAT CODE***
00240:  $008  "8000J"
00247:  $009  LDA   S   ,"OL"
00255:  LD   "0001"
00261:  LD   "0005"
00267:  SBSTR
00268:  LDA   "THIS"
00274:  COMPC   "1"
00275:  JUMPF  $010
00281:  FMT   $011  ,"1"
00287:  LDA   "STRING COMPARE 1 WORKS"
00311:  EDIT
00312:  PUT
00313:  JUMP  $010
     ***FORMAT CODE***
00318:  $011  "80000"
00325:  $010  LDA   S   ,"OL"
00333:  LD   "0001"
00339:  LD   "0004"
00345:  SBSTR
00346:  LDA   T   ,"OL"
00354:  LD   "0001"
00360:  LD   "0004"
00366:  SBSTR
00367:  COMPC   "1"
00369:  JUMPF  $013
00374:  FMT   $014  ,"1"
00380:  LDA   "STRING COMPARE 2 WORKS"
00404:  EDIT
00405:  PUT
00406:  JUMP  $013
***:FORMAT CODE***
00411 $014 "80000"
00418 $013 FMT $016 "1"
00424 LDA "EXIT STRINGS"
00430 EDIT
00434 PUT
00440 JUMP $017
***:FORMAT CODE***
00445 $016 "2000180000"
00457 $017 STKTP $STKTO $STK1
00468 ALLOC $STK1 M
***DOPE VECTOR CODE***
"10*Z0001Z0005"
00492 JUMP $018
00497 $018 FMT $019 "1"
00503 LDA "BEGIN I/O BLOCK"
00520 EDIT
00521 PUT
00522 JUMP $020
***:FORMAT CODE***
00527 $019 "2000000000"
00539 $020 FMT $021 "Z"
00545 LDA S "0D"
00553 GET
00554 JUMP $022
***:FORMAT 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
***:FORMAT CODE***
00598 $023 "30001:0005:0005:0005"
00620 $024  LDA  T ,"0D"
00623  FMT $025 ,"0"
00630  LD   A
00640  EDIT
00643  LD   B
00647  EDIT
0064F  LD   C
00654  EDIT
00655  PUT
00656  JUMP $026

***FORMAT CODE***
00661 $025 "0005"
00663 $026  FMT $027 ,"1"
00674  LDA  S ,"0D"
00682  EDIT
00683  LDA  T ,"0D"
00691  EDIT
00692  LD   A
00693  EDIT
00699  LD   B
00705  EDIT
00705  LD   C
00712  EDIT
00713  PUT
00714  JUMP $028

***FORMAT CODE***
00719 $027 "200180000D8000000000000000005000500050005"
00761 $028  FMT $029 ,"Z"
00767  LDA  S ,"0D"
00775  GET
00776  LDA  T ,"0D"
00784  GET
00785  JUMP $030

***FORMAT CODE***
00790 $029 "20018000D8000D"
00807 $030  LDA  S ,"0D"
00815  FMT $031 ,"Y"
0082: LDA X
0087: GET
0088: LDA Y
0089: GET
008b: JUMP $032

***FORMAT CODE***
0084: $031 "9000" "0D"
0085: LDA T
0086: FMT $033 "Y"
0087: LDA Z
0088: GET
0089: JUMP $034

***FORMAT CODE***
0083: $033 "9000"
0084: $034 FMT $035 "1"
0085: LD X
0086: EDIT
0087: LD Y
0088: EDIT
0089: LD Z
008a: EDIT
008b: LDA S
008c: EDIT
008d: LDA T
008e: EDIT
008f: EDIT
0090: PUT
0091: JUMP $036

***FORMAT CODE***
0093: $035 "200029000D9000D9000D200018000D8000D"
0096: $036 LDA A
0097: LD "0001"
0098: SST
0099: $038 LD "0005"
009a: STCKC "4"
009b: JUMPT $037
009c: LDA M
009d: LD A
01006  INDXA "1"
01008  LD  A
01014  STO
01015  LDA A
01021  LD  A
01027  LD  "0001"
01033  ADD
01034  SST
01035  JUMP $038
01040  $037  FMT  $039  "1"
01046  LDA A
01052  LD  "0001"
01058  SST
01059  $041  LD  "0005"
01065  STCKC "4"
01067  JUMPT $040
01072  LDA M
01073  LD  A
01084  INDXR "1"
01085  EDIT
01087  LDA A
01093  LD  A
01099  LD  "0001"
01105  ADD
01106  SST
01107  JUMP $041
01112  $040  PUT
01113  JUMP $042
  ***FORMAT CODE***
01118  $039  "0005"
01125  $042  FMT  $043  "1"
01131  LDA  "EXIT I/O BLOCK"
01147  EDIT
01148  PUT
01149  JUMP $044
  ***FORMAT CODE***
01154  $043  "2000180000"
01166 $044   STKTP $STK0  $STK1
01177   ALLOC $STK1  $M
***DOPE VECTOR CODE***
 "204Z0000Z0001Z0000"
01211    JUMP $045
01216   $045   FMT $046  "1"
01222    LDA "BEGIN DO GROUPS"
01237    EDIT
01240    PUT
01244    JUMP $047
***FORMAT CODE***
01246 $046   "2000380000"
01258 $047   LDA I
01264    LD  "0000"
01270   $ST
01271 $049   LD  "0004"
01277   STCKC "4"
01279   JUMPT $048
01284    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   "800000"
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      JUMPS $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
01503      $052 LDA I
01514      LD I
01520  LD   "0001"
01526  ADD
01527  SST
01528  JUMP $049
01535  $048   LDA I
01536  LD   "0001"
01545  SST
01546  $070   LD   "cccc"
01552  STCKC "2"
01554  JUMPT $069
01560  LDA M
01565  LD   I
01571  LD   "0005"
01577  INDXA "2"
01579  LD   "0000"
01585  STO
01586  LDA A
01592  LD   "0003"
01598  LD   "0002"
01604  MULT
01605  LD   "0005"
01611  SUB
01612  SST
01613  $072   LD   "0004"
01619  STCKC "4"
01621  JUMPT $071
01626  LD   A
01632  LD   "0005"
01639  STCKC "2"
01640  JUMPF $071
01645  LDA M
01651  LD   I
01657  LD   A
01663  INDXA "2"
01665  LD   A
01671  STO
01672  LDA A
01676 LD A
01684 LD "0001"
01690 ADD
01691 SST
01692 JUMP $072
01697 $071 FMT $073 ,"1"
01703 LDA "I = "
01705 EDIT
01711 LD I
01715 EDIT
01717 LDA , M(I,1) = "
01731 EDIT
01733 LDA M
01737 LD I
01745 LD "0001"
01749 INDXR "2"
01753 EDIT
01755 LDA , M(I,5) = "
01763 EDIT
01765 LDA M
01775 LD I
01775 LD "0005"
01781 INDXR "2"
01785 EDIT
01787 PUT
01788 JUMP $074

***FORMAT CODE***
01793 $073 "80000*0005"
01805 1074 LDA I
01813 LD I
01817 LD "0001"
01823 SUB
01829 SST
01825 JUMP $070
01830 $069 FMT $075 ,"1"
01836 LDA "EXIT DO GROUPS"
01852 EDIT
01853  PUT
01854  JUMP $076

*** FORMAT CODE ***
01859  $075  "2000180000"
01872  $076  STKTP $STKTO , $STKTI
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"
01933  $079  LDA A
01942  LD "0001"
01943  SST
01949  $081  LD "0001"
01953  STCKC "4"
01957  JUMPT $080
01962  LDA X
01963  LD A
01974  LD "EA@YD?01"
01984  MULT
01985  STO
01992  LDA Y
01998  LD X
01998  LD A
02004  DIV
02005  STO
02006  LDA Z
02012  LD A
02018  LD "0005"
02024  STCKC "2"
02026  JUMPF $082
02031  LD A
02037  JUMP $083
02042  $082  LD "00000000"
02052  $083  STO
02053  FMT  $084  "1"
02059  LD  X
02065  EDIT
02066  LD  Y
02072  EDIT
02073  LD  Z
02079  EDIT
02086  PUT
0208C  JUMP  $085

***FORMAT CODE***
02086  $084  "9000D"
02095  $085  FMT  $086  "1"
02096  LD  A
02105  EDIT
02106  PUT
02107  JUMP  $087

***FORMAT CODE***
02112  $086  "0005"
02115  $087  LDA  A
02125  LD  A
02136  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  RETR N
02247  $093  LDA  $094
02253  LD  $ACTIVE
02259  LD  $ACTIVE
02265  FLAG
02266  LDA  $095
02272  LDA  $095
02278  LD  I
02284  LD  "0001"
02290  SUB
02291  STO
02292  LD  $STK T2
02298  LDA  NFACT
02304  JUMPA
02305  $094  LD  I
02311  MULT
02312  SWAP
02313  RETR N
02314  RETR N
02315  $090  FMT  $096  "1"
02321  LDA  "ENTER RPROC"
02334  EDIT
02335  PUT
02335  JUMP  $097

***FORMAT CODE***
02341  $096  "2000380000"
02353  $097  FMT  $098  "1"
02359  LDA  "7 FACTORIAL ="
02374  EDIT
02375  LDA  $099
02381  LD  $ACTIVE
02387   LD $ACTIVE
02393   FLAG
02394   LDA $100
02400   LDA $100
02406   LDA $101
02412   LD $ACTIVE
02418   LD $ACTIVE
02424   FLAG
02425   LDA $102
02431   LDA $102
02437   LD "0003"
02442   STO
02444   LD $STKT1
02450   LDA NFACT
02456   JUMPA
02457   LDI $101
02463   ADD
02464   STO
02465   LD $STKT1
02471   LDA NFACT
02477   JUMPA
02478   $099
02479   EDIT
02480   PUT
02481   JUMP $103

***FORMAT CODE***
02485   $098 "2000180000*000*"
02502   $103
02506   FMT $104 "1"
02508   LDA "EXIT RPROC"
02520   EDIT
02521   PUT
02522   JUMP $105

***FORMAT CODE***
02527   $104 "2000180000"
02539   $105
02550   $106
02555   OUT ENTPK "2"
02557   DYNAM $095
<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>02562</td>
<td>JUMP</td>
<td>$108</td>
</tr>
<tr>
<td>02567</td>
<td>LDA</td>
<td>$108</td>
</tr>
<tr>
<td>02575</td>
<td>LD</td>
<td>&quot;1+&quot;</td>
</tr>
<tr>
<td>02581</td>
<td>LD</td>
<td>&quot;001&quot;</td>
</tr>
<tr>
<td>02587</td>
<td>SBSTh</td>
<td></td>
</tr>
<tr>
<td>02588</td>
<td>LDA</td>
<td>OUTCHAR</td>
</tr>
<tr>
<td>02594</td>
<td>STO</td>
<td></td>
</tr>
<tr>
<td>02597</td>
<td>LDA</td>
<td>J</td>
</tr>
<tr>
<td>02603</td>
<td>LD</td>
<td>J</td>
</tr>
<tr>
<td>02609</td>
<td>LD</td>
<td>&quot;001&quot;</td>
</tr>
<tr>
<td>02615</td>
<td>ADD</td>
<td></td>
</tr>
<tr>
<td>02616</td>
<td>STO</td>
<td></td>
</tr>
<tr>
<td>02617</td>
<td>RETRN</td>
<td></td>
</tr>
<tr>
<td>02618</td>
<td>NEXT</td>
<td></td>
</tr>
<tr>
<td>0262C</td>
<td>DYNAM</td>
<td>I</td>
</tr>
<tr>
<td>02625</td>
<td>JUMP</td>
<td>$110</td>
</tr>
<tr>
<td>02630</td>
<td>LDA</td>
<td>INPUT</td>
</tr>
<tr>
<td>02638</td>
<td>LD</td>
<td>I</td>
</tr>
<tr>
<td>02644</td>
<td>LD</td>
<td>&quot;001&quot;</td>
</tr>
<tr>
<td>02650</td>
<td>SBSTR</td>
<td></td>
</tr>
<tr>
<td>02651</td>
<td>LDA</td>
<td>&quot;1+&quot;</td>
</tr>
<tr>
<td>02654</td>
<td>COMPC</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>02656</td>
<td>JUMPF</td>
<td>&quot;1+&quot;</td>
</tr>
<tr>
<td>02661</td>
<td>LDA</td>
<td>I</td>
</tr>
<tr>
<td>02667</td>
<td>LD</td>
<td>I</td>
</tr>
<tr>
<td>02675</td>
<td>LD</td>
<td>&quot;001&quot;</td>
</tr>
<tr>
<td>02679</td>
<td>ADD</td>
<td></td>
</tr>
<tr>
<td>02680</td>
<td>STO</td>
<td></td>
</tr>
<tr>
<td>02681</td>
<td>LD</td>
<td>&quot;001+&quot;</td>
</tr>
<tr>
<td>02687</td>
<td>LD</td>
<td>&quot;001+&quot;</td>
</tr>
<tr>
<td>02693</td>
<td>STCKC</td>
<td>&quot;4&quot;</td>
</tr>
<tr>
<td>02695</td>
<td>JUMPF</td>
<td>$113</td>
</tr>
<tr>
<td>02700</td>
<td>LD</td>
<td>&quot;F&quot;</td>
</tr>
<tr>
<td>02703</td>
<td>Swap</td>
<td>RETRN</td>
</tr>
<tr>
<td>02705</td>
<td>LDA</td>
<td>INPUT</td>
</tr>
<tr>
<td>02713</td>
<td>LD</td>
<td>I</td>
</tr>
</tbody>
</table>
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  NUMFR ENTPR "2"
02754  DYNAM I
02759  JUMP $116
02764  $116 LDA $117
02770  LDA $ACTIVE
02776  LDA $ACTIVE
02782  FLAG
02783  LDA $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"
LDA "0"
COMPC "5"
AND
JUMP $119
LDA CHAR ,"01"
LDA INPUT ,"1+
LD I
LD "0001"
SbSTR
STO
LDA I
LD I
LD "0001"
ADD
STO
LD "T"
SWAP
RETRN
LD "F"
SWAP
RETRN
RETRN
ID ENTPR "2"
DYNAM I
JUMP $121
LDA $122
LD $ACTIVE
LD $ACTIVE
FLAG
LD $STKT2
LDA NEXT
JUMPA
$122 NOT
JUMPF $123
LD "F"
SWAP
RETRN
LDA INPUT ,"1+
LD I
LD "0001"
SBSTR
LDA "A"
COMPC "5"
LDA INPUT ,"1+
LD I
LD "0001"
SBSTR
LDA "Z"
COMPC "3"
AND
JUMPF $124
LDA CHAR ,"01"
LDA INPUT ,"1+
LD I
LD "0001"
SBSTR
STO
LDA I
LD I
LD "0001"
ADD
STO
LD "T"
SWAP
RETRN
RETRN
$124
LD "F"
SWAP
RETRN
RETRN
LITERAL ENTRP "2"
DYNAM $095
JUMP $126
LDA $127
03253   LDA  $133
03259   LDA  $133
03265   LDA  "(" 
03266   STO
03269   LD   $STKT2
03275   LDA  LITERAL
03281   JUMPA
03282   JUMPF $132
03287   LDA  $135
03292   LD   $ACTIVE
03295   LD   $ACTIVE
03305   FLAG
03306   LD   $STKT2
03312   LDA  EXP1
03316   JUMPA
03316   $135
03320   JUMPF $136
03325   LD   "F"
03328   SWAP
03329   RETRN
03330   LDA  $136
03336   LD   $ACTIVE
03342   LD   $ACTIVE
03348   FLAG
03349   LDA  $138
03355   LDA  $138
03361   LDA  ")" 
03364   STO
03365   LD   $STKT2
03377   LDA  LITERAL
03378   JUMPA
03378   $137
03379   JUMPF $139
03384   LD   "F"
03387   SWAP
03388   RETRN
03389   LD   "T"
03392  SWAP
03393  RETRN
03394  $134  LDA $140
03400  LD $ACTIVE
03406  LD $ACTIVE
03412  FLAG
03413  LD $STKT2
03419  LDA NUMBER
03425  JUMPA
03426  $140  JUMPF $141
03431  LDA $142
03437  LD $ACTIVE
03443  LD $ACTIVE
03449  FLAG
03450  LDA CHAR ,"01"
03456  LD $STKT2
03462  LDA OUT
03470  JUMPA
03471  $141  LD "T"
03476  SWAP
03477  RETRN
03478  $141  LDA $143
03484  LD $ACTIVE
03488  LD $ACTIVE
03494  FLAG
03495  LD $STKT2
03501  LDA ID
03503  JUMPA
03503  $143  JUMPF $144
03503  JUMPF $144
03508  LDA $145
03515  LD $ACTIVE
03525  LD $ACTIVE
03531  FLAG
03532  LDA CHAR ,"01"
03540  LD $STKT2
03546  LDA OUT
03552  JUMPA
03555 $145    LD "T"
03556          SWAP
03557          RETRN
03558 $144    LD "F"
03561          SWAP
03562          RETRN
03563          RETRN
03564          TERM ENTPR "2"
03566          DYNAM $146
03576 $147    LDA $148
03582          LD $ACTIVE
03588          LD $ACTIVE
03594          FLAG
03595          LD $STKT2
03601          LDA PRIMARY
03607          JUMPA
03608 $148    NOT
03609          JUMPF $149
03614          LD "F"
03617          SWAP
03619 $149    LDA $150
03625          LD $ACTIVE
03631          LD $ACTIVE
03637          FLAG
03638          LDA $133
03644          LDA $133
03650          LDA "*"
03653          STO
03654          LD $STKT2
03660          LDA LITERAL
03666          JUMPA
03667 $150    JUMPF $152
03672          LDA $153
03678          LD $ACTIVE
03684          LD $ACTIVE
03690  FLAG
03691  LD  $STKT2
03697  LDA  PRIMARY
03703  JUMPA
03704  $153  NOT
03705  JUMPF  $154
03710  LD  "F"
03713  SWAP
03714  RETRN
03715  $154  LDA  $155
03721  LD  $ACTIVE
03727  LD  $ACTIVE
03733  FLAG
03734  LDA  $138
03740  LDA  $138
03745  LDA  "*"
03749  STO
03750  LD  $STKT2
03755  LDA  OUT
03762  JUMPA
03763  $155  LDA  $149
03769  JUMPA
03770  $152  LDA  $157
03776  LD  $ACTIVE
03782  LD  $ACTIVE
03788  FLAG
03789  LDA  $158
03795  LDA  $158
03801  LDA  "/"
03804  STO
03805  LD  $STKT2
03811  LDA  LITERAL
03817  JUMPA
03818  $157  JUMPF  $159
03823  LDA  $160
03829  LD  $ACTIVE
03835  LD  $ACTIVE
03841   FLAG
03842   LD $STKT2
03844   LDA PRIMARY
03854   JUMPA
03855   $160 NOT
03856   JUMPF $161
03861   LD "F"
03864   SWAP
03865   RETRN
03866   $161 LDA $162
03872   LD $ACTIVE
03873   LD $ACTIVE
03884   FLAG
03885   LDA $163
03891   LDA $163
03897   LDA "/"
03900   STO
03901   LD $STKT2
03907   LDA $162
03913   JUMPA
03914   $162 LDA $149
03920   JUMPA
03921   $159 LD "T"
03924   SWAP
03925   RETRN
03926   RETRN
03927   EXP2 ENTPR "2"
03929   DYNAM $163
03934   JUMP $165
03939   $165 LDA $166
03945   LD $ACTIVE
03951   LD $ACTIVE
03957   FLAG
03958   LDA $133
03964   LDA $133
03970   LDA "-"
03973   STO
03974  LD  $STKT2
03980  LDA  LITERAL
03986  JUMPA
03987  $166  JUMPF $168
03992  LDA  $169
03996  LD  $ACTIVE
04000  LD  $ACTIVE
04010  FLAG
04011  LD  $STKT2
04017  LDA  TERM
04023  JUMPA
04024  $169  NOT
04025  JUMPF $170
04030  LD  "F"
04033  SWAP
04034  RETRN
04035  $170  LDA  $171
04041  LD  $ACTIVE
04046  LD  $ACTIVE
04051  FLAG
04055  LDA  $138
04059  LDA  $138
04064  LDA  "m"
04068  STO
04070  LD  $STKT2
04075  LDA  OUT
04081  JUMPA
04083  $171  LD  "T"
04086  SWAP
04087  RETRN
04088  $168  LDA  $173
04094  LD  $ACTIVE
04100  LD  $ACTIVE
04105  FLAG
04107  LDA  $158
04113  LDA  $158
04119  LDA  "*"
04122 STO
04123 LD $STKT2
04129 LDA LITERAL
04135 JUMPA
04136 $173 JUMPF $175
04141 LDA $176
04147 LD $ACTIVE
04153 LD $ACTIVE
04159 FLAG
04160 LD $STKT2
04166 LDA TERM
04172 JUMPA
04173 $176 NOT
04174 JUMPF $177
04175 LD "F"
04182 SWAP
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 EXP 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 LDA %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 LDA %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 STK2
04578 LDA $STK2
04580 LDA OUT
04586 JUMPA
04587 $196 LDA $183
04593 JUMPA
04594 $193 LD "T"
04597 SWAP
04598 RETRN
04599 RETRN
04600 $106 FMT $198 "$1"
04606 LDA "ENTER POSTFIX"
04621 EDIT
04622 PUT
04623 JUMP $199
**:FORMAT CODE**
04628 $198 "2000300000"
04640 $199 FMT $200 "$Z"
04645 LDA INPUT "$1+"
04651 GET
04653 JUMP $201
**:FORMAT CODE**
04660 $200 "200018001+"
04672 $201 LDA I
04678 LDA J
04684 LD "0001"
04690 SST
04691 STO
04692 FMT $202 "$1"
04698 LDA "INFIX EXPRESSION ="
04718 EDIT
04719 LDA INPUT "1+
04727 EDIT
04728 PUT
04729 JUMP $203
***FORMAT CODE***
04734 $202 "2000180000000028001+
04756 $203 LDA T "1+
04764 LDA INPUT "1+
04772 STO
04775 LDA $204
04778 LD $ACTIVE
04779 LD $ACTIVE
04785 FLAG
04788 LD $STKTI
04794 LDA EXP1
04804 JUMPA
04805 $204 JUMPF $205
04810 FMT $206 "1"
04814 LDA "POSTFIX EXPRESSION ="
04830 EDIT
04839 LDA T "1+
04847 LD "0001"
04850 LD J
04859 LD "0001"
04865 SUB
04866 SBSTR
04867 LDA ";"
04870 CAT
04871 EDIT
04872 PUT
04873 JUMP $207
***FORMAT CODE***
04878 $206 "20001800000000280000"
04900 $207 JUMP $208
04905 $205 FMT $209 "1"
04911 LDA "****ERROR****"
04926 EDIT
04927 PUT
04928 JUMP $209
***FORMAT CODE***
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 Q ENTPR "3"
05030 DYNAM $217
05035 ALLOC $STK4 F
***DOPE VECTOR CODE***
"10400012000"
05059 JUMP $219
05064 $218 LDA F
05070 LD "0001"
05075 INDXA "$1"
05078 LD "000"
05084 STO
05088 LDA $219
05091 LD $ACTIVE
05097 LDA $220
05103 LD "$300+" ****
05109 LD "$0004"
05115 ADD
05116 STO
05117  LD "0000"  *****
05123  FLAG
05124  LDA $221
05130  LDA $221
05136  LDA F
05142  LD "0001"
05148  INDXR "1"
05150  LD Y
05156  ADD
05157  STO
05158  LD $STKT4
05164  LDA Z
05170  JUMPA
05171 $219  RETRN
05172 $216  LDA $222
05178  LD $ACTIVE
05184  LD $ACTIVE
05190  FLAG
05194  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
05258  ENTPR "2"
05259  DYNAM $226
05264  ALLOC $STKT2 G
***DOPE VECTOR CODE***
"10420001Z0001"

05288  JUMP $227
05293  ST<TP $ST<T2 $ST<T3
05304  JUMP $228
05309  U  ENTRP "3"
05311  DYNAM F
05316  JUMP $230
05321  $230  LDA  G
05327  LD  I
05333  INDXA "1"
05335  LD  W
05341  STO
05342  RETRN
05343  $228  LDA  I
05349  LD  "0001"
05355  SST
05356  $232  LD  "0001"
05362  STCKC "4"
05364  JUMP T $231
05369  LDA  G
05375  LD  I
05381  INDXA "1"
05383  LD  "000G"
05389  STO
05390  LDA  $233
05396  LD  $ACTIVE
05402  LD  $ACTIVE
05408  FLAG
05409  LDA  $224
05415  LDA  $224
05421  LDA  "01B6" *****
05427  STO
05428  LDA  $235
05434  LD  $ACTIVE
05440  STO
05441  LDA  $236
05447: LDA $236
05450: LD "0001"
05460: STK $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 "000*"
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
05617  JUMP  $242
***#FORMAT CODE***
05622  $239  "'0007"
05629  $242  RETRN
05630  $213  LDA  $243
05636  LD  $ACTIVE
05642  LD  $ACTIVE
05648  FLAG
05649  LD  $STKT1
05655  LDA  R
05661  JUMPA
05662  $243  FMT  $244  \",1"
05668  LDA  "EXIT GLOBAL TEST"
05686  EDIT
05687  PUT
05688  JUMP  $245
***#FORMAT CODE***
05693  $244  "2000180000"
05705  $245  STKTP  $STKT0  ,$STKT1
05716  ALLOC  $STKT1  ,Z
***#DOPE VECTOR CODE***
"1042000120003"
05740  JUMP  $246
05745  LBL  ENTPK  "2"
05747  DYNAM  $095
05752  JUMP  $248
05758  $248  POPUP
05758  LDA  LABEL
05764  LD  "0003"
05770  INDXR  "1"
05772  RETRN
05773  RETRN
05774  $246  FMT  $249  \",1"
05780  LDA  "ENTER LABEL"
05793  EDIT
05794  PUT
05795  JUMP  $250
**FORMAT CODE**

05800 $249 "2000380000"
05812 $250 LDA Y
05818 LD "01#"
05824 STD
05825 STKTP $STKT1 $STKT2
05836 JUMP $251
05841 $251 LDA I
05847 LD "0001"
05853 STD
05854 LD "01Z"
05860 JUMPA
05867 LDA B
05867 LD I
05873 LD "0002"
05879 DIV
05880 STD
05881 LBL1 FMT $252 "1"
05887 LDA "LABEL TEST"
05893 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 STD
05950 LD Y
05956 JUMPA
05957 LBL2 FMT $254 "1"
05963 LDA "INCORRECT LABEL TEST"
05985 EDIT
05986 PUT
05987: JUMP $255
  ***FORMAT CODE***
05992: $254 "80000"
05995: $255 FMT $256 "1"
06005: LDA "INCORRECT LABEL TEST"
06027: EDIT
0602b: PUT
0602d: JUMP LBL2
  ***FORMAT CODE***
06034: $256 "80000"
06041: LBL2 FMT $258 "1"
06047: LDA "LABEL TEST"
06059: EDIT
06060: LD A
06066: EDIT
06067: PUT
06068: JUMP $259
  ***FORMAT CODE***
06073: $258 "20001800000005"
06090: $259 LDA Z
06096: LD "0001"
06102: INDXA "1"
06104: LD "01-ki"
06110: STO
06111: LDA Z
06117: LD "0002"
06123: INDXA "1"
06125: LD "01-ki"
06131: STO
06132: LDA Z
06138: LD "0003"
06144: INDXA "1"
06145: LD "01J-"
06152: STO
06153: LDA BADLAB
06159: LD $ACTIVE
06165: LD $ACTIVE
06171    FLAG
06172    LDA   Z
06173    LD    $STKT1
06174    LDA   LBL
06175    JUMPA
06176    BADLAB  FMT  $261  "1"
06177    LDA   "INCORRECT LABEL RETURN"
06178    JUMP  GOODLAB

***FORMAT CODE***
06228  $261  "2000180000"
06240  GOODLAB  FMT  $263  "1"
06246  LDA   "CORRECT LABEL RETURN"
06268  EDIT
06269  PUT
06270  JUMP  $264

***FORMAT CODE***
06275  $263  "2000180000"
06287  $264  FMT  $265  "1"
06293  LDA   "EXIT LABEL"
06303  EDIT
06305  PUT
06307  JUMP  $266

***FORMAT CODE***
06312  $265  "2000180000"
06324  $266  STOP
06325  END    PROGRAM
BEGIN STRINGS
THIS IS A STRING.
THIS IS A CONCATENATED SUBSTRING.
STRING COMPARE 1 WORKS
STRING COMPARE 2 WORKS
EXIT STRINGS

BEGIN I/O BLOCK
123  -12  20   123  -12  20   123  -12  20
5.12340  4.300000000E+015  -11.20000000E-07
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
I =  5, M(I,1) = 1, M(I,5) = 0
I =  4, M(I,1) = 1, M(I,5) = 0
I =  3, M(I,1) = 1, M(I,5) = 0
I =  2, M(I,1) = 1, M(I,5) = 0
I =  1, M(I,1) = 1, M(I,5) = 0
I =  0, M(I,1) = 1, M(I,5) = 0
I = -1, M(I,1) = 1, M(I,5) = 0
I = -2, M(I,1) = 1, M(I,5) = 0

EXIT DO GROUPS

ENTER ARITHMETIC BLOCK
1.330  1.330  1.0
1
2.660  1.330  2.0
2
3.990  1.330  3.0
3
5.320  1.330  4.0
4
6.650  1.330  5.0
5
7.980  1.330  6.0
6
9.310  1.330  7.0
7
10.640  1.330  8.0
8
11.970  1.330  9.0
9
13.30   1.330 10.0
10

EXIT ARITHMETIC BLOCK
ENTER RPROC
7 FACTORIAL = 5040
EXIT RPROC

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

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

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

//
ESYLST: PROCEDURE MAIN;

IDENTIFICATION:

PROGRAM-ID: ESYLST.
AUTHOR: J. R. VAN DOREN.
SOURCE LANGUAGE: PLEX.
OBJECT LANGUAGE: PLEX PSEUDO-MACHINE CODE.
OBJECT INTERPRETER: PLXINT.

PURPOSE:

ESYLST LISTS EASYCODER PROGRAMS AND BUILDS A CARD REFERENCE INDEX FOR SYMBOL DEFINITIONS.

DECLARE INPUT CHAR(80) » (CRDCNT, ITSYMCNT) FIXED, SYMBOL(500) CHAR(6),
(CRDREF(500), DECNT) CHAR(5);
I, CRDCNT=0:
SYMCNT=1;

GETCRD: GET EDIT(INPUT) (A(80));
IF SUBSTR(INPUT,1,4)="****" THEN GO TO SORT;
CRDCNT=CRDCNT+10;
PUT STRING(DECNT) EDIT (CRDCNT) (1(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 (J,K,L,M) FIXED, SYMTMP CHAR(6), REFTMP CHAR(5);
SYMCNT=SYMCNT-1;
M=SYMCNT;
LBL20: M=M/2;
    IF M=0 THEN GO TO LBL40;
    K=SYMCNT-M;
    J=1;
LBL41: I=J;
LBL49: L=I+M;
    IF SYMBOL(I) < SYMBOL(L) THEN GO TO LBL60;
    SYMTMP=SYMBOL(I);
    REFTMP=CRDREF(I);
    SYMBOL(I)=SYMBOL(L);
    CRDREF(I)=CRDREF(L);
    SYMBOL(L)=SYMTMP;
    CRDREF(L)=REFTMP;
    I=I-M;
    IF I > 0 THEN GO TO LBL49;
LBL60: J=J+1;
    IF J < < THEN GO TO LBL20; ELSE GO TO LBL41;
END SORT;
LBL40: PUT EDIT ("SYMBOL DEFINITION - CARD REFERENCE INDEX"," ")
      (SKIP(3),COL(20),A,SKIP(2),A);
      PUT EDIT ((SYMBOL(I),CRDREF(I)," ");  " DO I=1 TO SYMCNT))
      (A(6)*X(2),A(5)*A,A(6)*X(2),A(5)*A,A(6)*X(2),A(5)*A,A)
      A(6)*X(2),A(5)*A,A(6)*X(2),A(5)*A,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
FUNCTION PLACPL
INTERPRETER=MTXINT04.
ERROR: PROCEDURE MAIN;
                        /* IDENTIFICATION: */
                        ** PROGRAM-ID: ERROR. **
                        ** AUTHOR: J. R. VAN DOREN. **
                        ** SOURCE LANGUAGE: PLEX. **
                        ** PURPOSE: **
                        ** ERROR DEMONSTRATES THE ERROR DIAGNOSTICS OF PLXCPL. **
BEGIN; DECLARE (I,J) FIXED, N(10,15) FLOAT;
XYZ: PROCEDURE (A,B,C); DECLARE A ENTRY, B FIXED;
    *****ERROR**** w: INCORRECT ARG DCL COUNT
        A=0;
    *****ERROR**** F: SYNTAX
        END AbC;
    *****ERROR**** W: POSS PROC CLOSING ERR
        XYZ: PROCEDURE(A); DECLARE A FLOAT;
    *****ERROR**** W: DUP PROC DCL
        A=I;
        END XYZ;
LABEL: CALL XYZ(N(1),J*3+1/4),I);
    *****ERROR**** W: INCORRECT PARM CNT
        J=I****-1;
    *****ERROR**** F: SYNTAX
        END;
        END ERROR;
FATAL ERROR(S) ENCOUNTERED, JOB ABORTED
IDENTIFICATION:

PROGRAM-ID: PLXINT

AUTHOR: J. R. VAN DOREN

SOURCE LANGUAGE: EASYCODER

SOURCE COMPUTER: H-1200

OBJECT COMPUTER: H-1200

PURPOSE:

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

ADMODE4 ASSEMBLE IN FOUR CHAR ADDRESSING MODE

ORG 45056 EXECUTION LOCATION

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

- **INSTCTR EQU 209**  
  Instruction Counter

- **STKSTR EQU 223**  
  Stack Start

- **STKEND EQU 227**  
  Stack End

- **LODLOC EQU 219**  
  Starting Address for Execution

- **DYNSTR EQU 231**  
  Starting Address for Dynamic Storage

- **DYEND EQU 235**  
  Ending Address for Dynamic Storage

**INDEX REGISTER LOCATION DEFINITIONS AND USAGE DESCRIPTIONS**

- **IR1 EQU 4**  
  Conversion Subroutine Usage and Work Register

- **IR2 EQU 8**  
  Push Down Stack Pointer

- **IR3 EQU 12**  
  Program Counter

- **IR4 EQU 16**  
  Result Register for Address Computation

- **IR5 EQU 20**  
  Result Register for Address Computation

- **IR6 EQU 24**  
  Input Buffer Pointer

- **IR7 EQU 28**  
  Operation Code Register

- **IR8 EQU 32**  
  Output Buffer Pointer

- **IR9 EQU 36**  
  Work Register

- **IR10 EQU 40**  
  Work Register

- **IR11 EQU 44**  
  Work Register

- **IR12 EQU 48**  
  Work Register

- **IR13 EQU 52**  
  Work Register

- **IR14 EQU 56**  
  Active Display Pointer for Dynamic Storage

- **IR15 EQU 60**  
  Conversion Subroutine Usage and Work Register

**MACRO CALLS TO SOURCE LIBRARY TO ESTABLISH CONVERSION SUBROUTINES**

- **L F8/HD 4,BD, FLOATING BINARY TO FLOATING DECIMAL**

- **L FD/FB 4,DB, FLOATING DECIMAL TO FLOATING BINARY**
IFIX EQU INT RESOLVE CONVERSION NAME MISMATCH

PROGRAM INITIALIZATION:
SET REGISTERS.
INITIALIZE DYNAMIC STORAGE DISPLAY.

START EQU *

CAM 60
SET FOUR CHAR ADDRESSING FOR EXECUTION

SW STKEND-2
WORD MARK FOR MOVING AND TESTING

MCw STKSTR,IR2
STACK POINTER

SW IR2-2

SI IR2
ITEM MARK FOR RIGHT MOVE

MCw LADLOC,IR3
INITIALIZE INSTRUCTION COUNTER

SI IR3
ACCOMODATE RIGHT MOVE

LCA =4BO,IR5
PROPERLY

LCA =4BO,IR4
PUNCTUATE

SI IR4,IR5
INDEX REGISTERS IR4,IR5

BS IR7
CLEAR OP CODE REGISTER

SW IR7-1
SHORTEN FETCH ARITHMETIC

MCw DYNSTR,IR14
INITIALIZE ACTIVE STORAGE POINTER

SI IR14
ACCOMODATE RIGHT MOVE

SW DYNEND-2
WORD MARK FOR COMPARISON

MLWDR IR14,3+X14
INIT DYNAMIC STOR STACK TOP POINTER

MLWDR IR14,7+X14
LEVEL ONE STORAGE POINTER

MCw : ;OUTPUT+132
CLEAR PRINT BUFFER

MCw OUTPUT+132

MCw =1C21,OUTPUT
CARRIAGE CONTROL

SI INPUT+80
RESTORE LOST ITEM MARK ON INPUT BUFFER

:SKIP PRINT+57,
SKIP TO TOP OF PAGE

LCA *PRNTBF,IR8
OUTPUT BUFFER POINTER

B GETIPT
INITIALIZE INPUT BUFFER
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>FETCH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=1B1*INSTCT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FETCH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=1R7*1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INSTRUCTION FETCHING</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=1B1*INSTCT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FUTURE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=1R7*1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INSTRUCTION COUNT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=1B1*INSTCT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COUNT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=1R7*1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLEAR 2ND CHAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=1B1*INSTCT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=1R7*1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLEAR OP CODE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=1B1*INSTCT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CODE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=1R7*1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BUMP SEQUENCE COUNTER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=1B1*INSTCT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COUNT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=1R7*1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INSERT OP CODE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=1B1*INSTCT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CODE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=1R7*1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INSERT EXECUTION ADDRESS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=1B1*INSTCT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDRESS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=1R7*1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF NO ADDRESS COMPUTATION THEN EXECUTE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=1B1*INSTCT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDRESS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=1R7*1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ELSE DO ADDRESS ADDRESSES AND ADDRESS PARAMETER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=1B1*INSTCT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDRESS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=1R7*1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
- BA = BA + 1
- BS = BS + 1
- =1B1*INSTCT = INSTCT + 1
- =1R7*1 = R7 + 1

**Instructions:**
- FETCH
- BA
- =1B1*INSTCT
- FUTURE
- BS
- =1R7*1
- INSTRUCTION FETCHING
- BA
- =1B1*INSTCT
- FUTURE
- BS
- =1R7*1
- INSTRUCTION COUNT
- BA
- =1B1*INSTCT
- COUNT
- BS
- =1R7*1
- CLEAR 2ND CHAR
- BA
- =1B1*INSTCT
- CHAR
- BS
- =1R7*1
- CLEAR OP CODE
- BA
- =1B1*INSTCT
- CODE
- BS
- =1R7*1
- BUMP SEQUENCE COUNTER
- BA
- =1B1*INSTCT
- COUNT
- BS
- =1R7*1
- INSERT OP CODE
- BA
- =1B1*INSTCT
- CODE
- BS
- =1R7*1
- INSERT EXECUTION ADDRESS
- BA
- =1B1*INSTCT
- ADDRESS
- BS
- =1R7*1
- IF NO ADDRESS COMPUTATION THEN EXECUTE
- BA
- =1B1*INSTCT
- ADDRESS
- BS
- =1R7*1
- ELSE DO ADDRESS ADDRESSES AND ADDRESS PARAMETER
- BA
- =1B1*INSTCT
- ADDRESS
- BS
- =1R7*1

**Codes:**
- OP CODE 10
- OP CODE 11
- OP CODE 12
- OP CODE 20
- OP CODE 21
- OP CODE 22
- OP CODE 23
- OP CODE 24
- OP CODE 25
- OP CODE 26
- OP CODE 27
- OP CODE 28
- OP CODE 29
- OP CODE 30
- OP CODE 31
- OP CODE 32
- OP CODE 33
- OP CODE 34
- OP CODE 35
- OP CODE 36
ADDRESS COMPUTATIONS ARE PERFORMED BY THE ADCOMP SUBROUTINE. ONE OR TWO ADDRESSES ARE COMPUTED DEPENDING ON ADDRESSING PARAMETER IN TVEC WHICH IS ADDRESSED BY THE OP CODE REGISTER IR7. RETURN ADDRESS IN IR13.

ADCOMP SST 0+X3,STYPE,70 SAVE STORAGE TYPE
SST 0+X3,DTYPE,07 SAVE DATA TYPE
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01860</td>
<td>BCE</td>
<td>LITADD,STYPE,70</td>
</tr>
<tr>
<td>01870</td>
<td>LCOMP BCE</td>
<td>NOCOMP,1+X3,00</td>
</tr>
<tr>
<td>01880</td>
<td>BS IR9</td>
<td></td>
</tr>
<tr>
<td>01890</td>
<td>MRSD 1+X3,IR9</td>
<td></td>
</tr>
<tr>
<td>01900</td>
<td>BA IR9</td>
<td></td>
</tr>
<tr>
<td>01910</td>
<td>BA IR9</td>
<td></td>
</tr>
<tr>
<td>01920</td>
<td>BA IR14,IR9</td>
<td></td>
</tr>
<tr>
<td>01930</td>
<td>MRID 0+X9,IR4-3</td>
<td></td>
</tr>
<tr>
<td>01940</td>
<td>BA 4+X3,IR4</td>
<td></td>
</tr>
<tr>
<td>01950</td>
<td>BS IP4-3</td>
<td></td>
</tr>
<tr>
<td>01960</td>
<td>BA 1B5,IR3</td>
<td></td>
</tr>
<tr>
<td>01970</td>
<td>BCE</td>
<td>INDIRA,STYPE,60</td>
</tr>
<tr>
<td>01980</td>
<td>BCE</td>
<td>PRCADD,STYPE,20</td>
</tr>
<tr>
<td>01990</td>
<td>BCE 0+X13,TVEC+X7,01</td>
<td></td>
</tr>
<tr>
<td>02000</td>
<td>BS 1R9</td>
<td></td>
</tr>
<tr>
<td>02010</td>
<td>MRSD 1+X3,IR9</td>
<td></td>
</tr>
<tr>
<td>02020</td>
<td>BA 1R9</td>
<td></td>
</tr>
<tr>
<td>02030</td>
<td>BA 1R9</td>
<td></td>
</tr>
<tr>
<td>02040</td>
<td>BA 1R14,IR9</td>
<td></td>
</tr>
<tr>
<td>02050</td>
<td>MRID 0+X9,IR5-3</td>
<td></td>
</tr>
<tr>
<td>02060</td>
<td>BA 4+X3,IR5</td>
<td></td>
</tr>
<tr>
<td>02070</td>
<td>BS 1R5-3</td>
<td></td>
</tr>
<tr>
<td>02080</td>
<td>BA 1B5,IR3</td>
<td></td>
</tr>
<tr>
<td>02090</td>
<td>B 0+X13</td>
<td></td>
</tr>
<tr>
<td>02100</td>
<td>LITADD MRID</td>
<td>IR3-2,IR4-2</td>
</tr>
<tr>
<td>02110</td>
<td>BA 1B1,IR4</td>
<td></td>
</tr>
<tr>
<td>02120</td>
<td>MRIN 1+X3,0</td>
<td></td>
</tr>
<tr>
<td>02130</td>
<td>SAR IR3</td>
<td></td>
</tr>
<tr>
<td>02140</td>
<td>B 0+X13</td>
<td></td>
</tr>
<tr>
<td>02150</td>
<td>PRCADD MRID</td>
<td>0+X4,IR4-3</td>
</tr>
<tr>
<td>02160</td>
<td>INDIRA MRID</td>
<td>0+X4,IR4-3</td>
</tr>
<tr>
<td>02170</td>
<td>B 0+X13</td>
<td></td>
</tr>
<tr>
<td>02180</td>
<td>NOCOMP MRID</td>
<td>2+X3,IR4-3</td>
</tr>
<tr>
<td>02190</td>
<td>SAR IR3</td>
<td></td>
</tr>
<tr>
<td>02200</td>
<td>B 0+X13</td>
<td></td>
</tr>
<tr>
<td>02210</td>
<td>STYPE DCw</td>
<td>:0;</td>
</tr>
<tr>
<td>02220</td>
<td>DTYPE DCw</td>
<td>:0;</td>
</tr>
</tbody>
</table>
DYNAM - ALLOCATE FIXED DYNAMIC STORAGE (PROCEDURE ENTRY)

INCREMENT STACKTOP
CHECK FOR DYNAMIC STORAGE OVERFLW
INSTRUCTION COUNTER

STKTP - INITIALIZE NEW STACKTOP (BLOCK ENTRY)

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.

INSERT SUBSCRIPT COUNT
SAVE LOCATION OF STACK TOP VALUE
SAVE DYNAMIC DOPE VECTOR ADDRESS
MOVE IN ELEMENT LENGTH SIZE
INIT ARRAY SIZE COUNTER
CLEAR
SET OP CODE REGISTER TO ONE ADDRESS CODE
RETURN ADDRESS FROM 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  MRR  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  MRR  LNGTH-3,0+X12  ELEMENT LENGTH FACTOR
02760  SBR  IR12
02770  MRR  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,END  CHECK FOR DYNAMIC STORAGE ALLOC OVERFLOW
02820  BL  DYNEND
02830  B  FETCH
02840  LNGTH  DCW  =4B0
02850  SIZE  DCW  =4B0
02860  CONPRT  DCW  =4B0
02870 ***************************************************************************

**LDA - LOAD ADDRESS TO THE STACK, INCLUDING THE**
**DATATYPE CODE. IF A CHARACTER STRING ALSO LOAD**
**THE MAX STRING LENGTH.**

02880  LDA  MLWDR  IR4+4+X2  ADDRESS TO STACK
02890  LCA  DT/PE,0+X2  DATA TYPE
02900  BCE  LDLNG,DTYPE,04  IF A CHARACTER STRING LOAD LENGTH
02970 UPSTCK BA =1B9,IR2
           STACK POINTER
02980 B  FETCH
02990 LDLNG BCE LITLNG,STYPE,70
           IF A LITERAL COMPUTE LENGTH
03000 MRIDR O+X3,5+X2
           ELSE EXTRACT FROM INSTRUCTION
03010 SAR IR3
           BUMP SEQUENCE COUNTER
03020 B  UPSTCK
03030 LITLNG BS IR5
03040 MRIN O+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 O+X4,l+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  *  BYPASS IF STRING OR SUBSTRING
STT  EUU  *
BCE  CNVRTN,0-18+X2,04  IF TYPES NOT EQUAL
BCE  CNVRTN,0-18+X2,34
C  0-9+X2,0-18+X2  DO CONVERSION
BNE  SCNV
BCE  CNVRTN,0-18+X2,34
C  0-9+x2,0-18+X2  IF TYPES NOT EQUAL
BNE  SCNV  DO CONVERSION
BCE  CNVRTN,0-14+X2,IP4  RECEIVING ADDRESS
BCE  CHRSRT,0-9+X2,04  SPECIAL HANDLING IF CHARACTER STRING
BCE  CSBSTR,0-9+X2,34  CHECK FOR SENDING SUBSTRING
BCE  CSBSTR,0-9+X2,34  STORE IT
BCE  CHRSTR,0-9+X2,04  SPECIAL HANDLING IF CHARACTER STRING
BCE  CHRSRT,0-9+X2,04  CHECK FOR SENDING SUBSTRING
SCNV  BCE  PKMOVE,0-9+X2,34  STORE IT
BCE  PKMOVE,0-9+X2,34  SPECIAL HANDLING IF CHARACTER STRING
BCE  SAVTOP,0-1+X3,23  CHECK FOR SAVE AND STORE (SST)
B  =1B18,IR2  POP STACK TWO ELEMENTS
MCW  +ENDPRG,ENADDR  RESTORE STRING WORKING STORAGE POINTER
B  FETCH
SAVTOP  0-9+X2  PUNCTUATION FOR MOVE
MLRDR  0-1+X2,0-10+X2  DISCARD ADDRESS
CI  0-9+X2,0-18+X2  CLEAR ITEM MARKS
BS  =1B9,IR2  AND SAVE STACK TOP ON TOP
B  FETCH
SAVTOP  0-9+X2  PUNCTUATION FOR MOVE
MLRDR  0-1+X2,0-10+X2  DISCARD ADDRESS
CI  0-9+X2,0-18+X2  CLEAR ITEM MARKS
BS  =1B9,IR2  AND SAVE STACK TOP ON TOP
B  FETCH
SCNV  BCE  CNVRTN,0-18+X2,07  UNIVERSAL TYPE O.K.
BCE  CNVRTN,0-18+X2,07  UNIVERSAL TYPE O.K.
BCE  FIXFLT,0-9+X2,01  CHECK FOR SAVE AND STORE (SST)
BCE  FIXFLT,0-9+X2,01  CHECK FOR SAVE AND STORE (SST)
TMA  0-1+X2,00  GO CONVERT
IFIX
CNVERR  R  B  CONVERSION OVERLOW INSTRUCTION
BCE  CNVERR  R  B  CONVERSION OVERLOW INSTRUCTION
MCW  IR15,0-5+X2  INTEGER RESULT
SI  0-5+X2  TO
SI  0-5+X2  TO
LCA  =1B1,0-9+X2  STACK
LCA  =1B1,0-9+X2  STACK
BCE  CNVRTN
BCE  CNVRTN
FIXFLT  MLWDI  =4C00000027,0-1+X2  NORMALIZED LOAD
AMW  0-1+X2,70  CLEAR EXTRANEOUS PUNCTUATION
MLWDI  =8B00,0-1+X2  CLEAR EXTRANEOUS PUNCTUATION
TAM  0-1+X2,00  FLOATING
SI  0-1+X2  RESULT
SI  0-1+X2  RESULT
LCA  =1B2,0-9+X2  TO STACK
LCA  =1B2,0-9+X2  TO STACK
BCE  CNVRTN
03710 CHKSTR BS IR5
03720 MCh =1C10,IR5
03730 BCE CHRTRN,0-18+2,07
03740 MRID 0-13+2,IR5-1
03750 CHRTRN MCw 0-5+2,IR9
03760 Sw 0+X9
03770 BS IR10
03780 MRIN 0+X9,0+X4
03790 SBR IR10
03800 EA IR4,IR5
03810 C IR5,IR10
03820 BH TMOVE
03830 BCE SUBMV,0-18+2,34
03840 MRIDR 0+X9,0+X4
03850 SUBMV MRIU 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+2,IR15
03950 BS IR9
03960 MRIU 0-4+2,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+2,70
04020 B CHRSTR
04030 PRMOVE CI 0-1+X15
04040 MCw 0-5+2,IR9
04050 CW 0+X9
04060 B PNCRTN
04070***************************************************************************
0408(*)
0409(*)  FLAG - MARK STACK FOR THE LIMIT OF PARAMETER ADDRESSES
0410(*)  PRIOR TO A PROCEDURE CALL
0411(*)
0412(*)  ****************************************************************************
0413(*)  FLAG LCA = 1C77, 0+X2
0414(*)  B  UPSTCK
0415(*)  ****************************************************************************
0416(*)
0417(*)  ENTPRO - ESTABLISH DYNAMIC STORAGE DISPLAY, STORE PARAMETER
0418(*)  ADDRESSES IN NEW DYNAMIC STORAGE AREA AND THEN PASS
0419(*)  CONTROL TO THE CALLLD PROCEDURE.
0420(*)
0421(*)  ****************************************************************************
0422(*)  ENTPRO  MRID  0-8+X2,IR14-3  NEW DISPLAY AND STORAGE AREA
0423(*)  BS   = 1B9, IR2
0424(*)  BS   IR13
0425(*)  MRSID 0+X3,IR13  LEVEL NUMBER OF PROCEDURE
0426(*)  SAR  IR3  BUMP SEQUENCE COUNTER
0427(*)  BIM  = 4B4, IR13
0428(*)  BA   = 1B4, IR13
0429(*)  BA   IR14, IR13  LOCATION FOR PARAMETER ADDRESSES
0430(*)  BS   PRMCNT
0431(*)  FLGCHK BS  = 1B9, IR2  SEARCH DOWN
0432(*)  BCE  PRMSTO, 0+X2, 77  AND
0433(*)  BA   = 1B1, PRMCNT  COUNT
0434(*)  B  FLGCHK  PARMS
0435(*)  PRMSTO MCw  IR2, IR9
0436(*)  PRMCHK BCE  PRMDNE, PRMCNT, 00  IF DONE GO ELSEWHERE
0437(*)  MRIDR 10+X9, 0+X13  MOVE ADDRESS TO DYNAMIC STORAGE
0438(*)  SBR  IR13
0439(*)  BA   = 1B9, IR9
0440(*)  BS   = 1B1, PRMCNT
0441(*)  B  PRMCHK
0442(*)  PRMDNE MRID  0-8+X2, IR9-3  GLOBAL DISPLAY ADDRESS
0443(*)  BA   = 1B4, IR9  JUMP OVER STACK TOP VALUE
0444(*)  MRIDR 0-17+X2, 0-22+X2  PACK CALLING PROC DISPLAY ADDRESS WITH
RETURN ADDRESS
1ST DISPLAY ENTRY (STACK TOP VALUE)
NEXT DISPLAY LOCATION
MOVE GLOBALLY ACTIVE STORAGE LEVEL
ADDRESSES TO CURRENT DISPLAY
CURRENT DISPLAY BASE ADDRESS
B FETCH
**************************************************************************
* *
* SWAP - SWAP THE TOP TWO ELEMENTS OF THE STACK *
* *
**************************************************************************
SWAP SI 0-9+X2,0-18+X2
MLRDR 0-1+X2,HOLD
MLRDR 0-10+X2,0-1+X2
MLRDR HOLD,0-10+X2
CI 0-9+X2,0-18+X2
B FETCH
HOLD DCw =9
**************************************************************************
* *
* RETURN - RESTORE DISPLAY ADDRESS FOR CALLING PROCEDURE *
* AND SET INSTRUCTION COUNTER WITH RETURN ADDRESS *
**************************************************************************
RETNR MRID 0-8+X2,IR3-3 RETURN ADDRESS TO INST COUNTER
MRID 0-4+X2,IR14-3 DISPLAY AT POINT OF CALL
BS =189,IR2
B FETCH
**************************************************************************
<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>04820</td>
<td>JUMPA - JUMP TO THE ADDRESS IN THE STACK; POP THE STACK</td>
<td>*</td>
</tr>
<tr>
<td>04830</td>
<td>POPUP - POP THE STACK</td>
<td>*</td>
</tr>
<tr>
<td>04840</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>04850</td>
<td>JUMPA MRID 0-8×X2×IR3-3</td>
<td>SET INSTRUCTION COUNTER</td>
</tr>
<tr>
<td>04860</td>
<td>POPUP BS =189×IR2</td>
<td></td>
</tr>
<tr>
<td>04870</td>
<td>B FETCH</td>
<td></td>
</tr>
<tr>
<td>04880</td>
<td></td>
<td></td>
</tr>
<tr>
<td>04900</td>
<td>JUMP - JUMP TO THE ADDRESS FOLLOWING THE OP CODE</td>
<td></td>
</tr>
<tr>
<td>04910</td>
<td>JUMPT - JUMP IF TOP OF STACK IS TRUE; POP THE STACK</td>
<td></td>
</tr>
<tr>
<td>04920</td>
<td>JUMPF - JUMP IF TOP OF STACK IS FALSE; POP THE STACK</td>
<td></td>
</tr>
<tr>
<td>04930</td>
<td></td>
<td></td>
</tr>
<tr>
<td>04940</td>
<td></td>
<td></td>
</tr>
<tr>
<td>04950</td>
<td></td>
<td></td>
</tr>
<tr>
<td>04960</td>
<td>JUMP MRID 0×X3×IR3-3</td>
<td></td>
</tr>
<tr>
<td>04970</td>
<td>B FETCH</td>
<td></td>
</tr>
<tr>
<td>04980</td>
<td>JUMPT BCE JUMPP×0-8×X2×T</td>
<td></td>
</tr>
<tr>
<td>04990</td>
<td>BA =184×IR3</td>
<td></td>
</tr>
<tr>
<td>05000</td>
<td>B POPUP</td>
<td></td>
</tr>
<tr>
<td>05010</td>
<td>JUMPP MRID 0×X3×IR3-3</td>
<td></td>
</tr>
<tr>
<td>05020</td>
<td>B POPUP</td>
<td></td>
</tr>
<tr>
<td>05030</td>
<td>JUMPF BCE JUMPP×0-8×X2×F</td>
<td></td>
</tr>
<tr>
<td>05040</td>
<td>BA =184×IR3</td>
<td></td>
</tr>
<tr>
<td>05050</td>
<td>B POPUP</td>
<td></td>
</tr>
<tr>
<td>05060</td>
<td></td>
<td></td>
</tr>
<tr>
<td>05070</td>
<td></td>
<td></td>
</tr>
<tr>
<td>05080</td>
<td>STCKC - TESTS 2ND STACK ITEM AGAINST 1ST ACCORDING TO THE LITERAL</td>
<td>*</td>
</tr>
<tr>
<td>05090</td>
<td>CHARACTER FOLLOWING THE OP CODE.</td>
<td></td>
</tr>
<tr>
<td>05100</td>
<td>RESULTS IN A BOOLEAN VALUE ON TOP OF THE STACK.</td>
<td></td>
</tr>
<tr>
<td>05110</td>
<td></td>
<td></td>
</tr>
<tr>
<td>05120</td>
<td></td>
<td></td>
</tr>
<tr>
<td>05130</td>
<td>STCKC BCE CHK2ND×0-9×X2×02</td>
<td></td>
</tr>
<tr>
<td>05140</td>
<td>MLWDI =4C00000027×0-1×X2</td>
<td>CONVERT TO Floating INTEGER</td>
</tr>
<tr>
<td>05150</td>
<td>CHK2ND BCE SETCND×0-18×X2×02</td>
<td></td>
</tr>
<tr>
<td>05160</td>
<td>MLWDI =4C00000027×0-10×X2</td>
<td>CONVERT TO Floating INTEGER</td>
</tr>
<tr>
<td>05170</td>
<td>SETCND MRSD 0×X3×COND-1</td>
<td>SET CONDITION TO BE TESTED</td>
</tr>
<tr>
<td>05180</td>
<td>SAR IR3</td>
<td>BUMP INSTRUCTION COUNTER</td>
</tr>
</tbody>
</table>
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  =1CO3,0-18+x2  SET STACK
05250  B  POPUP
05260  COND FBA  SETT,00  BRANCH ON FLOATING REGISTER COND
05270  SI  0-17+x2  SET STACK
05280  MCW  =1CO3,0-18+x2  PUNCTUATION FOR POSSIBLE STORE
05290  B  POPUP
05300  ************************************************************

05310  **
05320  **  COMPC-COMPARE THE CHARACTER STRING WHOSE ADDRESS IS 2ND IN THE **
05330  **  STACK AGAINST THE STRING WHOSE ADDRESS IS 1ST. IF THE **
05340  **  STRINGS ARE OF UNEQUAL LENGTH THE SHORTER IS MOVED AND **
05350  **  PADDLED ON THE RIGHT WITH BLANKS. **
05360  **  THE COMPARISON CONDITION IS CONTAINED IN THE CHARACTER **
05370  **  FOLLOWING THE OP CODE. **
05380  **
05390  ******************************************************

05400  **
05410  **  COMPC BS  IR9  CLEAR
05420  **  BS  IR10
05430  **  BS  IR15
05440  **  MCW  0-14+x2,IR13  LEFT ADDRESSES OF STRINGS
05450  **  BCE SUBMV1,0-18+x2,34  IF SUBSTRING PUT IN WORKING STORE
05460  **  MVRT1 MCW  0-5+x2+IR12  TO INDEX REGISTERS
05470  **  MVRT2 MRIN  0+x13,0  IF SUBSTRING PUT IN WORKING STORE
05480  **  SAR  IR9  DETERMINE
05490  **  MRIN  0+x12,0
05500  **  SAR  IR10  STRING
05510  **  BS  IR13,IR9
05520  **  BS  IR12,IR10  LENGTHS
05530  **  SW  IR9-2
05540  **  C  IR10,IR9  TEST
05550  **  CW  IR9-2
05560 BL MOVNXT
05570 BH MOVTOP
05580 TSTRNG BS IR15 SET PROPER
05590 MRSD 0+X3,IR15 CONDITION
05600 SAR IR3
05610 MRSD CNDTBL+X15,CNDTST CODE
05620 SW 0+X13 WORK MARK FOR COMARE
05630 MRIN 0+X13,0+X12 POSITION
05640 SAR IR13 INDEX REGISTERS
05650 SBR IR12 TO RIGHT END
05660 MCw +ENDPRG,ENDADR RESTORE STRING WORKING STORAGE POINTER
05670 C 0-1+X12,0-1+X13 COMPARE
05680 CNDTSTBTC SETT,00 CONDITIONAL BRANCH
05690 B SETF
05700 MOVNXT EQU *
05710 MRIDR 0+X13,(ENDADR-3) MOVE TO TEMPORARY LOCATION
05720 SBR IR15
05730 CI 0-1+X15
05740 MCw ENADDR+IR13
05750 MCw ENADDR+IR11
05760 BA IR10,IR11 RIGHT END OF PADDED FIELD
05770 MRBLNK MRSDR BLNK+0+X15 PAD
05780 SBR IR15
05790 C IR11,IR15 WITH
05800 BEH SETPNC
05810 B MRBLNK BLANKS
05820 SETPNC SI 0-1+X11
05830 B TSTRNG
05840 MOVTOP MRIDR 0+X12,(ENDADR-3)
05850 SBR IR15
05860 CI 0-1+X15
05870 MCw ENADDR+IR12
05880 MCw ENADDR+IR11
05890 BA IR9,IR11 RIGHT END OF PADDED FIELD
05900 B MRBLNK
05910 SUBMV1 MRID 0-13+X2,IR10-1 GET LENGTH
05920 SW 0+X13 WORD MARK FOR MOVE
ADD - REPLACE THE TOP TWO NUMBERS ON THE STACK WITH THEIR SUM  
SUB - REPLACE THE TOP TWO NUMBERS ON THE STACK WITH THEIR  
DIFFERENCE (2ND - 1ST)  
MULT - REPLACE THE TOP TWO NUMBERS ON THE STACK WITH THEIR PRODUCT  
DIV - REPLACE THE TOP TWO NUMBERS ON THE STACK WITH THEIR  
RATIO (2ND / 1ST)  
NEG - UNARY MINUS OPERATION ON THE ELEMENT ON THE TOP OF THE STACK  
NOTE THE TYPE CONVERSIONS WHICH MAY TAKE PLACE.  
*
06300 BCE INTADD,ARI,01 
06310 TMA 0-1+x2,00 
06320 AMA 0-10+x2,00 
06330 SETBck FBI FERR+06 
06340 BS =1B99+IR2 
06350 MLWDR =8800+0-1+x2 
06360 SI 0-1+x2 
06370 TAM 0-1+x2,00 
06380 LCA =1B20+0-9+x2 
06390 B FETCH 
06400 INTADD Sw 0-17+x2 
06410 BA 0-5+x2,0-14+x2 
06420 BS =1B99+IR2 
06430 B FETCH 
06440 SUB B CNVCHK 
06450 BCE INTSUB,ARI,01 
06460 TMA 0-10+x2,00 
06470 SMA 0-1+x2,00 
06480 B SETBCK 
06490 INTSUB Sw 0-17+x2 
06500 BS 0-5+x2,0-14+x2 
06510 BS =1B99+IR2 
06520 B FETCH 
06530 MULT B CNVCHK 
06540 BCE INTMLT,ARI,01 
06550 TMA 0-1+x2,00 
06560 MAM 0-10+x2,00 
06570 B SETBCK 
06580 INTMLT BIM 0-5+x2,0-14+x2 
06590 BS =1B99+IR2 
06600 B FETCH 
06610 DIV MRSD 0-9+x2,ARI 
06620 BA 0-18+x2,ARI 
06630 BCE NUMER,0-9+x2,02 
06640 MLWDI =4C000000027,0-1+x2 
06650 NUMER BCE DODIV,0-18+x2,02 
06660 MLWDI =4C000000027,0-10+x2 

Stack Top to Fro
Add Next
Branch on Any Error
Reduce Stack
Clear Extraneous Punctuation
Proper Punctuation
Store Result
Data Type
Word Mark to Stop Addition
Binary Add
Reduce Stack
If Integers Branch
Floating Point
Integer Subtraction
Reduce Stack
If Integers Branch
Floating Point
Sum of Types for Later Testing
Convert to Unnormalized Fltng Pt
Convert to Unnormalized Fltng Pt
06670 DODIV AMA 0-1 + x2,71 NORMALIZED LOAD TO FRI
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 =1B9,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 =4B0,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;
06860 CNVCHK SrR CNVTR+4 SET RETURN
06870 MRSD 0-9 + x2,ARI SET
06880 SST 0-18 + x2,ARI,02 INDICATOR
06890 BCE (CNVTR+1),ARI,01 IF ALL INTEGER RETURN
06900 BCE (CNVTR+1),ARI,02 IF ALL FLTNG PT RETURN
06910 BCE CNVTOP,0-9 + x2,01 DO
06920 MLwD =4C00000027,0-10 + x2 NECESSARY
06930 B (CNVTR+1) CONVERSION
06940 CNVTOP MLwD =4C00000027,0-1 + x2
06950 CNVTR B *
06960**************************************************************************************************************************************************************************************************
06970* OR REPLACE TOP TWO ELEMENTS WITH LOGICAL SUM
06980* AND REPLACE TOP TWO ELEMENTS WITH LOGICAL PRODUCT
07000* NOT-LOGICAL NEGATION OF TOP ELEMENT
07010* *
07020**************************************************************************************************************************************************************************************************
07030 OR BS =1B9,IR2
andonce-Compute Address According to Dope Vector (Address
in Stack) and Indices on the Stack. Resulting Address
on Top of the Stack.

INDEXA-Same as INDEXA Except That Addressed Item Is Loaded
To the Stack.

INDEXA EQU *
INDEXR EQU *
BS IR13 CLEAR
MRSD O+X3,IR13 INSERT SUBSCRIPT COUNT
SUBCHK BCE INTSBS+0-9+X2,01 CHECK FOR
TMA 0-1+X2,00 SUBSCRIPT CONVERSION
B IFIX
R B CNVERR
MCW IR15,0-5+X2 MOVE INTEGER RESULT TO THE STACK
INTSBS BS =1B9,IR2
BS =1B1,IR13
BCE CMPADD+IR13,00
B SUBCHK
CMPADD MCW IR2+IR9 SAVE STACK POINTER
MCW 0-5+X2,IR10 ADDRESS OF DOPE VECTOR
MRSD 0+X3,IR13 SUBSCRIPT COUNT
07410 SAR IR3
07420 MRIV 1*X2,IR15-3 FIRST SUBSCRIPT
07430 MORIDX BS =IB1,IR13 REDUCE SUBSCRIPT COUNT
07440 BCE SUBDNE,IR13,00 IF NO MORE THEN FINISH
07450 BA =IB4,IR10 POINT TO NEXT MULTIPLIER
07460 BA =IB9,IR2 NEXT SUBSCRIPT
07470 BIM 3*X10,4*X12 SUBSCRIPT * MULTIPLIER
07480 BA 4*X2,IR15 ACCUMULATE VARIABLE PART
07490 B MORIDX
07500 SUBDNE BIM 7*X10,IR15 MULTIPLY TIMES ELEMENT SIZE
07510 BA 11*X10,IR15 ADD CONSTANT PART
07520 MCW IR9,IR2 RESTORE STACK POINTER
07530 C IR15,DYNEND CHECK FOR EXTREMELY WILD INDEX
07540 BL IDXERR
07550 MLW0 IR15,0-5*X2 ADDRESS TO STACK
07560 BCE FETCH,0-2*X3,67 IF INDEXA THEN DONE
07570 MRIDR 0*X15,0-8*X2 ELSE LOAD RESULT
07580 B FETCH
07590*******************************************************************************
07600* CAT -CONCATENATE TWO STRINGS WHOSE ADDRESSES ARE IN THE TOP
07601* TWO POSITIONS OF THE STACK. THE RESULTING STRING IS PLACED
07602* IN WORKING STRING STORAGE AND THE TWO ADDRESSES ARE REPLACED
07603* BY THE ADDRESS IN WORKING STORAGE.
07604*
07605*******************************************************************************
076060 CAT MCW 0-14+X2,IR13 ADDRESS TO REGISTER
076080 BCE CTSUB1,0-18+X2,34 CHECK FOR SUBSTRING
076090 MRIDR 0*X13,(ENDADR-3) CHECK FOR SUBSTRING
07700 SBR IR12
07710 CI 0-1*X12 CLEAR ITEM MARK
07720 CAT2 MCW 0-5*X2,IR13 CHECK FOR SUBSTRING
07730 BCE CTSUB2,0-9+X2,34 CHECK FOR SUBSTRING
07740 MRIDR 0*X13,0*X12 NEXT LOCATION IN WORKING STORAGE
07750 SBR IR13 CLEAR POSSIBLE WORD MARK
07760 CAT3 CW 0*X12 ADDRESS OF RESULTING STRING
07770 MCW ENDADR,0-14+X2
0778C  MCW  IR13,ENDADR  NEXT LOCATION IN WORKING STORAGE
0779C  BS  =1B9*IR2  POP STACK
0780C  B  FETCH
0781C  CTSUB1  BS  IR10
0782C  MRID  0-13+x2,IR10-1  MOVE
0783C  BA  IR13,IR10  SUBSTRING
0784C  SI  0-1+x10  TO
0785C  MRIDR  0+X13, (ENDADR-3)  WORKING
0786C  SBR  IR12  STORAGE
0787C  CI  0-1+x12
0788C  CI  0-1+x10
0789C  Sw  (ENDADR-3)
0790C  SST  =1B0,0-18+x2,70  REMOVE SUB STRING MARK
0791C  B  CAT2
0792C  CTSUB2  BS  IR10
0793C  MRID  0-4+x2,IR10  CATENATE
0794C  BA  IR13,IR10  SUBSTRING
0795C  SI  0-1+x10  TO
0796C  MRIDR  0+X13, 0+X12  WORKING
0797C  SBR  IR13  NEXT LOCATION IN WORKING STORAGE
0798C  CI  0-1+x10  STORAGE
0799C  B  CAT3

0800C**********************************************************************************************
0801C*  SUBSTR - PL/I SUBSTRING OPERATION  *
0802C*  **********************************************************************************************
0803C**********************************************************************************************
0804C********SUBSTR - PL/I SUBSTRING OPERATION********
0805C  SBSTR  Sw  0-26+x2  COMPUTE
0806C  SBSTR  BA  0-14+x2,0-23+x2  STARTING
0807C  BS  =1B1,0-23+x2  POINT OF STRING
0808C  C  0-5+x2,:00:  IF NO LENGTH GIVEN
0809C  BE  FNDLN  COMPUTE IT
0810C  MRIDI  0-6+x2,0-22+x2  ELSE STORE LENGTH
0811C  SBSTRL  BS  =1B18,IR2  POP STACK
0812C  LCA  =1C34,0-9+x2  MARK AS SUBSTRING ELEMENT
0813C  B  FETCH
0814C  FNDLN  MCW  0-23+x2,IR13  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 0+X3,FMTCD1-3 SET FORMAT CODE ADDRESS
SAR  IR3 BUMP SEQUENCE COUNTER
BA  =1B1,FMTCD1 JUMP OVER BEGINNING MARKER
MCW  FMTCD1,FMTCD2 SAVE FOR REPETITION
SST  0+X3,INOROT,70 INPUT OR OUTPUT CODE
SST  0+X3,DEVTYP,07 STRING OR UNIT RECORD
BA  =1B1,IR3 BUMP SEQUENCE COUNTER
BCE  STRSET,DEVTYP,00 IF STRING INITIALIZE
BCE  FETCH,INOROT,70 ELSE INITIALIZE FOR UNIT RECORD
MCW  +PRNTBF,IR8 OUTPUT PRINT BUFFER
MCW  +PRNTBF+132,STREND
B  FETCH
FMTCD1 DCW  =4
FMTCD2 DCW  =4
INOROT DCW  :0:
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,LNGSTR-1 LENGTH
MCW  STRADD,STREND
BA  LNGSTR,STREND STRING END ADDR (PLUS ONE)
08520  BS  =1B9,IR2  POP STACK
08530  BCE  FETCH,INOROT,70  IF INPUT STRING ALL DONE
08540  MCW  STRADD,IR8  ELSE SET OUTPUT POINTER
08550  MCW  STRADD,IR9
08560  CLRMOR  MRSDR  BLNK,0+X9  CLEAR OUTPUT
08570  SBR  IR9  STRING
08580  C  IR9,STREN  TO
08590  BH  CLRMOR  BLNK
08600  SI  0-1+X9
08610  B  FETCH

08620 ***************************************************************************
08630  */
08640  PUT- IF OUTPUT IS TO THE PRINTER PRINT AND RESET
08650  */  OUTPUT BUFFER POINTER
08660  */  IF OUTPUT IS TO A STRING SET RIGHT END PUNCTUATION
08670  */  AND RESET BUFFER POINTER TO PRINTER BUFFER.
08680  */
08690 ***************************************************************************

08700  PUT  BCE  STRPUT,DEVTyp,00
08710  :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

POSITION TO PROPER CHARACTER

ONE ORIGIN INDEX FOR BUFFER POINTER

IF STRING THEN ERROR

IF STRING THEN ERROR

DECREMENT LINE COUNT

SET BUFFER POINTER
09260L:PUT PRINT,DUMMY, REQUIRED
09270 BS =1B1,IR10 NO OF
09280 B SKPTST BLANK LINES
09290 DUMMY DCw :A:
09300 L DCw =1C45
09310 FMTRST MCw FMTCD2,FMTCD1 RESET FORMAT POINTER
09320 B EDIT
09330 DATFMT EQU * CHECK FOR A FORMAT CODE
09340 BCE AFORM,(FMTCD1-3),10 JUMP OVER CODE TYPE
09350 BCE LFORM,(FMTCD1-3),11 A FIELD LENGTH
09360 BCE IFORM,(FMTCD1-3),12 NEXT FORMAT CODE LOCATION
09370 BCE LFORM,(FMTCD1-3),13 ADDRESS OF STRING
09380 B FMTERR IF NO A FORMAT LENGTH GET FROM STRING
09390 AFRM 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 FIELKD
09460 BA IR8,IR10 BUFFER POINTER PLUS LENGTH
09470 CHKbff C IR10,*STREND CHECK FOR
09480 BL BUFOFL OVERFLOW PROBLEMS
09490 SW 0*X8 MOVE STOPPER IN RECEIVING FIELD
09500 MCw 0-1*X9,0-1*X10 MOVE DATA TO RECEIVING FIELD
09510 CI 0-1*X10 CLEAR POSSIBLE ITEM MARK
09520 Cw 0*X8 REMOVE MOVE STOPPER
09530 MCw IR10,IR8 NEXT BUFFER LOCATION
09540 BS =1B9,IR2 POP STACK
09550 B FETCH
09560 GTALN BCE GTSBLN,0-9*X2,34 IF SUBSTRING GET LENGTH FROM STACK
09570 MRIN 0*X9,0*X8 ELSE COMPUTE END LOCATIONS
09580 SAR IR9 STRING RIGHT END ( PLUS ONE )
09590 SBR IR10 BUFFER RIGHT END ( PLUS ONE )
09600 B CHKbff
09610 GTSBLN MRID 0-4*X2*IR10-1 LENGTH FROM STACK
09620 B AFRMA
09630 BUFDFL EQU *
09640 :PUT PRINT•BUFDFL
09650 B ERRLOC
09660 BUFDFL DCW: A ****BUFFER OVERFLOW****:
09670 L DCW=1C45
09680 IFORM BA=1B1•FORMCD1 BUMP FORMAT CODE POINTER
09690 MRID (FORMCD1-3)•IR10-3 I FIELD LENGTH
09700 SAR FORMCD1 NEXT FORMAT CODE POINTER
09710 BA IR8•IR10 BUFFER POINTER PLUS LENGTH
09720 C IR10•STREND CHECK
09730 BL BUFDFL OVERFLOW
09740 BS CNVFLD CONVERT
09750 SW 0-8+X2 MARK FOR MOVE
09760 BBE IFMNEG•0-8+X2•40 CHECK FOR MINUS SIGN
09770 ML WD 0-5+X2•CNVFLD-2 BINARY
09780 IFMCNV TMA CNVFLD•00 INTEGER
09790 G TD CNVFLD•00 TO DECIMAL
09800 MCW; 0•0-2+X10 ZERO SUPPRESSION SYMBOl
09810 SW 0•X8 LEFT END OF EDIT CONTROL
09820 MCE CNVFLD•0-1•X10 MOVE AND EDIT DECIMAL INTEGER
09830 MCW IR10•IR8 NEXT BUFFER LOCATION
09840 BS =189•IR2 POP STACK
09850 BBE IFMNEG•1+X2•40 CHECK FOR NEGATIVE SIGN PLACEMENT
09860 B FETCH
09870 IFMNEG SW CNVFLD•5 CONVERT TO POSITIVE
09880 BS 0-5+X2•CNVFLD-2 GO CONVERT TO DECIMAL
09890 CW CNVFLD•5 FIND BLANK
09900 B IFMCNV NEXT POSITION TO THE LEFT
09910 IFMSGN BCE IFMSST•0-1•X10•15
09920 BS =181•IR10
09930 B IFMSGN
09940 IFMSST MCW;:•0-1•X10
09950 B FETCH
09960 LFORM BA=1B1•FORMCD1
09970 MRID (FORMCD1-3)•IR10-3 NEXT FORMAT CODE POINTER
09980 SAR FORMCD1
09990 BA IR10•IR8
MOVE TRUE

MOVE FALSE

OUTPUT LENGTH

NEXT FORMAT CODE

OPERAND TO FRO

POP STACK

CONVERT TO FLOATING DECIMAL

ADDRESS OF FLOATING DECIMAL FIELD

PREPARE FOR BASIC E FORMAT

NO SIGN IF PLUS

CLEAR SIGN

MINUS SIGN

PREPARE FOR ROUNDING OVERFLOW

ROUND RESULT

TEST NO OVERFLOW

MOVE RIGHT ONE CHARACTER ON OVERFLOW

ADJUST EXPONENT

RESTORE FIELD MARKER

RIGHT END OF OUTPUT FIELD

BRANCH IF NEGATIVE EXPONENT

STRIP SIGN

TEST EXPONENT

TEST EXPONENT

RIGHT MOVE STOPPER

ZERO EXPONENT TEST

MOVE DIGITS LEFT OF
**10370** MVPT MCW :*:1+X5  MOVE IN .
**10380** MRID DECRES-13+X1:2+X5  MOVE DIGITS TO THE RIGHT OF .
**10390** SBR IR5
**10400** NZTST BBE NZTST,0-1+X5,77  REPLACE
**10410** MCW :*:O+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 :*: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 NXTF LD
**10530** DCW :0:
**10540** DCW =11
**10550** DECRES DCW =3
**10560** NXTF LD MRIN 0+X5,0
**10570** SAR IR5
**10580** CI 0-1+X5  REMOVE ITEM MARK
**10590** BA IR10,IR3  NEXT OUTPUT BUFFER LOCATION
**10600** B Fetch

---

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

---

**10660*** 
**10670** GET BCE GETFM,DEVTYPE,00  IF STRING INPUT BYPASS EOF TEST
**10680** BCE ENDFMT,ENDF,T  TEST FOR CLOSED FILE
**10690** GETFM BCE IFMRST,(FMTCD1-3),77  CHECK FOR FORMAT RESET
**10700** BBE 1DTFMT,(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     FMTERR
10750  IFMRST MCW  FMTCD2, FMTCD1
10760  B     GET
10770  NXTIFM LA  =1BI, FMTCD1
10780  B     GET
10790  ISPCE BA  =1B4, FMTCD1
10800  ISPCE MCW  (FMTCD1-3), IR9
10810  SPCCNT C  IR9, :000:
10820  BE    NXTIFM
10830  BCE   STRSPC, DEVTYP, 00
10840  EA    =1B1, IR6
10850  C     +INPUT+79, IR6
10860  BH    GETIPT
10870  BS    =1B1, IR9
10880  B     SPCCNT
10890  STRSPC BA  IR9, STRADD
10900  C     STREN, STRADD
10910  BEH   BUFOFL
10920  B     NXTIFM
10930  I-Col BCE  STRCOL, DEVTYP, 00
10940  I-Col MCW  +INPUT, IR6
10950  BA    =1B4, FMTCD1
10960  BA    (FMTCD1-3), IR6
10970  BS    =1B1, IR6
10980  C     +INPUT+79, IR6
10990  BH    BUFOFL
11000  B     NXTIFM
11010  STRCOL MCW  STREN, STRADD
11020  BS    LNEGSTR, STRADD
11030  BA    =1B5, FMTCD1
11040  BA    (FMTCD1-3), STRADD
11050  BS    =1B1, STRADD
11060  C     STREN, STRADD
11070  BEH   BUFOFL
11080  B     NXTIFM
11090  ISKP BCE  FMTERR, DEVTYP, 00
11100  BA    =1B1, FMTCD1

IF NONE OF THE ABOVE THEN ERROR
RESET FORMAT POINTER
NEXT FORMAT CODE
POINT TO SPACE COUNT
MOVE IT TO INDEX REG.
TEST FINISH
TEST FOR STRING INPUT
TEST FOR
NEW RECORD REQUIRED
DECREMENT SPACE COUNT
SPACE
AND
TEST FOR STRING OVER FLOW
TEST FOR STRING
TEST FOR STRING
CHECK FOR ERROR
ESTABLISH
ORIGINAL ADDRESS
POINT TO COL INDICATOR
ONE ORIGIN INDEX FOR COLUMN POINTER
TEST
FOR ERROR
IF STRING INPUT THEN ERROR
MRIL (FMTCD1-3), IR10-3 nearby
SAR FMTCD1 nearby
ISKPTS BCE GET, IR10, 00 nearby
B GETIPT nearby
BS = IB1, IR10
B ISKPTS nearby
GETIPT SBR GETRTN, 4 nearby
BCE ENDFMS, ENDF, T nearby
CL :GET READ,
MCI (+INPUT, IR6) nearby
C INPUT + 3, :IEOF: nearby
BNE GETRTN nearby
MCW :T, *ENDF
GETRTN B *
ENDF DCW :F:
ENDFMS EQU *
PUT PRINT, EOFMES,
B ERRLOC
EOFMES DCW :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 LFRMI, (FMTCD1-3), 13
B FMTERR
BA = IB1, FMTCD1 jump over code type
MRID (FMTCD1-3), IR10-3 nearby
SAR FMTCD1 nearby
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 nearby
BA IR10, IR9 nearby
AMORE SW 0 + X6 nearby
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 FIFLD
GO FINISH MOVE
END ADDRESS OF INPUT BUFFER
CHECK LENGTH
MOVE STOPPER
FINISH PUNCTUATION AND EXIT
JUMP OVER CODE TYPE
I FIELD LENGTH
NEXT FORMAT CODE LOCATION
CHECK FOR MISMATCH
11850  IOK  BS  IR9
11860  MCw  :+:+ISGN
11870  BS  CNVFLD
11880  BCE  ISTRING,DEVTPY,00  CLEAR CONVERSION FIELD
11890  IMORE  BCE  STISGN,0+X6,-  CHECK FOR STRING INPUT
11900  BCE  STISGN,0+X6,=  TEST FOR STRING INPUT
11910  SST  0+X6,DFLD+X9,17  MOVE NUMERIC BITS ONLY
11920  BA  =1B1,IR9
11930  IENDT  BA  =1B1,IR6  BUMP
11940  C  IR6,INPEND  COUNTERS
11950  BL  GETIPT  TEST FOR
11960  C  IR10,IR9  NEXT RECORD
11970  BL  IMORE
11980  IDECMV  MCw  DFLD+1+X9, CNVFLD
11990  SST  ISGN,CNVFLD,60  MOVE DECIMAL DIGITS
12000  DTB  CNVFLD+00
12010  TAM  CNVFLD+00
12020  MRID  0-8*X2,IR9-3  ADDRESS OF RECEIVING FIELD
12030  SI  CNVFLD-2  SET SIGN IN CONVERSION FIELD
12040  MRID  CNVFLD-5,0+X9  PUNCTUATION FOR MOVING
12050  CI  CNVFLD-2  MOVE RESULT
12060  B  POPUP  CLEAR ITEM MARK
12070  DFLD  DCw  =30  POP STACK AND EXIT
12080  ISGN  DCw  :+:  0+X6,ISGN
12090  STISGN  MRSU  0+X6,ISGN  SET SIGN
12100  BS  =1B1,IR10  ADJUST CHARACTER COUNT
12110  E  IENDT  CHECK FOR RECORD END
12120  ISTRING  SST  (STRADD-3),DFLD+X9,17  MOVE DIGIT
12130  BCE  ISTISGN,(STRADD-3),+
12140  BCE  ISTISGN,(STRADD-3),-
12150  ISTRUP  EA  =1B1,IR9
12160  BA  =1B1,STRADD  IF FIELD END
12170  C  1K10,IR9  IF FIELD END
12180  EH  IDECMV  THEN SET UP FOR CONVERSION
12190  C  STREND,STRADD  CHECK BUFFER
12200  BEH  BUFOPFL  OVERFLOW
12210  B  ISTRING  GET ANOTHER DIGIT
12220  ISTSGN  MRSD  (STRADD-3) * ISGN
12230   B    ISTRUP
12240  EFRMI  EA  =161,FMTCD1
               JUMPOVER CODE TYPE
12250  MRID  (FMTCD1-3) * IR10-3
               FIELD LENGTH
12260  SAR  FMTCD1
               NEXT FORMAT CODE LOCATION
12270  BCE  EOK * 0-9 * X2,02
               CHECK FOR TYPE MISMATCH
12280   B    CNVERR
12290  EOK  BS  DEXP
               INITIALIZE DECIMAL EXPONENT
12300  BS  IR9
12310  BCE  ESTRNG,DEVTyp*00
               CHECK FOR STRING INPUT
12320  BS  IR11
12330  EMORE  MRSD  0 * X6 * EFHLD * X9
12340  SAR  IR6
12350  BA  =181,IR9
12360  C    IR6,INPND
               CHECK FOR
12370  BL    GETIPT
               NEW RECORD REQUIREMENT
12380  C    IR10,IR9
               DETERMINE
12390  BL    EMORE
               FINISH
12400   B    ECNV
               GO CONVERT
12410  EFHLD  EQU  DFLD
12420  ESTRNG  MCW  STRADD * IR9
12430  BA  IR10,IR9
12440  C    IR9,STREND
12450  BL    BUF0FL
12460  MCW  0-1 * X9 * EFHLD=1 * X10
12470  BA  IR10,STRADD
               NEXT STRING INPUT
12480  ECNV  EQU  *
12490  BS  DFRACT + 10
               CLEAR DECIMAL MANTISSA
12500  MCW  ** * FSGN
12510  MCW  ** * ESGN
12520  BS  DECEXP
               CLEAR DECIMAL EXPONENT FIELD
12530  BS  IR9
               CLEAR CHAR COUNTER
12540  BS  IR11
               MANTISSA COUNTER
12550  ECKBLK  BCE  ECNT1,EFHLD+X9=15
               CHECK FOR BLANKS
12560  BCE  FSGNST * EFHLD+X9,+
               CHECK
12570  BCE  FSGNST * EFHLD+X9,=
               SIGNS
12580  BCE  FRACT * EFHLD+X9,.
               CHECK BEGINNING OF FRACTION
<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>12590</td>
<td>BCE EXPON,EFHLD+x9,E</td>
<td>Decimal Fraction Pointer</td>
</tr>
<tr>
<td>12600</td>
<td>SST EFHLD+x9,DFRACT+x11,17</td>
<td></td>
</tr>
<tr>
<td>12610</td>
<td>BA =1BI,IR11</td>
<td>Increment Exponent (Decimal)</td>
</tr>
<tr>
<td>12620</td>
<td>A :001,DECX</td>
<td>Input Char Counter</td>
</tr>
<tr>
<td>12630</td>
<td>ECNT1 BA =1BI,IR9</td>
<td>If More</td>
</tr>
<tr>
<td>12640</td>
<td>C IR9,IR10</td>
<td>If No More</td>
</tr>
<tr>
<td>12650</td>
<td>BH ECKBLK</td>
<td>Then go check it</td>
</tr>
<tr>
<td>12660</td>
<td>B CFDFB</td>
<td>Else Convert FD/FB</td>
</tr>
<tr>
<td>12670</td>
<td>FSGNST MRSD EFHLD+x9,FSGN</td>
<td></td>
</tr>
<tr>
<td>12680</td>
<td>B ECNT1</td>
<td></td>
</tr>
<tr>
<td>12690</td>
<td>FRACT BA =1BI,IR9</td>
<td>Bump Char Pointer</td>
</tr>
<tr>
<td>12700</td>
<td>C IR9,IR10</td>
<td>If No More</td>
</tr>
<tr>
<td>12710</td>
<td>HEL CFDFB</td>
<td>Then Convert FD/FB</td>
</tr>
<tr>
<td>12720</td>
<td>BCE EXPON,EFHLD+x9,E</td>
<td>Else Check Exponent</td>
</tr>
<tr>
<td>12730</td>
<td>BCX EXPON,EFHLD+x9,15</td>
<td>If Blank then look for Exponent</td>
</tr>
<tr>
<td>12740</td>
<td>SST EFHLD+x9,DFRACT+x11,17</td>
<td>Else Move Digit</td>
</tr>
<tr>
<td>12750</td>
<td>BA =1BI,IR11</td>
<td></td>
</tr>
<tr>
<td>12760</td>
<td>B FRACT</td>
<td>Go look for more</td>
</tr>
<tr>
<td>12770</td>
<td>EXPON BCE EXPR,EFHLD+x9,E</td>
<td></td>
</tr>
<tr>
<td>12780</td>
<td>BA =1BI,IR9</td>
<td>Bump Char Pointer</td>
</tr>
<tr>
<td>12790</td>
<td>C IR9,IR10</td>
<td>If No More</td>
</tr>
<tr>
<td>12800</td>
<td>BL CFDFB</td>
<td>Then Convert FD/FB</td>
</tr>
<tr>
<td>12810</td>
<td>B EXPON</td>
<td></td>
</tr>
<tr>
<td>12820</td>
<td>EXPR MCW : ++ ,ESGN</td>
<td>Default Sign</td>
</tr>
<tr>
<td>12830</td>
<td>BA =1BI,IR9</td>
<td></td>
</tr>
<tr>
<td>12840</td>
<td>BCE ESGNST,EFHLD+x9,-</td>
<td>Check for</td>
</tr>
<tr>
<td>12850</td>
<td>BCE ESGNST,EFHLD+x9,+</td>
<td>Exponent Sign</td>
</tr>
<tr>
<td>12860</td>
<td>B EXPNUM</td>
<td>Get Exponent</td>
</tr>
<tr>
<td>12870</td>
<td>ESGNST MRSD EFHLD+x9,FSGN</td>
<td>Set Sign</td>
</tr>
<tr>
<td>12880</td>
<td>BA =1BI,IR9</td>
<td></td>
</tr>
<tr>
<td>12890</td>
<td>EXPNUM SW EFHLD+x9</td>
<td></td>
</tr>
<tr>
<td>12900</td>
<td>BS DECEXP</td>
<td></td>
</tr>
<tr>
<td>12910</td>
<td>MCW EFHLD-1+x10,DECEXP</td>
<td></td>
</tr>
<tr>
<td>12920</td>
<td>CW EFHLD+x9</td>
<td></td>
</tr>
<tr>
<td>12930</td>
<td>CFDFB SST ESGN,DECEXP,60</td>
<td>Set Sign</td>
</tr>
<tr>
<td>12940</td>
<td>SST FSGN,DFRACT+10,60</td>
<td>Set Fraction Sign for Conversion</td>
</tr>
<tr>
<td>12950</td>
<td>A DECEXP,DEXP</td>
<td>Accumulate Dec Exp for Conversion</td>
</tr>
</tbody>
</table>
CALL CONVERSION ROUTINE
ADDRESS OF RIGHT END OF 14 CHAR
FLOATING DECIMAL FIELD
ERROR INSTRUCTION
ADDRESS OF RESULT
CLEAR ANY EXTRANEOUS PUNCTUATION
STORE RESULT
ITEM MARK RIGHT
POP STACK AND FETCH NEXT INSTR

JUMP OVER CODE TYPE
FIELD LENGTH
NEXT FORMAT CODE LOCATION
CHECK FOR TYPE MISMATCH
CHECK FOR STRING INPUT
CHECK
NEW RECORD REQUIREMENT
MOVE ADDRESS
DETERMINE EXCESS CHAR
NEW LENGTH
CHECK OVERFLOW
ADDRESS OF RECEIVING FIELD
MRSRSR 0-1*9*0*10 MOVE INPUT
MCW IK9*STRADD NEXT STRING INPUT
B POPUP POP STACK AND FETCH
*********************************************************************
* STOP - PRINT INSTRUCTION COUNT MESSAGE AND EXIT. *
* *********************************************************************

STOP BS CNVFLD
MCW INSCT,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)
L :PUT PRINT,CNTMES,
B (164) EXIT
CNTMESDCw :A *****INSTRUCTION COUNT = : 
PRTCNT DC =9
DC :****:
L DCw =1C45
*********************************************************************
* ERROR MESSAGES *
* *********************************************************************
ERROR EQU *
:PUT PRINT,BADOP,
B ERRLOC
BADOP DCw :A *****ILLEGAL OP CODE***:
L DCw =1C45
dynofl EQU *
:PUT PRINT,STROFL,
B 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 FERMESDCw :A ****FLTNG PT OVFLW OR ZERO DIVIDE****:
13770 L DCw =1C45
13780 IDXERR EQU *
13790 L :PUT PRINT,IDXMES,
13800 B ERRLOC
13810 IDXMESDCw :A ****INDEXED ADDRESS BEYOND DYNAMIC STORAGE****:
13820 L DCw =1C45
13830 TYPERR EQU *
13840 L :PUT PRINT,TYPMES,
13850 B ERRLOC
13860 TYPMESDCw :A ****INCONSISTENT DATA TYPE ERROR****:
13870 L DCw =1C45
13880 FMTERR EQU *
13890 L :PUT PRINT,FMTMES,
13900 B ERRLOC
13910 FMTMESDCw :A ****FORMAT CODE ERROR****:
13920 L DCw =1C45
13930 ERRLOC BS LODLOC,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 :****:
DCW =1C45
CNVFLD DCW =11B0
ENDPRG EQU *

SYMBOL DEFINITION - CARD REFERENCE INDEX

ADCQV 01640; ADD 06290; AFIN 11560; AFORM 09390; AFRMA 09450;
AFRMI 11370; ALCRT1 02600; ALCRT2 02630; ALLOC 02490; AMORE 11470;
AND 07060; AOK 11420; ARI 06850; ASPLIT 03180; ASTRNG 11710;
BADUP 13610; BFOFL 09660; BGSTR 09050; BLNK 06160; BSTF 07130;
BSTT 07100; BUFQFL 09630; CAT2 07720; CAT3 07680; CNVRTN 03410;
CFDFB 12930; CH<2NF 05150; CHKBFF 09470; CHKTYP 03180; CHRSTR 03710;
CHRTPN 03750; CLRMTR 08560; CMPADD 07380; CNDTBL 06150; CNDTST 05680;
CNTMES 13490; CNVCHK 06860; CNVERR 13680; CNVFLD 14080; CNVRTN 03410;
COND 05210; CONPRT 02860; CSBSTR 03930; CTSUBL 07810; CTSUB2 07920;
CURLEV 04550; CVEMS 13710; DATFMT 09330; DECEXP 13070; DECRE5 10550;
DEVTYP 08440; DEXP 13060; DFLD 12070; DFRCT 13050; DISPLY 04490;
DIV 06610; DODIV 06670; DTYPE 02220; DUMMY 09290; DVDNE 02750;
DYNAM 02280; DYEND 04670; DYNOFI 13630; DYNSTR 00450; ECKBLK 12550;
ECNT1 12630; ECNV 12480; EDIT 08870; EFHLD 12410; EFORM 10070;
ECNTL 12630; ENDF 11250; ENDFMS 11260; ENDPGR 14100; ENTPRO 04220; EFORM 11290;
EOK 12290; EERRLOC 13930; ERROR 13580; ESGN 13080; ESGNST 12870;
ESTRNG 12420; EWORD 14030; EXPNUM 12890; EXPON 12770; EXPR 12820;
FERMS 13760; FERR 13730; FETCH 01170; FIXFLT 03640; FLAG 04130;
FILECK 04310; FMT 08290; FMTCMD 08410; FMTCMD2 08420; FMTERP 13880;
FMTMES 13910; FMTST 09310; FNDLNG 08140; FOUT 10050; FRACT 12690;
FSIZE 13090; FNSG 12670; GET 10670; GETFM 10690; GETIP 11170;
GETRTN 11240; GTALN 09560; GTSBLN 09610; HALT 14020; HOLD 04690;
ICOL 10930; IDECMV 11980; IDTFMT 11310; IDXXERR 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 OCTAL ADDRESS DEFINITIONS OF PERTINENT SYMBOLS IN THE RESIDENT INPUT/OUTPUT ROUTINE.

00400 #RDWR CEGU =4C00000754
00410 READ CEQU =4C00005430
00420 INPUT CEQU =4C00006144
00430 OUTPUT CEQU =4C00006265
00440 PRINT CEQU =4C00005647
00450 #SKP CEGU =4C00000756

00460 GENFLD EQU 215
00470 LODFLD EQU 219
00480 STCKF1 EQU 223
00490 STCKF2 EQU 227
00500 SYMF1 EQU 231
00510 SYMF2 EQU 235
00520 GENFLD EQU 215
00530 LODFLD EQU 219
00540 STCKF1 EQU 223
00550 STCKF2 EQU 227
00560 SYMF1 EQU 231
00570 SYMF2 EQU 235
CONTAINS ADDRESS OF CODE GENERATION LOC
CONTAINS METAX PROGRAM LOADING ADDRESS
BEGINNING PUSH DOWN STACK ADDRESS
CONTAINS STACK LIMIT ADDRESS
ADDRESS OF SYMBOL TABLE START
(START OF DYNAMIC STORAGE FOR PLEX
OBJECT PROGRAMS)
CONTAINS SYMBOL TABLE LIMIT ADDRESS
(LIMIT OF DYNAMIC STORAGE FOR PLEX
OBJECT PROGRAMS)
COMPLETION CODE FIELD SET BY COMPILERS
DISK LOADING OPTION FIELD
GO OPTION FIELD
POST LISTING OPTION FIELD
METAX PROG NAME FIELD
INTERPRETER NAME FIELD
SYMBOL FIELD USED BY COMPILERS
INTERPRETER INSTRUCTION COUNT FIELD
LOCATION FOR RESIDENT METAX PROGRAM
LOCATION FOR RESIDENT CONTROL RECORD ANALYZER
SET FOUR CHAR ADDRESSING MODE
GET NAMES OF RESIDENT METAX PROGRAMS
SAVE FOR LATER USE
SET LOCATION FOR CONTROL RECORD ANALYZER
NAME TO METAX COMMUNICATIONS FIELD
SEGMENT AND
NAME OF DISK TO MEMORY LOAD PROGRAM
SET UP RETURN FOR RETURN START
FETCH AND EXECUTE
SET UP RETURN FOR RETURN START
EXIT POINT
RETURN TO LOADER
SET NAME OF RESIDENT METAX PROGRAM
SET LOCATION FOR RESIDENT METAX PROGRAM
EXIT POINT
FETCH RESIDENT METAX PROGRAM
ZERO INSTRUCTION COUNT
COMMUNICATIONS AREA
FIELDS
FOR EXECUTING
CONTROL RECORD ANALYZER
INTERPRETER SEGMENT
AND NAME
RETURN POINT
FETCH AND EXECUTE
ZERO INSTRUCTION COUNT
INITIALIZE COMPLETION CODE
SET UP
MEMORY
CLEAR
<table>
<thead>
<tr>
<th>Line</th>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01120</td>
<td>Sw 0</td>
<td>x15</td>
<td>OPERATION</td>
</tr>
<tr>
<td>01130</td>
<td>B CLEAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01140</td>
<td>MCw LODFLD,LODSAV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01150</td>
<td>C PROGNAME,SAVNME</td>
<td></td>
<td>TEST EXECUTION OF RESIDENT MFTAX PROGRAM</td>
</tr>
<tr>
<td>01160</td>
<td>BNE MFETCH</td>
<td></td>
<td>IF NOT GO THE RIGHT ONE</td>
</tr>
<tr>
<td>01170</td>
<td>MCw MTFPRG,LODFLD</td>
<td></td>
<td>ALTER EXECUTION LOCATION TO RESIDENT PROG</td>
</tr>
<tr>
<td>01180</td>
<td>GETINT MCw INTNAME,75</td>
<td></td>
<td>INTERPRETER SEGMENT</td>
</tr>
<tr>
<td>01190</td>
<td></td>
<td></td>
<td>AND NAME</td>
</tr>
<tr>
<td>01200</td>
<td>BNE MFETCH</td>
<td></td>
<td>RETURN POINT</td>
</tr>
<tr>
<td>01210</td>
<td>B (168)</td>
<td></td>
<td>FETCH AND EXECUTF</td>
</tr>
<tr>
<td>01220</td>
<td>INTRT2 BCE FATAL,CMPLCD,F</td>
<td></td>
<td>TEST FOR FATAL ERROR ACTION</td>
</tr>
<tr>
<td>01230</td>
<td>BCE LTODSK,DSKLUD,Y</td>
<td></td>
<td>LOAD COMPILTED PROGRAM TO DISK IF REQUESTED</td>
</tr>
<tr>
<td>01240</td>
<td>BCE EOFSTST,EXCPPG,N</td>
<td></td>
<td>IF NO GO THEN SEARCH FOR END OF FILE</td>
</tr>
<tr>
<td>01250</td>
<td>GO EQU *</td>
<td></td>
<td>ELSE MOVE AND RELOCATE COMPILTED</td>
</tr>
<tr>
<td>01260</td>
<td>MCw LODSAV,LODFLD</td>
<td></td>
<td>PROGRAM FOR EXECUTION</td>
</tr>
<tr>
<td>01270</td>
<td>SI 1+x5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01280</td>
<td>Sw 1+x5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01290</td>
<td>MLwD GENFLD,IR15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01300</td>
<td>MLwD LODFLD,IR14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01310</td>
<td>MORPRG MRwDR 0+x15,0+x14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01320</td>
<td>SAR IR15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01330</td>
<td>SBR IR14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01340</td>
<td>MRLDR 0+x15,0+x14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01350</td>
<td>SAR IR15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01360</td>
<td>SBR IR14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01370</td>
<td>BA LODFLD,0-1+x14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01380</td>
<td>CW 0-3+x14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01390</td>
<td>CW 0-1+x14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01400</td>
<td>BCE 0LKCNT,0-4+x14,00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01410</td>
<td>ENDTST C IR15,IR5</td>
<td></td>
<td>TEST POSSIBLE BLOCK PSEUDO OP CODE</td>
</tr>
<tr>
<td>01420</td>
<td>BH MORPRG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01430</td>
<td>MLwD =3C117776,IR15</td>
<td></td>
<td>PREPARE FOR CLEARING REMAINING MEMORY</td>
</tr>
<tr>
<td>01440</td>
<td>MRSR =480,0+x14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01450</td>
<td>SBR IR14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01460</td>
<td>SI 0+x15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01470</td>
<td>B CLEAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01480</td>
<td>MCw SYMF1,IR14</td>
<td></td>
<td>CLEAR</td>
</tr>
</tbody>
</table>
SYMBOL (OR DYNAMIC STORAGE FOR PLFX)

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

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

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

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

SYMBOL DEFINITION - CARD REFERENCE INDEX

BLKCNT 01570; CLEAR 01640; CLRRTN 01760; CMPLCD 00660; CPACLL 00960;
DSKLOD 00670; ENDP RG 02030; ENDTST 01410; EOFTST 01810; EXCPPG 00680;
EXIT 01770; FATAL 01850; FTLMES 01880; FTRTN 02010; FTRTST 01990;
GENFLD 00560; GETINT 01180; GO 01250; INPUT 00470; INSTCT 00730;
INTNAME 00710; INTRT1 01060; INTRT2 01220; INTSEG 01660; IR13 00360;
INSTRUCTION COUNT = 47,601***
XIII. APPENDIX E

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

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

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

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

The only explicit use of word marks in object code is to mark the left hand character of an 18 bit address field as a relocatable address or pseudo-address. During loading by either the control program or MTXLDR these word marks provide a convenient scheme for marking addresses to be relocated.

The RESOLVE primitive, discussed in Chapter III, also utilizes word marks to examine object code for potential pseudo-addresses. A pseudo-address is marked by a one in the left most bit of an 18 bit address in addition to the word mark. The remaining 17 bits comprise a symbol table address as described.

The use of punctuation bits represents a significant dependence on the structure of the host machine for all of the METAX processors.

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

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

$$\text{BA } = \text{1B1,IR13}$$

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

The specification of indexing is exemplified by

$$\text{MCW TVEC}+3+X7,\text{IR14}$$

which specifies that the first operand is to be moved to index register 14. The address computation TVEC+3 is effected at translation time while the indexing via index register seven (specified by +X7) takes place at execution time.

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

$$\text{SAR IR1}$$
and

\[
\text{SBR IR10}
\]

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

\[
\text{MRIDI 0X6,SYMBOL}
\]

specifies the data and item mark bits of the first operand
are to be moved from left to right to SYMBOL with the move terminated by the first item mark in the sending field while

```mrin 0+x14,0+x13```
doing nothing more than position the A and B registers according to the first item mark found in the first operand.

Item marks are used extensively in the object code of the pseudo-machines to delimit address fields and literal operands. This scheme is not essential for addresses because the address size is fixed but it does speed up interpretation in that arithmetic instructions for updating index registers are not required in many cases.

With respect to symbolic addressing within the respective assembler programs instructions are normally addressed on the left and data fields on the right.

A reversal of these rules is used on occasion by indenting the location field by one position.

The reader is referred to the appropriate Honeywell publications (27,28) for more information on the assembler language and hardware characteristics.

The system supervisor (24) under which the METAX system operates utilizes its own communications region. Several field in this region are used by the METAX system. Decimal positions 67-75 are used to communicate the name of a program to be loaded. An indirect branch to the address in positions 168-171 (B (168)) is then a supervisor call to fetch and ex-
execute the named program. A return address may be set in positions 164-167 which is used by programs loaded into the transient region to return to the METAX control program.

All input and output operations are coded using macro routines outlined in (26). These include unit record and disk I/O functions. The METAX library is maintained in a partitioned sequential data file on disk. Additional information about certain aspects of Honeywell's version of this type of data file may be found in (25).