

72-5264

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

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

University Microfilms, A XEROX Company, Ann Arbor, Michigan

**An experimental compiler-compiler system**

**by**

**James Robert Van Doren**

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

**Major Subject: Computer Science**

**Approved:**

Signature was redacted for privacy.

**In Charge of Major Work**

Signature was redacted for privacy.

**For the Major Department**

Signature was redacted for privacy.

**For the Graduate College**

**Iowa State University  
Of Science and Technology  
Ames, Iowa**

**1971**

**PLEASE NOTE:**

**Some Pages have indistinct  
print. Filmed as received.**

**UNIVERSITY MICROFILMS**

## TABLE OF CONTENTS

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

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

## I. INTRODUCTION

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

We refer frequently to the METAX (META eXperimental) system due to the influence of the META series of translator writing systems (43,44,45,52,54) and we also refer to the compiler-compiler system because compilers for compiling a compiler written in a metalanguage are included. It should be noted that "metalanguage" and "compiler writing language" are used almost synonymously throughout this dissertation although it is certainly true that one may metalinguistically describe a language without necessarily implying anything about a computer program. Additionally, "semantic" may be

used in a manner meaning postsyntactic, that is, not syntactic.

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

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

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

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

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

In Chapter VI we review the results, present some conclusions and suggest some areas for future investigation.

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

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

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



## II. REVIEW OF THE LITERATURE

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

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

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

With this scheme Schorre (54) constructed translators

for two languages he called VALGOL I and VALGOL II based on restricted subsets of ALGOL.

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

BNF as it stands has a limitation as a metalanguage in that it is not possible to describe the syntax of BNF in BNF due to metasympol conflicts. The syntax specifications used in META II resolve this problem by making certain changes in metasympol 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

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

in META II as opposed to

$$\langle \text{SUBLIST} \rangle ::= \langle \text{SUBLIST} \rangle , \langle \text{EXP} \rangle \mid \langle \text{EXP} \rangle ;$$

in BNF.

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

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

.SYNTAX PROGRAM

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

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

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

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

EX3 := .ID | .STRING | ".ID" | ".NUMBER" | ".STRING"

      | "(" EX1 ")" | ".EMPTY" | "\$" EX3 ;

OUTPUT := ".OUT" "(" \$ OUT1 ")" | ".LABEL" "(" OUT1 )";

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

.END .

Certain additional liberties with BNF may be evident in that .ID and .STRING are assumed to be terminal symbols which

will in fact be references to corresponding pseudo-machine interpreter segments for recognition purposes.

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

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

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

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

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

META5, reported by Oppenheim and Haggerty (45), was developed at System Development Corporation and has been used primarily for source-to-source language translations and for

data conversion purposes. Presser (46) reports an implementation of META5 at UCLA as well as some of the history of development of META5 and the META series in general.

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

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

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

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

CWIC/360 appears to be a significant follow on to the earlier members of the META series but unfortunately detailed information does not seem to be available from System De-

velopment Corporation which has begun pursuit of profit making goals.

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

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

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

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

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

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

Additionally there are provisions for optional syntax and iteration in lieu of left recursion which are found in a

different form in the META series.

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

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

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

Reynold's COGENT system (48), which has been used largely for problems in symbolic mathematics, contains an interesting approach to syntactical recognition in that alternatives are processed in parallel in a modified top-down

analysis. COGENT is another example of a system written in its own language.

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

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

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

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



process the description of SOL to produce the requisite tables. Thus a hand compiled version of a metalanguage was used to create a compiler for another language.

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

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

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

Higman (23) describes many of the problems introduced by

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

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

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

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

### III. THE METAX METALANGUAGES

#### A. Developmental Rationale

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

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

Furthermore, the system was planned as a new effort, utilizing the ideas of many others to be certain, but not necessarily as a modification of an existing operational system. Finally, it was considered that even with the use of very sophisticated compiler writing aids, suitable restrictions would have to be placed on the goal language in order to bring the matter within the range of feasibility.

Faced with the above factors, bootstrapping a workable system with which to further develop itself was of paramount importance. Thus it was that ease of implementation and the bootstrapping capabilities of the META series of translating systems (43,44,45,52,54) provided a convenient starting point. The success of O'Neil's META-PI system (43,44) in demonstrating the approach could be extended to include considerable postsyntactic processing was a significant factor.

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

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

Moreover, the goal of implementing a PL/I-like language, albeit with the target machine being a pseudo-machine, suggested the necessity of a degree of postsyntactic processing which implied significant duplication of effort if symbolic code were to be generated which would then have to be assembled.

In view of the goal in mind and the degree of development anticipated at the outset, an intermediate goal of implementing a translator for a relatively simple language such as BASIC (33) was established in order to gain experience and experimental insight. Although this translator is not presented here, its development comprised a valuable step in reaching the main results from the author's point of view.

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

#### B. Bootstrapping

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

The hand compiled version of METAX0, with patches for initial hand translation errors, comprised 491 six bit characters of pseudo-code exclusive of the control stack and symbol table used by the interpreter. In the debugging phase of the initial step METAX0 written in METAX0 was used as source input and the output of this used to compile itself again to determine if the results were the same. The machine compiled version of METAX0 resulted in 512 characters of code, the difference being attributable to certain redundant instructions being left out of the original hand compiled version.

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

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

### C. The METAX9 Metalanguage

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

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

#### 1. Compiler definition commands

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

The definition command specified by the key word .PROG and followed by a procedure identifier must occur at the beginning of a program and nowhere else. It specifies the

first procedure to be executed which corresponds to the goal symbol in a top-down syntax analysis.

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

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

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

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



in the communications region.

Example:       PUSHLB .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).

**.STRING (STRTST):**

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

**.ID (ID):**

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

**.TSTTBA (TSTTBA):**

A test is made against the symbol table entry addressed by the last search of the symbol table. Two arguments are required, the first being an octal number pointing to the left most position of the symbol table field within the current entry to be tested (zero origin pointer) and the second

being a compile time variable against which the symbol table field is to be tested. The test is a raw binary comparison suitable for testing character fields and unsigned binary integers, the length of the test being controlled by the second argument. If equality results the true-false flag is set true, otherwise false. It should be noted that it would be an easy matter to extend this test for order relationships if desired.

Example:        .TSTTBA(07,PARMCNT)

.TSTTBL (TSTTBL):

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

Example:        .TSTTBL(00,11)

Both .TSTTBL AND .TSTTBA permit a rather wide latitude in testing symbol table entries in that any part of an entry may be examined. One could, for example, easily determine default data types for undeclared identifiers from an entry name by testing the leading character, particularly if these operators were extended to include relational testing.

FORTTRAN-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

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

as opposed to

SUBLIST := SUBLIST    "," EXP / EXP ;.

The above example also partially illustrates the use of parentheses metasyntactically; that is to say, the iteration

operator applies to both syntactic elements enclosed in parentheses. Such metasyntactic expressions may be nested to any desired level with parentheses.

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

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

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

#### 4. Semantic commands

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

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

Example:        .OUT(DYNAM,\*1)

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

The code specified by the enclosed output operators specifies that such code is to be included directly in the compiler being generated. There is a distinct difference in function between .DO and .OUT. If one discusses the difference in terms of the action the compiler takes when processing these commands then .OUT causes the generation of code



that will generate the specified code and .DO specifies directly the code to be generated perhaps in a manner more akin to an assembler language. The output operators which may be used with .DO are a subset of the operators usable with .OUT and are outlined in the section on output operators.

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

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

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

in the code stack is the amount of storage space available in the stack area.

**.MARK (MARK) :**

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

**.NEVLAB (PUSHLB) :**

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

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

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

has been previously defined the new value overlays the old one.

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

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

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

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

Some caution must be exercised, however, in determining that the top of the stack has the right elements in it. If an attempt is made to .UNSTACK a code marker or internally generated label anomolous behavior may occur due to the behavior of the POP primitive. The implementation of this

command, which may be found in the METAX9 translator in Appendix A is a relatively clear and uncluttered example of code reordering.

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

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

Example:       **.SET(DYNAMP=DYNAMB)**

**.BLKENT (BLKENT):**

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

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

The specific purposes of this command are to restore the block list pointers for the surrounding block and, if requested, to perform certain functions with respect to any unresolved symbols remaining in the terminating block. Upon detection of such a symbol the immediately surrounding block

is searched. If it is found to be resolved then certain values are moved to the entry in the terminating block. If the symbol is not found then it is entered into the surrounding block and the chain field is set in the terminating entry to point to the entry in the surrounding block and a parameter is set to mark it is a resolution link for the terminal RESOLVE primitive to use in replacing a pseudo-address in the object code with the actual address after it is defined. A resolution chain may be formed which may extend outward over a number of enclosing blocks. It is this command, or perhaps more specifically its interpretation, which permits proper resolution of label references in a block structure.

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

`.CAT(...)` (CAT):

This is a concatenation command causing the output code specified by the enclosed output operators to be suffixed to the top entry of the variable length code stack. This is effected by marking the top of the control stack with the current output pointer, restoring the top of the code stack to the output area, executing the specified output operators and then saving all the code back to the code mark.

An example of its use is in the implementation of the DO CASE construction in the PLEX translator.

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

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

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

Example:      **.ENTERL(00,01)**

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

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

Example:      **.ENTERA(07,DIMCNT)**

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

An explicit search of the symbol table is directed with the entry in SYMBOL being used as a search argument. The search may be a block search or a full table search depending

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

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

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

#### .STKSYM (STKSYM):

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

#### .SCAN(...) (SCAN):

The purpose of this semantic command is to cause a scan of the input string for the enclosed character string, this

being the only semantic command affecting the input pointer. It may be used, for example, in scanning for the terminating "\*" delimiter of a comment or scanning for the end of a statement in error recovery.

Example:        .SCAN("\*/")

.RETURN (R) :

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

## 5. Output operators

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

\*1 (LB1,EVAL) :

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

\*2 (LB2,EVAL) :

This is identical in function to \*1 except that the second label in the control stack is extracted.



**\*\*(...)** or **\*\* (EVAL):**

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

Example:       **\*\* (04,01)**

**\* (OUTSYM):**

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

**# (RESTORE):**

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

**Identifier, octal number, string (OUT):**

Using any one of these as output operators causes a literal value, or symbol table value in the case of an identifier, to be generated as the operand of an OUT operation code. Execution of such code then causes the literal operand

to be appended to the output code. It is an identifier as an output operator in conjunction with the .OEQU definition command which allows symbolic reference to operation codes but literally to have the octal equivalent generated.

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

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

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

## 6. Error processing

### .CANCEL (CANCEL) :

If a latched call was made to the procedure containing this command then its effect is to turn off the backup latch, thus preventing backup in the case of a subsequent error. If

a latched call was not made then its effect is that of a null operation.

#### **.ERRLATCH (MOVI):**

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

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

ALTERN of METAX9 clearly represents the conditions under which error handling code will be generated.

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

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

Pending fatal messages in the manner discussed provide a method for avoiding multiple error messages.

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

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

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

META PI (43,44) has an .ERR command but the error recovery scheme in an interactive environment for line oriented languages such as FORTRAN is somewhat less demanding than for a block structured language such as PLEX in a batch processing environment where it is considered desirable to be able

to continue compilation but not to lose knowledge of certain program structure already gained. For example, if one detects a fatal error in the middle of a DO group or block and the error recovery scheme exits to a procedure that takes certain standard action and then continues processing by calling for recognition of certain program segments, say statements, but does not make use of the fact that the head of a DO group or block has been processed, then an END statement may cause interesting problems.

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

#### D. The METAX9 Pseudo-Machine

##### 1. Primitive operations

There are 47 operation codes or primitives available on the METAX9 pseudo-machine. Only selected primitives are reviewed here due to the pertinent discussions with the cor-

responding source language elements and also due to the extensive comments in the assembler language listing of the corresponding interpreter in Appendix B. In some cases the symbolic representation of a primitive differs between the interpreter and the corresponding translator, the name in the interpreter being given in parentheses in that case. The index at the end of the interpreter listing may be used for consulting the interpreter.

**B ABC (BRANCH):**

Jump unconditionally to ABC.

**BT ABC (BRANCHT):**

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

**BF ABC (BRANCHF):**

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

**R (RETURN):**

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

**POP:**

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

SWAP:

Exchange the top two elements on the control stack.

SETP:

Set the true-false code false.

MOVE ABC,DEF:

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

MOVI ABC,"\$ACTIVE " (MOVLIT):

Move the literal operand to ABC in a manner similar to MOVE.

A FOUR,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 PLEX 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 FI. With the scheme used here a label may be created on one procedure level and used or defined on another

lower level. Perhaps this provides a useful degree of source language control over the creation, use and value definitions of labels while retaining source language expressive power.

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

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

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

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

### 3. Symbol table structure

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

communications region.

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

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

**DTYPE (0-0):**

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

**LEVEL (1-1):**

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

**ADDR (2-4):**

An eighteen bit field is used to represent addresses relative to a dynamic storage block except in the case of a

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

#### LENGTH (5-6):

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

#### DIMCNT (7-7):

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

#### CHAIN (8-10):

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

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

**ETYPE (11-11):**

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

**NAME (12-19):**

This is an eight character field containing the identifier or symbol table search argument used in referencing the

table. If the identifier is shorter than eight characters then it is padded on the right with blanks.

#### 4. Addressing structure

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

#### 5. Block list

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

Each entry in the block list comprises a single character surrounding block number and an address of the first entry in the symbol table for the particular block. The block number represents the relative position within the block list of the entry for the immediately surrounding block, this being used for full table searches and for proper restoration to the surrounding block upon execution of the



BLKEXT primitive. The block list represents a tree structure that is searched from bottom-to-top or leaf-to-root during a full table search. The scope of identifiers is thus properly preserved.

## 6. General comments

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

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

## E. METAX8 and METAX9 Pseudo-Machine Differences

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

ror handling and symbol table processing.

The symbol table for METAX8 is a straight forward 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 .TYPEN 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, MTXLDR is called to load the requested METAX program and then the specified interpreter segment is called.

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

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

An internal memory clearing routine is executed at sev-

eral places in the main loop to clear certain segments of main storage.

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

#### B. Control Record Analyzer

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

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

Specifically a control record may specify the METAX program and corresponding interpreter to be executed as well as the main storage area to be utilized for pushdown stacks, symbol table space (dynamic storage space in the case of PLEX object programs), object program execution and code generation. Furthermore, parameters for postlisting, METAX library updating and a "go" option may be set in the case a translation is to be performed.



In all cases default parameters will be set if none is specified.

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

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

### C. Communications Region

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

INSTRCT (205-209);

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

GENFLD (213-215)\*:

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

LODFLD (217-219)\*:

This field specifies the beginning location for loading

object programs for execution and is used by MTXLDR and the control program for loading purposes.

STCKF1 (221-223)\* and STCKF2 (225-227)\*:

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

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 MTXCRA 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

(38) .

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

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

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

Character string assignments may include the use of SUBSTR either as a pseudo-variable in a left part or as a

built-in function in an expression. The concatenation operation is denoted by a pair of slashes (//) due to character set limitations of the host computer. Proper truncation on the right occurs with character string assignment statements as required.

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

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

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

Conditional statements (if statements) may include relational tests on character string expressions or arithmetic expressions. The ELSE option may be used but in any case the statement following the THEN may not be a conditional

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

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

The object of a GO TO statement may be a label constant or label variable. If it is a label constant then it must occur within the currently active procedure although jumps



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

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

Format lists may contain both data format items and control format items although somewhat limited in scope compared to PL/I, the intent being to provide only enough format capability to effectively demonstrate the rest of the language and its translator. No explicit decimal point specifications and no repetition factors are permitted. The E format item represents a generalized floating point format while the L

format item represents a logical format. Control format items include X, COL, PAGE and SKIP, the latter two causing execution time errors if used in the wrong context.

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

A procedure may be invoked with a CALL statement if a RETURNS attribute has not been established or otherwise as a function reference. LABEL functions, however, are not permitted because of restrictions on label assignments delineated below. A return from a procedure may be effected by a RETURN statement, by the flow of control reaching the

procedure END statement or by a jump to a formal parameter as specified by a GO TO as outlined above. However, a direct jump to a label outside the procedure is not permitted, the restrictions on label assignment having been established to prevent this from happening indirectly via a label variable. As a further explanation the denial of a direct jump is based on dynamic storage and pushdown stack considerations.

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

It should be noted that there are certain implementation restrictions on actual parameters which are not necessarily restrictions on the methods used but rather are limitations of the effort expended. For example, a determination has to be made at compile time whether a particular actual parameter is a simple reference requiring only an address to be passed

or an expression requiring evaluation with the result being put in dynamic or temporary storage and the address of that being passed. To further complicate the matter a function procedure reference presents a potential ambiguity between classification as a procedure parameter or as an expression requiring evaluation. Further discussion about the exact limitations and methods for their removal are discussed below. Suffice it to say, as has been previously mentioned, that the implementation has not been carried to its logical conclusion in all cases but rather to the point of demonstrating certain capabilities.

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

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

## B. The PLEX Pseudo-Machine

The discussion which follows is divided into two basic parts with run-time storage administration and related primitives being considered first and then followed by a descrip-

tion of the pseudo-machine interpreter.

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

cedure 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 PLXCPL, the PLEX translator, may enhance the meaning of what follows.

Procedure PROGRAM effects certain initialization and permits leading comments to be processed prior to calling BHDBDY (block head and block body). The .DO(RESOLVE) command at the end of the procedure is normally the last command executed. BLKBDY (block body), which is called by BHDBDY, specifies that an arbitrary number of statements, possibly none, satisfies the syntactical requirements for that procedure.

The 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 an END statement and then backing up so that it may be processed as the end of a procedure, DO group or block. Although it isn't necessary to back over the END delimiter only to allow another procedure to test for it, it does perhaps make the procedures representing DO group, block or procedure syntax somewhat more readable.

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

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

BHDBDY contains a semantic check of fundamental importance in compiling dynamic storage administration code. Variable DYNAMB represents the discernable dynamic storage requirements, excluding arrays, for the block just processed

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

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

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

Procedure PARM and the two procedures called by it partially represent certain implementation restrictions to which

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

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

As 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 straight forward and on which only selective comments are given.

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

Procedure LTERM (label term) also requires special con-

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

Certainly the translator contains errors of commission or omission as extensive testing on it has not been undertaken. The speed of the translator is relatively good, processing input at essentially card reader speed, 400 cards per minute, except perhaps in cases of multiple statements per card. It is true, however, that certain improvements in speed could be made primarily because an initial identifier in a statement may under go considerable rescanning and use as a symbol table search argument in classfying 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 postsynactic command structure which permits a significant amount of processing to be expressed in a single pass translator.

Certainly, then, a fundamental aspect of the METAX experimental compiler-compiler system is the METAX9 compiler writing language with which the PLEX translator was written. A major factor to be considered, however, is that in designing and developing a translator writing system of reasonable generality and efficacy it is rather difficult to conceive and implement a single compiler writing language which anticipates all of the features which may be required or considered useful for implementing a translator for a particular language. An alternative to be considered is to have available a basic compiler writing language with which one can implement a new compiler writing language satisfying the

requirements at hand. That is to say, it is suggested that a fixed language for compiler writing need not be a necessary goal of a translator writing system. It has certainly been the author's experience that this alternative approach is a viable one.

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

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

As a result of the experience with matters covered in this dissertation there are a number of additions, extensions and/or revisions which come to mind as being worthy of

further investigation.

In view of the similarity of PLEX to PL/I further investigation into the implementation of additional PL/I-like features within the basic framework presented suggests itself.

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

Contemplating some additional features which may be of use in writing translators, particularly if code for most conventional machines is to be generated, it firstly appears that the ability to declare directly the specifications for certain data structures such as last-in-first-out and first-in-first-out queues of both the fixed and variable length variety as well as symbol table structures may be of considerable utility. Of course a metalanguage command structure

would also be needed for data transmission from one structure to another. A limited form of this suggestion may be found in Feldman's semantic language (15).

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

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

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

## VII. BIBLIOGRAPHY

1. Book, E., Schorre, V. and Sherman, S. J. The CWIC/360 system, a compiler for writing and implementing compilers. ACM SIGPLAN Notices, 1970: 11-29. June, 1970.
2. Branstadt, Dennis K. A computer aided instructional system for teaching formal languages. Unpublished Ph.D. dissertation. Ames, Iowa, Library, Iowa State University of Science and Technology. 1970.
3. Brooker, R. A. and Morris, D. An assembly program for a phrase structure language. Computer Journal 3, No. 3: 168-174. 1960.
4. Brooker, R. A. and Morris, D. A general translator program for phrase structure languages. Journal of the ACM 9, No. 1: 1-10. 1962.
5. Brooker R. A., MacCallum, I. R., Morris, D. and Rohl, J. S. The compiler compiler. Annual Review in Automatic Programming 3: 229-275. 1963.
6. Burroughs B6500/7500 information processing systems characteristics manual. Detroit, Michigan, Burroughs Corporation. 1968.
7. Burroughs B6500 information processing processing systems reference manual. Detroit, Michigan, Burroughs Corporation. 1969.
8. Carr, J. W. and Weiland, J. A nonrecursive method of syntax specification. Communications of the ACM 9, No. 4: 267-269. 1966.
9. Cheatham, T. E., Jr. and Sattley, K. Syntax directed compiling. AFIPS Conference Proceedings 28: 31-57. 1964.
10. Chomsky, N. On certain formal properties of grammars. Information and Control 2: 137-167. 1959.
11. Chomsky, N. Formal properties of grammars. In Handbook of Mathematical Psychology. Vol. II. Chapter 12. New York, N.Y., John Wiley and Sons, Inc. 1963.



12. DeRemer, F. Generating parsers for BNF grammars. AFIPS Conference Proceedings 34: 793-799.
13. Dijkstra, E. W. Recursive programming. In Rosen, S., ed. Programming Systems and Languages. Pp. 221-227. New York, N.Y., McGraw-Hill Book Company. 1966.
14. Evans, A. An ALGOL 60 compiler. Annual Review in Automatic Programming 4: 87-124. 1964.
15. Feldman, J. A formal semantics for computer languages and its application in a compiler-compiler. Communications of the ACM 9, No. 3-9. 1966.
16. Feldman, J. and Gries, D. Translator writing systems. Communications of the ACM 11, No. 2: 77-113. 1968.
17. Floyd, R. W. An algorithm for coding efficient arithmetic operations. Communications of the ACM 4, No. 1: 42-51. 1961.
18. Freiburghouse, R. A. The MULTICS PL/I compiler. AFIPS Conference Proceedings 35: 187-199. 1969.
19. Garwick, J. V. GARGOYLE, a language for compiler writing. Communications of the ACM 7, No. 1: 16-20. 1964.
20. Gries, D. Unpublished notes of the Kansas State University translator writing systems conference. Ithaca, N.Y., Computer Science Dept., Cornell Univ. 1971.
21. Gries, D., Paul, M. and Wiehle, H. R. Some techniques in the ALCOR ILLINOIS 7090. Communications of the ACM 8, No. 8: 496-500. 1965.
22. Hedrick, G. E. User error analysis and automatic correction for compiling. Unpublished Ph.D. dissertation. Ames, Iowa, Library, Iowa State University of Science and Technology. 1970.
23. Higman, B. Towards an ALGOL translator. Annual Review in Automatic Programming 3: 121-162. 1963.
24. Honeywell extended mod 1 supervisor. File number 123.5005.141H.1-B32. Wellesley Hills, Mass., Honeywell, Inc. 1970.

25. Honeywell operating system 200 and mod 1 extended data management, file organization. File number 123.0005.141H.0-B34. Wellesly Hills, Mass., Honeywell, Inc. 1969.
26. Honeywell operating system 200 and mod 1 extended data management, file processing. File number 123.0005.141H.0-B35. Wellesly Hills, Mass., Honeywell, Inc. 1969.
27. Honeywell operating system 200 and mod 1 extended Easy-coder assembler. File number 123.1105.1410-B41. Wellesley, Hills, Mass., Honeywell, Inc. 1969.
28. Honeywell series 200 programmers' reference manual. File number 113.005.0000.4-139. Wellesley Hills, Mass., Honeywell Information Systems, Inc. 1971.
29. IBM system/360 PL/I reference manual. Form C28-8202-2. Poughkeepsie, N.Y., IBM Corporation. 1968.
30. Irons, E. T. A syntax directed compiler for ALGOL 60. Communications of the ACM 4, No. 1: 51-55. 1961.
31. Irons, E. T. and Feurzig, W. Comments on the implementation of recursive procedures and blocks in ALGOL 60. Communications of the ACM 4, No. 1: 65-69. 1961.
32. Johnson, W. L., Porter, J. H., Ackley, S. I. and Ross, D. T. Automatic generation of efficient lexical processors using finite state techniques. Communications of the ACM 11, No. 12: 805-813. 1968.
33. Kemeny, John G. and Kurtz, T. E. BASIC Programming. New York, N.Y., John Wiley and Sons, Inc. 1968.
34. Knuth, D. On the translation of languages from left to right. Information and Control 8: 607-639. 1965.
35. Knuth, D. and McNealey, J. L. A formal definition of SOL. IEEE Transactions EC13: 409-414. 1964.
36. Lucas, P. and Walk, K. On the formal description of PL/I. Annual Review in Automatic Programming 6: 105-182. 1969.
37. McClure, R. M. TMG: a syntax directed compiler. ACM National Conference Proceedings 20: 262-274. 1965.

38. McKeeman, W. M., Hornung, J. J. and Wortman, D. B. A Compiler Generator. Englewood Cliffs, N.J., Prentice-Hall, Inc. 1970.
39. Metcalfe, H. H. A parameterized compiler. Annual Review in Automatic Programming 4: 125-165. 1964.
40. Naur, P. The design of the Gier ALGOL compiler. Annual Review in Automatic Programming 4: 49-85. 1964.
41. Naur, P., ed. Report on the algorithmic language ALGOL 60. Communications of the ACM 3, No. 5: 299-314. 1960.
42. Naur, P., ed. Revised report on the algorithmic language ALGOL 60. Communications of the ACM 6, No. 1: 1-17. 1963.
43. O'Neil, J. T., Jr. META-PI: an on-line interactive compiler-compiler. Fall Joint Computer Conference Proceedings, 1968: 210-218. 1968.
44. O'Neil, J. T., Jr. An interactive language design system. Unpublished Ph.D. dissertation. Philadelphia, 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 Proceedings 21: 465-468. 1966.
46. Presser, L. The structure, specification and evaluation of translators and translator writing systems. Unpublished Ph.D. dissertation. Los Angeles, California, Library, University of California at Los Angeles. 1968.
47. Randell, B. and Russell, L. J. ALGOL 60 Implementation. New York, N. Y., Academic Press, Inc. 1964.
48. Reynolds, J. C. An introduction to the COGENT programming system. ACM National Conference Proceedings 20: 422-436. 1965.
49. Ritland, Steven R. Implementation of a general purpose simulation language using a syntax directed translator. Unpublished M.S. thesis. Ames, Iowa, Library, Iowa State University of Science and Technology. 1968.

50. Rosen, S. A compiler writing system developed by Brooker and Morris. Communications of the ACM 7, No. 7: 403-414. 1964.
51. Schaefer, M. DBL: a language for converting data bases. Datamation 16, No. 6: 123-130. 1970.
52. Schneider, F. W. and Johnson, G. D. META-3: a syntax-directed compiler writing compiler to generate efficient code. ACM National Conference Proceedings 19: D1.5.1-D1.5.8. 1964.
53. Schorre, V. Syntax directed SMALGOL. ACM National Conference Abstracts. Communications of the ACM 6, No. 7: 365. 1963.
54. Schorre, V. META-II, a syntax oriented compiler writing language. ACM National Conference Proceedings 19: D1.3.1-D1.3.11. 1964.
55. Theys, M. E. An over view of META compiler systems. Unpublished M.S. thesis. Ames, Iowa, Library, Iowa State University of Science and Technology. 1970.
56. Whitney, Gordon. An extended BNF for specifying the syntax of declarations. Spring Joint Computer Conference Proceedings, 1969: 801-812. 1969.
57. Wilkes, M. V. An experiment with a self-compiling compiler for a simple list-processing language. Annual Review in Automatic Programming 4: 1-48. 1964.

## VIII. ACKNOWLEDGEMENTS

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

A special note of appreciation is given to the author's wife, Sharon, for her unswerving support and patience during the last four years.

IX. APPENDIX A

```
// CONTROL RECORD.  
    FUNCTION METAX8.  
    INTERPRETER=MTXINT03.  
    STORE=YES.  
    GO=NO.  
//
```

•PROG MTx009;

```

/*****
/*
/*  IDENTIFICATION:
/*  -----
/*
/*  PROGRAM-ID:  METAX9.
/*  AUTHOR:  J. R. VAN DOREN.
/*  SOURCE LANGUAGE:  METAX8.
/*  OBJECT LANGUAGE:  METAX8 PSEUDO-MACHINE CODE.
/*  OBJECT INTERPRETER:  MTXINT03.
/*
/*
/*  PURPOSE:
/*  -----
/*
/*  METAX9 IS A REVISION OF THE METAX8 METALANGUAGE AND ASSOCIATED
/*  COMPILER-COMPILER AND IS INTENDED FOR USE IN IMPLEMENTING THE PLEX
/*  COMPILER.
/*
/*
/*:*****/

```

```

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

```

```

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

```

```

BEM      .OEQU 16 .TYPE0;  MOVSYM  .OEQU 27 .TYPE0;  DELETE  .OEQU 32 .TYPE0;
TYPTST   .OEQU 34 .TYPE0;  ENTTP   .OEQU 35 .TYPE0;  ENTER   .OEQU 72 .TYPE0;
DECNUM   .OEQU 75 .TYPE0;

```



```
/* OPERATION CODES COMMON TO METAX8 AND METAX9 OBJECT PROGRAMS.
```

```
*/
```

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

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

```
*/
```

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

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

```
*/
```

```
/* OBSERVE THE NONRELOCATABLE TYPE CODE USED.
```

```
*/
```

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

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

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

```
*/
```

```
PROGHD :=".PROG" .ID  ";"
        .DO(MOVSYM      ,PRGNME)
        .OUT(B,**) $ ( DECLPT / COMMENT ) ;
```

```

/*  DECLARATIVE STATEMENTS SPECIFY THE INITIAL VALUE AND SIZE OF VARIABLES  */
/*  BY THE SPECIFIED STRING, OCTAL OR DECIMAL INTEGER VALUE                  */
/*                                                                           */
DECLPT := ".DECLARE" $(.ID .DEFLAB(*)
                      (.STRING /
                      .OUT(*) ("," / ";" .RETURN) ) ;
                      .ONUM)

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

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

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

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

/*  THE ALTERN PROCEDURE CONTAINS THE SYNTAX AND SEMANTICS FOR THE PART   */
/*  OF A METAX9 EXPRESSION CONTAINED WITHIN A SYNTACTICAL ALTERNATIVE.    */
/*                                                                           */

```

```

/* THE METALANGUAGE CONSTRUCTS CONTAINED THEREIN ARE CLASSIFIED AS      */
/* ELEMENTARY SYNTAX OR SEMANTIC ACTION.                                  */
/*                                                                           */

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

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

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

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

/ "(" MTXEXP ")" .ERR("w: EXPECTED "),SET)

/* THE NEXT ALTERNATIVE PERMITS ITERATIVE EXPRESSIONS.                  */

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

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

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

SEMACT := OUTPUT / COMMENT / SEM1 ;

```

```

/* 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(LB1,EVAL,04,01) / "*2" .OUT(LB2,EVAL,04,01)
      / "***" .OUT(EVAL) ( "(" .ONUM .OUT(*)
                        ( "," .ONUM .OUT(*) / .EMPTY .OUT(00) ) )"
                        / .EMPTY .OUT(05,00) )
      / "*" .OUT(OUTSYM) / "*" .OUT(RESTORE)
      / .ID .OUT(OUT,**) / .ONUM .OUT(OUT,*)
      / .STRING .OUT(OUT,*) ;

```

```

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

```

```

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

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

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

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

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

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

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

.END
***COMPILED PROGRAM SIZE =      2,653;  METAX INSTRUCTION COUNT =      33,888***
***SYMTAB SEARCH COUNT =           895;  SYMTAB COMPARE COUNT =      143,068***
***SYMBOL TABLE ENTRY COUNT =       346***

```

X. APPENDIX B

```
// CONTROL RECORD.  
  FUNCTION MTX009.  
  INTERPRETER=MTXINT03.  
  GO=NO.  
  START SYMTAB AT 5096, END SYMTAB AT 15096.  
  EXECUTE AT 15097.  
  POSTLIST=YES.  
  STORE=YES.  
//
```

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

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

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

/*  VARIABLES USED FOR SYNTACTIC ANALYSIS  */

•DECLARE DOSYM "DO      ", IFSYM "IF      ", CALLCON "CALL      ",
      RPAREN ")", SYMSAV "      ", STMLAB "      ";
```



```

/* VARIABLES USED FOR ATTRIBUTE PROCESSING */

.DECLARE TYPE " ", LENGTH 0000, DIMCNT 00, FUNCT 0000;

/* VARIABLES USED FOR PROCEDURE PROCESSING (DECLARATIONS AND CALLS) */

.DECLARE ARGCNT 00, ADECNT 00, PARMCNT 00, OCTEN 10, OCTL60 60,
      OCTL20 20;

/* VARIABLES USED FOR LABEL PROCESSING (PRIMARILY FOR PROCEDURE EXITS) */

.DECLARE PEXITF 77, PEXITT 00;

/* VARIABLES USED FOR TESTING I/O ITERATION LOOPS */

.DECLARE IOITER 77, IOSW 00;

/* OTHER VARIABLES AND VALUES FOR GENERAL USE */

.DECLARE ONE 01, FOUR "0004", EIGHT "0008", BLNK8 "      ", ZERO 00;

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

/* OPERATION CODES FOR METAX9 OBJECT PROGRAMS. */

MOVE      .OEQU 12 .TYPE0;  A      .OEQU 13 .TYPE0;  M      .OEQU 14 .TYPE0;
BEF       .OEQU 16 .TYPE0;  EXIT   .OEQU 17 .TYPE0;  RESOLVE .OEQU 20 .TYPE0;
R         .OEQU 21 .TYPE0;  BT     .OEQU 22 .TYPE0;  BF      .OEQU 23 .TYPE0;
BM        .OEQU 24 .TYPE0;  SET    .OEQU 25 .TYPE0;  SETF     .OEQU 15 .TYPE0;
SCAN      .OEQU 26 .TYPE0;  MOVI   .OEQU 30 .TYPE0;  SEARCH  .OEQU 33 .TYPE0;
COMP      .OEQU 32 .TYPE0;  TSTTBL .OEQU 34 .TYPE0;  TSTTBA  .OEQU 35 .TYPE0;
LATCH     .OEQU 36 .TYPE0;  CANCEL .OEQU 37 .TYPE0;  CLM      .OEQU 42 .TYPE0;

```

```

R      .OEQU 43 .TYPE0;  PUSHLB .OEQU 46 .TYPE0;  POP      .OEQU 47 .TYPE0;
LBI    .OEQU 50 .TYPE0;  LB2     .OEQU 51 .TYPE0;  ENTA     .OEQU 55 .TYPE0;
ENTL   .OEQU 56 .TYPE0;  OUT     .OEQU 62 .TYPE0;  OUTSYM  .OEQU 63 .TYPE0;
TEST   .OEQU 64 .TYPE0;  ID      .OEQU 65 .TYPE0;  ONUM    .OEQU 66 .TYPE0;
STRIST .OEQU 67 .TYPE0;  EVAL   .OEQU 70 .TYPE0;  ENTLOC  .OEQU 71 .TYPE0;
INUM   .OEQU 73 .TYPE0;  FNUM   .OEQU 74 .TYPE0;  BLKENT  .OEQU 75 .TYPE0;
BLKEXT .OEQU 76 .TYPE0;  STKSYM .OEQU 52 .TYPE0;  CHKSYM  .OEQU 53 .TYPE0;
SWAP   .OEQU 54 .TYPE0;  MARK   .OEQU 40 .TYPE0;  SAVE    .OEQU 41 .TYPE0;
RESTORE .OEQU 45 .TYPE0;  ERASE  .OEQU 31 .TYPE0;

```

```
/* OPERATION CODES FOR PLEX OBJECT PROGRAMS
```

```
*/
```

```

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

```

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

```
*/
```

```
/* OBSERVE THE NONRELOCATABLE TYPE CODE USED.
```

```
*/
```

```

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

```

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

```
*/
```

```
PLXCPL :=.EMPTY .ERRLATCH PROGRAM .ERR("F: COMPILER ABORT - BAD PROGRAM",EXIT);
```

```
PROGRAM:=.ERRLATCH .NEWLAB .ID ":" .STKSYM /* SAVE NAME FOR END CHECK
```

```
*/
```

```

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

BLKEDY := $ ( .SET (STMLAB=BLNK8) STMENT ) ;

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

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

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

LABEL := .ID ":" .SET (STMLAB=SYMBOL) /* SAVE LABEL */
.SEARCH ("1") .DO (ENTLOC) /* ENTER LABEL AND VALUE */
.ENTRERL (00,06) ; /* LABEL CONSTANT TYPE */

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

ENDSTMT := "END" .DO (SETF,BEF) / .EMPTY ;

```

```

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

BHDEBY :=BLKHD BLKBDY /* DECLARATIONS AND BLOCK BODY */
      ( .TEST(DYNAMB > DYNAMP) /* DTERMINE LEVEL OF STORAGE REQUIRED */
      .SET(DYNAMP=DYNAMB) /* SET NEW LEVEL */
      / .EMPTY ) ; /* OLD LEVEL O.K. */

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

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

```

```

•DO(M,LEVNO,DYNAMP)          /* COUNTERS */
•SET(DYNAMB=DYNAMP)          /* FOR THIS PROCEDURE */

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

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

```

```
TARGCNT:=•TEST(ARGCNT=ZERO) •DO(SETF,R) / •EMPTY;
```

```

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

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

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

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

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

CALLPRT:=.SAV(LDA,**,JUMPA) .EMPTY /* SAVE CODE TO JUMP TO PROC */
        .SET(STMLAB=SYMBOL) /* SAVE PROC NAME FOR CHECKING */
        .NEWLAB .OUT(LDA,06,*1) /* CODE TO LOAD RETURN ADDRESS */
        .OUT(LD,0100000065) /* CODE TO LOAD CURRENT DISPLAY ADDR */
        .SET(SYMBOL=STMLAB) .SEARCH /* REPOSITION SYMBOL TABLE POINTER */
        ( ( .TSTTBL(00,10) /* IF GLOBALLY KNOW PROC THEN */
        / .TSTTBL(00,11)
        / .TSTTBL(00,12)
        / .TSTTBL(00,13)
        / .TSTTBL(00,14) )
        .OUT(LD,0100000065) /* CURRENT DISPLAY ADDR IS GLOBAL */
        / ( .TSTTBL(00,20) /* IF INDIRECTLY KNOWN PROC THEN */
        / .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 .OUT(LDA,01,*1)      /* CODE TO LOAD ADDR FOR STORING
                                      COMPUTED GLOBAL DISPLAY ADDR      */
        .OUT(*)
        .OUT(LD,71,"0004",ADD)      /* COMPUTE ADDR OF GLOBAL DISPLAY ADDR*/
        .OUT(STO)                    /* CODE TO STORE ADDR OF GLOBAL
                                      DISPLAY ADDR IN NEXT INSTR      */
        .OUT(LD,01) .DEFLAB(*1)      /* CODE TO LOAD GLOBAL DISPLAY ADDR */
        .ENTERL(00,01)
        .OUT("0000")
        .DO(POP)                      /* POP *1 LABEL                      */
        .DO(SET)                      /* SET TF CODE FOR COMPILER TESTING */
    )      .ERR("F:  INVALID PROC CALL")
CLLPRM      /* COMPILE CODE TO LOAD PARM ADDRESSES*/
        .SET(SYMBOL=STACKTP)          /* CURRENT STACK TOP SYMBOL      */
        .OUT(LD,**)                    /* CODE TO LOAD STACK TOP VALUE  */
        .OUT(*)                        /* RESTORE PROC JUMP CODE        */
        .DEFLAB(*1) ;                  /* DEFINE RETURN ADDRESS        */

CLLPRM :=.OUT(FLAG)
        .STACK(PARMCNT) .SET(PARMCNT=ZERO)
        .STACK(STMLAB)                /* SAVE PROC NAME                */
        ( PARMPT / .EMPTY )
        .DO(POP)                      /* PROC NAME OFF STACK FOR SEARCHING */
        .SEARCH("0")
        ( ( .TSTTBL(00,11) / .TSTTBL(00,12) / .TSTTBL(00,13) / .TSTTEL(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")
        ( .TSTBL(00,10) .CANCEL GPROC      /* CHECK FOR GLOBAL PROC PARM   */
        / .TSTBL(00,20) .CANCEL IPROC ) ; /* CHECK FOR INDIRECT PROC PARM */

PARMEXP:= .NEWLAB .DO(LB1) .SEARCH("1") .ENTERA(01,DYNAMB)
        .ENTERL(00,07) .DO(A,EIGHT,DYNAMB) .OUT(LDA,**,LDA,**)
        ( EXP / SSE / LTERM / BPRMRY )
        .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 DISPLY*/
                                          /* 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)
      .NEWLAB
      .OUT(LDA,01,*1)            /* CODE TO LOAD ADDR FOR STORING
                                  COMPUTED ADDR OF PASSED GLOBAL
                                  DISPLAY ADDRESS */
      .OUT(*)                    /* RESTORE ADDR OF PROC ADDR CODE */
      .OUT(LD,71,"0004",ADD)     /* COMPUTE ADDR OF GLOBAL DISPLAY ADDR */
      .OUT(STO)                  /* CODE TO STORE IN NEXT INSTR */
      .OUT(LD,01) .DEFLAB(*1)    /* CODE TO LOAD GLOBAL DISPLAY ADDR */
      .ENTERL(00,01)            /* MARK INTEGER */
      .OUT("0000")
      .OUT(STO) ;                /* 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           /* STACK IDENTIFIER FOR LATER USE */
      (ARRYPT                   /* CHECK FOR ARRAY ATTRIBUTES */
      .STACK(DIMCNT)
      IDLIST                    /* RECURSIVELY SEARCH FOR MORE */
      .UNSTACK(SYMSAV,DIMCNT)  /* RESTORE DIMCNT, SAVE ID FOR ENTRY */
      ARRYSEM                   /* PERFORM ARRAY SEMANTICS */
      /IDLIST                  /* RECURSIVE SEARCH */
      .DO(POP).SEARCH("1")     /* POP ID AND POSITION SYMTAB POINTER */
      IDSEM                    /* PERFORM IDENTIFIER SEMANTICS */
      .DO(A,LENGTH,DYNAMB) ) ; /* BUMP DYNAMIC STORAGE COUNTER */

```

```

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

```

```

ATTRIBT := ("CHARACTER(" / "CHAR(").INUM ")"

```

```

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

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

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

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

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

IDELMNT:=.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 */
                                        /* FOLLOWED BY SDV */
        .SET(SYMBOL=BLNK8)             /* CLEAR SYMBOL TO BLANKS */
        .SET(SYMBOL=SYMSAV) .SEARCH("1") /* POSITION SYMTAB POINTER */
        .ENTERA(07,DIMCNT)             /* ENTER DIMENSION COUNT */
        IDSEM                          /* ENTER TYPE, LENGTH AND ADDRESS */
        .OUT(**)                      /* DYNAMIC DOPE VECTOR ADDRESS */
        .OUT(**(01,07))               /* DIMCNT ALSO TO SDV */
        .OUT(**(02,05))               /* LENGTH TO SDV */
        .OUT(0)                      /* FINISH SDV WITH BOUND PAIR CODE */
        .DO(A,DUPFIX,DYNAMB)           /* ADD FIXED DOPE VECTOR SIZE */
        .SET(SYMBOL=FOUR)
        .DO(M,DIMCNT,SYMBOL)
        .DO(A,SYMBOL,DYNAMB)           /* ADJUST FOR MULTIPLIER STORAGE */
        .DO(SET) ;

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

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

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

CASE := "CASE" .NEWLAB .ERRLATCH      /* TRANSFER VECTOR LABEL */
        .NEWLAB                      /* LABEL FOR TV ADDRESS CONSTANT */
        EXP .ERR("F: BAD CASE EXPRESSION",SCAN,";" ,CLM,TAIL,R) ";"
        .OUT(LD,7100000005,MULT)      /* CODE TO MULT BY TV ELEMENT SIZE */
        .DO(LB1) .SEARCH("1") .ENTERL(00,01) /* MARK 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) .DO(POP) /* FIRST TV ENTRY */
    $(.NEWLAB .DEFLAB(*1) STMENT /* LABEL AND COMPILE STATEMENT */
    .OUT(JUMP,*2)          /* CODE TO JUMP OUT OF CASE GROUP */
    .CAT(JUMP,*1) .DO(POP) /* CATENATE CODE TO TRANSFER VECTOR */
    .DO(POP)               /* POP EXTRANEIOUS 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 */
                           /* LABEL ON TOP */
    .OUT(*2)               /* GENERATE ADCON */
    .DEFLAB(*2)             /* DEFINE TV ADDRESS */
    .OUT(*)                 /* OUTPUT TV TO CODE STRING */
    ENDING .DEFLAB(*1) ;    /* DEFINE CASE GROUP EXIT ADDRESS */

```

```

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

```

```

WHILE := "WHILE" WHLPT ";" ;

```

```

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

```

```

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

```

```

INTERPT:==" .ERRLATCH EXP
      .ERR("F:  INVALID INITIAL INDEX")  .OUT(SST)
      "TO"                                /* INITIAL VALUE */
      .NEWLAB .DEFLAB(*1)                /* ITERATION LABEL AND POSSIBLE VALUE */
      .MARK                               /* POSSIBLE CODE REORDERING POINT */
      (.LATCH(LOGPRM)                    /* COMPILE SIMPLE PRIMARY IF ANY */
      / .NEWLAB .DEFLAB(*1)              /* LABEL FOR TEMPORARY IN DYNAMIC */
        .ENTERA(01,DYNAMB)              /* STORAGE AND VALUE */
        .ENTERL(00,07)                  /* DATA TYPE ANY */
        .DO(A,EIGHT,DYNAMB)             /* SPACE FOR INTEGER OR REAL */
        .OUT(LDA) .DO(LB1) .OUT(**)      /* CODE TO LOAD DYNAM STOR ADDRESS */
      EXP .OUT(STO)                      /* COMPILE LIMIT EXPRESSION */
      .MARK                               /* NEW REORDERING POINT */
      .DEFLAB(*2)                        /* REDEFINE ITERATION LABEL */
      .OUT(LD) .DO(LB1) .OUT(**)         /* CODE TO LOAD EXPRESSION VALUE */
      .DO(SWAP,POP)                      /* DISCARD LABEL AND LEAVE CODE */
    ) .ERR("F:  INVALID INDEX LIMIT")
      /* MARKER ON TOP */
      ("BY" .OUT(STCKC)                  /* 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(LOGPRM)                    /* COMPILE SIMPLE PRIMARY IF ANY */
      / .DO(POP) .SAV                    /* DISCARD CODE MARK, SAVE TEST CODE */
        .NEWLAB .DEFLAB(*1)              /* LABEL FOR TEMPORARY IN DYNAMIC */
        .ENTERA(01,DYNAMB)              /* STORAGE AND VALUE */
        .ENTERL(00,07)                  /* DATA TYPE ANY */
        .DO(A,EIGHT,DYNAMB)             /* SPACE FOR INTEGER OR REAL */
        .OUT(LDA) .DO(LB1) .OUT(**)      /* CODE TO LOAD DYNAM STOR ADDRESS */
      EXP .OUT(STO)
        .DEFLAB(*2)                      /* REDEFINE ITERATION LABEL */
        .OUT(*)                          /* RESTORE STACKED TEST CODE */
        .MARK                             /* LOOP END CODE FOLLOWING TO BE SAVED */
        .OUT(LD) .DO(LB1) .OUT(**)       /* CODE TO LOAD EXPRESSION VALUE */
        .DO(SWAP,POP)                   /* CODE MARK ON TOP, DISCARD LABEL */
    ) .ERR("F:  INVALID INCREMENT")

```

```

        .OUT(*)                      /* RESTORE INCREMENT UP CODE      */
/ .EMPTY .OUT(STCKC,04)              /* DEFAULT TEST                */
IOCHK .MARK
        .OUT(LD,71,"0001",ADD) )    /* AND INCREMENT CODE          */
.OUT(SST,JUMP,*1)                   /* FINAL LOOP END CODE         */
.SAV ;                               /* SAVE ALL LOOP END CODE      */

/* RESTORE ILIST CODE IF I/O ITERATION */

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

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

UNDTST := .SEARCH .TSTTBL(00,00)
        .DO(BM,"w: UNDECLARED IDENTIFIER") / .EMPTY ;

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

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

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

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

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

BSCNDRY := BPRMRY / ".NOT." BPRMRY .OUT(NOT) ;

```

```

BPRMRY :=BVALUE / .LATCH(BVARBLE) / .LATCH(BFNCT) / RELATN /
        "(" BOOLEXP ")" .ERR("W: EXPECTED ") ,SET) ;

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

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

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

BVARBLE:=.ID .SEARCH("0") ( .TSTTBL(00,03) / .TSTTBL(00,63) ) .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 :=PRMRY $ ( "*" 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,*) ;

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

EXP :=.LATCH(IFCLSE) .NEWLAB .OUT(JUMPF,*1) SAE
        "ELSE" .NEWLAB .OUT(JUMP,*1)

```

```

        .DEFLAB(*2) EXP.DEFLAB(*1)
        / SAE ;

SEXP :=.LATCH(IFCLSE) .NEWLAB .OUT(JUMPF,*1) SSE
      "ELSE" .NEWLAB .OUT(JUMP,*1)
      .DEFLAB(*2) SEXP .DEFLAB(*1)
      / SSE ;

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

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

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

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

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

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

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

LBLCON :=.TSTTBL(00,06) .OUT(LDA,**);

```



```

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

CONDST :=.LATCH(IFCLSE) .NEWLAB .OUT(JUMPF,*1)
      BSCSTM ( "ELSE" .NEWLAB .OUT(JUMP,*1)
      .DEFLAB(*2) STMENT .DEFLAB(*1)
      / .EMPTY .DEFLAB(*1) ) ;
IFCLSE :=.ID .TEST(SYMBOL=IFSY4) .CANCEL BOOLEXP "THEN" ;

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

UNCOND :=.LATCH(GOTOST) / INOUT / .LATCH(CLLSTMT) / .LATCH(RTNSTMT)
      / STPSTM / ASSGNST / IDENT ;
STPSTM := "STOP" .OUT(STOP) ;

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

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

ASSGNST:=AASSGN / BASSGN / SASSGN / LASSGN ;

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

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

SASSGN :=SLFTPT .SAV(STO)
      $ ( "," SLFTPT .MARK .OUT(SST,*) .SAV )
      "=" SEXP .OUT(*) ;

```

```

LASSGN :=.LATCH(LLFTPT) .SAV(STO)
$ ( "," LLFTPT .MARK .OUT(SST,#) .SAV )
"=" LTRM .TEST(PEXITT=ZERO) .ERR("F: INDIRECT LABEL ASSIGNMENT")
.OUT(#) ;

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

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

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

SLFTPT :=.LATCH(SVARIABLE) / LSUBSTR ;

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

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

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

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

ILIST :=IELMNT $ ( "," IELMNT ) ;

```

```

IELMNT :=IRPLIST
        /(.LATCH(SLFTPT) / .LATCH(ALFTPT) / .LATCH(BLFTPT) ) .OUT(GET)    ;

IRPLIST:="(" .MARK ILIST .SAV 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" .SAV(00) "(" SLFTPT ")"
        / .EMPTY .SAV(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 .SAV ERPLIST  ;

FMTLIST:="(" .OUT(77) FMTITEM $ ( "," FMTITEM ) ")" .OUT(77)  ;

FMTITEM:=CTRLFMT / DATAFMT  ;

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

DATAFMT:="A" .OUT(10) ( SPEC / .EMPTY .OUT( "0000" ) )

```

```

      / "E" .OUT(11) SPEC
      / "I" .OUT(12) SPEC / "L" .OUT(13) SPEC ;

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

.END
****COMPILED PROGRAM SIZE =      7,362;  METAX INSTRUCTION COUNT =  128,483****
****SYMTAB SEARCH COUNT =      2,308;  SYMTAB COMPARE COUNT =      857,907****
****SYMBOL TABLE ENTRY COUNT =      846****

```

	B	PLXCPL	
DYNAMP		"1008"	
DYNAME		"0000"	
STACKT		"\$STKTO	"
ONELEV		"1000"	
LEVNO		"1"	
ONELNG		"00000100"	
DOPFIX		"08"	
DOSYM		"DO	"
IFSYM		"IF	"
CALLCO		"CALL	"
RPAREN		")"	
SYMSAV		"	"
STMLAB		"	"
TYPE		" "	
LENGTH		"00"	
DIMCNT		"0"	
FUNCT		"00"	
ARGCNT		"0"	
ADECNT		"0"	
PARMCN		"0"	
OCTEN		"8"	
OCTL60		"<"	
OCTL20		"+"	
PEXITF		"c"	
PEXITT		"0"	
IOITER		"c"	
IOSW		"0"	
ONE		"1"	
FOUR		"0004"	
EIGHT		"0008"	
BLNK8		"	"
ZERO		"0"	
PLXCPL	SET		
	BF	\$002	
	MOVI	ELATCH	
		"1"	

CLM	PROGRA
BT	\$002
BM	"F: COMPILER ABORT
EXIT	
\$002 R	
PROGRA MOVI	ELATCH
	"1"
PUSHLB	
ID	
BF	\$005
TEST	":"
BEF	
STKSYM	
MOVE	SYMBOL
	PRGNME
MOVI	INTNME
	"PLXINT00"
MOVI	SYMBOL
	"\$ACTIVE "
SEARCH	
	"0"
ENTL	"0"
	IOCHK
MOVE	DYNAMP
	DYNAMB
TEST	"PROCEDURE"
BEF	
TEST	"MAIN"
BEF	
BLKENT	
TEST	":"
BEF	
OUT	DYNAM
LB1	
EVAL	
	"4"
	"1"

	MOVE	STACKT		CLM	STMENT
		SYMBOL		BF	\$015
	SEARCH		\$015	BT	BLKBDY
		"0"		SET	
	ENTL	"0"		BF	\$012
		"51000"	\$012	R	
\$006	CLM	COMMEN	STMENT	MOVI	FLATCH
	BT	\$006			"1"
	SET		\$018	CLM	COMMEN
	BEF			BT	\$018
	CLM	BHDBDY		SET	
	BEF			BF	\$017
	TEST	"END"	\$019	LATCH	LABEL
	BEF			BT	\$019
	OUT	STOP		SET	
	LBI			BEF	
	ENTLOC			CLM	ENDTST
	ENTL			BF	\$021
		"0"		SETH	
		"6"		R	
	ENTA	"1"		B	\$020
		DYNAMP	\$021	CLM	CONDST
	ID			BF	\$022
	BF	\$008		R	\$020
	CHKSYM		\$022	CLM	BSCSTM
	BT	\$009		BF	\$020
	BM	"w: POSSI	\$020	BT	\$017
	SET			BM	"F: BAD STATEMENT"
\$009	B	\$007		SET	
\$008	SET			SCAN	DOSYM
	BF	\$007	\$017	R	
\$007	BEF		BSCSTM	LATCH	DOGROU
	BLKEXT	"1"		BF	\$028
	RESOLV			B	\$027
\$005	R		\$028	LATCH	BLOCK
BLKBDY	MOVE	BLNK8		BF	\$029
		STMLAB		B	\$027

\$029	CLM	UNCOND
	BF	\$030
	B	\$027
\$030	SET	
	BF	\$027
\$027	BF	\$026
	TEST	" ; "
	BEF	
\$026	R	
COMMEN	TEST	"/*"
	BF	\$033
	SCAN	DOSYM
\$033	R	
LABEL	ID	
	BF	\$035
	TEST	": "
	BEF	
	MOVE	SYMBOL
		STMLAB
	SEARCH	"1"
	ENTLOC	
	ENTL	"0"
		"6"
\$035	R	
ENDTST	LATCH	ENDSTM
	BF	\$037
	SETF	
	B	\$036
\$037	SET	
	BF	\$036
\$036	R	
ENDSTM	TEST	"END"
	BF	\$040
	SETF	
	BEF	
	B	\$039
\$040	SET	

\$039	BF	\$039
	R	
BLOCK	TEST	"BEGIN"
	BF	\$043
	TEST	" ; "
	BEF	
	BLKENT	
	MOVI	ELATCH
		"1"
	MOVE	STMLAB
		SYMBOL
	STKSYM	
	MOVE	DYNAMB
		SYMBOL
	STKSYM	
	MOVE	STACKT
		SYMBOL
	STKSYM	
	OUT	STKTOP
	EVAL	
		"5"
		"0"
	A	ONELNG
		STACKT
	MOVE	STACKT
		SYMBOL
	SEARCH	"1"
	ENTA	"1"
		DYNAMB
	ENTL	"0"
		"5"
	EVAL	
		"5"
		"0"
	A	FOUR
		DYNAMB
	CLM	BHDBDY

	BEF		
	POP		
	MOVE	SYMBOL	
		STACKT	
	POP		
	MOVE	SYMBOL	
		DYNAMB	
	TEST	"END"	
	BEF		
	BLKEXT		
		"0"	
	ID		
	BF	\$045	
	CHKSYM		
	BT	\$046	
	BM	"W: POSSI	
	SET		
\$046	B	\$044	
\$045	SET		
	BF	\$044	
\$044	BEF		
\$043	R		
BHDBDY	CLM	BLKHD	
	BF	\$049	
	CLM	BLKBDY	
	BEF		
	COMP	DYNAMB	
		DYNAMP	
		"1"	
	BF	\$051	
	MOVE	DYNAMB	
		DYNAMP	
		\$050	
\$051	B		
	SET		
	BF	\$050	
\$050	BEF		
\$049	R		

BLKHD	CLM	DECLAR
	BT	PLKHD
	SET	
	BF	\$054
	PUSHLB	
	OUT	JUMP
	LB1	
	EVAL	
		"4"
		"1"
\$056	LATCH	PROCDE
	BT	\$056
	SET	
	BEF	
	LB1	
	ENTLOC	
	ENTL	
		"0"
		"6"
\$054	R	
PROCDE	ID	
	BF	\$058
	TEST	": "
	BEF	
	TEST	"PROCEDURE"
	BEF	
	CANCEL	
	PUSHLB	
	SEARCH	"1"
	ENTLOC	
	TSTTBL	"0"
		"8"
	BF	\$060
	BM	"W: DUP PROC DCL"
	SET	
	B	\$059
\$060	TSTTBL	"0"



		"0"
	BF	\$061
	ENTL	"0"
		"8"
	B	\$059
\$061	SET	
\$059	BEF	
	BLKENT	
	MOVI	ELATCH
		"1"
	MOVE	SYMBOL
		STMLAB
	STKSYM	
	MOVE	DYNAMP
		SYMBOL
	STKSYM	
	MOVE	DYNAMB
		SYMBOL
	STKSYM	
	MOVE	LEVNO
		SYMBOL
	STKSYM	
	MOVE	STACKT
		SYMBOL
	STKSYM	
	A	ONE
		LEVNO
	MOVE	ONELEV
		DYNAMP
	M	LEVNO
		DYNAMP
	MOVE	DYNAMP
		DYNAMB
	A	ONELNG
		STACKT
	MOVE	STACKT
		SYMBOL

	SEARCH	"1"
	ENTA	"1"
		DYNAMB
	ENTL	"0"
		"5"
	MOVE	FOUR
		SYMBOL
	M	LEVNO
		SYMBOL
	A	FOUR
		SYMBOL
	A	SYMBOL
		DYNAMB
	OUT	ENTPRO
	MOVE	LEVNO
		SYMBOL
	OUTSYM	
	OUT	DYNAM
	LB1	
	EVAL	
		"4"
		"1"
	MOVE	ZERO
		ARGCNT
	TEST	"("
	BF	\$064
	CLM	ARGID
\$065	BEF	
	TEST	","
	BF	\$067
	CLM	ARGID
	BEF	
\$067	BT	\$065
	SET	
	BEF	
	TEST	"),"
	BEF	

	B	\$063
\$064	SET	
	BF	\$063
\$063	BEF	
	TEST	";"
	BEF	
	MOVE	STMLAB
		SYMBOL
	SEARCH	
		"0"
	ENTA	"7"
		ARGCNT
	MOVE	ZERO
		ADECNT
	CLM	TARGCN
	BF	\$070
	TEST	"DECLARE"
	BF	\$072
	CLM	ARGDEC
	BEF	
\$073	TEST	","
	BF	\$075
	CLM	ARGDEC
	BEF	
\$075	BT	\$073
	SET	
	BEF	
	TEST	";"
	BEF	
	B	\$071
\$072	SET	
	BF	\$071
\$071	BEF	
	COMP	ARGCNT
		ADECNT
		"2"
	BT	\$077

	BM	"W: INCORRECT ARG
	SET	
\$077	B	\$069
\$070	SET	
	BF	\$069
\$069	BEF	
	CLM	RHDBDY
	BEF	
	TEST	"END"
	BEF	
	OUT	RETURN
	LBI	
	ENTLOC	
	ENTL	
		"0"
		"6"
	ENTA	"1"
		DYNAMP
	BLKEXT	"1"
	POP	
	MOVE	SYMBOL
		STACKT
	POP	
	MOVE	SYMBOL
		LEVNO
	POP	
	MOVE	SYMBOL
		DYNAMB
	POP	
	MOVE	SYMBOL
		DYNAMP
	ID	
	BT	\$079
	BM	"W: EXPECTED CLOS
	SET	
\$079	CHKSYM	
	BT	\$080

	BM	"W: POSS
	SET	
\$080	TEST	";"
	BEF	
\$058	R	
TARGCN	COMP	ARGCNT
		ZERO
		"2"
	BF	\$082
	SETF	
	R	
	B	\$081
\$082	SET	
	BF	\$081
\$081	R	
ARGID	ID	
	BF	\$085
	SEARCH	"1"
	ENTA	"1"
		DYNAMB
	ENTL	"0"
		"<"
	A	ONE
		ARGCNT
	A	FOUR
		DYNAMB
	SET	
\$085	R	
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
		DIMCNT
	TEST	"*"
	BEF	
	A	ONE
		DIMCNT
\$091	TEST	","
	BF	\$093
	TEST	"*"
	BEF	
	A	ONE
		DIMCNT
\$093	BT	\$091
	SET	
	BEF	
	TEST	"),"
	BEF	
	ENTA	"7"
		DIMCNT
		\$089
\$090	B	
	SET	
	BF	\$089
\$089	R	
ARGATR	CLM	ATTRIB
	BF	\$096
	ENTA	"5"
		LENGTH
	A	OCTL60

	ENTA	TYPE	CALLPR	MARK	
		"0"		OUT	LDA
		TYPE		EVAL	
	B	\$095			"5"
\$096	TEST	"ENTRY"			"0"
	BF	\$095		OUT	JUMPA
	TEST	"RETURNS("		SAVE	
	BF	\$099		SET	
	CLM	ATTRIB		BF	\$104
	BEF			MOVE	SYMBOL
	TEST	")"			STMLAB
	BEF			PUSHLB	
	A	OCTL20		OUT	LDA
		TYPE		OUT	"6"
	ENTA	"0"		LB1	
		TYPE		EVAL	
	B	\$098			"4"
\$099	SET				"1"
	BF	\$098		OUT	LD
	ENTL	"0"		OUT	IOCHK
		"+"		MOVE	STMLAB
\$098	BEF				SYMBOL
\$095	R			SEARCH	
CLLSTM	ID				"0"
	BF	\$102		TSTTBL	"0"
	COMP	SYMBOL			"8"
		CALLCO		BF	\$108
		"2"		B	\$107
	BEF		\$108	TSTTBL	"0"
	ID				"9"
	BEF			BF	\$109
	CANCEL			B	\$107
	SEARCH		\$109	TSTTBL	"0"
		"0"			"#"
	CLM	CALLPR		BF	\$110
	BEF			B	\$107
\$102	R		\$110	TSTTBL	"0"

		"="
	BF	\$111
	B	\$107
\$111	TSTTBL	"0"
		":"
	BF	\$107
\$107	BF	\$106
	OUT	LD
	OUT	IOCHK
	B	\$105
\$106	TSTTBL	"0"
		"+"
	BF	\$115
	B	\$114
\$115	TSTTBL	"0"
		"A"
	BF	\$116
	B	\$114
\$116	TSTTBL	"0"
		"B"
	BF	\$117
	B	\$114
\$117	TSTTBL	"0"
		"C"
	BF	\$118
	B	\$114
\$118	TSTTBL	"0"
		"D"
	BF	\$114
\$114	BF	\$105
	MARK	
	OUT	LD
	OUT	"1"
	EVAL	"4"
		"1"
	SAVE	
	PUSHLB	

	OUT	LDA
	OUT	"1"
	LBI	
	EVAL	
		"4"
		"1"
	RESTOR	
	OUT	LD
	OUT	"2"
	OUT	"0004"
	OUT	ADD
	OUT	STO
	OUT	LD
	OUT	"1"
	LBI	
	ENTLOC	
	ENTL	
		"0"
		"6"
	ENTL	"0"
		"1"
	OUT	"0000"
	POP	
	SET	
\$105	BT	\$120
	BM	"F: INVALID PROC
\$120	CLM	CLLPRM
	BEF	
	MOVE	STACKT
		SYMBOL
	OUT	LD
	EVAL	
		"5"
		"0"
	RESTOR	
	LBI	
	ENTLOC	

	ENTL		TSTTBA	"7"
		"0"		PARMCN
		"6"	BT	\$133
\$104	R		BM	"W: INCORRECT PAR
CLLPRM	OUT	FLAG	SET	
	MOVE	PARMCN	\$133 B	\$126
		SYMBOL	\$127 SET	
	STKSYM		BF	\$126
	MOVE	ZERO	\$126 BEF	
		PARMCN	POP	
	MOVE	STMLAB	MOVE	SYMBOL
		SYMBOL		PARMCN
	STKSYM		\$122 R	
	CLM	PARMPR	PARMPR TEST	"("
	BF	\$124	BF	\$136
	B	\$123	CLM	PARM
\$124	SET		BEF	
	BF	\$123	\$137 TEST	"."
\$123	BF	\$122	BF	\$139
	POP		CLM	PARM
	SEARCH	"0"	BEF	
	TSTTBL	"0"	\$139 BT	\$137
		"9"	SET	
	BF	\$129	BEF	
	B	\$128	TEST	"")"
\$129	TSTTBL	"0"	BEF	
		"."	\$136 R	
	BF	\$130	PARM CLM	PARMID
	B	\$128	BF	\$143
\$130	TSTTBL	"0"	B	\$142
		"="	\$143 CLM	PARMEX
	BF	\$131	BF	\$142
	B	\$128	\$142 BF	\$141
\$131	TSTTBL	"0"	A	ONE
		"8"		PARMCN
	BF	\$128	\$141 R	
\$128	BF	\$127	PARMID LATCH	ALFTPT

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

	BEF	
\$156	BEF	
\$152	R	
PARMEX	PUSHLB	
	LB1	
	SEARCH	"1"
	ENTA	"1"
		DYNAMB
	ENTL	"0"
		"7"
	A	EIGHT
		DYNAMB
	OUT	LDA
	EVAL	
		"5"
		"0"
	OUT	LDA
	EVAL	
		"5"
		"0"
	CLM	EXP
	BF	\$162
	B	\$161
\$162	CLM	SSE
	BF	\$163
	B	\$161
\$163	CLM	LTERM
	BF	\$164
	B	\$161
\$164	CLM	BPRMRY
	BF	\$161
\$161	BF	\$160
	OUT	STO
\$160	R	
GPROC	MARK	
	OUT	LDA
	OUT	"1"

EVAL	"4"
	"1"
SAVE	
PUSHLB	
LB1	
SEARCH	"1"
ENTL	"0"
	"1"
ENTA	"1"
	DYNAMB
OUT	LDA
EVAL	
	"5"
	"0"
OUT	LDA
EVAL	
	"5"
	"0"
RESTOR	
OUT	STO
A	FOUR
	DYNAMB
PUSHLB	
LB1	
SEARCH	"1"
ENTA	"1"
	DYNAMB
ENTL	"0"
	"1"
OUT	LDA
EVAL	
	"5"
	"0"
OUT	LD
OUT	IOCHK
OUT	STO
A	FOUR

\$166  
IPROC

R	DYNAMB
MARK	
OUT	LD
OUT	"1"
EVAL	"4"
	"1"
SAVE	
MARK	
OUT	LD
OUT	"/"
EVAL	"4"
	"1"
SAVE	
PUSHLB	
LB1	
SEARCH	"1"
ENTL	"0"
	"1"
ENTA	"1"
	DYNAMB
OUT	LDA
EVAL	
	"5"
	"0"
OUT	LDA
EVAL	
	"5"
	"0"
RESTOR	
OUT	STO
A	FOUR
	DYNAMB
PUSHLB	
LB1	
SEARCH	"1"
ENTA	"1"



	ENTL	DYNAMB			"1"
		"0"		CLM	DECL1
		"1"		BT	\$172
	OUT	LDA		BM	"F"
	EVAL			SCAN	";"
		"5"		SET	
		"0"	\$172	TEST	","
	A	FOUR		BF	\$175
		DYNAMB		CLM	DECL1
	PUSHLB			BT	\$175
	OUT	LDA		BM	"F"
	OUT	"1"		SCAN	";"
	LBI			SET	
	EVAL		\$175	BT	\$172
		"4"		SET	
		"1"		BEF	
	RESTOR			TEST	";"
	OUT	LD		BEF	
	OUT	"2"	\$171	R	
	OUT	"0004"	DECL1	CLM	IDELMN
	OUT	ADD		BF	\$178
	OUT	STO		B	\$177
	OUT	LD	\$178	CLM	IDGROU
	OUT	"1"		BF	\$177
	LBI		\$177	R	
	ENTLOC		IDGROU	TEST	"{"
	ENTL			BF	\$181
		"0"		CLM	LISTPT
		"6"		BEF	
	ENTL	"0"	\$181	R	
		"1"	LISTPT	ID	
	OUT	"0000"		BF	\$183
	OUT	STO		STKSYM	
\$168	R			CLM	ARRYPT
DECLAR	TEST	"DECLARE"		BF	\$185
	BF	\$171		MOVE	DIMCNT
	MOVI	ELATCH			SYMBOL

	STKSYM	
	CLM	IDLIST
	BEF	
	POP	
	MOVE	SYMBOL
		DIMCNT
	POP	
	MOVE	SYMBOL
		SYMSAV
	CLM	ARRAYSE
	BEF	
\$185	B	\$184
	CLM	IDLIST
	BF	\$184
	POP	
	SEARCH	"1"
	CLM	IDSEM
	BEF	
	A	LENGTH
		DYNAMB
\$184	BEF	
\$183	R	
IDLIST	TEST	","
	BF	\$188
	CLM	LISTPT
	BEF	
	B	\$187
\$188	TEST	")"
	BF	\$187
	CLM	ATTRIB
	BEF	
\$187	R	
ATTRIB	TEST	"CHARACTER (
	BF	\$193
	B	\$192
\$193	TEST	"CHAR (
	BF	\$192

\$192	BF	\$191
	INUM	
	BEF	
	TEST	")"
	BEF	
	MOVE	SYMPTW
		LENGTH
	MOVI	TYPE
		"4"
	B	\$190
\$191	TEST	"RETURNS (
	BF	\$195
	CLM	ATTRIB
	BEF	
	TEST	")"
	BEF	
	A	OCTEN
		TYPE
	MOVI	LENGTH
		"00"
	B	\$190
\$195	TEST	"FIXED"
	BF	\$196
	MOVI	LENGTH
		LEVNO
	MOVI	TYPE
		"1"
	B	\$190
\$196	TEST	"FLOAT"
	BF	\$197
	MOVI	LENGTH
		LEVNO
	MOVI	TYPE
		"2"
	B	\$190
\$197	TEST	"LOGICAL"
	BF	\$198

	MOVI	LENGTH
		LEVNO
	MOVI	TYPE
		"3"
\$198	B	\$190
	TEST	"LABEL"
	BF	\$190
	MOVI	LENGTH
		LEVNO
	MOVI	TYPE
		"5"
\$190	R	
ARRYPT	TEST	"("
	BF	\$201
	MOVI	DIMCNT
		"0"
	CLM	BDPRLS
	BEF	
	TEST	")"
	BEF	
\$201	R	
BDPRLS	MARK	
	CLM	BNDPAI
	BF	\$203
	SAVE	
\$204	TEST	","
	BF	\$206
	MARK	
	RESTOR	
	CLM	BNDPAI
	BEF	
	SAVE	
\$206	BT	\$204
	SET	
	BEF	
\$203	R	
BNDPAI	A	ONE

	CLM	DIMCNT
		INTBND
	BF	\$208
	TEST	":"
	BF	\$210
	RESTOR	
	CLM	INTBND
	BEF	
	RESTOR	
	B	\$209
\$210	SET	
	BF	\$209
	OUT	"Z0001"
	RESTOR	
\$209	BEF	
\$208	R	
INTBND	ID	
	BF	\$213
	SEARCH	
		"0"
	TSTIBL	"0"
		"1"
	ET	\$214
	BM	"W: INVALID ARRAY
\$214	MARK	
	EVAL	
		"5"
		"0"
	SAVE	
	B	\$212
\$213	INUM	
	BF	\$212
	MARK	
	OUT	"Z"
	OUTSYM	
	SAVE	
\$212	R	

IDE	LMN	ID		BF	\$222
		BF	\$217	EVAL	
		STKSYM			"5"
		CLM	ARRYPT		"0"
		BF	\$219	EVAL	"1"
		CLM	ATTRIB		"7"
		BEF		EVAL	"2"
		POP			"5"
		MOVE	SYMBOL	RESTOR	
			SYMSAV	A	DOPFIX
		CLM	ARRYSE		DYNAMB
		BEF		MOVE	FOUR
		B	\$218		SYMBOL
\$219		CLM	ATTRIB	M	DIMCNT
		BF	\$218		SYMBOL
		POP		A	SYMBOL
		SEARCH	"1"		DYNAMB
		CLM	IDSEM	SET	
		BEF		\$222 R	
		A	LENGTH	IDSEM ENTA	"0"
			DYNAMB		TYPE
\$218		BEF		ENTA	"5"
\$217		R			LENGTH
ARRYSE		MOVE	STACKT	COMP	LENGTH
			SYMBOL		FUNCT
		OUT	ALLOC		"2"
		EVAL		BF	\$226
			"5"	R	
			"0"	B	\$225
		MOVE	BLNK8	\$226 SET	
			SYMBOL	BF	\$225
		MOVE	SYMSAV	ENTA	"1"
			SYMBOL		DYNAMB
		SEARCH	"1"	\$225 BF	\$224
		ENTA	"7"	\$224 R	
			DIMCNT	DOGROU ID	
		CLM	IDSEM	BF	\$229

	COMP	SYMBOL		CLM	EXP
		DOSYM		BT	\$241
		"2"		BM	"F: BAD CASE EXPR
	BEF			SCAN	";"
	MOVI	ELATCH		CLM	TAIL
		"1"		R	
	TEST	";"	\$241	TEST	";"
	BF	\$231		BEF	
	CLM	TAIL		OUT	LD
	BEF			OUT	"Z0005"
	B	\$230		OUT	MULT
\$231	CLM	CASE		LB1	
	BF	\$232		SEARCH	"1"
	B	\$230		ENTL	"0"
\$232	CLM	DOWHIL			"1"
	BF	\$233		OUT	LD
	B	\$230		EVAL	
\$233	CLM	LOOP			"5"
	BF	\$230			"0"
\$230	BT	\$229		OUT	ADD
	BM	"F: INVALID		OUT	JUMPA
	CLM	TAIL		PUSHLB	
\$229	R			PUSHLB	
TAIL	CLM	BLKBDY		LB1	
	BF	\$237		ENTLOC	
	CLM	ENDING		ENTL	
	BT	\$237			"0"
	BM	"W: INVALID			"6"
	SCAN	"; "		CLM	STMENT
	SET			BEF	
\$237	R			OUT	JUMP
CASE	TEST	"CASE"		LB2	
	BF	\$240		EVAL	
	PUSHLB				"4"
	MOVI	ELATCH			"1"
		"1"		MARK	
	PUSHLB			OUT	JUMP

	LB1		"0"		
	EVAL		"6"		
		"4"		ENTL	"0"
		"1"			"1"
	SAVE			SWAP	
	POP			POP	
\$242	PUSHLB			LB2	
	LB1			EVAL	
	ENTLOC		"4"		
	ENTL		"1"		
		"0"		LB2	
		"6"		ENTLOC	
	CLM	STMENT		ENTL	
	BF	\$244			"0"
	OUT	JUMP			"6"
	LB2			RESTOR	
	EVAL			CLM	ENDING
		"4"		BEF	
		"1"		LB1	
	MARK			ENTLOC	
	RESTOR			ENTL	
	OUT	JUMP			"0"
	LB1				"6"
	EVAL			\$240 R	
		"4"		DOWHIL TEST	"WHILE"
		"1"		BF	\$246
	SAVE			PUSHLB	
	POP			LB1	
\$244	BT	\$242		ENTLOC	
	SET			ENTL	
	BEF				"0"
	POP				"6"
	OUT	LD		PUSHLB	
	OUT	"6"		CLM	WHLPT
	LB2			BEF	
	ENTLOC			TEST	" ; "
	ENTL			BEF	

	CLM	TAIL
	BEF	
	OUT	JUMP
	LB2	
	EVAL	"4"
		"1"
	LB1	
	ENTLOC	
	ENTL	"0"
		"6"
\$246	R	
WHILE	TEST	"WHILE"
	BF	\$248
	CLM	WHLPT
	BEF	
	TEST	" ; "
	BEF	
\$248	R	
WHLPT	TEST	" ( "
	BF	\$250
	CLM	BOOLEX
	BT	\$251
	BM	"F: LOGICAL
	SCAN	" ; "
	R	
\$251	OUT	JUMPF
	LB1	
	EVAL	"4"
		"1"
	TEST	" ) "
	BEF	
\$250	R	
LOOP	ID	
	BF	\$253

	CLM	UNDTST
	BEF	
	PUSHLB	
	OUT	LDA
	EVAL	"5"
		"0"
	STKSYM	
	CLM	ITERPR
	BEF	
	OUT	JUMPT
	LB1	
	EVAL	"4"
		"1"
	CLM	WHILE
	BF	\$255
	B	\$254
\$255	TEST	" ; "
	BF	\$254
\$254	BT	\$257
	BM	"w: EXPECTED ; "
	SET	
\$257	CLM	TAIL
	BEF	
	POP	
	OUT	LDA
	EVAL	"5"
		"0"
	OUT	LD
	EVAL	"5"
		"0"
	RESTOR	
	LB1	
	ENTLOC	

	ENTL	"0"		LB1	
		"6"		EVAL	
\$253	R				"5"
ITERPR	TEST	"="			"0"
	BF	\$259		CLM	EXP
	MOVI	ELATCH		BF	\$261
		"1"		OUT	STO
	CLM	EXP		MARK	
	BT	\$260		LB2	
	BM	"F: INVALID		ENTLOC	
\$260	OUT	SST		ENTL	
	TEST	"TO"			"0"
	BEF				"6"
	PUSHLB			OUT	LD
	LB1			LB1	
	ENTLOC			EVAL	
	ENTL				"5"
		"0"			"0"
		"6"		SWAP	
	MARK			POP	
	LATCH	LOOPRM	\$261	BT	\$264
	BF	\$262		BM	"F: INVALID INDEX
	B	\$261	\$264	TEST	"BY"
\$262	PUSHLB			BF	\$266
	LB1			OUT	STCKC
	ENTLOC			TEST	"_"
	ENTL			BF	\$268
		"0"		OUT	"2"
		"6"		CLM	IOCHK
	ENTA	"1"		BEF	
		DYNAMB		MARK	
	ENTL	"0"		OUT	SUB
		"7"		SAVE	
A	EIGHT		\$268	B	\$267
	DYNAMB			SET	
	OUT	LDA		BF	\$267
				OUT	"4"



	CLM	IOCHK		RESTOR	
	BEF			MARK	
	MARK			OUT	LD
	OUT	ADD		LB1	
	SAVE			EVAL	
\$267	BEF				"5"
	MARK				"0"
	LATCH	LOOPRM		SWAP	
	BF	\$271		POP	
	B	\$270	\$270	BT	\$273
\$271	POP			BM	"F: INVALID INCRE
	SAVE		\$273	RESTOR	
	PUSHLB			B	\$265
	LB1		\$266	SET	
	ENTLOC			BF	\$265
	ENTL			OUT	STCKC
		"0"		OUT	"4"
		"6"		CLM	IOCHK
	ENTA	"1"		BEF	
		DYNAMB		MARK	
	ENTL	"0"		OUT	LD
		"7"		OUT	"2"
	A	EIGHT		OUT	"0001"
		DYNAMB		OUT	ADD
	OUT	LDA	\$265	BEF	
	LB1			OUT	SST
	EVAL			OUT	JUMP
		"5"		LB1	
		"0"		EVAL	
	CLM	EXP			"4"
	BF	\$270			"1"
	OUT	STO		SAVE	
	LB2		\$259	R	
	ENTLOC		IOCHK	COMP	IOSW
	ENTL				IOITER
		"0"			"2"
		"6"		BF	\$276

	OUT	JUMPT
	LB2	
	EVAL	
		"4"
		"1"
	RESTOR	
	SET	
	B	\$275
\$276	SET	
	BF	\$275
\$275	R	
LOOPRM	ID	
	BF	\$281
	OUT	LD
	EVAL	
		"5"
		"0"
	B	\$280
\$281	INUM	
	BF	\$282
	OUT	LD
	OUT	"Z"
	OUTSYM	
	B	\$280
\$282	FNUM	
	BF	\$280
	OUT	LD
	OUT	"@"
	OUTSYM	
\$280	BF	\$279
	TEST	"+"
	BF	\$287
	B	\$286
\$287	TEST	"_"
	BF	\$288
	B	\$286
\$288	TEST	"("

	BF	\$289
	B	\$286
\$289	TEST	"/"
	BF	\$290
	B	\$286
\$290	TEST	"*"
	BF	\$286
\$286	BF	\$285
	SETF	
	BEF	
	B	\$284
\$285	SET	
	BF	\$284
\$284	BEF	
\$279	R	
UNDTST	SEARCH	
		"0"
	TSTTBL	"0"
		"0"
	BF	\$294
	BM	"W: UNDECLARED IDE
	B	\$293
\$294	SET	
	BF	\$293
\$293	R	
DUPTST	SEARCH	"1"
	SET	
	BF	\$297
	TSTTBL	"0"
		"0"
	BT	\$297
	BM	"W: MULTIPLE DECLA
\$297	R	
ENDING	TEST	"END"
	BF	\$300
	ID	
	BF	\$302

	B	\$301
\$302	SET	
	BF	\$301
\$301	BEF	
\$300	R	
BOOLEX	LATCH	IFCLSE
	BF	\$307
	PUSHLB	
	OUT	JUMPF
	LB1	
	EVAL	
		"4"
		"1"
	CLM	BTERM
	BEF	
	PUSHLB	
	OUT	JUMP
	LB1	
	EVAL	
		"4"
		"1"
	TEST	"ELSE"
	BEF	
	LB2	
	ENTLOC	
	ENTL	
		"0"
		"6"
	CLM	BOOLEX
	BEF	
	LB1	
	ENTLOC	
	ENTL	
		"0"
		"6"
\$307	BF	\$305
	B	\$304

\$305	CLM	BTERM
	BF	\$304
\$304	R	
BTERM	CLM	BFACTO
	BF	\$310
\$311	TEST	".OR."
	BF	\$313
	CLM	BFACTO
	BEF	
	OUT	OR
\$313	BT	\$311
	SET	
	BEF	
\$310	R	
BFACTO	CLM	BSCNDR
	BF	\$315
\$316	TEST	".AND."
	BF	\$318
	CLM	BSCNDR
	BEF	
	OUT	AND
\$318	BT	\$316
	SET	
	BEF	
\$315	R	
BSCNDR	CLM	BPRMRY
	BF	\$320
	B	\$319
\$320	TEST	".NOT."
	BF	\$319
	CLM	BPRMRY
	BEF	
	OUT	NOT
\$319	R	
BPRMRY	CLM	BVALUE
	BF	\$323
	B	\$322

\$323	LATCH	BVARBL
	BF	\$324
	B	\$322
\$324	LATCH	BFNCT
	BF	\$325
	B	\$322
\$325	CLM	RELATN
	BF	\$326
	B	\$322
\$326	TEST	"("
	BF	\$322
	CLM	BOOLEX
	BEF	
	TEST	" ) "
	BT	\$322
	BM	"W: EXPE
	SET	
\$322	R	
RELATN	CLM	SAE
	EF	\$330
	CLM	RELOP
	BEF	
	CLM	SAE
	BEF	
	OUT	STCKC
	RESTOR	
	B	\$329
\$330	CLM	SSE
	BF	\$329
	CLM	RELOP
	BEF	
	CLM	SSE
	BEF	
	OUT	COMPC
	RESTOR	
\$329	R	
BFNCT	ID	

	BF	\$333
	SEARCH	"0"
	TSTTBL	"0"
		"="
	BF	\$335
	B	\$334
\$335	TSTTBL	"0"
		"C"
	BF	\$334
\$334	BEF	
	CANCEL	
	CLM	CALLPR
	BEF	
\$333	R	
BVALUE	TEST	".T."
	BF	\$338
	OUT	LD
	OUT	", "
	OUT	"T"
	B	\$337
\$336	TEST	".F."
	BF	\$337
	OUT	LD
	OUT	", "
	OUT	"F"
\$337	R	
BVARBL	ID	
	BF	\$341
	SEARCH	"0"
	TSTTBL	"0"
		"3"
	BF	\$343
	B	\$342
\$343	TSTTBL	"0"
		"T"
	BF	\$342
\$342	BEF	

	CANCEL		OUT	"2"
	TEST	"("	SAVE	
	BF	\$346	B	\$348
	OUT	LDA	\$351 TEST	">="
	EVAL		BF	\$352
		"5"	MARK	
		"0"	OUT	"5"
	CLM	SUBLIS	SAVE	
	BEF		B	\$348
	TEST	")"	\$352 TEST	">"
	BEF		BF	\$353
	OUT	INDXR	MARK	
	OUTSYM		OUT	"4"
	B	\$345	SAVE	
\$346	SET		B	\$348
	BF	\$345	\$353 TEST	"≠"
	OUT	LD	BF	\$348
	EVAL		MARK	
		"5"	OUT	"6"
		"0"	SAVE	
\$345	BEF		\$348 R	
\$341	R		SAE	CLM
RELOP	TEST	"="		TERM
	BF	\$349	BF	\$358
	MARK		B	\$357
	OUT	"1"	\$358 TEST	"-"
	SAVE		BF	\$359
	B	\$348	CLM	TERM
\$349	TEST	"<="	BEF	
	BF	\$350	OUT	NEG
	MARK		B	\$357
	OUT	"3"	\$359 TEST	"+"
	SAVE		BF	\$357
	B	\$348	CLM	TERM
\$350	TEST	"<"	BEF	
	BF	\$351	\$357 BF	\$356
	MARK		\$361 TEST	"+"
			BF	\$363

	CLM	TERM
	BEF	
	OUT	ADD
	B	\$362
\$363	TEST	"_"
	BF	\$362
	CLM	TERM
	BEF	
	OUT	SUB
\$362	BT	\$361
	SET	
	BEF	
\$356	R	
TERM	CLM	PRIMRY
	BF	\$366
\$367	TEST	"*"
	BF	\$369
	CLM	PRIMRY
	BEF	
	OUT	MULT
	B	\$368
\$369	TEST	"/"
	BF	\$368
	CLM	PRIMRY
	BEF	
	OUT	DIV
\$368	BT	\$367
	SET	
	BEF	
\$366	R	
PRIMRY	TEST	"("
	BF	\$372
	CLM	EXP
	BEF	
	TEST	")"
	BEF	
	B	\$371

\$372	CLM	CONST
	BF	\$373
	B	\$371
\$373	LATCH	VARBLE
	BF	\$374
	B	\$371
\$374	LATCH	AFUNCT
	BF	\$371
\$371	R	
AFUNCT	ID	
	BF	\$377
	SEARCH	"0"
	TSTTBL	"0"
		"9"
	BF	\$379
	B	\$378
\$379	TSTTBL	"0"
		"1"
	BF	\$380
	B	\$378
\$380	TSTTBL	"0"
		"A"
	BF	\$381
	E	\$378
\$381	TSTTBL	"0"
		"B"
	BF	\$378
\$378	BEF	
	CANCEL	
	CLM	CALLPR
	BEF	
\$377	R	
CONST	INUM	
	BF	\$384
	OUT	LD
	OUT	"Z"
	OUTSYM	

	B	\$383
\$384	FNUM	
	BF	\$383
	OUT	LD
	OUT	"e"
	OUTSYM	
\$383	R	
VARIABLE	ID	
	BF	\$387
	SEARCH	"0"
	TSTTBL	"0"
		"1"
	BF	\$389
	B	\$388
\$389	TSTTBL	"0"
		"2"
	BF	\$390
	B	\$388
\$390	TSTTBL	"0"
		"/"
	BF	\$391
	B	\$388
\$391	TSTTBL	"0"
		"5"
	BF	\$388
\$388	BEF	
	CANCEL	
	TEST	"("
	BF	\$394
	OUT	LDA
	EVAL	
		"5"
		"0"
	CLM	SUBLIS
	BEF	
	TEST	" ) "
	BEF	

	OUT	INDXR
	OUTSYM	
	B	\$393
\$394	SET	
	BF	\$393
	OUT	LD
	EVAL	
		"5"
		"0"
\$393	BEF	
\$387	R	
EXP	LATCH	IFCLSE
	BF	\$397
	PUSHLB	
	OUT	JUMPF
	LB1	
	EVAL	
		"4"
		"1"
	CLM	SAE
	BEF	
	TEST	"ELSE"
	BEF	
	PUSHLB	
	OUT	JUMP
	LB1	
	EVAL	
		"4"
		"1"
	LB2	
	ENTLOC	
	ENTL	
		"0"
		"6"
	CLM	EXP
	BEF	
	LB1	

	ENTLOC				"6"
	ENTL				\$399
		"0"	\$400	R	SSE
		"6"		BF	\$399
\$397	B	\$396	\$399	R	
	CLM	SAE	SSE	CLM	STERM
	BF	\$396		BF	\$403
\$396	R		\$404	TEST	"//"
SEXP	LATCH	IFCLSE		BF	\$406
	BF	\$400		CLM	STERM
	PUSHLB			BEF	
	OUT	JUMPF		OUT	CAT
	LB1		\$406	BT	\$404
	EVAL			SET	
		"4"		BEF	
		"1"	\$403	R	
	CLM	SSE	STERM	CLM	SBSTRN
	BEF			BF	\$408
	TEST	"ELSE"		B	\$407
	BEF		\$408	STRTST	
	PUSHLB			BF	\$409
	OUT	JUMP		OUT	LDA
	LB1			OUT	"("
	EVAL			OUTSYM	
		"4"		B	\$407
		"1"	\$409	LATCH	SVARBL
	LB2			BF	\$410
	ENTLOC			R	\$407
	ENTL		\$410	LATCH	SFUNCT
		"0"		BF	\$407
		"6"	\$407	R	
	CLM	SEXP	SFUNCT	ID	
	BEF			BF	\$413
	LB1			SEARCH	"0"
	ENTLOC			TSTTBL	"0"
	ENTL				":"
		"0"		BF	\$415



	B	\$414
\$415	TSTTBL	"0"
		"D"
	BF	\$414
\$414	BEF	
	CANCEL	
	CLM	CALLPR
	BEF	
\$413	R	
SBSTRN	TEST	"SUBSTR("
	BF	\$418
	CLM	SEXP
	BEF	
	TEST	",,"
	BEF	
	CLM	EXP
	BEF	
	TEST	",,"
	BF	\$420
	CLM	EXP
	BEF	
	B	\$419
\$420	SET	
	BF	\$419
	OUT	LD
	OUT	"Z"
	OUT	"0000"
\$419	BEF	
	TEST	")"
	BEF	
	OUT	SUBSTR
\$418	R	
SVARBL	ID	
	BF	\$423
	SEARCH	"0"
	TSTTBL	"0"
		"4"

	BF	\$425
	B	\$424
\$425	TSTTBL	"0"
		"U"
	BF	\$424
\$424	BEF	
	CANCEL	
	CUT	LDA
	EVAL	
		"5"
		"0"
	EVAL	"2"
		"5"
	TEST	"("
	BF	\$428
	CLM	SUBLIS
	BEF	
	TEST	")"
	BEF	
	OUT	INDXA
	OUTSYM	
	B	\$427
\$428	SET	
	BF	\$427
\$427	BEF	
\$423	R	
SUBLIS	MOVE	ONE
		DIMCNT
	STKSYM	
	CLM	EXP
	BF	\$431
\$432	TEST	",,"
	BF	\$434
	CLM	EXP
	BEF	
	A	ONE
		DIMCNT

\$434	BT	\$432	TEST	) "
	SET		BEF	
	BEF		OUT	INDXR
	POP		OUTSYM	
	SEARCH		B	\$442
	TSTTbA	"0"	\$443	SET
		"7"	BF	\$442
		DIMCNT	OUT	LD
	BT	\$435	OUT	" "
	BM	"w: INCOR	EVAL	"4"
\$435	MOVE	DIMCNT		"1"
		SYMBOL	\$442	BF
			\$438	BEF
\$431	R		\$437	R
LTERM	ID		LBLCON	TSTTBL
	BF	\$437		"0"
	SEARCH	"1"		"6"
	CLM	LBLCON	BF	\$449
	BF	\$439	OUT	LDA
	B	\$438	EVAL	
\$439	CLM	LBLVAR		"5"
	BF	\$440		"0"
	B	\$438	\$449	R
\$440	TEST	" ("	LBLVAR	TSTTBL
	BF	\$443		"0"
	OUT	LDA		"V"
	OUT	"5"	BF	\$453
	EVAL	"4"	MOVE	PEXITF
		"1"		PEXITT
	CLM	EXP	\$453	B
	BEF		MOVI	PEXITT
\$444	TEST	" ,"		"0"
	BF	\$446	TSTTBL	"0"
	CLM	EXP		"5"
	BEF		BF	\$452
\$446	BT	\$444	\$452	BF
	SET		TEST	" ("
	BEF		BF	\$456
			OUT	LDA

	EVAL	"5"
		"0"
	CLM	SUBLIS
	BEF	
	TEST	")"
	BEF	
	OUT	INDXR
	OUTSYM	
	B	\$455
\$456	SET	
	BF	\$455
	OUT	LD
	EVAL	
		"5"
		"0"
\$455	BEF	
\$451	R	
CONDST	LATCH	IFCLSE
	BF	\$459
	PUSHLB	
	OUT	JUMPF
	LB1	
	EVAL	
		"4"
		"1"
	CLM	BSCSTM
	BEF	
	TEST	"ELSE"
	BF	\$461
	PUSHLB	
	OUT	JUMP
	LB1	
	EVAL	
		"4"
		"1"
	LB2	

	ENTLOC	
	ENTL	
		"0"
		"6"
	CLM	STMENT
	BEF	
	LB1	
	ENTLOC	
	ENTL	
		"0"
		"6"
	B	\$460
\$461	SET	
	BF	\$460
	LB1	
	ENTLOC	
	ENTL	
		"0"
		"6"
\$460	BEF	
\$459	R	
IFCLSE	ID	
	BF	\$464
	COMP	SYMBOL
		IFSYM
		"2"
	REF	
	CANCEL	
	CLM	BOOLEX
	BEF	
	TEST	"THEN"
	BEF	
\$464	R	
IDENT	ID	
	BF	\$466
	SEARCH	"0"
	TSTTBL	"0"

		"0"		BF	\$483
	BEF			B	\$481
	BM	"W: UNDEC	\$483	CLM	BPRMRY
\$466	R			BF	\$481
UNCOND	LATCH	GOTOST	\$481	BEF	
	BF	\$468		TEST	" ) "
	B	\$467		BEF	
\$468	CLM	INOUT		OUT	SWP
	BF	\$469		B	\$479
	B	\$467	\$480	SET	
\$469	LATCH	CLLSTM		BF	\$479
	BF	\$470	\$479	BEF	
	B	\$467		OUT	RETURN
\$470	LATCH	RTNSTM	\$478	R	
	BF	\$471	GOTOST	TEST	"GO"
	B	\$467		BF	\$487
\$471	CLM	STPSTM		TEST	"TO"
	BF	\$472		BEF	
	B	\$467		CANCEL	
\$472	CLM	ASSGNS		MARK	
	BF	\$473		CLM	LTERM
	B	\$467		BEF	
\$473	CLM	IDENT		COMP	PEXITT
	BF	\$467			PEXITF
\$467	R				"2"
STPSTM	TEST	"STOP"		BF	\$489
	BF	\$476		SAVE	
	OUT	STOP		OUT	POPUP
\$476	R			RESTOR	
RTNSTM	TEST	"RETURN"		OUT	RETURN
	BF	\$478		B	\$488
	TEST	" ( "	\$489	SET	
	BF	\$480		BF	\$488
	CLM	EXP		OUT	JUMPA
	BF	\$482	\$488	BEF	
	B	\$481	\$487	R	
\$482	CLM	SSE	ASSGNS	CLM	AASSGN

	BF	\$492
	B	\$491
\$492	CLM	BASSGN
	BF	\$493
	B	\$491
\$493	CLM	SASSGN
	BF	\$494
	B	\$491
\$494	CLM	LASSGN
	BF	\$491
\$491	R	
AASSGN	LATCH	ALFTPT
	BF	\$497
	MARK	
	OUT	STO
	SAVE	
\$498	TEST	","
	BF	\$500
	CLM	ALFTPT
	BEF	
	MARK	
	OUT	SST
	RESTOR	
	SAVE	
\$500	BT	\$498
	SET	
	BEF	
	TEST	"="
	BEF	
	CLM	EXP
	BEF	
	RESTOR	
\$497	R	
BASSGN	LATCH	BLFTPT
	BF	\$502
	MARK	
	OUT	STO

	SAVE	
\$503	TEST	","
	BF	\$505
	CLM	BLFTPT
	BEF	
	MARK	
	OUT	SST
	RESTOR	
	SAVE	
\$505	BT	\$503
	SET	
	BEF	
	TEST	"="
	BEF	
	CLM	BOOLEX
	BEF	
	RESTOR	
\$502	R	
SASSGN	CLM	SLFTPT
	BF	\$507
	MARK	
	OUT	STO
	SAVE	
\$508	TEST	","
	BF	\$510
	CLM	SLFTPT
	BEF	
	MARK	
	OUT	SST
	RESTOR	
	SAVE	
\$510	BT	\$508
	SET	
	BEF	
	TEST	"="
	BEF	
	CLM	SEXP

	BEF	
	RESTOR	
\$507	R	
LASSGN	LATCH	LLFTPT
	BF	\$512
	MARK	
	OUT	STO
	SAVE	
\$513	TEST	"."
	BF	\$515
	CLM	LLFTPT
	BEF	
	MARK	
	OUT	SST
	RESTOR	
	SAVE	
\$515	BT	\$513
	SET	
	BEF	
	TEST	"="
	BEF	
	CLM	LTERM
	BEF	
	COMP	PEXITT
		ZERO
		"2"
	BT	\$516
	BM	"F: INDIR
\$516	RESTOR	
\$512	R	
ALFTPT	ID	
	BF	\$518
	SEARCH	"0"
	TSTTBL	"0"
		"1"
	BF	\$520
	B	\$519

\$520	TSTTBL	"0"
		"2"
	BF	\$521
	B	\$519
\$521	TSTTBL	"0"
		"/"
	BF	\$522
	B	\$519
\$522	TSTTBL	"0"
		"5"
	BF	\$519
\$519	BEF	
	CANCEL	
	OUT	LDA
	EVAL	
		"5"
		"0"
	CLM	SUBPAR
	BEF	
\$518	R	
BLFTPT	ID	
	BF	\$525
	SEARCH	"0"
	TSTTBL	"0"
		"3"
	BF	\$527
	B	\$526
\$527	TSTTBL	"0"
		"T"
		\$526
	BF	
\$526	BEF	
	CANCEL	
	OUT	LDA
	EVAL	
		"5"
		"0"
	CLM	SUBPAR

	BEF	
\$525	R	
LLFTPT	ID	
	BF	\$530
	SEARCH	"0"
	TSTTBL	"0"
		"5"
	BF	\$532
	B	\$531
\$532	TSTTBL	"0"
		"V"
	BF	\$531
\$531	BEF	
	CANCEL	
	OUT	LDA
	EVAL	
		"5"
		"0"
	CLM	SUBPAR
	BEF	
\$530	R	
SLFTPT	LATCH	SVARBL
	BF	\$535
	B	\$534
\$535	CLM	LSUBST
	BF	\$534
\$534	R	
LSUBST	TEST	"SUBSTR("
	BF	\$538
	CLM	SVARBL
	BEF	
	TEST	",,"
	BEF	
	CLM	EXP
	BEF	
	TEST	",,"
	BF	\$540

	CLM	EXP
	BEF	
	E	\$539
\$540	SET	
	BF	\$539
	OUT	LD
	OUT	"2"
	OUT	"0000"
\$539	BEF	
	TEST	")"
	BEF	
	OUT	SUBSTR
\$538	R	
SUBPAR	TEST	"("
	BF	\$543
	CLM	SUBLIS
	BEF	
	TEST	")"
	BEF	
	OUT	INDXA
	OUTSYM	
	B	\$542
\$543	SET	
	BF	\$542
\$542	R	
INOUT	LATCH	INPUT
	BF	\$546
	B	\$545
\$546	LATCH	OUTPUT
	BF	\$545
\$545	R	
INPUT	TEST	"GET"
	BF	\$549
	TEST	"STRING"
	BF	\$551
	MARK	
	OUT	"Y"

	SAVE	
	TEST	"("
	BEF	
	CLM	SLFTPT
	BEF	
	TEST	")"
	BEF	
	B	\$550
\$551	SET	
	BF	\$550
	MARK	
	OUT	"Z"
	SAVE	
\$550	BEF	
	TEST	"EDIT"
	BEF	
	CANCEL	
	PUSHLB	
	OUT	FMT
	LB1	
	EVAL	
		"4"
		"1"
	RESTOR	
	TEST	"("
	BEF	
	CLM	ILIST
	BEF	
	TEST	")"
	BEF	
	PUSHLB	
	OUT	JUMP
	LB1	
	EVAL	
		"4"
		"1"
	LB2	

	ENTLOC	
	ENTL	
		"0"
		"6"
	CLM	FMTLIS
	BEF	
	LB1	
	ENTLOC	
	ENTL	
		"0"
		"6"
\$549	R	
ILIST	CLM	IELMNT
	BF	\$554
\$555	TEST	"."
	BF	\$557
	CLM	IELMNT
	BEF	
\$557	BT	\$555
	SET	
	BEF	
\$554	R	
IELMNT	CLM	IRPLIS
	BF	\$559
	B	\$558
\$559	LATCH	SLFTPT
	BF	\$562
	B	\$561
\$562	LATCH	ALFTPT
	BF	\$563
	B	\$561
\$563	LATCH	BLFTPT
	BF	\$561
\$561	BF	\$558
	OUT	GET
\$558	R	
IRPLIS	TEST	"("



	BF	\$566
	MARK	
	CLM	ILIST
	BEF	
	SAVE	
	CLM	ERPLIS
	BEF	
\$566	R	
ERPLIS	TEST	"D0"
	BF	\$568
	ID	
	BEF	
	CLM	UNDTST
	BEF	
	PUSHLB	
	OUT	LDA
	EVAL	
		"5"
		"0"
	STKSYM	
	MOVE	IOITER
		IOSW
	CLM	ITERPR
	BEF	
	MOVI	IOSW
		"0"
	POP	
	OUT	LDA
	EVAL	
		"5"
		"0"
	OUT	LD
	EVAL	
		"5"
		"0"
	RESTOR	
	LB1	

	ENTLOC	
	ENTL	
		"0"
		"6"
	TEST	" ) "
	BEF	
\$568	R	
OUTPUT	TEST	"PUT"
	BF	\$570
	TEST	"STRING"
	BF	\$572
	MARK	
	OUT	"0"
	SAVE	
	TEST	" ( "
	BEF	
	CLM	SLFTPT
	BEF	
	TEST	" ) "
	BEF	
	B	\$571
\$572	SET	
	BF	\$571
	MARK	
	OUT	"1"
	SAVL	
\$571	BEF	
	TEST	"EDIT"
	BEF	
	CANCEL	
	PUSHLB	
	OUT	FMT
	LB1	
	EVAL	
		"4"
		"1"
	RESTOR	

	TEST	"("		BF	\$580
	BEF			b	\$579
	CLM	OLIST	\$580	CLM	EXP
	BEF			BF	\$583
	TEST	")"		B	\$582
	BEF		\$583	CLM	SSE
	OUT	PUT		BF	\$584
	PUSHLB			B	\$582
	OUT	JUMP	\$584	LATCH	BVARBL
	LB1			BF	\$582
	EVAL		\$582	BF	\$579
		"4"		OUT	EDIT
		"1"	\$579	R	
	LB2		ORPLIS	TEST	"("
	ENTLOC			BF	\$587
	ENTL			MARK	
		"0"		CLM	OLIST
		"6"		BEF	
	CLM	FMTLIS		SAVE	
	BEF			CLM	ERPLIS
	LB1			BEF	
	ENTLOC		\$587	R	
	ENTL		FMTLIS	TEST	"("
		"0"		BF	\$589
		"6"		OUT	"c"
\$570	R			CLM	FMTITE
OLIST	CLM	OELMNT		BEF	
	BF	\$575	\$590	TEST	","
\$576	TEST	","		BF	\$592
	BF	\$578		CLM	FMTITE
	CLM	OELMNT	\$592	BEF	
	BEF			BT	\$590
\$578	BT	\$576		SET	
	SET			BEF	
	BEF			TEST	")"
\$575	R			BEF	
OELMNT	CLM	ORPLIS		OUT	"c"

\$589	R		CLM	SPEC	
FMTITE	CLM	CTRLFM	BF	\$607	
	BF	\$594	B	\$606	
	B	\$593	\$607	SET	
\$594	CLM	DATAFM	BF	\$606	
	BF	\$593	OUT	"0000"	
\$593	R		\$606	BEF	
CTRLFM	TEST	"X"		B	\$604
	BF	\$597	\$605	TEST	"E"
	OUT	"0"		BF	\$609
	CLM	SPEC		OUT	"9"
	BEF			CLM	SPEC
	B	\$596		BEF	
\$597	TEST	"PAGE"		B	\$604
	BF	\$598	\$609	TEST	"I"
	OUT	"1"		BF	\$610
	B	\$596		OUT	" "
\$598	TEST	"SKIP"		CLM	SPEC
	BF	\$599		BEF	
	OUT	"2"		B	\$604
	CLM	SPEC	\$610	TEST	"L"
	BF	\$601		BF	\$604
	B	\$600		OUT	"="
\$601	SET			CLM	SPEC
	BF	\$600		BEF	
	OUT	"0001"	\$604	R	
\$600	BEF		SPEC	TEST	"("
	B	\$596		BF	\$613
\$599	TEST	"COL"		INUM	
	BF	\$596		BEF	
	OUT	"3"		OUTSYM	
	CLM	SPEC		TEST	" ) "
	BEF			BEF	
\$596	R		\$613	R	
DATAFM	TEST	"A"			END
	BF	\$605			
	OUT	"8"			

```

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

```

```

00750*
00760*****
00770  START  EQU  *
00780          CAM  60          SET FOUR CHAR ADDRESSING FOR EXECUTE
00790          SW   STKEND-2     WORD MARK FOR MOVING AND TESTING
00800          MCW  STKSTR,IR2   INITIALIZE STACK POINTER
00810          SW   IR2-2       SHORTEN ARITHMETIC
00820          SI   IR2         ITEM MARK FOR RIGHT MOVE
00830          MCW  LODLOC,IR1   INITIALIZE INSTRUCTION COUNTER
00840          SW   IR1-2       SHORTEN INDEX ARITHMETIC
00850          SI   IR1         ACCOMMODATE RIGHT MOVE
00860          BS   IR3         ZERO PROGRAM COUNTER
00870          SW   IR3-2       SHORTEN ARITHMETIC
00880          SI   IR3         ACCOMMODATE RIGHT MOVE
00890          SI   IR15        ACCOMMODATE RIGHT MOVE
00900          MCW  GENLOC,IR5   INITIALIZE CODE GENERATION LOCATION
00910          SW   IR7-1       SHORTEN FETCH ARITHMETIC
00920          MCW  SYMSTR,NEWSYM INITIALIZE NEXT LOCATION IN SYMBOL TABLE
00930          BS   =1B4,SYMEND  INITIALIZE
00940          MCW  SYMEND,IR14
00950          LCA  =1C77,1+X14  BLOCK
00960          LCA  NEWSYM,4+X14 LIST
00970          MCW  SYMEND,CRBLKT INIT CURRENT BLOCK LIST POINTER
00980          MCW  NEWSYM,IR15  INIT SYMTAB POINTER
00990          MCW  SYMEND,SVSYME SAVE INITIAL SYMTAB END AS START OF BLOCK
01000*          LIST
01010          MCW  : : ,OUTPUT+132 CLEAR PRINT
01020          MCW  OUTPUT+132   BUFFER
01030          MCW  =1C21        CARRIAGE CONTROL
01040          SI   INPUT+80     RESTORE LOST ITEM MARK ON INPUT BUFFER
01050*          SKIP TO TOP OF PAGE AND INIT INPUT BUFFER
01060L          :SKIP PRINT,57,
01070L          :GET  READ,
01080L          :PUT  PRINT,INPUT-1,
01090          MCW  +INPUT,IR6
01100          B     FIRST
01110*****

```

```

01120*
01130*      INSTRUCTION FETCHING
01140*
01150*****
01160  FETCH3  BA      =1B3,IR1
01170  FETCH  BA      =1B1,INSTCT      INSTRUCTION COUNT
01180  FIRST  BS      IR7-1            CLEAR SECOND CHAR
01190          MRSD    0+X1,IR7        INSERT OP CODE
01200          SAR     IR1             BUMP SEQUENCE COUNTER
01210          BA      IR7            MULTIPLY
01220          BA      IR7            BY 4
01230          MCW     TVEC+3+X7,IR14
01240          B       0+X14
01250  TVEC   EQU     *
01260          REP     10
01270          DSA     ERROR
01280          DSA     MOVE
01290          DSA     ADD
01300          DSA     MULT
01310          DSA     SETF
01320          DSA     BEF
01330          DSA     EXITI
01340          DSA     RESOLV
01350          DSA     BRANCH
01360          DSA     BRNCHT
01370          DSA     BRNCHF
01380          DSA     BM
01390          DSA     SET
01400          DSA     SCANI
01410          DSA     ERROR
01420          DSA     MOVLIT
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	STKSYM
01610	DSA	CHKSYM
01620	DSA	SWAP
01630	DSA	ENTA
01640	DSA	ENTL
01650	REP	3
01660	DSA	ERROR
01670	DSA	OUT
01680	DSA	OUTSYM
01690	DSA	TEST
01700	DSA	ID
01710	DSA	ONUM
01720	DSA	STRTST
01730	DSA	EVAL
01740	DSA	ENTLOC
01750	DSA	ERROR
01760	DSA	INUM
01770	DSA	ENUM
01780	DSA	BLKENT
01790	DSA	BLKEXT
01800	DSA	ERROR
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  *
01940  *      THE SET AND SETF PRIMITIVES SET THE TRUE-FALSE INDICATOR.
01950  *
01960  *****
01970  SETF  MCW  :F:,TF
01980  B      FETCH
01990  SET   MCW  :T:,TF
02000  B      FETCH
02010  *****
02020  *
02030  *      THE CLM PRIMITIVE (CALL META PROCEDURE) STACKS THE RETURN ADDRESS
02040  *      AND ERROR LATCH CODES, RESETS THE ERROR LATCH AND SETS THE
02050  *      INSTRUCTION COUNTER TO THE BEGINNING OF THE CALLED PROCEDURE.
02060  *
02070  *****
02080  CLM  EQU  *
02090  MRID 0+X1,IR1-2
02100  SAR  4+X2
02110  MCW  :00:,1+X2
02120  MRSU ELATCH,5+X2
02130  MCW  :F:,ELATCH
02140  BA   =1B9,IR2
02150  B      STKOVF
02160  *****
02170  *
02180  *      THE RETURN PROCEDURE POPS THE CONTROL STACK UNTIL A RETURN ADDRESS
02190  *      IS FOUND WHICH IS SENT TO THE INSTRUCTION COUNTER. THE ERROR LATCH
02200  *      PREVIOUSLY STACKED WITH THE RETURN ADDRESS IS RESTORED.
02210  *
02220  *****

```

```

02230 RETURN BS      =1B9,IR2          SEARCH AND POP
02240             BCE DORET,0+X2,00    UNTIL RETURN
02250             B   RETURN          ADDRESS IS FOUND
02260 DORET MCW      4+X2,IR1          RETURN ADDRESS TO LOCATION COUNTER
02270             SI   IR1            RESTORE ITEM MARK
02280             MRSU 5+X2,ELATCH     RESTORE ERROR LATCH UPON RETURN
02290             B   CANCEL          RETURN CANCELS ANY BACKUP LATCH
02300*****
02310*
02320*     PUSHLB GENERATES A NEW INTERNAL LABEL AND PUSHES IT ON THE
02330*     CONTROL STACK.
02340*
02350*****
02360 PUSHLB A       :1:,LABEL
02370             MRIDI LABEL-4,0+X2
02380             BA    =1B9,IR2
02390             B     FETCH
02400             DCW   :6$:
02410 RLABEL DCW     :000:             ITEM MARK RIGHT
02420*****
02430*
02440*     POP POPS THE CONTROL STACK RESTORING THE VALUE TO SYMBOL IF THE
02450*     TOP OF THE STACK IS MARKED AS A SYMBOL.
02460*
02470*****
02480 POP          BS      =1B9,IR2
02490             BCE     POPSYM,0+X2,S    BRANCH IF STACK TOP IS SYMBOL
02500             B       FETCH
02510 POPSYM MCW      8+X2,SYMBOL+7      STACK SYMBOL TO SYMBOL AREA
02520             B       FETCH
02530*****
02540*
02550*     LBI SEARCHES THE CONTROL STACK TO FIND THE FIRST LABEL SYMBOL
02560*     WHICH IS THEN MOVED TO SYMBOL. THE STACK IS NOT AFFECTED.
02570*
02580*****
02590 LBI      LCA      IR2,IR14

```

```

02600 TESTLB BS      =1B9,IR14
02610 BCE          MOVLAB,0+X14,6
02620 B           TESTLB
02630 MOVLAB MCW    :      :;SYMBOL+7  CLEAR SYMBOL AREA
02640 MRID1 1+X14,SYMBOL
02650 B           FETCH
02660*****
02670*
02680* LB2 IS SIMILAR TO LB1 EXCEPT THAT THE SECOND LABEL IN THE
02690* CONTROL STACK IS RETRIEVED.
02700*
02710*****
02720 LB2          LCA      IR2,IR14
02730 TESTLB1 BS   =1B9,IR14
02740 BCE          TESTLB2,0+X14,6
02750 B           TESTLB1
02760 TESTLB2 BS   =1B9,IR14
02770 BCE          MOVLAB,0+X14,6
02780 E           TESTLB2
02790*****
02800*
02810* THE OUT PRIMITIVE CAUSES THE LITERAL VALUE FOLLOWING THE OP
02820* CODE TO BE OUTPUT DIRECTLY INTO THE CODE STRING.
02830*
02840*****
02850 OUT          MRID1 0+X1,0+X5
02860 SBR          IR5
02870 MRIN 0+X1,0+X3
02880 SBR          IR3
02890 SAR          IR1
02900 B           FETCH
02910*****
02920*
02930* OUTSYM PLACES THE CONTENTS OF SYMBOL INTO THE OUTPUT CODE STRING.
02940*
02950*****
02960 OUTSYM CM      0+X5
      CLEAR ANY POSSIBLE WORD MARK

```

```

02970 MRID1 SYMBOL,0+X5
02980 SBR IR5
02990 MRIN SYMBOL,0+X3
03000 SBR IR3
03010 B FETCH
03020*****
03030*
03040* TEST EXAMINES THE INPUT STRING FOR THE SPECIFIED LITERAL STRING
03050* FOLLOWING THE OP CODE SETTING THE TRUE-FALSE INDICATOR APPROPRIATLY
03060*
03070*****
03080 TEST E NEXT
03090 SW 0+X1
03100 MRIN 0+X1,0+X6
03110 SAR IR1
03120 SBR IR10
03130 C 0-1+X10,0-1+X1
03140 BE TEST1
03150 MCW :F:,TF
03160 B FETCH
03170 MCW :T:,TF
03180 LCA IR10,IR6
03190 B FETCH
0320C*****
0321C*
0322C* ID TESTS THE INPUT STRING FOR A VALID IDENTIFIER SETTING THE
0323C* TRUE-FALSE INDICATOR AND MOVING THE IDENTIFIER TO SYMBOL IF FOUND.
0324C*
0325C*****
0326C ID E NEXT
0327C MCW : :
0328C LCA =3B0,IR13
0329C LCA IR6,IR14
0330C MRSD 0+X14,IR13
0331C RR TSTAN,IDTAB+X13
0332C MCW :F:,TF
0333C B FETCH

```

TEST FOR ALPHA CHAR

:SYMBOL+7 CLEAR SYMBOL AREA

SCAN FOR ITEM MARK

```

03340 TSTAN BA =1B1,IR14
03350 MRSD 0+X14,IR13
03360 BI TSTAN,IDTAB+X13 TEST ALPHANUMERIC
03370 SI 0-1+X14 SAVE ID
03380 MRID 0+X6,SYMBOL IN
03390 SAR IR6 SYMBOL
03400 CI 0-1+X6 AREA
03410 SI SYMBOL+7
03420 MCW :T:,TF
03430 B FETCH
03440 DCW :0:
03450 IDTAB EQU *
03460 REP 10
03470 L DC : : 0-9
03480 DC : :
03490 REP 9
03500 L DCW : : A-I
03510 DC : :
03520 REP 9
03530 L DCW : : J-K
03540 DC : :
03550 REP 8
03560 L DCW : : S-Z
03570 DC : :
03580*****
03590*
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 BS SYMBOL+32 CLEAR TO ZEROES
03670 MCW :F:,TF
03680 LCA =3B0,IR13
03690 BS IR14
03700 ONMTST MRSD 0+X6,IR14

```

```

03710      B10      CONUM,IDTAB+X14      BCE      FETCH,TF,F      SI      SYMBOL-1+X13      CW      IRI4      B      FETCH      CONUM      BA      IRI4      BA      IRI4      BA      IRI4      BA      IRI4      SW      IRI4      HA      IRI4,SYMBOL+X13      MRSD      IRI4,SYMBOL+X13      SAR      IRI4      HA      IRI4,SYMBOL+X13      CI      SYMBOL+X13      BA      =IR1,IR13      MCW      :T:,TF      B      UNMIST      03870      03860      03850      03840      03830      03820      03810      03800      03790      03780      03770      03760      03750      03740      03730      03720      03710
CONUM      BCE      FETCH,TF,F      SI      SYMBOL-1+X13      CW      IRI4      B      FETCH      CONUM      BA      IRI4      BA      IRI4      BA      IRI4      BA      IRI4      SW      IRI4      HA      IRI4,SYMBOL+X13      MRSD      IRI4,SYMBOL+X13      SAR      IRI4      HA      IRI4,SYMBOL+X13      CI      SYMBOL+X13      BA      =IR1,IR13      MCW      :T:,TF      B      UNMIST      03870      03860      03850      03840      03830      03820      03810      03800      03790      03780      03770      03760      03750      03740      03730      03720      03710
STRIST      B      NEXT      BCE      STRYS,0+X6,"      MCW      :F:,TF      B      FETCH      STRYS      BA      =IR1,IR6      LCA      IRI6,IR13      BCE      STRDN,0+X13,"      BA      =IR1,IR13      B      STRNXT      SI      0-1+X13      MRIDI      0+X6,SYMBOL      SAR      IRI6      BA      =IR1,IR6      04070      04060      04050      04040      04030      04020      04010      04000      03990      03980      03970      03960      03950      03940
STRIST      B      NEXT      BCE      STRYS,0+X6,"      MCW      :F:,TF      B      FETCH      STRYS      BA      =IR1,IR6      LCA      IRI6,IR13      BCE      STRDN,0+X13,"      BA      =IR1,IR13      B      STRNXT      SI      0-1+X13      MRIDI      0+X6,SYMBOL      SAR      IRI6      BA      =IR1,IR6      04070      04060      04050      04040      04030      04020      04010      04000      03990      03980      03970      03960      03950      03940
*****
03930*      IF TRUE.
03920*      SETTING THE TRUE-FALSE INDICATOR AND MOVING THE STRING TO SYMBOL
03910*      STRIST EXAMINES THE INPUT STRING FOR A CHARACTER STRING LITERAL
03900*      03890*      03880*****
*****

```

MOVE STRING TO SYMBOL

```

04080      CI      0-1+X13
04090      MCW     :T:,TF
04100      B       FETCH
04110*****
04120*
04130*      THE EVAL PRIMITIVE CAUSES A SEARCH OF THE SYMBOL TABLE AND
04140*      THEN OUTPUTING OF A SYMBOL TABLE VALUE TO THE CODE STREAM
04150*      ACCORDING TO TWO ONE CHARACTER LITERAL PARAMETERS:
04160*      1)  THE LENGTH OF THE FIELD
04170*      2)  THE RELATIVE POSITION WITHIN THE TABLE ENTRY
04180*
04190*****
04200  EVAL  MCW     :0:,SRCHTP      SET SEARCH MODE
04210      B       SEARCH            GO SEARCH
04220      BS      IR13
04230      MRSD    0+X1,IR13        INSERT LENGTH
04240      SAR     IR1
04250      BS      IR14
04260      MRSD    0+X1,IR14        INSERT POSITION
04270      SAR     IR1
04280      BA      IR15,IR14        FIND ABSOLUTE POSITION
04290      BA      IR14,IR13        FIND FIELD END
04300      SI      0-1+X13
04310      MRILR   0+X14,0+X5      OUTPUT CODE
04320      SBR     IR5              NEXT OUTPUT LOCATION
04330      MRIN    0+X14,0+X3
04340      SBR     IR3              PROGRAM COUNTER
04350      CI      0-1+X13
04360      B       FETCH
04370*****
04380*
04390*      THE ENTLOC PRIMITIVE CAUSES A FULL SEARCH OF THE SYMBOL TABLE
04400*      AND THEN CAUSES THE PROGRAM COUNTER TO BE ENTERED AS A VALUE
04410*      FOR THE ADDRESS OF THE SPECIFIED SYMBOL.  LEVEL 0 (STATIC)
04420*      IS ASSIGNED FOR THE STORAGE LEVEL.
04430*
04440*****

```

```

04450  ENTLOC MCW    :0:,SRCHTP      SEARCH MODE
04460      B        SEARCH
04470      MRID     IR3-2,ADDR-2+X15  MOVE IN ADDRESS
04480      SW       ADDR-2+X15        MARK RELOCATABLE
04490      MCW      :0:,LEVEL+X15     STATIC STORAGE INDICATOR
04500      B        FETCH
04510*****
04520*
04530*  ENTL AND ENTA ARE PRIMITIVES FOR INTERING LITERAL AND ADDRESSED *
04540*  VALUES, RESPECTIVELY, INTO THE SYMBOL TABLE.  INDEX REGISTER 15 *
04550*  MUST POINT TO THE PROPER SYMBOL TABLE ENTRY PRIOR TO EXECUTION. *
04560*  A SIX BIT LITERAL NUMBER FOLLOWS EACH OP CODE SPECIFYING THE *
04570*  RELATIVE POSITION WITHIN THE TABLE ENTRY TO BE ALTERED. *
04580*
04590*****
04600  ENTA  EQU    *
04610  ENTL  EQU    *
04620      BS      IR14              COMPUTE LEFTMOST
04630      MRSD    0+X1,IR14        ADDRESS
04640      SAR      IR1              OF
04650      BA       IR15,IR14        RECEIVING FIELD
04660      CW       0+X14            CLEAR POSSIBLE WORD MARK
04670      BCE      ENTLIT,0-2+X1,56 TEST FOR LITERAL ENTRY
04680      MRID     0+X1,IR13-2      ADDRESS TO IR13
04690      SAR      IR1              UPDATE INSTRUCTION COUNTER
04700      CW       0+X13            AVOID INADVERTENT RELOCATION MARKER
04710      MRIDw    0+X13,0+X14      ENTER, CLEAR ANY WORD MARKS
04720      B        FETCH
04730  ENTLIT MRIDw  0+X1,0+X14      ENTER LITERAL
04740      SAR      IR1              UPDATE INSTRUCTION COUNTER
04750      B        FETCH
04760*****
04770*
04780*  SEARCH IS A SUBROUTINE FOR SEARCHING A BLOCK STRUCTURED SYMBOL *
04790*  TABLE.  NOTE THE SEARCH TYPE PARAMETER (SRCHTP) WHICH MAY *
04800*  BE USED TO CONTROL THE SEARCH MODE.  SEARCH MAY BE CALLED *
04810*  BY PRIMITIVES EVAL, ENTLOC, SRCHP, OR BLKEXT. *

```



04820\*  
 04830\*  
 \*\*\*\*\*

04840	SYMCNT	DCW	=380
04850	SRCHTP	DCW	=1
04860	NEWSYM	DCW	=3
04870	SVSYME	DCW	=4
04880*			
04890*			
04900	NAME	DCW	19
04910	ETYP	DCW	11
04920	CHAIN	DCW	10
04930	DIMCNT	DCW	7
04940	LENGTH	DCW	6
04950	ADDR	DCW	4
04960	LEVEL	DCW	1
04970	DTYP	DCW	0
04980	SEARCH	DCW	SRCHRT+4
04990		DCW	BA
05000		DCW	=1B1,SRCHCT
05010	SRCHBL	DCW	CRBLK1,IR14
05020	SRCHC	DCW	0+4+X14,IR15
05030		DCW	BA
05040		DCW	=1B1,CMPCNT
05050		DCW	NAME+X15,SYMBOL+7
05060		DCW	BE
05070		DCW	CHAIN+X15,IR15
05080		DCW	B
05090	FNDSYM	DCW	SRCHC
05100	SRCHRT	DCW	* :1:,TF
05110	NFND	DCW	DOENT,SRCHTP,01
05120		DCW	BCE
05130	DOENT	DCW	BA
05140		DCW	=1B20,NEWSYM
05150		DCW	BA
05160		DCW	=1B1,SYMCNT
05170		DCW	C
05180		DCW	SYMEND,NEWSYM
		DCW	OVFLW
		DCW	NEWSYM,CHAIN+X15
		DCW	NEWSYM,IR15

SYMBOL TABLE ENTRY COUNT  
 SPACE FOR SEARCH MODE TYPE CODE  
 POINTER TO NEXT AVAILABLE SLOT IN SYMTAB  
 SPACE TO SAVE ORIGINAL SYMBOL TABLE END  
 FIELD WHICH EFFECTIVELY POINTS TO THE  
 START OF THE BLOCK LIST  
 SYMBOLIC  
 NAMES  
 FOR  
 SYMBOL  
 TABLE  
 STRUCTURE  
 SAVE RETURN TO CALLER  
 TALLY SEARCH COUNT  
 SET STARTING POINT FOR CURRENT BLOCK  
 START OF BLOCK  
 TALLY COMPARISON COUNT  
 TEST END OF BLOCK  
 NEXT SYMBOL TABLE ENTRY  
 TEST SINGLE BLOCK SEARCH  
 IF FULL SEARCH GET NEW BLOCK  
 COUNT THE NUMBER OF ENTRIES  
 CHECK FOR  
 TABLE OVERFLOW  
 FILL IN CHAIN ADDRESS FOR LAST ENTRY

```

05190 MLWD* SYMBOL+7,NAME+X15 SET NAME
05200 MCW IR15,ADDR+X15 UNDEFINED VALUE POINTER TO THIS ENTRY
05210 MCW :00:,LEVEL+X15 ZERO LEVEL AND DATA TYPE
05220 SW ADDR-2+X15 MARK RELOCATABLE (DEFAULT)
05230 HA =IB32,ADDR-2+X15 MARK UNRESOLVED
05240 MCW :F:,TF
05250 E (SRCHRT+1)
05260 NEWBLK BCE DONT,1+X14,77 CHECK FOR LAST BLOCK
05270 BS IR13 CLEAR
05280 MRSD 1+X14,IR13 INSERT SURROUNDING BLOCK NUMBER
05290 BIM =4B4,IR13 MULTIPLY BY BLOCK LIST ENTRY SIZE
05300 MCW SVSYME,IR14 START OF BLOCK LIST
05310 BS IR13,IR14 COMPUTE LOC OF BLOCK LIST ENTRY
05320 E SRCHBL GO SEARCH BLOCK
05330*****
05340*
05350* SRCHP IS A SYMBOL TABLE SEARCH PRIMITIVE WHICH CALLS UPON
05360* SUBROUTINE SEARCH. NOTE THE ONE CHARACTER PARAMETER FOLLOWING
05370* THE OP CODE WHICH CONTROLS THE SEARCH MODE.
05380*
05390*****
0540C SRCHP MRSD 0+X1,SRCHTP SET SEARCH MODE
0541C SAR IR1
0542C B SEARCH
0543C B FETCH
05440*****
05450*
05460* 1STBA AND 1STBL ARE PRIMITIVES FOR TESTING ADDRESSED AND
05470* LITERAL VALUES, RESPECTIVELY, AGAINST SYMBOL TABLE VALUES.
05480* A SIX BIT LITERAL NUMBER FOLLOWS EACH OP CODE SPECIFYING THE
05490* RELATIVE POSITION WITHIN THE TABLE ELEMENT TO BE TESTED.
05500* INDEX REGISTER IS MUST POINT TO THE PROPER SYMBOL TABLE ENTRY
05510* PRIOR TO EXECUTION.
05520*
05530*****
05540 1STBA EQU *
05550 1STBL EQU *

```

```

05560 BS IR14
05570 MRSD 0+X1,IR14
05580 SAR IR1
05590 BA IR15,IR14
05600 BCE TSTLIT,0-2+X1,34
05610 MRID 0+X1,IR13-2
05620 SAR IR1
05630 SW 0+X13
05640 MRIN 0+X13,0+X14
05650 SAR IR13
05660 SBR IR14
05670 C 0-1+X14,0-1+X13
05680 BE SET
05690 B SETF
05700 SW 0+X1
05710 MRIN 0+X1,0+X14
05720 SAR IR1
05730 SBR IR14
05740 C 0-1+X14,0-1+X1
05750 BE SET
05760 B SETF
05770 *****
05780*
05790* THE COMPARE PRIMITIVE COMPARES THE 2ND ADDRESSED OPERAND TO
05800* THE FIRST. THE SIX BIT CHARACTER FOLLOWING THE OPERAND ADDRESSES
05810* IS USED AS THE VARIANT OF THE CONDITIONAL BRANCH INSTRUCTION
05820* FOLLOWING THE COMPARISON.
05830*
05840*****
05850 COMP 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 RA =IR1,IR1
05920 SW 0+X14 WORD MARK TO STOP COMPARE

```

COMPUTE LEFTMOST ADDRESS  
 OF  
 TABLE FIELD  
 TO BE TESTED  
 TEST FOR LITERAL TEST  
 ADDRESS TO IR13  
 UPDATE INSTRUCTION COUNTER  
 WORD MARK TO STOP COMPARE  
 POSITION TO RIGHT END  
 SET  
 INDEX REGISTERS

WORD MARK TO STOP COMPARE  
 POSITION TO RIGHT END  
 SET  
 INDEX REGISTERS

TSTLIT SW 0+X1  
 MRIN 0+X1,0+X14  
 SAR IR1  
 SBR IR14  
 C 0-1+X14,0-1+X1  
 BE SET  
 B SETF

\*\*\*\*\*  
 THE COMPARE PRIMITIVE COMPARES THE 2ND ADDRESSED OPERAND TO  
 THE FIRST. THE SIX BIT CHARACTER FOLLOWING THE OPERAND ADDRESSES  
 IS USED AS THE VARIANT OF THE CONDITIONAL BRANCH INSTRUCTION  
 FOLLOWING THE COMPARISON.

\*\*\*\*\*  
 COMP EQU \*  
 MRID 0+X1,IR13-2 FIRST ADDRESS TO IR13  
 SAR IR1  
 MRID 0+X1,IR14-2 SECOND ADDRESS TO IR14  
 SAR IR1  
 SST 0+X1,COMPT,07 INSERT CONDITIONAL BRANCH CODE  
 RA =IR1,IR1  
 SW 0+X14 WORD MARK TO STOP COMPARE

```

05930      MRIN  0+X13,0+X14      POSITION A AND B ADDRESS REGISTERS
05940      SAR   IR13
05950      SBR   IR14
05960      C     0-1+X13,0-1+X14
05970      COMPT BCT  SET,40      SET TF
05980      B     SETF      INDICATOR
05990      *****
06000      *
06010      *   LATCH SETS POINTERS AND INDICATORS AND THEN INVOKES CLM. THE
06020      *   EFFECT IS THAT AN APPARENT ERROR IN THE CALLED PROCEDURE CAUSES
06030      *   A RETURN TO THE CALLING PROCEDURE.
06040      *
06050      *****
06060      LATCH  B     NEXT      AVOID CARD BOUNDARY PROBLEMS
06070      MCW    IR6,SAVIN      SAVE INPUT POINTER
06080      MCW    IR5            SAVE OUTPUT POINTER
06090      MCW    :T:,BACKUP     SET BACKUP INDICATOR
06100      B      CLM            CALL THE SPECIFIED ROUTINE
06110      SAVOUT DCW  =3
06120      SAVIN  DCW  =3
06130      BACKUP DCW  :F:      BACKUP INITIALLY TURNED OFF
06140      *****
06150      *
06160      *   CANCEL TURNS OFF ANY BACK UP LATCH.
06170      *
06180      *****
06190      CANCEL MCW  :F:,BACKUP  TURN OFF BACKUP SWITCH
06200      B      FETCH      NEXT INSTRUCTION
06210      *****
06220      *
06230      *   SCAN1 SCANS THE INPUT STRING FOR THE SPECIFIED LITERAL STRING
06240      *   FOLLOWING THE OP CODE.
06250      *
06260      *****
06270      SCAN1  B     NEXT
06280      SW      0+X1
06290      MRIN  0+X1,0+X6      SCAN FOR ITEM MARK

```

```

06300          SAR    IR1
06310          SBR    IR6
06320  SCNCMP  C      0-1+X6,0-1+X1
06330          BE     SCANT
06340          BI     NEXT,0+X6
06350          BA     =1B1,IR6
06360          B      SCNCMP
06370  SCANT   MCW    :T:,TF
06380          B      FETCH
06390)*****
06400)*
06410)*      MOVLIT MOVES THE LITERAL CHARACTER STRING FOLLOWING THE ADDRESS
06420)*      (WHICH FOLLOWS THE OP CODE) TO THE ADDRESSED LOCATION.
06430)*
06440)*****
06450  MOVLIT  MRID   0+X1,IR13-2      ADDRESS OF RECEIVING CHAR FIFD
06460          SAR    IR14
06470          MRID   0+X14,0+X13      MOVE LITERAL DATA
06480          SAR    IR1              UPDATE LOCATION COUNTER
06490          SW     0+X14
06500          B      FETCH
06510)*****
06520)*
06530)*      INUM CALLS INM FOR AN ATTEMPTED RECOGNITION OF AN INTEGER NUMBER.
06540)*
06550)*****
06560  INUM    B      INM
06570          B      FETCH
06580)*****
06590)*
06600)*      SUBROUTINE INMTESTS THE INPUT STRING FOR AN INTEGER NUMBER SETTING
06610)*      TRUE-FALSE CODE AND CONVERTINGTHE NUMBER TO BINARY IN SYMBOL
06620)*      IF TRUE.
06630)*
06640)*****
06650  INM     SBR     INMRTN+4
06660          B      NEXT

```

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	INMTST	
06750	ISGN	DCW	=1	
06760	STISGN	MRSD	0+X6,ISGN	SAVE SIGN
06770		SAR	IR6	UPDATE INPUT POINTER
06780	INMTST	MRSD	0+X6,IR14	
06790		BCE	NOINT,0+X6,,	TEST FOR POSSIBLE FLTNG PT NM
06800		BIO	MVNUM,IDTAB+X14	TEST FOR INTEGER DIGIT
06810		BCE	NOINT,TF,F	HAVE WE FOUND AN INTEGER
06820		B	CONVRT	YES, GO CONVERT TO BINARY
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	
06870		SAR	IR6	
06880		BA	=1B1,IR13	
06890		BCE	INMERR,IR13,10	TEST TOO MANY DIGITS
06900		MCW	:T:,TF	WE HAVE PART OF AN INTEGER
06910		B	INMTST	GO LOOK FOR MORE
06920	CONVRT	BS	CVBFLD	CLEAR HOLD FIELD
06930		MCW	SYMBOL-1+X13,CVBFLD	MOVE IN DECIMAL INTEGER
06940		SST	ISGN,CVBFLD,60	SET SIGN IN CONVERSION FIELD
06950		DTB	CVBFLD,00	BINARY MANTISSA IN FR 0
06960		TAM	CVBFLD,00	STORE IT
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	

```

07040 INMRTN B      *
07050 CVBFLD DCW    =11
07060 INMERR B      ERMGRP
07070          MCW   =20AW: INTEGER TOO LARGE,OUTPUT+39
07080          B      ERRPRT
07090*****
07100*
07110*      ERASE ERASES THE SPECIFIED NUMBER OF CHARACTERS FROM THE CODE STRING.*
07120*
07130*****
07140 ERASE  BS      IR13
07150          MRSD  SYMBOL,IR13      MOVE ERASE COUNT TO INDEX REG
07160          BS      IR13,IR3      ADJUST PROGRAM COUNTER
07170          BS      IR13,IR5      AND OUTPUT POINTERS
07180          MCW     IR5,IR14
07190 ERTST  BCE     FETCH,IR13,00    TEST ERASE LOOP FINISHED
07200          MRSDR  IR13-1,0+X14    ERASE A CHARACTER
07210          SBR     IR14          NEXT CHARACTER TO ERASE
07220          BS      =1B1,IR13      DECREMENT LOOP COUNT
07230          B      ERTST
07240*****
07250*
07260*      BEF AND BM COMPRISE THE ERROR MESSAGING PRIMITIVES.  OBSERVE
07270*      THE SPECIAL ACTION IF THE BACK UP OR ERROR LATCHES ARE SET.  NOTE
07280*      ALSO THE DIFFERENCE BETWEEN A WARNING MESSAGE AND A FATAL MESSAGE
07290*      WITH PRIMITIVE BM.  NOTE THAT BACKING UP OVER A CARD BOUNDARY IS
07300*      NOT PROVIDED.
07310*
07320*****
07330 BEF      EQU      *
07340          BCE     FETCH,TF,T      IF TRUE CONTINUE
07350          BCE     DEFMES,BCKUP,F  IF NO BACKUP BYPASS BACKUP MECHANISM
07360          MCW     SAVIN,IR6      RESTORE INPUT POINTER
07370          BS      SAVOUT,IR5     COMPUTE ERASE
07380          MCW     IR5,IR13        COUNT
07390          BS      IR5,IR3        ADJUST PROGRAM COUNTER
07400          MCW     SAVOUT,IR5     RESTORE OUTPUT POINTER

```

07410		MCW	IR5,IR14	
07420		MCW	+ERSRTN,ERTST+4	SET UP RETURN FROM CODE ERASURE
07430		B	ERTST	ERASE CODE
07440	ERSRTN	MCW	+FETCH,ERTST+4	RESTORE INSTRUCTION IN ERASE ROUTINE
07450		B	RETURN	BACKUP CANCELS AND RETURNS
07460	DEFMES	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	0+X1,OUTPUT+20	
07520		SAR	IR1	
07530	ERRPRT	EQU	*	
07540		MCW	:****ERROR****:,OUTPUT+18	
07550	L	:PUT	PRINT,OUTPUT,	
07560		MCW	: : ,OUTPUT+132	CLEAR
07570		MCW	OUTPUT+132	PRINT LINE
07580		MCW	=1C21,OUTPUT	CARRIAGE CONTROL
07590		SI	INPUT+80	RESTORE LOST ITEM MARK ON INPUT BUFFER
07600		B	FETCH	
07610	BM	EQU	*	
07620		BCE	BEF,BCKUP,T	TEST FOR BACKUP ACTION
07630		BCE	FERR,0+X1,F	IF FATAL CONTINUE TESTING
07640		B	ERMES	ELSE PRINT MESSAGE AND CONTINUE
07650	ERPASS	MRIN	0+X1,0	SCAN BY ERROR MESSAGE
07660		SAR	IR1	
07670		B	ERRRTN	
07680	FERR	BCE	ERPASS,OUTPUT+20,F	TEST PREVIOUS FATAL MESSAGE PENDING
07690		B	ERMPRP	NO, SO SET UP
07700		MRID	0+X1,OUTPUT+20	MOVE IN MESSAGE
07710		SAR	IR1	
07720	ERRHD	MCW	:F:,CMPLCD	SET FATAL COMPLETION CODE
07730	ERRRTN	BCE	RETURN,ELATCH,F	IF NO LATCH RETURN
07740		B	ERRPRT	ELSE PRINT AND CONTINUE
07750	ERMPRP	SBR	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  EXITI  EQU     ERRFLG
07870  OVFLW  EQU     *
07880L     :PUT    PRINT,OVMES,
07890  ERRFLG MCW     :F:,CMPLCD      FATAL ERROR FLAG
07900      B       EXIT
07910  OVMESDCW :1SYMBOL TABLE OVERFLOW, JOB ABORTED:
07920L     DCW     =1C45      RECORD MARK
07930*****
07940*
07950*      ERROR IS EXECUTED IF AN ATTEMPT IS MADE TO INTERPRET AN INVALID
07960*      OP CCDE.  THE JOB IS ABORTED.
07970*
07980*****
07990  ERROR  EQU     *
08000L     :PUT    PRINT,OPCDMS,
08010      H
08020      B       EXIT
08030  OPCDMSDCW :1INVALID OP CODE, JOB ABORTED:
08040L     DCW     =1C45      RECORD MARK
08050*****
08060*
08070*      THE ENUM PRIMITIVE EXAMINES THE INPUT STRING FOR A FLOATING POINT
08080*      NUMBER SETTING THE TRUE-FALSE INDICATOR AND CONVERTING THE NUMBER
08090*      TO BINARY IN SYMBOL IF TRUE.
08100*
08110*****
08120  ENUM    MCW     +DHOLD,IR12      SET  POINTER TO DECIMAL HOLD FIELD
08130      BS      SCALE      CLEAR SCALE EXPONENT FIELD
08140      B       NEXT

```

08150		B	DNUM	CALL DECIMAL NUMBER RECOGNIZER
08160		BCE	DMVE,TF,T	IF DNUM.TRUE THEN GO SAVE INPUT
08170		BBE	FETCH,0+X6,60	AVOID POSSIBLE LOGICAL CONSTANT
08180		BCE	FRACT,0+X6,.	DETERMINE FRACTION FOLLOWING
08190		B	FETCH	IF NONE THEN FETCH NEXT INSTRUCTION
08200	DMVE	BS	DHOLD+31	CLEAR HOLD AREA
08210		MRID	SYMBOL,DHOLD	MOVE DECIMAL CHARACTERS
08220		SBR	IR12	SAVE POINTER FOR MOVING FRACTION
08230		BCE	FRACT,0+X6,.	DETERMINE FRACTION FOLLOWING
08240		B	DECMVE	IF NONE MOVE DECIMAL FIELD FOR CONVERSION
08250	FRACT	BA	=1B1,IR6	UPDATE INPUT POINTER
08260		BI	NEXT,0+X6	IF RECORD END GO GET MORE
08270		B	DNUM	LOOK FOR DECIMAL FRACTION
08280		BCE	DECMVE,TF,F	IF NONE MOVE DECIMAL FIELD FOR CONVERSION
08290		MRID	SYMBOL,0+X12	CONCATENATE FRACTION WITH INTEGER PART
08300		SBR	IR14	COMPUTE
08310		BS	IR12,IR14	SCALE EXPONENT ADJUSTMENT
08320		BS	IR14,SCALE	ADJUST IT
08330		BA	IR14,IR12	RESTORE X12
08340	DECMVE	BS	SYMBOL+10	CLEAR SYMBOL AREA
08350		C	+DHOLD+12,IR12	TEST IF FIELD TOO LONG
08360		BL	LNGOK	
08370		B	ERMPPR	WARNING MESSAGE
08380		MCW	:SIGNIFICANT DIGITS LOST:,OUTPUT+107	
08390L		:PUT	PRINT,OUTPUT,	
08400		B	EXPTST	
08410	LNGOK	MCW	0-1+X12,SYMBOL+10	10 SET UP DECIMAL FIELD
08420		MCW	SYMBOL+10,DHOLD+10	FOR DECIMAL TO BINARY CONVERSION
08430*				LOOK FOR EXPONENT
08440	EXPTST	B	NEXT	
08450		BCE	ESGN,0+X6,E	TEST FOR EXPONENT
08460		B	FCNVRT	NO EXPONENT, GO CONVERT
08470	ESGN	BA	=1B1,IR6	
08480		B	NEXT	
08490		BCE	SETSM,0+X6,-	TEST MINUS SIGN
08500		BCE	SETSP,0+X6,+	TEST PLUS SIGN
08510		MCW	=1C34,ESGNOP	SET ADDITION OP CODE

08520		B	PLUSOP	NO SIGN, TREAT AS PLUS
08530	SETSM	MCW	=1C35,ESGNOP	SET SUBTRACTION OP CODE
08540	UPDATE	BA	=1B1,IR6	UPDATE INPUT POINTER
08550	PLUSOP	BI	NEXT,0+X6	IF END OF RECORD GO GET MORE
08560		B	INM	LOOK FOR DECIMAL EXPONENT
08570		BCE	BEF,TF,F	IF INVALID THEN SIGNAL ERROR
08580	ESGNOP	BA	SYMBOL+3,SCALE	ADJUST SCALE FACTOR EXPONENT
08590				BINARY FORM OF EXPONENT
08600		B	FCNVRT	GO CONVERT
08610	SETSP	MCW	=1C34,ESGNOP	SET ADDITION OP CODE
08620		B	UPDATE	
08630	DNUM	SBR	DNMRTN+4	SAVE RETURN ADDRESS
08640		BS	IR14	CLEAR INDEX
08650		BS	IR13	AND COUNTER
08660		MCW	:F:,TF	INITIAL TF SWITCH
08670	DNMTST	MRSI	0+X6,IR14	INPUT CHARACTER TO INDEX REGISTER
08680		BIO	MVDEC,IDTAB+X14	TEST FOR NUMERIC CHARACTER
08690		BCE	(DNMRTN+1),TF,F	IF NO NUMERICS RETURN
08700		SI	SYMBOL-1+X13	MARK RIGHT END OF NUMERIC FIELD
08710	DNMRTN	B	*	RETURN TO CALLER
08720	MVDEC	MRSI	0+X6,SYMBOL+X13	MOVE NUMERIC CHAR TO SYMBOL FIELD
08730		SAR	IR6	UPDATE INPUT POINTER
08740		BI	NEXT,0+X6	TEST END OF RECORD
08750		BA	=1B1,IR13	UPDATE CHARACTER COUNT
08760		MCW	:T:,TF	SET TF FLAG TRUE
08770		B	DNMTST	LOOK FOR MORE
08780	FCNVRT	DTB	DHOLD+10,00	CONVERT DECIMAL FIELD TO BINARY IN FRO
08790		AAA	70	NORMALIZE IT
08800		MCW	:P:,EXPSGN	SET EXPONENT SIGN FLAG
08810		BBE	SETN,SCALE-2,40	TEST NEGATIVE EXPONENT FOR SCALE FACTOR
08820		E	TSTSCL	
08830	SETN	BS	IR14	CONVERT NEGATIVE
08840		BS	SCALE,IR14	SCALE EXPONENT
08850		MCW	IR14,SCALE	TO POSITIVE
08860		MCW	:M:,EXPSGN	SET EXPONENT SIGN FLAG
08870	TSTSCL	C	SCALE,=3B600	TEST SCALE EXPONENT FOR VALID RANGE
08880		BH	SCLOK	

08890		B	ERMPRP	SET UP
08900		MCW	:EXPONENT OUT OF RANGE:,OUTPUT+25	ERROR MESSAGE
08910		MCW	:F:,CMPLCD	FATAL COMPLETION CODE
08920		B	ERRPRT	PRINT IT
08930	SCLOK	C	SCALE,=3B0	IF EXPONENT ZERO CONVERSION FINISHED
08940		BE	FCVEND	
08950		BS	IR14	CLEAR INDEX TO ZERO
08960		SST	SCALE,IR14,17	INSERT LOW 4 BITS FOR INDEXING
08970		EIM	=4B8,IR14	LEFT 3 BITS TO INDEX CVTTAB
08980		TMA	CVTTAB+X14,01	CONVERSION FACTOR TO FR1
08990		ES	FHOLD+7	CLEAR EIGHT CHAR FLOATING POINT FIELD
09000		MCW	SCALE,FHOLD+5	3 CHAR FIELD TO 8 CHAR FLTNG PT FIELD
09010		TMA	FHOLD+7,03	LOAD IT TO FR3
09020		BMS	31,04	SHIFT RIGHT 4 BITS
09030		TAM	FHOLD+7,30	STORE IT
09040		BS	IR14	CLEAR
09050		SST	FHOLD+5,IR14,17	INSERT LOW 4 BITS TO INDEX
09060		BIM	=4B8,IR14	SHIFT LEFT 3 BITS TO INDEX CVTTAB
09070		TMA	CTAB16+X14,02	CONVERSION FACTOR TO FR2
09080	FINTM	MAA	21	COMPUTE INTERMEDIATE FACTOR
09090		TLA	02	SAVE LOW ORDER FOR DOUBLE PRECISION
09100		BMS	31,04	NEXT 4 BITS
09110		TAM	FHOLD+7,30	STORE IT
09120		BS	IR14	CLEAR
09130		SST	FHOLD+5,IR14,03	MAX 10 BITS FOR SCALE EXPONENT
09140		BIM	=4B8,IR14	SHIFT FOR INDEX
09150		TMA	CTB256+X14,03	CONVERSION FACTOR TO FR3
09160	FLOW	MAA	32	LOW ORDER FACTOR
09170		MAA	31	HIGH ORDER FACTOR
09180		TLA	03	SAVE LOW ORDER
09190		AAA	32	ACCUMULATE LOW ORDER FACTORS
09200		BCE	FDIV,EXPSGN,M	TEXT EXPONENT SIGN
09210		MAA	01	MULTIPLY BY HIGH ORDER SCALE FACTOR
09220		TLA	03	SAVE LOW ORDER
09230		MAA	02	LOW ORDER SCALE FACTOR
09240		AAA	32	ACCUMULATE LOW ORDER FACTORS
09250		AAA	21	ADD TO UNROUNDED RESULT

09260		TLA	03
09270		AAA	33
09280		AAA	31
09290		TAA	10
09300		B	FCVEND
09310	FDIV	AAA	21
09320		TLA	03
09330		AAA	33
09340		AAA	31
09350		DAA	10
09360		TLA	03
09370		DAA	13
09380		AAA	33
09390		AAA	30
09400	FCVEND	TAM	FHOLD+7,00
09410		MRIDI	FHOLD,SYMBOL
09420		MCW	:T:,TF
09430		B	FETCH
09440	SCALE	DCW	=3
09450	DHOLD	DCW	=32
09460	R FHOLD	DCW	=8
09470	EXPSGN	DCW	=1
09480	CVTTAB	DCW	F1E0
09490		DCW	F1E1
09500		DCW	F1E2
09510		DCW	F1E3
09520		DCW	F1E4
09530		DCW	F1E5
09540		DCW	F1E6
09550		DCW	F1E7
09560		DCW	F1E8
09570		DCW	F1E9
09580		DCW	F1E10
09590		DCW	F1E11
09600		DCW	F1E12
09610		DCW	F1E13
09620		DCW	F1E14

ROUND

IT  
 PUT IT IN FRO  
 CONVERSION DONE  
 ADJUST DIVIDEND BY ACCUMULATED LOW ORDER

ROUND

IT  
 DIVIDE CONVERTED NUM BY SCALE FACTOR  
 SAVE REMAINDER  
 DIVIDE  
 AND  
 ROUND QUOTIENT  
 STORE CONVERTED NUMBER  
 SYMBOL FIELD FOR OUTPUT  
 SET TF FLAG  
 NEXT INSTRUCTION  
 SCALE EXPONENT FIELD  
 DECIMAL CHAR HOLD FIELD  
 FLOATING POINT HOLD FIELD

CONVERSION TABLE

```

09630          DCW    F1E15
09640  CTAB16  DCW    F1E0
09650          DCW    F1E16
09660          DCW    F1E32
09670          DCW    F1E48
09680          DCW    F1E64
09690          DCW    F1E80
09700          DCW    F1E96
09710          DCW    F1E112
09720          DCW    F1E128
09730          DCW    F1E144
09740          DCW    F1E160
09750          DCW    F1E176
09760          DCW    F1E192
09770          DCW    F1E208
09780          DCW    F1E224
09790          DCW    F1E240
09800  CTB256  DCW    F1E0
09810          DCW    F1E256
09820          DCW    F1E512
09830*****
09840*
09850*   THE MARK PRIMITIVE PUSHES THE ADDRESS OF THE NEXT OUTPUT STRING   *
09860*   LOCATION ON THE CONTROL STACK FOR LATER USE BY THE SAVE PRIMITIVE. *
09870*
09880*****
09890  MARK    BA      =1B9,IR2          BUMP STACK POINTER
09900          MCW     IR5,0-5+X2        SAVE OUTPUT CODE ADDRESS
09910          MCW     :M:,0-9+X2        MARK STACK ELEMENT TYPE
09920          B       STKOVF            CHECK STACK OVERFLOW
09930*****
09940*
09950*   SAVE PUSHES THE CODE GENERATED SINCE THE LAST MARK OPERATION   *
09960*   ON THE VARIABLE LENGTH CODE STACK AND RESETS THE OUTPUT LOCATION *
09970*   BACK TO THE MARKED LOCATION.                                     *
09980*
09990*****

```

1000C	SAVE	BCE	DOSAVE,0-9+X2,M	TREAT AS NO-OP IF STACK TOP
1001C		B	FETCH	IS NOT A CODE MARK
1002C	DOSAVE	MCW	0-5+X2,IR14	MARKED ADDRESS TO IR14
1003C		MCW	STKEND,IR13	STACK END VALUE TO IR13
1004C		BS	=1B1,IR5	ADJUST CODE POINTER TO LAST OUTPUT CHAR
1005C	SVTEST	C	IR14,IR5	TEST MOVE NOTE POSSIBLE NULL
1006C		BL	SVEXIT	COMPLETION SAVE OPERATION
1007C		MLSDR	0+X5,0+X13	MOVE AND
1008C		SAR	IR5	ADJUST
1009C		SBR	IR13	POINTERS
1010C		CW	1+X5	CLEAR ANY POSSIBLE WORD MARK
1011C		BS	=1B1,IR3	
1012C		B	SVTEST	
1013C	SVEXIT	LCA	STKEND,0+X13	NEXT ELEMENT POINTER
1014C		LCA	=1C77,0-3+X13	MARKER FOR NO-OP TEST
1015C		SBR	STKEND	NEW SYSTEM STACK END
1016C		MCW	IR14,IR5	RESTORE OUTPUT MARKER
1017C		BS	=1B9,IR2	POP CODE MARKER OFF STACK
1018C		B	STKOVF	CHECK STACK OVERFLOW
1019C	STKOVF	C	IR2,STKEND	TEST STACK
1020C		BH	FETCH	OVERFLOW
10210L		:PUT	PRINT,STKMES,	
1022C		H		HALT INTERRUPT, DUMP REQUEST
1023C	STKMESDCW	:	1 STACK OVERFLOW:	
1024C	L	DCW	=1C45	RECORD MARK
1025C	*****			
1026C	*			
1027C	* THE STKSYM PRIMITIVE STACKS THE CURRENT SYMBOL IN SYMBOL ON THE *			
1028C	* CONTROL STACK. *			
1029C	*			
1030C	*****			
1031C	STKSYM	MCW	SYMBOL+7,8+X2	MOVE SYMBOL TO STACK
1032C		MCW	:S:,0+X2	MARK STACK ELEMENT AS SYMBOL
1033C		BA	=1B9,IR2	BUMP STACK POINTER
1034C		B	STKOVF	CHECK FOR STACK OVERFLOW
1035C	*****			
1036C	*			

```

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              STCK
10440 DORES  MCW      4+X14,STKEND        RESTORE PREVIOUS STK END POINTER
10450         MCW      4+X14,IR13         ALSO TO IR13 FOR LOOP TEST
10460         BA       =1B5,IR14         POINT TO CODE TO MOVE
10470 RESTST C        IR13,IR14          TEST MOVE              NOTE POSSIBLE
10480         BH       FETCH              COMPLETION            NULL RESTORE
10490         MRSDB 0+X14,0+X5          MOVE CODE
10500         SAR     IR14              AND ADJUST
10510         SBR     IR5              POINTERS
10520         BA       =1B1,IR3
10530         B       RESTST
10540*****
10550*
10560*   THE CHKSYM PRIMITIVE TESTS THE EIGHT CHARACTER FIELD ON TOP OF THE   *
10570*   CONTROL STACK AGAINST SYMBOL SETTING THE TRUE-FALSE CODE.          *
10580*                                                                       *
10590*****
10600 CHKSYM BS       =1B9,IR2          ADJUST STACK POINTER
10610         C        8+X2,SYMBOL+7    TEST AGAINST SYMBOL VALUE
10620         BE       SET              IF EQUAL SET TRUE
10630         B        SETF             ELSE SET FALSE
10640*****
10650*
10660*   SWAP SWAPS THE TOP TWO ELEMENTS ON THE CONTROL STACK.                *
10670*                                                                       *
10680*****
10690 SWAP  MCW      0-1+X2,SWPTMP      MOVE TOP TO TEMPORARY
10700         SI      0-18+X2          ITEM MARK FOR NEXT MOVE
10710         MLIDI 0-10+X2,0-1+X2      SWAP
10720         MCW      SWPTMP,0-10+X2
10730         B       FETCH

```



```

10740 SWPTMP DCW    =9
10750*****
10760*
10770*   THE ADD AND MULT PRIMITIVES COMPRISE THE BINARY ARITHMETIC
10780*   CAPABILITIES (TWO ADDRESS) OF THE METAX9 PSEUDO-MACHINE.
10790*
10800*****
10810 ADD      B      GTPRA      GO GET OPERAND ADDRESSES
10820          SW      0+X11      WORD MARK TO STOP ADDITION
10830          SW      0+X10
10840          MRIN    0+X10,0     FIND RIGHT POSITION
10850          SAR     IR10        SET INDEX
10860          MRIN    0+X11,0
10870          SAR     IR11        REGISTERS
10880          BA      0-1+X10,0-1+X11 BINARY ADD
10890          B      FETCH
10900 MULT     B      GTPRA      GO GET OPERAND ADDRESSES
10910          SW      0+X10      WORD MARK TO STOP MOVE
10920          MRIN    0+X10,0     FIND
10930          SAR     IR10        RIGHT END
10940          BS      MFLD1       CLEAR RECEIVING FIELD
10950          MCW     0-1+X10,MFLD1 MOVE MULTIPLIER
10960          SW      0+X11      WORD MARK TO STOP MOVE
10970          MRIN    0+X11,0     FIND
10980          SAR     IR11        RIGHT END
10990          BS      MFLD2       CLEAR RECEIVING FIELD
11000          MCW     0-1+X11,MFLD2 MOVE MULTIPLICAND
11010          BIM     MFLD1,MFLD2 BINARY MULTIPLY REQUIRES 24 BITS
11020          MCW     MFLD2,0-1+X11 PUT BACK RESULT
11030          B      FETCH
11040 MFLD1    DCW     =4B0
11050 MFLD2    DCW     =4B0
11060 GTPRA    SBR     GTRTN      SET RETURN
11070          MRID    0+X1,IR10-2 FIRST OPERAND ADDRESS
11080          SAR     IR1         UPDATE LOCATION COUNTER
11090          MRID    0+X1,IR11-2 SECOND OPERAND ADDRESS
11100          SAR     IR1         UPDATE LOCATION COUNTER

```

```

11110   GTRTN B      *                      RETURN TO CALLER
11120*****
11130*                                                    *
11140*   THE MOVE PRIMITIVE MOVES THE FIRST ADDRESSED FIELD TO THE SECOND.  *
11150*                                                    *
11160*****
11170  MOVE      B      GTOPRA              GET ADDRESSES IN IR10 AND IR11
11180          MRIDI 0+X10,0+X11          MOVE DATA AND TERMINATING ITEM MARK
11190          B      FETCH
11200*****
11210*                                                    *
11220*   THE BLOCK ENTRY PRIMITIVE CONSTRUCTS A NEW TABLE ENTRY IN THE      *
11230*   BLOCK LIST AND MAINTAINS BLOCK COUNTERS AND POINTERS.                *
11240*                                                    *
11250*****
11260  BLKCNT DCW    :0:                     BLOCK COUNTER
11270  PRVBLK DCW    =1C00                 PREVIOUS BLOCK NUMBER
11280  CRBLKT DCW    =3                     INITIAL SYMTAB SEARCH ENTRY
11290  BLKENT BA     =1B1,BLKCNT           BLOCK COUNT
11300          BS     =1B4,SYMEND         REDUCE SYMBOL TABLE END LOCATION
11310*   TO ACCOMMODATE NEW BLOCK LIST ENTRY
11320          MCW     SYMEND,IR13
11330          LCA     PRVBLK,1+X13        SET SURROUNDING BLOCK NUMBER
11340          MCW     BLKCNT,PRVBLK       SET UP FOR NEXT BLOCK ENTRY
11350          BA      =1B20,NEWSYM        SPACE FOR DUMMY ENTRY
11360          LCA     NEWSYM,4+X13        STORE IN BLOCK LIST
11370          MCW     SYMEND,CRBLKT       SET FOR CURRENT SYMTAB SEARCH
11380          B      FETCH
11390*****
11400*                                                    *
11410*   THE BLOCK EXIT PRIMITIVE RESTORES CRBLKT POINTER AND PRVPLK          *
11420*   NUMBER FOR THE SURROUNDING BLOCK.  THE SYMBOL TABLE ENTRIES FOR      *
11430*   THE TERMINATING BLOCK ARE SCANNED FOR UNRESOLVED SYMBOLS.              *
11440*   UNRESOLVED ENTRIES ARE ADDED TO THE SURROUNDING BLOCK IF NOT          *
11450*   FOUND IN THAT PORTION OF THE TABLE.  APPROPRIATE LINKING               *
11460*   PARAMETERS ARE SET FOR THE RESOLVE PRIMITIVE TO USE.  THUS            *
11470*   DIABOLOGICAL LABEL REFERENCES IN A BLOCK STRUCTURE ARE RESOLVABLE.    *

```

```

11480*
11490*****
11500  BLKSAV DCW    =3
11510  BLKPRM DCW    =1
11520  BLKEXT MCW    CRBLKT,BLKSAV    SAVE FOR UNRESOLVED SEARCH
11530      MRSD    0+X1,BLKPRM        BLOCK EXIT PARAMETER
11540      SAR      IR1
11550      BS       IR13                CLEAR
11560      MCW      CRBLKT,IR14
11570      MCW      1+X14,IR13          INSERT PREVIOUS BLOCK NUMBER
11580      MCW      1+X14,PRVBK         NEW SURROUNDING BLOCK NUMBER
11590      BIM      =4B4,IR13          COMPUTE
11600      MCW      SVSYME,IR14        BLOCK LIST
11610      BS       IR13,IR14          LOCATION FOR SURROUNDING BLOCK
11620      MCW      IR14,CRBLKT        BLOCK LIST POINTER
11630      MCW      BLKSAV,IR12        TERMINATING BLOCK LIST POINTER
11640      MCW      4+X12,IR12         SYMBOL TABLE POINTER
11650      BCE      FETCH,BLKPRM,01    TEST NO LABEL LINK UP
11660  CHKUND C      CHAIN+X12,:000:    CHECK FOR END OF
11670      BE       FETCH              BLOCK TABLE ENTRIES
11680      MCW      CHAIN+X12,IR12      NOTE 1ST TIME JUMP OVER DUMMY ENTRY
11690  UDCHK BCE     CHKPRV,DTYPE+X12,00 CHECK FOR UNDEFINED SYMBOLS
11700      B       CHKUND              LOOK FOR MORE
11710  CHKPRV MCW    NAME+X12,SYMBOL+7  SET NAME TO USE SEARCH SUBROUTINE
11720      MCW      :1:,SRCHTP         BLOCK ONLY SEARCH MODE
11730      B       SEARCH
11740      BCE      ADDSYM,TF,F         IF FALSE SYMBOL ADDED TO SURROUNDING
11750*      BLOCK. GO SET MARKERS
11760      BCE      PRVUN,0+X15,00     IF FOUND BUT STILL UNDEFINED SET MARKERS
11770      SI       0+X15
11780      MLIDW    DIMCNT+X15,DIMCNT+X12 FOUND, SET VALUES FOR RESOLVE
11790      CI       0+X15
11800      B       CHKUND
11810  ADDSYM EQU     *
11820  PRVUN SW       IR15-2
11830      MCW      IR12,IR9           SAVE CURRENT ENTRY POINTER
11840      MCW      CHAIN+X12,IR12     SAVE NEXT ENTRY POINTER

```

```

1185C      MCW   IR15,CHAIN+X9      SET CHAIN FIELD
1186C      MCW   =1C77,ETYPE+X9    LET RESOLVE KNOW ABOUT IT
1187C      CW    IR15-2
1188C      C     CHAIN+X12,:000:
1189C      BE    FETCH
1190C      B     UDCHK
1191C*****
1192C*
1193C*      RESOLVE IS A TERMINAL PRIMITIVE WHICH RESOLVES FORWARD
1194C*      REFERENCES AND DETECTS ANY UNDEFINED ADDRESSES.  THE OBJECT
1195C*      TEXT IS SCANNED FOR WORD MARKS TO FIND RELOCATABLE ADDRESSES.
1196C*      THE LEFTMOST BIT OF THE ADDRESS MARKS UNRESOLVED ADDRESSES.
1197C*
1198C*****
1199C  RESOLV BCE  EXIT,CMPLCD,F      EXIT IF FATAL COMPILATION TO THIS POINT
1200C      BS    CVBFLD              CLEAR
1201C      MCW   IR3,CVBFLD-2        MOVE PROGRAM SIZE
1202C      TMA    CVBFLD,00           LOAD TO FRO
1203C      BTD    CVBFLD,00           CONVERT TO DECIMAL AND STORE
1204C      LCA    EWORD,PSIZE         EDIT CONTROL WORD
1205C      MCE    CVBFLD,PSIZE        MOVE AND EDIT
1206C      CW     PSIZE-8             CLEAR WORD MARK
1207C      BS    CVBFLD              CLEAR
1208C      MCW   INSTCT,CVBFLD-2      MOVE INSTRUCTION COUNT
1209C      TMA    CVBFLD,00           LOAD TO FRO
1210C      BTD    CVBFLD,00           CONVERT TO DECIMAL AND STORF
1211C      LCA    EWORD,ICOUNT        EDIT CONTROL WORD
1212C      MCE    CVBFLD,ICOUNT       MOVE AND EDIT
1213C      CW     ICOUNT-8            CLEAR WORD MARK
1214C      :PUT   PRINT,EXITMS,       PRINT IT
1215C      BS    CVBFLD
1216C      MCW   SRCHCT,CVBFLD-2      MOVE SEARCH COUNT
1217C      TMA    CVBFLD,00
1218C      BTD    CVBFLD,00
1219C      LCA    EWORD,SCOUNT
1220C      MCE    CVBFLD,SCOUNT
1221C      BS    CVBFLD

```

12220		MCW	CMPCNT,CVBFLD-2	MOVE COMPARISON COUNT
12230		TMA	CVBFLD,00	
12240		BTD	CVBFLD,00	
12250		LCA	EWORD,TCOUNT	
12260		MCE	CVBFLD,TCOUNT	
12270L		:PUT	PRINT,TABMES,	
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		MCE	CVBFLD,ECOUNT	MOVE AND EDIT
12340L		:PUT	PRINT,TABCNT,	
12350		LCA	GENLOC,IR15	START OF COMPILED CODE
12360	SCAN	MRWN	0+X15,0	SCAN FOR WORD MARK
12370		SAR	IR15	SAVE NEXT POSITION
12380		C	IR15,IR5	DETERMINE
12390		BL	PLIST	COMPLETION
12400		BBE	GETADD,0-1+X15,40	TEST UNRESOLVED ADDRESS
12410		B	SCAN	
12420	GETADD	MRIDI	0-1+X15,IR14-2	UNRESOLVED ADDRESS IS POINTER TO SYMBOL
12430*				TABLE. MOVE IT TO IR14.
12440		HA	=1C40,IR14-2	REMOVE UNRESOLVED MARKER
12450		C	SYMSTR,IR14	TEST FOR
12460		EL	STERR	VALID
12470		C	SYMEND,IR14	SYMBOL
12480		BH	STERR	ADDRESS
12490	CHNTST	BCE	CHNADD,ETYPE+X14,77	
12500		BBE	NTDEFN,ADDR-2+X14,40	IF STILL UNDEFINED PRINT ERROR
12510		B	ADDRSL	
12520	CHNADD	MCW	CHAIN+X14,IR14	CHAIN TO SURROUNDING BLOCK
12530		B	CHNTST	GO TEST FOR ADDITIONAL CHAINING
12540	ADDRSL	SI	ADDR+X14	
12550		MRIDR	LEVEL+X14,0-2+X15	SET LEVEL AND DISPLACEMENT
12560		BCE	LBVTST,0-3+X15,76	TEST POSSIBLE LABEL VARIABLE - LABEL
12570*				CONSTANT RESOLUTION
12580		B	SCAN	

12590	LBVTST	BCE	LBV,DTYPE+X14,05	TEST LABEL VARIABLE IN SYMTAB
12600		B	SCAN	ELSE CONTINUE
12610	LBV	BNP	SCAN,0-4+X15	MAKE SURE OP CODE PRECEDES ADDR
12620		MRSD	:5:,0-3+X15	CHANGE DATA TYPE TO LABEL VARIABLE
12630		B	SCAN	AND CONTINUE
12640	STERR	EQU	*	
12650	L	:	PUT PRINT,CMPLMS,	
12660		H		
12670		B	EXIT	
12680	CMPLMS	SDCW	:ACOMPILER ERROR DISCOVERED DURING RESOLVE:	
12690	L	DCW	=1C45	
12700	NTDEFN	MCW	NAME+X14,PSYM	
12710	L	:	PUT PRINT,NDFMES,	
12720		MCW	:F:,CMPLCD	SET COMPLETION CODE
12730		B	SCAN	
12740	NDFMES	SDCW	:1UNDEFINED SYMBOL :	
12750	PSYM	DCW	:	:
12760	L	DCW	=1C45	
12770	TABCNT	DCW	:A ****SYMBOL TABLE ENTRY COUNT =:	
12780	ECOUNT	DC	=9	
12790		DC	:****:	
12800	L	DCW	=1C45	
12810	EXITMS	SDCW	: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	TABMES	SDCW	:B ****SYMTAB SEARCH COUNT = :	
12900	SCOUNT	DC	=9	
12910		DC	:; SYMTAB COMPARE COUNT = :	
12920	TCOUNT	DC	=9	
12930		DC	:****:	
12940	L	DCW	=1C45	
12950	WORD	DCW	: , ,0 :	

```

12960*****
12970*
12980*   EXIT IS A TERMINAL POINT IN THE PROGRAM, CLEARING CERTAIN
12990*   PUNCTUATION BEFORE EXITING.
13000*
13010*****
13020  EXIT    EQU    *
13030          CI     IR1
13040          CI     IR2
13050          CI     IR3
13060          CI     IR15
13070          CW     IR7-1
13080          B      (164)
13090*****
13100*
13110*   NEXT IS A SUBROUTINE WHICH SCANS THE INPUT STRING FOR THE NEXT
13120*   NON-BLANK CHARACTER READING NEW RECORD(S) IF REQUIRED.  IF AN
13130*   END OF FILE IS SENSED A MESSAGE IS PRINTED AND THE PROGRAM EXITS.
13140*
13150*****
13160  NEXT    SBR     NXTRTN+4
13170  ENDTST  BI      GETCRD,0+X6
13180  BLKTST  BCE     NBLNK,0+X6,15
13190  NXTRTN  B       *
13200  NBLNK   BA      =1B1,IR6
13210          B      ENDTST
13220  GETCRD  EQU     *
13230L          :GET   READ,
13240          MCW     =1C21,INPUT-1      CARRIAGE CONTROL
13250L          :PUT   PRINT,INPUT-1,
13260          MCW     +INPUT,IR6
13270          C      INPUT+3,:1EOF:      END OF FILE TEST
13280          BNE     BLKTST
13290L          :PUT   PRINT,E0FMES,
13300          B      ERRFLG
13310  E0FMES  DCW     :1UNEXPECTED END OF FILE, JOB ABORTED:
13320 L      DCW     =1C45      RECORD MARK

```

```

1333()*****
13340*
13350*    PLIST IS EXECUTED IF A POST LISTING IS REQUESTED.
13360*
13370()*****
13380  PLIST  BCE    EXIT,PSTLST,N      EXIT IF POST LIST NOT REQUESTED
13390L      :SKIP PRINT,57,
13400      LCA    =3B0,IR13            PROGRAM COUNTER
13410      LCA    GENLOC,IR15          START OF GENERATED CODE
13420      SW     IR15-2
13430  NEWLIN MCW   : : ,OUTPUT+132     CLEAR
13440      MCW    OUTPUT+132            PRINT LINE
13450      MCW    =1C21,OUTPUT          CARRIAGE CONTROL
13460      SI     INPUT+80              RESTORE LOST ITEM MARK ON INPUT BUFFER
13470      BS     CVBFLD                CLEAR CONVERSION FIELD
13480      MCW    IR13,CVBFLD-2
13490      TMA    CVBFLD,00             LOAD TO FRO
13500      BTD    CVBFLD,00             CONVERT TO DECIMAL
13510      SST    :0: ,CVBFLD,60        REMOVE SIGN BITS
13520      SW     CVBFLD-4
13530      MCW    CVBFLD,OUTPUT+5
13540      B      SYMADD                 DETERMINE SYMBOLIC LABEL, IF ANY
13550  LITOPR BA    =1B1,IR15           JUMP OVER LITERAL DESIGNATOR
13560      BA     =1B1,IR13
13570  LITRL  MRID  0+X15,OUTPUT+24    SET UP
13580      SBR    IR12
13590      MRIN   0+X15,0+X13
13600      SAR    IR15
13610      SBR    IR13
13620      MCW    :": ,0+X12            LITERAL OPERAND
13630      MCW    :": ,OUTPUT+23        FOR PRINTING
13640  PLSTPR EQU   *
13650L      :PUT  PRINT,
13660      B      NEWLIN
13670  OPCODE C     IR15,IR5           TEST
13680      BEL    PEXIT                 COMPLETION
13690      MCW    =1C40,SYMADD          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		BIM	=4B6,IR12	MULT BY TABLE ENTRY SIZE
13780		MCW	OPTAB+4+X12,OUTPUT+21	OP CODE TO PRINT
13790		BCE	PLSTPR,OPTAB+5+X12,00	TEST FOR NO OPERANDS
13800		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,	
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

14070		BIM	=4B10,IR12	MULT BY SIZE OF BOUND PAIR CODE
14080		BA	=1B3,IR12	SIZE OF LENGHT 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		MCW	:";,OUTPUT+15	
14130		MRRD	0+X15,OUTPUT+16	MOVE IT
14140		SAR	IR15	
14150		SBR	IR11	
14160		CW	0-1+X12	
14170		MCW	:";,0+X11	
14180		B	PLSTPR	GO PRINT
14190	DPVMES	DCW	:A ***DOPE VECTOR CODE***:	
14200	L	DCW	=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,OTYPE,07	SAVE TYPE
14250		B	ADDFV	GET OPERAND SYMBOL
14260		BCE	LNGCDE,OTYPE,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		LENGTH CODE
14310		SAR	IR15	
14320		MCW	:";,OUTPUT+35	
14330		BA	=1B2,IR13	
14340		B	PLSTPR	
14350	OTYPE	DCW	:0:	
14360	ADDR4	B	ADDFR	
14370		BCE	IOTYPE,0-5+X15,50	CHECK I/O SETUP
14380		B	PLSTPR	
14390	IOTYPE	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	AFVRTN+4	
14630		MRSD	0+X15,ADD5CN-4	
14640		EXM		GET ALL FIVE CHARACTERS
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,AD5CN1-4	MOVE DATA, NO WORD MARKS
14730		C	ADD5CN,AD5CN1	TEST EQUALITY
14740		BE	ADFVFD	
14750		BA	=1B20,IR14	
14760		C	IR14,NEWSYM	TEST
14770		BEH	A5COMP	
14780		BS	=1B5,IR15	
14790		BS	=1B5,IR13	
14800		B	MISSAD	

14810	ADVFVD	MCW	NAME+X14,OUTPUT+30	
14820	AFVRTN	B	*	
14830	ADD5CN	DCW	=5	
14840	AD5CN1	DCW	=5	
14850	SYMADD	B	OPCODE	FIRST TIME ONLY
14860		MCW	SYMSTR,IR14	TABLE START
14870	RELOCT	BNP	NOSYM,2+X14	IGNORE NON-RELOCATABLE SYMBOLS
14880		C	4+X14,IR13	TEST
14890		BE	SYMFND	EQUALITY
14900	NOSYM	BA	=1B20,IR14	NEXT
14910		C	SYMEND,IR14	TEST
14920		BH	OPCODE	TABLE END
14930		B	RELOCT	TEST NEXT
14940	SYMFND	MCW	NAME+X14,OUTPUT+15	
14950		B	OPCODE	
14960	PEXIT	MCW	:END	PROGRAM:,OUTPUT+32
14970L		:PUT	PRINT,	
14980		B	EXIT	
14990	TLITRL	SST	0+X15,LITCHR,70	LEFT THREE BITS OF TYPE CHAR
15000		BCE	LITOPR,LITCHR,70	TEST FOR LITERAL
15010		B	ADDR5	MUST BE FIVE CHAR ADDR
15020	LITCHR	DCW	:0:	
15030	FMTCDE	EQU	*	
15040L		:PUT	PRINT,FMTMES,	MESSAGE
15050		MCW	:"::,OUTPUT+15	BUILD
15060		MCW	+OUTPUT+16,IR12	FORMAT
15070	NFMTCH	MRSD	0+X15,0+X12	LITERAL
15080		SAR	IR15	
15090		SBR	IR12	
15100		BA	=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***:	

15180	L	DCW	=1C45	
15190	-OPTAB	EQU	*	OP CODE OPERAND TABLE FOR POSTLISTING
15200		REP	8	
15210		DCW	:ERROR0:	
15220		DCW	:DYNAM4:	OP CODE 10
15230		DCW	:STKTP8:	OP CODE 11
15240		DCW	:ALLOC4:	OP CODE 12
15250		REP	5	
15260		DCW	:ERROR0:	
15270		DCW	:LDA 4:	OP CODE 20
15280		DCW	:LD 4:	OP CODE 21
15290		DCW	:STO 0:	OP CODE 22
15300		DCW	:SST 0:	OP CODE 23
15310		DCW	:FLAG 0:	
15320		DCW	:ENTPR1:	OP CODE 25
15330		DCW	:RETRN0:	OP CODE 26
15340		DCW	:JUMPA0:	OP CODE 27
15350		DCW	:JUMP 4:	OP CODE 30
15360		DCW	:JUMPT4:	OP CODE 31
15370		DCW	:JUMPF4:	OP CODE 32
15380		DCW	:STCKC1:	OP CODE 33
15390		DCW	:COMPC1:	OP CODE 34
15400		DCW	:SWAP 0:	OP CODE 35
15410		DCW	:POPUP0:	OP CODE 36
15420		DCW	:ERROR0:	
15430		DCW	:ADD 0:	OP CODE 40
15440		DCW	:MULT 0:	OP CODE 41
15450		DCW	:SUB 0:	OP CODE 42
15460		DCW	:DIV 0:	OP CODE 43
15470		DCW	:NEG 0:	OP CODE 44
15480		REP	3	
15490		DCW	:ERROR0:	
15500		DCW	:FMT 4:	OP CODE 50
15510		DCW	:GET 0:	OP CODE 51
15520		DCW	:PUT 0:	OP CODE 52
15530		DCW	:EDIT 0:	OP CODE 53
15540		REP	4	

15550	DCW	:ERROR0:	
15560	DCW	:OR 0:	OP CODE 60
15570	DCW	:AND 0:	OP CODE 61
15580	DCW	:NOT 0:	OP CODE 62
15590	REP	3	
15600	DCW	:ERROR0:	
15610	DCW	:INDXR1:	OP CODE 66
15620	DCW	:INDXA1:	OP CODE 67
15630	DCW	:CAT 0:	OP CODE 70
15640	DCW	:SBSTR0:	OP CODE 71
15650	REP	4	
15660	DCW	:ERROR0:	
15670	DCW	:STOP 0:	OP CODE 76
15680	DCW	:ERROR0:	LAST
15690	LITORG*		
15700	END	START	

# SYMBOL DEFINITION - CARD REFERENCE INDEX

A4COMP	14520;	A5COMP	14710;	AD5CN1	14840;	ADD4CN	14610;	ADD5CN	14830;
ADD	10810;	ADDFR	14470;	ADDFV	14620;	ADDR4	14360;	ADDR5	14450;
ADDR	04950;	ADDRSL	12540;	ADDSYM	11810;	ADFRFD	14590;	ADFVFD	14810;
AFRRTN	14600;	AFVRTN	14820;	ALLOC	13930;	BCKUP	06130;	BEF	07330;
BLKCNT	11260;	BLKENT	11290;	BLKEXT	11520;	BLKPRM	11510;	BLKSAV	11500;
BLKTST	13180;	BM	07610;	BRANCH	01860;	BRNCHF	01900;	BRNCHT	01880;
CANCEL	06190;	CHAIN	04920;	CHKPRV	11710;	CHKSYM	10600;	CHKUND	11660;
CHNADD	12520;	CHNTST	12490;	CLM	02080;	CMPCNT	12880;	CMPLCD	00420;
CMPLMS	12680;	COMP	05850;	COMPT	05970;	CONUM	03760;	CONVRT	06920;
CRBLKT	11280;	CTAB16	09640;	CTB256	09800;	CVBFLD	07050;	CVTTAB	09480;
DECMVE	08340;	DEFMES	07460;	DHOLD	09450;	DIMCNT	04930;	DMVE	08200;
DNMRTN	08710;	DNMTST	08670;	DNUM	08630;	DOENT	05130;	DORES	10440;
DORET	02260;	DOSAVE	10020;	DPVMES	14190;	DTYPE	04970;	ECOUNT	12780;
ELATCH	00400;	ENDTST	13170;	ENTA	04600;	ENTL	04610;	ENTLIT	04730;
ENTLOC	04450;	ENUM	08120;	EOFMES	13310;	ERASE	07140;	ERMES	07500;

ERMPRP	07750;	ERPASS	07650;	ERRFLG	07890;	ERRHD	07720;	ERROR	07990;
ERRPRT	07530;	ERRRTN	07730;	ERSRTN	07440;	ERTST	07190;	ESGN	08470;
ESGNOP	08580;	ETYPE	04910;	EVAL	04200;	EWORD	12950;	EXIT	13020;
EXITI	07860;	EXITMS	12810;	EXPSGN	09470;	EXPTST	08440;	FCNVRT	08780;
FCVEND	09400;	FDIV	09310;	FERR	07680;	FETCH3	01160;	FETCH	01170;
FHOLD	09460;	FINTM	09080;	FIRST	01180;	FLOW	09160;	FMTCD	15030;
FMTDNE	15130;	FMTMES	15170;	FNDSYM	05090;	FRACT	08250;	GENLOC	00450;
GETADD	12420;	GETCRD	13220;	GTOPRA	11060;	GTRTN	11110;	ICOUNT	12840;
ID	03260;	IDTAB	03450;	INM	06650;	INMERR	07060;	INMOK	07010;
INMRTN	07040;	INMTST	06780;	INPUT	00310;	INSTCT	00440;	INUM	06560;
IOTYPE	14390;	IR10	00650;	IR11	00660;	IR12	00670;	IR13	00680;
IR14	00690;	IR15	00700;	IR1	00560;	IR2	00570;	IR3	00580;
IR4	00590;	IR5	00600;	IR6	00610;	IR7	00620;	IR8	00630;
IR9	00640;	ISGN	06750;	LABEL	02410;	LATCH	06060;	LB1	02590;
LB2	02720;	LBV	12610;	LBVTST	12590;	LDA	14210;	LENGTH	04940;
LEVEL	04960;	LITCHR	15020;	LITERL	13810;	LITOPR	13550;	LITRL	13570;
LNGCDE	14280;	LNGOK	08410;	LODLOC	00460;	MARK	09890;	MFLD1	11040;
MFLD2	11050;	MISSAD	14570;	MOVE	11170;	MOVLAB	02630;	MOVLIT	06450;
MULT	10900;	MVDEC	08720;	MVNUM	06860;	NAME	04900;	NBLNK	13200;
NDFMES	12740;	NEWBLK	05260;	NEWLIN	13430;	NEWSYM	04860;	NEXT	13160;
NFMTCH	15070;	NFND	05110;	NOINT	06830;	NOSYM	14900;	NTDEFN	12700;
NXTRTN	13190;	ONMTST	03700;	ONUM	03650;	OPCDMS	08030;	OPCODE	13670;
OPTAB	15190;	OTYPE	14350;	OUT	02850;	OUTPUT	00320;	OUTSYM	02960;
OVFLW	07870;	OVFMES	07910;	PEXIT	14960;	PLIST	13380;	PLSTPR	13640;
PLUSOP	08550;	POP	02480;	POPSYM	02510;	PRINT	00330;	PRPRTN	07850;
PRVBLK	11270;	PRVUN	11820;	PSIZE	12820;	PSTLST	00430;	PSYM	12750;
PUSHLB	02360;	READ	00300;	RELOCT	14870;	RESOLV	11990;	RESTOR	10410;
RESTST	10470;	RETURN	02230;	*RDWR	00290;	*SKP	00340;	SAVE	10000;
SAVIN	06120;	SAVOUT	06110;	SCALE	09440;	SCAN	12360;	SCANI	06270;
SCANT	06370;	SCLOK	08930;	SCNCMP	06320;	SCOUNT	12900;	SEARCH	04980;
SET	01990;	SETF	01970;	SETN	08830;	SETSM	08530;	SETSP	08610;
SRCHBL	05010;	SRCHC	05020;	SRCHCT	12870;	SRCHP	05400;	SRCHRT	05100;
SRCHTP	04850;	START	00770;	STERR	12640;	STISGN	06760;	STKEND	00500;
STKMES	10230;	STKOVF	10190;	STKSTR	00490;	STKSYM	10310;	STRDN	04040;
STRNXT	04010;	STRTST	03950;	STRYS	03990;	SVEXIT	10130;	SVSYME	04870;
SVTEST	10050;	SWAP	10690;	SWPTMP	10740;	SYMADD	14850;	SYMBOL	00410;
SYMCNT	04840;	SYMEND	00480;	SYMEND	14940;	SYMSTR	00470;	TABCNT	12770;

TABMES	12890;	TCOUNT	12920;	TEST	03080;	TESTLB	02600;	TESTT	03170;
TF	00710;	TLITRL	14990;	TSTAN	03340;	TSTLB1	02730;	TSTLB2	02760;
TSTLIT	05700;	TSTSCL	08870;	TSTT6A	05540;	TSTTEL	05550;	TVEC	01250;
TWOOP	13860;	UDCHK	11690;	UPDATE	08540;				

\*\*\*\*INSTRUCTION COUNT = 310,489\*\*\*\*



XI. APPENDIX C

```
// CONTROL RECORD.  
    FUNCTION PLXCPL.  INTERPRETER=MTXINT04.  
    START SYMTAB AT 8000, END SYMTAB AT 20000.  
    STACK START AT 5000, END STACK AT 7990.  
    EXECUTE AT 20001.  
    POSTLIST=YES.  
//
```

TEST: PROCEDURE MAIN;

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

```
/*  
  STRINGS AND SUBSTRINGS  
*/
```

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

```

/*
INPUT / OUTPUT (INCLUDING STRING I/O)
*/

```

```

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

```

```

/*
DO GROUPS
*/

```

```

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

```

```

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(3),A);
  DO A=1 TO 10;
    X=A*1.33; Y=X/A;
    Z=IF A<5 THEN A ELSE 0.0;
    PUT EDIT (X,Y,Z) (E(20));
    PUT EDIT (A) (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 RPROC;

```

```

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

```

```

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

```

```

LITERAL:PROCEDURE (TEST); DECLARE TEST CHAR(1);
    IF .NOT. NEXT THEN RETURN(.F.);
    IF SUBSTR(INPUT,I,1)=TEST THEN DO; I=I+1; RETURN(.T.); END;
    RETURN(.F.);
END LITERAL;
PRIMARY:PROCEDURE;
    IF LITERAL("(") THEN
        DO;
            IF .NOT. EXP1 THEN RETURN(.F.);
            IF .NOT. LITERAL(")") THEN RETURN(.F.);
            RETURN(.T.);
        END;
    IF NUMBER THEN DO; CALL OUT(CHAR); RETURN(.T.); END;
    IF ID THEN DO; CALL OUT(CHAR); RETURN(.T.); END;
    RETURN(.F.);
END PRIMARY;
TERM: PROCEDURE;
    IF .NOT. PRIMARY THEN RETURN(.F.);
    MULT: IF LITERAL("*") THEN
        DO;
            IF .NOT. PRIMARY THEN RETURN(.F.);
            CALL OUT("*"); GO TO MULT;
        END;
    IF LITERAL("/") THEN
        DO;
            IF .NOT. PRIMARY THEN RETURN(.F.);
            CALL OUT("/"); GO TO MULT;
        END;
    RETURN(.T.);
END TERM;
EXP2: PROCEDURE;
    IF LITERAL("-") THEN
        DO;
            IF .NOT. TERM THEN RETURN(.F.);
            CALL OUT("-"); RETURN(.T.);
        END;
    IF LITERAL("+") THEN

```

```

        DO;
          IF .NOT. TERM THEN RETURN(.F.);
          RETURN(.T.);
        END;
      IF TERM THEN RETURN(.T.); RETURN(.F.);
    END EXP2;
  EXP1:  PROCEDURE;
    IF .NOT. EXP2 THEN RETURN(.T.);
    PLUS: IF LITERAL("+") THEN
      DO;
        IF .NOT. TERM THEN RETURN(.F.);
        CALL OUT("+"); GO TO PLUS;
      END;
    IF LITERAL("-") THEN
      DO;
        IF .NOT. TERM THEN RETURN(.F.);
        CALL OUT("-"); GO TO PLUS;
      END;
    RETURN(.T.);
  END EXP1;
  /* START HERE */
  PUT EDIT ("ENTER POSTFIX") (SKIP(3),A);
  GET EDIT (INPUT) (SKIP,A(80));
  I,J=1;
  PUT EDIT ("INFIX EXPRESSION =",INPUT) (SKIP,A,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(1:10) FIXED;
      F(1)=13;
      CALL Z((F(1)+Y));
    END Q;
    CALL Q(X);
  END;
END P;
R: PROCEDURE;
  DECLARE (I,G(1:10)) FIXED;
  BEGIN;
    U: PROCEDURE(W); DECLARE W FIXED;
      G(I)=W;
    END U;
    DO I=1 TO 10;
      G(I)=23;
      CALL P(U,1);
    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("INCORPECT 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;  METAX INSTRUCTION COUNT =      55,991****
    ****SYMTAB SEARCH COUNT =      1,484;  SYMTAB COMPARE COUNT =      78,626****
    ****SYMBOL TABLE ENTRY COUNT =      370****

```

```

00000      DYNAM $001
00005      JUMP  $002
00010  $002 STKTP $STKT0 , $STKT1
00020      JUMP  $003
00026  $003 FMT   $004      , "1"
00032      LDA   "BEGIN STRINGS"
00047      EDIT
00048      PUT
00049      JUMP  $005
    ***FORMAT CODE***
00054  $004 "2000380000"
00066  $005 LDA   S          , "OL"
00074      LDA   "THIS IS A STRING."
00093      STO
00094      FMT   $006      , "1"
00100      LDA   S          , "OL"
00103      EDIT
00109      PUT
00110      JUMP  $007
    ***FORMAT CODE***
00115  $006 "80000"
00122  $007 LDA   T          , "OL"
00130      LD    "0001"
00136      LD    "0004"
00142      SBSTR
00143      LDA   "THIS"
00149      STO
00150      LDA   T          , "OL"
00158      LD    "0005"
00164      LD    "000%"
00170      SBSTR
00171      LDA   S          , "OL"
00179      LD    "0005"
00185      LD    "0006"
00191      SBSTR
00192      LDA   "CONCATENATED SUBSTRING."
00217      CAT

```

```

00218          STO
00219          FMT    $008    ,"1"
00225          LDA    T      ,"0L"
00233          EDIT
00234          PUT
00235          JUMP   $009
      ***FORMAT CODE***
00240    $008    "8000J"
00247    $009    LDA    S      ,"0L"
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      ,"0L"
00333          LD     "0001"
00339          LD     "0004"
00345          SBSTR
00346          LDA    T      ,"0L"
00354          LD     "0001"
00360          LD     "0004"
00366          SBSTR
00367          COMPC "1"
00369          JUMPF $013
00374          FMT    $014    ,"1"
00380          LDA    "STRING COMPARE 2 WORKS"
00404          EDIT
00405          PUT
00406          JUMP   $013

```

```

***FORMAT CODE***
00411 $014 "80000"
00418 $013 FMT $016 , "1"
00424 LDA "EXIT STRINGS"
00438 EDIT
00439 PUT
00440 JUMP $017
***FORMAT CODE***
00445 $016 "2000180000"
00457 $017 STKTP $STKT0 , $STKT1
00468 ALLOC $STKT1 , M
***DOPE VECTOR CODE***
"1042000120005"
00492 JUMP $018
00497 $018 FMT $019 , "1"
00503 LDA "BEGIN I/O BLOCK"
00520 EDIT
00521 PUT
00522 JUMP $020
***FORMAT CODE***
00527 $019 "2000380000"
00539 $020 FMT $021 , "Z"
00545 LDA S , "0D"
00553 GET
00554 JUMP $022
***FORMAT CODE***
00559 $021 "80006"
00566 $022 FMT $023 , "Z"
00572 LDA A
00578 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 , "OD"
00628 FMT $025 , "O"
00634 LD A
00640 EDIT
00641 LD B
00647 EDIT
00648 LD C
00654 EDIT
00655 PUT
00656 JUMP $026
***FORMAT CODE***
00661 $025 "0005"
00668 $026 FMT $027 , "1"
00674 LDA S , "OD"
00682 EDIT
00683 LDA T , "OD"
00691 EDIT
00692 LD A
00698 EDIT
00699 LD B
00705 EDIT
00706 LD C
00712 EDIT
00713 PUT
00714 JUMP $028
***FORMAT CODE***
00719 $027 "20001800003000D800003000Q*0005*0005*0005"
00761 $028 FMT $029 , "Z"
00767 LDA S , "OD"
00775 GET
00776 LDA T , "OD"
00784 GET
00785 JUMP $030
***FORMAT CODE***
00790 $029 "200018000D8000D"
00807 $030 LDA S , "OD"
00815 FMT $031 , "Y"

```

```

00821      LDA    X
00827      GET
00828      LDA    Y
00834      GET
00835      JUMP   $032
      ***FORMAT CODE***
00840  $031    "9000"
00847  $032    LDA    T      ,"0D"
00855      FMT    $033    ,"Y"
00861      LDA    Z
00867      GET
00868      JUMP   $034
      ***FORMAT CODE***
00873  $033    "9000"
00880  $034    FMT    $035    ,"1"
00885      LD     X
00892      EDIT
00893      LD     Y
00899      EDIT
00900      LD     Z
00906      EDIT
00907      LDA    S      ,"0D"
00915      EDIT
00916      LDA    T      ,"0D"
00924      EDIT
00925      PUT
00926      JUMP   $036
      ***FORMAT CODE***
00931  $035    "200029000D9000D9000D200018000D8000D"
00968  $036    LDA    A
00974      LD     "0001"
00980      SST
00981  $038    LD     "0005"
00987      STCKC  "4"
00989      JUMPT  $037
00994      LDA    M
01000      LD     A

```

```

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
01078      LD      A
01084      INDXR "1"
01086      EDIT
01087      LDA      A
01093      LD      A
01099      LD      "0001"
01105      ADD
01106      SST
01107      JUMP     $041
01112      $040    PUT
01113      JUMP     $042
      ***FORMAT CODE***
01118      $039    "0005"
01125      $042    FMT      $043      ,"1"
01131      LDA      "EXIT I/O BLOCK"
01147      EDIT
01148      PUT
01149      JUMP     $044
      ***FORMAT CODE***
01154      $043    "2000180000"

```



```

01166 $044 STKTP $STKTO , $STKTI
01177 ALLOC $STKTI , M
***DOPE VECTOR CODE***
      "204ZccccZ000*Z0001Z000*"
01211 JUMP $045
01216 $045 FMT $046 , "1"
01222 LDA "BEGIN DO GROUPS"
01239 EDIT
01240 PUT
01241 JUMP $047
***FORMAT CODE***
01246 $046 "2000380000"
01258 $047 LDA I
01264 LD "0000"
01270 SST
01271 $049 LD "0004"
01277 STCKC "4"
01279 JUMPT $048
01284 LD "0004"
01290 LD I
01296 SUB
01297 LD "0005"
01303 MULT
01304 LD $051
01310 ADD
01311 JUMPA
01312 $053 FMT $054 , "1"
01318 LDA "CASE 0"
01326 EDIT
01327 PUT
01328 JUMP $055
***FORMAT CODE***
01333 $054 "80000"
01340 $055 JUMP $052
01345 $056 FMT $057 , "1"
01351 LDA "CASE 1"
01359 EDIT

```

```

01360          PUT
01361          JUMP  $058
    ***FORMAT CODE***
01366  $057    "80000"
01373  $058    JUMP  $052
01378  $059    FMT   $060  ,"1"
01384          LDA   "CASE 2"
01392          EDIT
01393          PUT
01394          JUMP  $061
    ***FORMAT CODE***
01399  $060    "80000"
01406  $061    JUMP  $052
01411  $062    FMT   $063  ,"1"
01417          LDA   "CASE 3"
01425          EDIT
01426          PUT
01427          JUMP  $064
    ***FORMAT CODE***
01432  $063    "80000"
01439  $064    JUMP  $052
01444  $065    FMT   $066  ,"1"
01450          LDA   "CASE 4"
01458          EDIT
01459          PUT
01460          JUMP  $067
    ***FORMAT CODE***
01465  $066    "80000"
01472  $067    JUMP  $052
01477  $068    LD    $050
01483  $050    JUMP  $053
01488          JUMP  $056
01493          JUMP  $059
01498          JUMP  $062
01503          JUMP  $065
01508  $052    LDA   I
01514          LD    I

```

01520		LD	"0001"
01526		ADD	
01527		SST	
01528		JUMP	\$049
01533	\$048	LDA	I
01539		LD	"0001"
01545		SST	
01546	\$070	LD	"cccc"
01552		STCKC	"2"
01554		JUMPT	\$069
01559		LDA	M
01565		LD	I
01571		LD	"0005"
01577		INDXA	"2"
01579		LD	"0000"
01585		STO	
01586		LDA	A
01592		LD	"0003"
01598		LD	"0002"
01604		MULT	
01605		LD	"0005"
01611		SUB	
01612		SST	
01613	\$072	LD	"0001"
01619		STCKC	"4"
01621		JUMPT	\$071
01626		LD	A
01632		LD	"0005"
01638		STCKC	"2"
01640		JUMPF	\$071
01645		LDA	M
01651		LD	I
01657		LD	A
01663		INDXA	"2"
01665		LD	A
01671		STO	
01672		LDA	A

```

01678      LD      A
01684      LD      "0001"
01690      ADD
01691      SST
01692      JUMP    $072
01697  $071  FMT    $073      ,"1"
01703      LDA     "I = "
01709      EDIT
01710      LD      I
01716      EDIT
01717      LDA     ", M(I,1) = "
01730      EDIT
01731      LDA     M
01737      LD      I
01743      LD      "0001"
01749      INDXR  "2"
01751      EDIT
01752      LDA     ", M(I,5) = "
01765      EDIT
01766      LDA     M
01772      LD      I
01778      LD      "0005"
01784      INDXR  "2"
01786      EDIT
01787      PUT
01788      JUMP    $074
      ***FORMAT CODE***
01793  $073  "800000005"
01805  $074  LDA     I
01811      LD      I
01817      LD      "0001"
01823      SUB
01824      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"
01871  $076      STKTP $STKT0 , $STKT1
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"
01936  $079      LDA  A
01942          LD   "0001"
01948          SST
01949  $081      LD   "0001"
01955          STCKC "4"
01957          JUMPT $080
01962          LDA  X
01968          LD   A
01974          LD   "EA@YD?01"
01984          MULT
01985          STO
01986          LDA  Y
01992          LD   X
01998          LD   A
02004          DIV
02005          STO
02006          LDA  Z
02012          LD   A
02018          LD   "0005"
02024          STCKC "2"
02026          JUMPF $082
02031          LD   A
02037          JUMP  $083
02042  $082      LD   "00000000"

```

```

02052 $083 STO
02053 FMT $084 , "1"
02059 LD X
02065 EDIT
02066 LD Y
02072 EDIT
02073 LD Z
02079 EDIT
02080 PUT
02081 JUMP $085
***FORMAT CODE***
02086 $084 "9000D"
02093 $085 FMT $086 , "1"
02099 LD A
02105 EDIT
02106 PUT
02107 JUMP $087
***FORMAT CODE***
02112 $086 "'0005"
02119 $087 LDA A
02125 LD A
02131 LD "0001"
02137 ADD
02138 SST
02139 JUMP $081
02144 $080 FMT $088 , "1"
02150 LDA "EXIT ARITHMETIC BLOCK"
02173 EDIT
02174 PUT
02175 JUMP $089
***FORMAT CODE***
02180 $088 "2000180000"
02192 $089 STKTP $STKT0 , $STKT1
02203 JUMP $090
02208 NFACT ENTPR "2"
02210 DYNAM $091
02215 JUMP $092

```

```

02220 $092 LD I
02226 LD "0000"
02232 STCKC "1"
02234 JUMPF $093
02239 LD "0001"
02245 SWAP
02246 RETRN
02247 $093 LDA $094
02253 LD $ACTIVE
02259 LD $ACTIVE
02265 FLAG
02266 LDA $095
02272 LDA $095
02278 LD I
02284 LD "0001"
02290 SUB
02291 STO
02292 LD $STKT2
02298 LDA NFACT
02304 JUMPA
02305 $094 LD I
02311 MULT
02312 SWAP
02313 RETRN
02314 RETRN
02315 $090 FMT $096 , "1"
02321 LDA "ENTER RPROC"
02334 EDIT
02335 PUT
02336 JUMP $097
***FORMAT CODE***
02341 $096 "2000380000"
02353 $097 FMT $098 , "1"
02359 LDA "7 FACTORIAL ="
02374 EDIT
02375 LDA $099
02381 LD $ACTIVE

```

```

02387          LD      $ACTIVE
02393          FLAG
02394          LDA      $100
02400          LDA      $100
02406          LDA      $101
02412          LD      $ACTIVE
02418          LD      $ACTIVE
02424          FLAG
02425          LDA      $102
02431          LDA      $102
02437          LD      "0003"
02443          STO
02444          LD      $STKT1
02450          LDA      NFACT
02456          JUMPA
02457    $101    LD      "0001"
02463          ADD
02464          STO
02465          LD      $STKT1
02471          LDA      NFACT
02477          JUMPA
02478    $099    EDIT
02479          PUT
02480          JUMP    $103
      ***FORMAT CODE***
02485    $098    "2000180000*000*"
02502    $103    FMT    $104    , "1"
02508          LDA      "EXIT RPROC"
02520          EDIT
02521          PUT
02522          JUMP    $105
      ***FORMAT CODE***
02527    $104    "2000180000"
02539    $105    STKTP $STKT0    , $STKT1
02550          JUMP    $106
02555    OUT     ENTPR "2"
02557          DYNAM $095

```



02562		JUMP	\$108	
02567	\$108	LDA	T	,"1+"
02575		LD	J	
02581		LD	"0001"	
02587		SBSTR		
02588		LDA	OUTCHAR	,"01"
02596		STO		
02597		LDA	J	
02603		LD	J	
02609		LD	"0001"	
02615		ADD		
02616		STO		
02617		RETRN		
02618	NEXT	ENTPR	"2"	
02620		DYNAM	I	
02625		JUMP	\$110	
02630	\$110	LDA	INPUT	,"1+"
02638		LD	I	
02644		LD	"0001"	
02650		SBSTR		
02651		LDA	" "	
02654		COMPC	"1"	
02656		JUMPF	\$112	
02661		LDA	I	
02667		LD	I	
02673		LD	"0001"	
02679		ADD		
02680		STO		
02681		LD	I	
02687		LD	"001+"	
02693		STCKC	"4"	
02695		JUMPF	\$113	
02700		LD	"F"	
02703		SWAP		
02704		RETRN		
02705	\$113	LDA	INPUT	,"1+"
02713		LD	I	

02719		LD	"0001"
02725		SBSTR	
02726		LDA	";"
02729		COMPC	"1"
02731		JUMPF	\$114
02736		LD	"F"
02739		SWAP	
02740		RETRN	
02741	\$114	JUMP	\$110
02746	\$112	LD	"T"
02749		SWAP	
02750		RETRN	
02751		RETRN	
02752	NUMBFR	ENTPR	"2"
02754		DYNAM	I
02759		JUMP	\$116
02764	\$116	LDA	\$117
02770		LD	\$ACTIVE
02776		LD	\$ACTIVE
02782		FLAG	
02783		LD	\$STKT2
02789		LDA	NEXT
02795		JUMPA	
02796	\$117	NOT	
02797		JUMPF	\$118
02802		LD	"F"
02805		SWAP	
02806		RETRN	
02807	\$118	LDA	INPUT , "1+"
02815		LD	I
02821		LD	"0001"
02827		SBSTR	
02828		LDA	"9"
02831		COMPC	"3"
02833		LDA	INPUT , "1+"
02841		LD	I
02847		LD	"0001"

02853		SBSTR
02854		LDA "0"
02857		COMPC "5"
02859		AND
02860		JUMPF \$119
02865		LDA CHAR , "01"
02873		LDA INPUT , "1+"
02881		LD I
02887		LD "0001"
02893		SBSTR
02894		STO
02895		LDA I
02901		LD I
02907		LD "0001"
02913		ADD
02914		STO
02915		LD "T"
02918		SWAP
02919		RETRN
02920	\$119	LD "F"
02923		SWAP
02924		RETRN
02925		RETRN
02926	ID	ENTPR "2"
02928		DYNAM I
02933		JUMP \$121
02938	\$121	LDA \$122
02944		LD \$ACTIVE
02950		LD \$ACTIVE
02956		FLAG
02957		LD \$STKT2
02963		LDA NEXT
02969		JUMPA
02970	\$122	NOT
02971		JUMPF \$123
02976		LD "F"
02979		SWAP

02980		RETRN	
02981	\$123	LDA	INPUT , "1+"
02989		LD	I
02995		LD	"0001"
03001		SBSTR	
03002		LDA	"A"
03005		COMPC	"5"
03007		LDA	INPUT , "1+"
03015		LD	I
03021		LD	"0001"
03027		SBSTR	
03028		LDA	"Z"
03031		COMPC	"3"
03033		AND	
03034		JUMPF	\$124
03039		LDA	CHAR , "01"
03047		LDA	INPUT , "1+"
03055		LD	I
03061		LD	"0001"
03067		SBSTR	
03068		STO	
03069		LDA	I
03075		LD	I
03081		LD	"0001"
03087		ADD	
03088		STO	
03089		LD	"T"
03092		SWAP	
03093		RETRN	
03094	\$124	LD	"F"
03097		SWAP	
03098		RETRN	
03099		RETRN	
03100	LITERAL	ENTPR	"2"
03102		DYNAM	\$095
03107		JUMP	\$126
03112	\$126	LDA	\$127

03118	LD	\$ACTIVE	
03124	LD	\$ACTIVE	
03130	FLAG		
03131	LD	\$STKT2	
03137	LDA	NEXT	
03143	JUMPA		
03144	NOT		\$127
03145	JUMPF	\$128	
03150	LD	"F"	
03153	SWAP		
03154	RETRN		
03155	LDA	INPUT	\$128
03163	LD	I	
03165	LD	"0001"	
03175	SBSTR		
03176	LDA	OUTCHAR	
03184	COMPC	"1"	
03186	JUMPF	\$129	
03191	LDA	I	
03197	LD	I	
03203	LD	"0001"	
03209	ADD		
03210	STO		
03211	LD	"T"	
03214	SWAP		
03215	RETRN		
03216	LD	"F"	\$129
03219	SWAP		
03220	RETRN		
03221	RETRN		
03222	ENTPR	"2"	PRIMARY
03224	DYNAM	\$130	
03229	JUMP	\$131	
03234	LDA	\$132	\$131
03240	LD	\$ACTIVE	
03246	LD	\$ACTIVE	
03252	FLAG		

03253	LDA	\$133
03259	LDA	\$133
03265	LDA	"("
03268	STO	
03269	LD	\$STKT2
03275	LDA	LITERAL
03281	JUMPA	
03282	JUMPF	\$134
03287	LDA	\$135
03293	LD	\$ACTIVE
03299	LD	\$ACTIVE
03305	FLAG	
03306	LD	\$STKT2
03312	LDA	EXP1
03318	JUMPA	
03319	NOT	
03320	JUMPF	\$136
03325	LD	"F"
03328	SWAP	
03329	RETRN	
03330	LDA	\$137
03336	LD	\$ACTIVE
03342	LD	\$ACTIVE
03348	FLAG	
03349	LDA	\$138
03355	LDA	\$138
03361	LDA	"")"
03364	STO	
03365	LD	\$STKT2
03371	LDA	LITERAL
03377	JUMPA	
03378	NOT	
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"
03458		LD	\$STKT2
03464		LDA	OUT
03470		JUMPA	
03471	\$142	LD	"T"
03474		SWAP	
03475		RETRN	
03476	\$141	LDA	\$143
03482		LD	\$ACTIVE
03488		LD	\$ACTIVE
03494		FLAG	
03495		LD	\$STKT2
03501		LDA	ID
03507		JUMPA	
03508	\$143	JUMPF	\$144
03513		LDA	\$145
03519		LD	\$ACTIVE
03525		LD	\$ACTIVE
03531		FLAG	
03532		LDA	CHAR , "01"
03540		LD	\$STKT2
03545		LDA	OUT
03552		JUMPA	

03553	\$145	LD	"T"
03556		SWAP	
03557		RETRN	
03558	\$144	LD	"F"
03561		SWAP	
03562		RETRN	
03563		RETRN	
03564	TERM	ENTPR	"2"
03566		DYNAM	\$146
03571		JUMP	\$147
03576	\$147	LDA	\$148
03582		LD	\$ACTIVE
03588		LD	\$ACTIVE
03594		FLAG	
03595		LD	\$STKT2
03601		LDA	PRIMARY
03607		JUMPA	
03608	\$148	NOT	
03609		JUMPF	\$149
03614		LD	"F"
03617		SWAP	
03618		RETRN	
03619	\$149	LDA	\$150
03625		LD	\$ACTIVE
03631		LD	\$ACTIVE
03637		FLAG	
03638		LDA	\$133
03644		LDA	\$133
03650		LDA	"*"
03653		STO	
03654		LD	\$STKT2
03660		LDA	LITERAL
03666		JUMPA	
03667	\$150	JUMPF	\$152
03672		LDA	\$153
03678		LD	\$ACTIVE
03684		LD	\$ACTIVE



03690		FLAG
03691		LD \$STKT2
03697		LDA PRIMARY
03703		JUMPA
03704	\$153	NOT
03705		JUMPF \$154
03710		LD "F"
03713		SWAP
03714		RETRN
03715	\$154	LDA \$155
03721		LD \$ACTIVE
03727		LD \$ACTIVE
03733		FLAG
03734		LDA \$138
03740		LDA \$138
03746		LDA "*"
03749		STO
03750		LD \$STKT2
03755		LDA OUT
03762		JUMPA
03763	\$155	LDA \$149
03769		JUMPA
03770	\$152	LDA \$157
03776		LD \$ACTIVE
03782		LD \$ACTIVE
03788		FLAG
03789		LDA \$158
03795		LDA \$158
03801		LDA "/"
03804		STO
03805		LD \$STKT2
03811		LDA LITERAL
03817		JUMPA
03818	\$157	JUMPF \$159
03823		LDA \$160
03829		LD \$ACTIVE
03835		LD \$ACTIVE

03841		FLAG
03842		LD \$STKT2
03848		LDA PRIMARY
03854		JUMPA
03855	\$160	NOT
03856		JUMPF \$161
03861		LD "F"
03864		SWAP
03865		RETRN
03866	\$161	LDA \$162
03872		LD \$ACTIVE
03873		LD \$ACTIVE
03884		FLAG
03885		LDA \$163
03891		LDA \$163
03897		LDA "/"
03900		STO
03901		LD \$STKT2
03907		LDA OUT
03913		JUMPA
03914	\$162	LDA \$149
03920		JUMPA
03921	\$159	LD "T"
03924		SWAP
03925		RETRN
03926		RETRN
03927	EXP2	ENTPR "2"
03929		DYNAM \$163
03934		JUMP \$165
03939	\$165	LDA \$166
03945		LD \$ACTIVE
03951		LD \$ACTIVE
03957		FLAG
03958		LDA \$133
03964		LDA \$133
03970		LDA "-"
03973		STO

03974		LD	\$STKT2
03980		LDA	LITERAL
03986		JUMPA	
03987	\$166	JUMPF	\$168
03992		LDA	\$169
03998		LD	\$ACTIVE
04004		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
04047		LD	\$ACTIVE
04053		FLAG	
04054		LDA	\$138
04060		LDA	\$138
04066		LDA	"■"
04069		STO	
04070		LD	\$STKT2
04076		LDA	OUT
04082		JUMPA	
04083	\$171	LD	"T"
04086		SWAP	
04087		RETRN	
04088	\$168	LDA	\$173
04094		LD	\$ACTIVE
04100		LD	\$ACTIVE
04106		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
04179		LD "F"
04182		SWAP
04183		RETRN
04184	\$177	LD "T"
04187		SWAP
04188		RETRN
04189	\$175	LDA \$178
04195		LD \$ACTIVE
04201		LD \$ACTIVE
04207		FLAG
04208		LD \$STKT2
04214		LDA TERM
04220		JUMPA
04221	\$178	JUMPF \$179
04226		LD "T"
04229		SWAP
04230		RETRN
04231	\$179	LD "F"
04234		SWAP
04235		RETRN
04236		RETRN
04237	EXP1	ENTPR "2"
04239		DYNAM \$146

04244		JUMP	\$181
04249	\$181	LDA	\$182
04255		LD	\$ACTIVE
04261		LD	\$ACTIVE
04267		FLAG	
04268		LD	\$STKT2
04274		LDA	EXP2
04280		JUMPA	
04281	\$182	NOT	
04282		JUMPF	\$183
04287		LD	"T"
04290		SWAP	
04291		RETRN	
04292	\$183	LDA	\$184
04298		LD	\$ACTIVE
04304		LD	\$ACTIVE
04310		FLAG	
04311		LDA	\$133
04317		LDA	\$133
04323		LDA	"+"
04326		STO	
04327		LD	\$STKT2
04333		LDA	LITERAL
04339		JUMPA	
04340	\$184	JUMPF	\$186
04345		LDA	\$187
04351		LD	\$ACTIVE
04357		LD	\$ACTIVE
04363		FLAG	
04364		LD	\$STKT2
04370		LDA	TERM
04376		JUMPA	
04377	\$187	NOT	
04378		JUMPF	\$188
04383		LD	"F"
04386		SWAP	
04387		RETRN	

04388	\$188	LDA	\$189
04394		LD	\$ACTIVE
04400		LD	\$ACTIVE
04406		FLAG	
04407		LDA	\$138
04413		LDA	\$138
04419		LDA	"+"
04422		STO	
04423		LD	\$STKT2
04429		LDA	OUT
04435		JUMPA	
04436	\$189	LDA	\$183
04442		JUMPA	
04443	\$186	LDA	\$191
04449		LD	\$ACTIVE
04455		LD	\$ACTIVE
04461		FLAG	
04462		LDA	\$158
04468		LDA	\$158
04474		LDA	"-"
04477		STO	
04478		LD	\$STKT2
04484		LDA	LITERAL
04490		JUMPA	
04491	\$191	JUMPF	\$193
04495		LDA	\$194
04502		LD	\$ACTIVE
04508		LD	\$ACTIVE
04514		FLAG	
04515		LD	\$STKT2
04521		LDA	TERM
04527		JUMPA	
04528	\$194	NOT	
04529		JUMPF	\$195
04534		LD	"F"
04537		SWAP	
04538		RETRN	

```

04539 $195 LDA $196
04545 LD $ACTIVE
04551 LD $ACTIVE
04557 FLAG
04558 LDA $163
04564 LDA $163
04570 LDA "-"
04573 STC
04574 LD $STKT2
04580 LDA OUT
04586 JUMPA
04587 $196 LDA $183
04593 JUMPA
04594 $193 LD "T"
04597 SWAP
04598 RETRN
04599 RETRN
04600 $106 FMT $198 , "1"
04606 LDA "ENTER POSTFIX"
04621 EDIT
04622 PUT
04623 JUMP $199
***FORMAT CODE***
04628 $198 "2000380000"
04640 $199 FMT $200 , "Z"
04646 LDA INPUT , "1+"
04654 GET
04655 JUMP $201
***FORMAT CODE***
04660 $200 "200018001+"
04672 $201 LDA I
04678 LDA J
04684 LD "0001"
04690 SST
04691 STO
04692 FMT $202 , "1"
04698 LDA "INFIX EXPRESSION ="

```

```

04718          EDIT
04719          LDA    INPUT    ,"1+"
04727          EDIT
04728          PUT
04729          JUMP   $203
      ***FORMAT CODE***
04734  $202    "2000180000000028001+"
04756  $203    LDA    T        ,"1+"
04764          LDA    INPUT    ,"1+"
04772          STO
04773          LDA    $204
04775          LD     $ACTIVE
04785          LD     $ACTIVE
04791          FLAG
04792          LD     $STKT1
04798          LDA    EXP1
04804          JUMPA
04805  $204    JUMPF  $205
04810          FMT    $206      ,"1"
04816          LDA    "POSTFIX EXPRESSION ="
04838          EDIT
04839          LDA    T        ,"1+"
04847          LD     "0001"
04853          LD     J
04859          LD     "0001"
04865          SUB
04866          SBSTR
04867          LDA    ";"
04870          CAT
04871          EDIT
04872          PUT
04873          JUMP   $207
      ***FORMAT CODE***
04873  $206    "20001800000000280000"
04900  $207    JUMP   $208
04905  $205    FMT    $209      ,"1"
04911          LDA    "*****ERROR*****"

```



```

04926          EDIT
04927          PUT
04928          JUMP  $209
    ***FORMAT CODE***
04933  $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    STKTP $STKT0  , $STKT1
04995          JUMP  $213
05000  P      ENTPR "2"
05002          DYNAM $163
05007          JUMP  $215
05012  $215    STKTP $STKT2  , $STKT3
05023          JUMP  $216
05028  Q      ENTPR "3"
05030          DYNAM $217
05035          ALLOC $STKT4  ,F
    ***DOPE VECTOR CODE***
    "104200012000"
05059          JUMP  $218
05064  $218    LDA  F
05070          LD   "0001"
05076          INDXA "1"
05078          LD   "000 "
05084          STO
05085          LDA  $219
05091          LD   $ACTIVE
05097          LDA  $220
05103          LD   "300+"  *****
05109          LD   "0004"
05115          ADD
05116          STO

```

05117		LD	"0000"	*****
05123		FLAG		
05124		LDA	\$221	
05130		LDA	\$221	
05136		LDA	F	
05142		LD	"0001"	
05148		INDXR	"1"	
05150		LD	Y	
05156		ADD		
05157		STO		
05158		LD	\$STKT4	
05164		LDA	Z	
05170		JUMPA		
05171	\$219	RETRN		
05172	\$216	LDA	\$222	
05178		LD	\$ACTIVE	
05184		LD	\$ACTIVE	
05190		FLAG		
05191		LDA	\$223	
05197		LDA	\$223	
05203		LD	I	
05209		STO		
05210		LDA	\$224	
05216		LDA	\$225	
05222		LD	G	
05228		LD	"0004"	
05234		ADD		
05235		STO		
05236		LD	"0000"	*****
05242		STO		
05243		LD	\$STKT3	
05249		LDA	Q	
05255		JUMPA		
05256	\$222	RETRN		
05257	R	ENTPR	"2"	
05259		DYNAM	\$226	
05264		ALLOC	\$STKT2	,G

\*\*\*DOPE VECTOR CODE\*\*\*

"104200012000"

```

05288      JUMP  $227
05293  $227  STKTP $STKT2 , $STKT3
05304      JUMP  $228
05309  U     ENTPR "3"
05311      DYNAM F
05316      JUMP  $230
05321  $230  LDA   G
05327      LD    I
05333      INDXA "1"
05335      LD    W
05341      STO
05342      RETRN
05343  $228  LDA   I
05349      LD    "0001"
05355      SST
05356  $232  LD    "000"
05362      STCKC "4"
05364      JUMPT $231
05369      LDA   G
05375      LD    I
05381      INDXA "1"
05383      LD    "000G"
05389      STO
05390      LDA   $233
05396      LD    $ACTIVE
05402      LD    $ACTIVE
05408      FLAG
05409      LDA   $224
05415      LDA   $224
05421      LDA   "018" *****
05427      STO
05428      LDA   $235
05434      LD    $ACTIVE
05440      STO
05441      LDA   $236

```

```

05447      LDA    $236
05453      LD     "0001"
05459      STO
05460      LD     $STKT3
05466      LDA    P
05472      JUMPA
05473  $233   LDA    I
05479      LD     I
05485      LD     "0001"
05491      ADD
05492      SST
05493      JUMP   $232
05493  $231   FMT    $237      ,"1"
05504      LDA    "GLOBAL DISPLAY TEST"
05525      EDIT
05526      PUT
05527      JUMP   $238
      ***FORMAT CODE***
05532  $237   "2000360000"
05544  $238   FMT    $239      ,"1"
05550      LDA    I
05556      LD     "0001"
05562      SST
05563  $241   LD     "0001"
05569      STCKC  "4"
05571      JUMPT  $240
05576      LDA    G
05582      LD     I
05588      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 $STKTO  , $STKT1
05716          ALLOC $STKT1  ,2
***DOPE VECTOR CODE***
          "1042000120003"
05740          JUMP  $246
05745  LBL     ENTPK "2"
05747          DYNAM $095
05752          JUMP  $248
05757  $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.I"
05824 STO
05825 STKTP $STKT1 , $STKT2
05836 JUMP $251
05841 $251 LDA I
05847 LD "0001"
05853 STO
05854 LD "01.Z"
05860 JUMPA
05861 LDA B
05867 LD I
05873 LD "0002"
05879 DIV
05880 STO
05881 LBL1 FMT $252 , "1"
05887 LDA "LABEL TEST"
05899 EDIT
05900 LD I
05906 EDIT
05907 PUT
05908 JUMP $253
***FORMAT CODE***
05913 $252 "2000180000.0005"
05930 $253 LDA A
05936 LD I
05942 LD "0001"
05948 ADD
05949 STO
05950 LD Y
05956 JUMPA
05957 LBL2 FMT $254 , "1"
05963 LDA "INCORRECT LABEL TEST"
05985 EDIT
05986 PUT

```

```

05987          JUMP    $255
    ***FORMAT CODE***
05992  $254    "80000"
05999  $255    FMT     $256    , "1"
06005          LDA     "INCORRECT LABEL TEST"
06027          EDIT
06028          PUT
06029          JUMP    LBL2
    ***FORMAT CODE***
06034  $256    "80000"
06041  LBL2    FMT     $258    , "1"
06047          LDA     "LABEL TEST"
06059          EDIT
06060          LD      A
06066          EDIT
06067          PUT
06068          JUMP    $259
    ***FORMAT CODE***
06073  $258    "2000180000*0005"
06090  $259    LDA     Z
06096          LD      "0001"
06102          INDXA   "1"
06104          LD      "01-½"
06110          STO
06111          LDA     Z
06117          LD      "0002"
06123          INDXA   "1"
06125          LD      "01-½"
06131          STO
06132          LDA     Z
06133          LD      "0003"
06144          INDXA   "1"
06145          LD      "01J-"
06152          STO
06153          LDA     BADLAB
06159          LD      $ACTIVE
06165          LD      $ACTIVE

```

```

06171          FLAG
06172          LDA    Z
06178          LD     $STKT1
06184          LDA    LBL
06190          JUMPA
06191  BADLAB   FMT     $261      ,"1"
06197          LDA    "INCORRECT LABEL RETURN"
06221          EDIT
06222          PUT
06223          JUMP   GOODLAB
      ***FORMAT CODE***
06228  $261     "2000180000"
06240  GOODLAB  FMT     $263      ,"1"
06246          LDA    "CORRECT LABEL RETURN"
06268          EDIT
06269          PUT
06270          JUMP   $264
      ***FORMAT CODE***
06275  $263     "2000180000"
06287  $264     FMT     $265      ,"1"
06293          LDA    "EXIT LABEL"
06305          EDIT
06306          PUT
06307          JUMP   $266
      ***FORMAT CODE***
06312  $265     "2000180000"
06324  $266     STOP
06325          END      PROGRAM

```



```

BEGIN STRINGS
THIS IS A STRING.
THIS IS A CONCATENATED SUBSTRING.
STRING COMPARE 1 WORKS
STRING COMPARE 2 WORKS

```

```

EXIT STRINGS

```

```

BEGIN I/O BLOCK

```

```

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

```

```

5.12340      .43000000000E+015  -.11200000000E-007
  5.1234    4.3E+14  -11.2E-9
   1      2      3      4      5

```

```

EXIT I/O BLOCK

```

```

BEGIN DO GROUPS

```

```

CASE 4
CASE 3
CASE 2
CASE 1
CASE 0

```

```

I = 10, M(I,1) = 1, M(I,5) = 0
I = 9, M(I,1) = 1, M(I,5) = 0
I = 8, M(I,1) = 1, M(I,5) = 0
I = 7, M(I,1) = 1, M(I,5) = 0
I = 6, M(I,1) = 1, M(I,5) = 0

```

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		
2.650	1.330	2.0
2		
3.990	1.330	3.0
3		
5.320	1.330	4.0
4		
6.650	1.330	.0
5		
7.980	1.330	.0
6		
9.310	1.330	.0
7		
10.640	1.330	.0
8		
11.970	1.330	.0
9		
13.30	1.330	.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

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.  
//
```

ESYST: PROCEDURE MAIN;

```

. /**:*****
/*
/* IDENTIFICATION:
/* -----
/*
/* PROGRAM-ID: ESYST.
/* AUTHOR: J. R. VAN DOREN.
/* SOURCE LANGUAGE: PLEX.
/* OBJECT LANGUAGE: PLEX PSEUDO-MACHINE CODE.
/* OBJECT INTERPRETER: PLXINT.
/*
/*
/* PURPOSE:
/* -----
/*
/* ESYST LISTS EASYCODER PROGRAMS AND BUILDS A CARD REFERENCE INDEX
/* FOR SYMBOL DEFINITIONS.
/*
/*
/**:*****

```

```

        DECLARE INPUT CHAR(80), (CRDCNT, I, SYMCNT) FIXED, SYMBOL(500) CHAR(6),
                (CRDREF(500), DECNT) CHAR(5);
        I, CRDCNT=0;
        SYMCNT=1;
GETCRD: GET EDIT(INPUT) (A(80));
        IF SUBSTR(INPUT, 1, 4) = "****" THEN GO TO SORT;
        CRDCNT=CRDCNT+10;
        PUT STRING(DECNT) EDIT (CRDCNT) (I(5));
        /* INSERT HIGH ORDER ZEROES */
        DO I=1 TO 3;
        IF SUBSTR(DECNT, I, 1) = " " THEN SUBSTR(DECNT, I, 1) = "0";
        END;
        PUT EDIT (DECNT, SUBSTR(INPUT, 6, 75)) (A(5), A(75));
        /* TEST FOR COMMENT CARD */
        IF SUBSTR(INPUT, 6, 1) = "*" THEN GO TO GETCRD;
        /* TEST FOR SYMBOL DEFINITION. ENTER SYMBOL AND CARD

```

```

        REFERENCE IF FOUND.                                */
        IF SUBSTR(INPUT,8,7)=" " THEN GO TO GETCRD;
        IF SUBSTR(INPUT,8,1)=" " THEN
        SYMBOL(SYMCNT)=SUBSTR(INPUT,9,6);
        ELSE SYMBOL(SYMCNT)=SUBSTR(INPUT,8,6);
        CRDREF(SYMCNT)=DECNT;
        SYMCNT=SYMCNT+1;
        GO TO GETCRD;
SORT:  BEGIN; DECLARE (I,J,K,L,M) FIXED, SYMTMP CHAR(6), REFTMP CHAR(5);
        SYMCNT=SYMCNT-1;
        M=SYMCNT;
        LBL20: M=M/2;
        IF M=0 THEN GO TO LBL40;
        K=SYMCNT-M;
        J=1;
        LBL41: I=J;
        LBL49: L=I+M;
        IF SYMBOL(I) < SYMBOL(L) THEN GO TO LBL60;
        SYMTMP=SYMBOL(I);
        REFTMP=CRDREF(I);
        SYMBOL(I)=SYMBOL(L);
        CRDREF(I)=CRDREF(L);
        SYMBOL(L)=SYMTMP;
        CRDREF(L)=REFTMP;
        I=I-M;
        IF I > 0 THEN GO TO LBL49;
        LBL60: J=J+1;
        IF J > K THEN GO TO LBL20; ELSE GO TO LBL41;
        END SORT;
        LBL40: PUT EDIT ("SYMBOL DEFINITION - CARD REFERENCE INDEX"," ")
        (SKIP(3),COL(20),A,SKIP(2),A);
        PUT EDIT ((SYMBOL(I),CRDREF(I)," " DO I=1 TO SYMCNT))
        (A(6),X(2),A(5),A,A(6),X(2),A(5),A,A(6),X(2),A(5),A,
        A(6),X(2),A(5),A,A(6),X(2),A(5),A,SKIP);
        STOP;
END ESYLST;

```

\*\*\*\*COMPILED PROGRAM SIZE = 1,397; METAX INSTRUCTION COUNT = 14,274\*\*\*\*  
\*\*\*\*SYMTAB SEARCH COUNT = 393; SYMTAB COMPARE COUNT = 4,819\*\*\*\*  
\*\*\*\*SYMBOL TABLE ENTRY COUNT = 54\*\*\*\*

// CONTROL RECORD.  
FUNCTION PLXCPL.  
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+I/14),I);
          *
          ***** ERROR ***** W:  INCORRECT PARM CNT
              J=I***-1;
              *
          ***** ERROR ***** F: SYNTAX
              END;
          END ERROR;
FATAL ERROR(S) ENCOUNTERED, JOB ABORTED

```

```

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

```

```

00380*      COMMUNICATION AREA FIELD LOCATION DEFINITIONS      *
00390*                                                         *
00400*****
00410  INSTCT EQU    209      INSTRUCTION COUNTER
00420  STKSTR EQU    223      STACK START
00430  STKEND EQU    227      STACK END
00440  LODLOC EQU    219      STARTING ADDRESS FOR EXECUTION
00450  DYNSTR EQU    231      STARTING ADDRESS FOR DYNAMIC STORAGE
00460  DYNEND EQU    235      ENDING ADDRESS FOR DYNAMIC STORAGE
00470*****
00480*                                                         *
00490*      INDEX REGISTER LOCATION DEFINITIONS AND USAGE DESCRIPTIONS      *
00500*                                                         *
00510*****
00520  IR1      EQU    4      CONVERSION SUBROUTINE USAGE AND WORK REGISTER
00530  IR2      EQU    8      PUSH DOWN STACK POINTER
00540  IR3      EQU   12      PROGRAM COUNTER
00550  IR4      EQU   16      RESULT REGISTER FOR ADDRESS COMPUTATION
00560  IR5      EQU   20      RESULT REGISTER FOR ADDRESS COMPUTATION
00570  IR6      EQU   24      INPUT BUFFER POINTER
00580  IR7      EQU   28      OPERATION CODE REGISTER
00590  IR8      EQU   32      OUTPUT BUFFER POINTER
00600  IR9      EQU   36      WORK REGISTER
00610  IR10     EQU   40      WORK REGISTER
00620  IR11     EQU   44      WORK REGISTER
00630  IR12     EQU   48      WORK REGISTER
00640  IR13     EQU   52      WORK REGISTER
00650  IR14     EQU   56      ACTIVE DISPLAY POINTER FOR DYNAMIC STORAGE
00660  IR15     EQU   60      CONVERSION SUBROUTINE USAGE AND WORK REGISTER
00670*****
00680*                                                         *
00690*      MACRO CALLS TO SOURCE LIBRARY TO ESTABLISH CONVERSION      *
00700*      SUBROUTINES      *
00710*                                                         *
00720*****
00730L      FB/FD 4,BD,      FLOATING BINARY TO FLOATING DECIMAL
00740L      FD/FB 4,DB,      FLOATING DECIMAL TO FLOATING BINARY

```

```

00750L      MUL/D 4,MLT,   DOUPLE PRECISION FLOATING BINARY MULTIPLY
00760L      DIV/D 4,DIV,   DOUPLE PRECISION FLOATING BINARY DIVIDE
00770L      INT   4,ITG,   FLOATING BINARY TO INTEGER BINARY (TRUNCATED)
00780      IFIX  EQU   INT   RESOLVE CONVERSION NAME MISMATCH
00790*****
00800*                                           *
00810*      PROGRAM INITIALIZATION:                     *
00820*      SET REGISTERS.                               *
00830*      INITIALIZE DYNAMIC STORAGE DISPLAY.          *
00840*                                           *
00850*****
00860      START  EQU   *
00870      CAM    60              SET FOUR CHAR ADDRESSING FOR EXECUTION
00880      SW     STKEND-2        WORD MARK FOR MOVING AND TESTING
00890      MCW    STKSTR,IR2      STACK POINTER
00900      SW     IR2-2
00910      SI     IR2              ITEM MARK FOR RIGHT MOVE
00920      MCW    LODLOC,IR3      INITIALIZE INSTRUCTION COUNTER
00930      SI     IR3              ACCOMMODATE RIGHT MOVE
00940      LCA    =4B0,IR5        PROPERLY
00950      LCA    =4B0,IR4        PUNCTUATE
00960      SI     IR4,IR5         INDEX REGISTERS IR4,IR5
00970      BS     IR7              CLEAR OP CODE REGISTER
00980      SW     IR7-1           SHORTEN FETCH ARITHMETIC
00990      LCA    =4B0,IR14
01000      MCW    DYNSTR,IR14     INITIALIZE ACTIVE STORAGE POINTER
01010      SI     IR14            ACCOMMODATE RIGHT MOVE
01020      SW     DYNEND-2        WORD MARK FOR COMPARISON
01030      MLWDR  IR14,3+X14      INIT DYNAMIC STOR STACK TOP POINTER
01040      MLWDR  IR14,7+X14      LEVEL ONE STORAGE POINTER
01050      MCW    : : ,OUTPUT+132 CLEAR PRINT BUFFER
01060      MCW    OUTPUT+132
01070      MCW    =1C21,OUTPUT    CARRIAGE CONTROL
01080      SI     INPUT+80        RESTORE LOST ITEM MARK ON INPUT BUFFER
01090L      :SKIP PRINT,57,      SKIP TO TOP OF PAGE
01100      LCA    +PRNTBF,IR8     OUTPUT BUFFER POINTER
01110      B      GETIPT          INITIALIZE INPUT BUFFER

```

```

0120*****
0130*
0140* INSTRUCTION FETCHING
0150*
0160*****
0170 FETCH BA =1B1,INSTCT
BS IR7-1
MRSD 0+X3,IR7
SAR IR3
BIM =4B5,IR7
MCW TVEC+4+X7,IR13
BCE 0+X13,TVEC+X7,00
B ADCOMP
EQU *
REP 8
01260
01250 TVEC
01240
01230
01220
01210
01200
01190
01180
01170
01160*****
01150*****
01140*****
01130*****
01120*****
01270
01280
01290
01300
01310
01320
01330
01340
01350
01360
01370
01380
01390
01400
01410
01420
01430
01440
01450
01460
01470
01480
00ERROR
00POPUP
00SWAP
00COMP
00STCKC
00JUMPF
00JUMPT
00JUMP
00JUMPA
00RETRN
00ENTPRO
00FLAG
00ST
00STO
01LD
01LDA
00ERROR
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 30
OP CODE 31
OP CODE 32
OP CODE 33
OP CODE 34
OP CODE 35
OP CODE 36
OP CODE 10
OP CODE 11
OP CODE 12
REP 5
03ALLO
03STKP
00DYNAM
00ERROR
OP CODE 10
OP CODE 11
OP CODE 12
EXECUTION ADDRESSES AND ADDRESS PARAMETER
ELSE DO ADDRESS COMPUTATION
IF NO ADDRESS COMPUTATION THEN EXECUTE
INSERT EXECUTION ADDRESS
SHIFT LEFT BY TV ENTRY SIZE
BUMP SEQUENCE COUNTER
INSERT OP CODE
CLEAR 2ND CHAR
INSTRUCTION COUNT

```

01490		00ADD	OP CODE 40
01500		00MULT	OP CODE 41
01510		00SUB	OP CODE 42
01520		00DIV	OP CODE 43
01530		00NEG	OP CODE 44
01540	REP	3	
01550		00ERROR	
01560		00FMT	OP CODE 50
01570		00GET	OP CODE 51
01580		00PUT	OP CODE 52
01590		00EDIT	OP CODE 53
01600	REP	4	
01610		00ERROR	
01620		00OR	OP CODE 60
01630		00AND	OP CODE 61
01640		00NOT	OP CODE 62
01650	REP	3	
01660		00ERROR	
01670		00INDXR	OP CODE 66
01680		00INDXA	OP CODE 67
01690		00CAT	OP CODE 70
01700		00SBSTR	OP CODE 71
01710	REP	4	
01720		00ERROR	
01730		00STOP	OP CODE 76
01740		00ERROR	LAST
01750	*****		
01760	*		
01770	ADDRESS COMPUTATIONS ARE PERFORMED BY THE ADCOMP		
01780	SUBROUTINE. ONE OR TWO ADDRESSES ARE COMPUTED		
01790	DEPENDING ON ADDRESSING PARAMETER IN TVEC WHICH IS		
01800	ADDRESSED BY THE OP CODE REGISTER IR7. RETURN		
01810	ADDRESS IN IR13.		
01820	*		
01830	*****		
01840	ADCOMP SST	0+X3,SType,70	SAVE STORAGE TYPE
01850	SST	0+X3,DType,07	SAVE DATA TYPE

01860		BCE	LITADD,STYPE,70	CHECK FOR LITERAL OPERAND
01870	LCOMP	BCE	NOCOMP,1+X3,00	CHECK STATIC LEVEL
01880		BS	IR9	CLEAR
01890		MRSD	1+X3,IR9	INSERT STORAGE LEVEL
01900		BA	IR9	SHIFT LEFT TWO
01910		BA	IR9	BITS FOR DISPLAY ADDRESSING
01920		BA	IR14,IR9	DISPLAY ADDRESS OF STORAGE LEVEL ADDR
01930		MRID	0+X9,IR4-3	LEVEL BASE ADDRESS
01940		BA	4+X3,IR4	ADD DISPLACEMENT
01950		BS	IP4-3	CLEAR HIGH BITS
01960		BA	=1B5,IR3	SEQUENCE COUNTER
01970		BCE	INDIRA,STYPE,60	CHECK FOR INDIRECT ADDRESS
01980		BCE	PRCADD,STYPE,20	CHECK FOR REMOTE PROCEDURE ADDR
01990		BCE	0+X13,TVEC+X7,01	IF ONE ADDRESS THEN RETURN
02000		BS	IR9	ELSE COMPUTE 2ND (DYNAMIC ONLY)
02010		MRSD	1+X3,IR9	
02020		BA	IR9	
02030		BA	IR9	
02040		BA	IR14,IR9	
02050		MRID	0+X9,IR5-3	
02060		BA	4+X3,IR5	
02070		BS	IR5-3	CLEAR HIGH BITS
02080		BA	=1B5,IR3	
02090		B	0+X13	RETURN
02100	LITADD	MRID	IR3-2,IR4-2	LITERAL
02110		BA	=1B1,IR4	ADDRESS
02120		MRIN	1+X3,0	
02130		SAR	IR3	SEQUENCE COUNTER
02140		E	0+X13	RETURN
02150	PRCADD	MRID	0+X4,IR4-3	DO IT TWICE FOR REMOTE PROCEDURE
02160	INDIRA	MRID	0+X4,IR4-3	GET ACTUAL ADDRESS
02170		B	0+X13	RETURN
02180	NOCOMP	MRID	2+X3,IR4-2	INSERT STATIC STORAGE ADDRESS
02190		SAR	IR3	SEQUENCE COUNTER
02200		B	0+X13	RETURN
02210	STYPE	DCW	:0:	
02220	DTYPE	DCW	:0:	

```

02230*****
02240*
02250*   DYNAM  -  ALLOCATE FIXED DYNAMIC STORAGE (PROCEDURE ENTRY)
02260*
02270*****
02280   DYNAM  BA    3+X3,3+X14    INCREMENT STACKTOP
02290          C    3+X14,DYNEND   CHECK FOR DYNAMIC STORAGE
02300          BL    DYNOWL        OVERFLW
02310          BA    =1B4,IR3      INSTRUCTION COUNTER
02320          B     FETCH
02330*****
02340*
02350*   STKTP  -  INITIALIZE NEW STACKTOP (BLOCK ENTRY)
02360*
02370*****
02380   STKTP  MRIDR 0+X4,0+X5
02390          B     FETCH
02400*****
02410*
02420*   ALLOC  -  SET DOPE VECTOR FOR INDICATED ARRAY
02430*           IN DYNAMIC STORAGE. ALLOCATE DYNAMIC
02440*           STORAGE FOR THE ARRAY ITSELF.
02450*           NOTE THAT ADCOMP IS CALLED TO COMPUTE THE ADDRESSES
02460*           OF LOWER AND UPPER BOUND SUBSCRIPT LIMITS.
02470*
02480*****
02490   ALLOC  MRSD  0+X3,IR10      INSERT SUBSCRIPT COUNT
02500          SAR   IR3
02510          MLWD  IR4,IR11      SAVE LOCATION OF STACK TOP VALUE
02520          MLWD  IR5,IR12      SAVE DYNAMIC DOPE VECTOR ADDRESS
02530          MRIDI 0+X3,LENGTH-1 MOVE IN ELEMENT LENGTH SIZE
02540          SAR   IR3
02550          MCW   LENGTH,SIZE    INIT ARRAY SIZE COUNTER
02560          BS    CONPRT         CLEAR
02570          MCW   =2C0120,IR7   SET OP CODE REGISTER TO ONE ADDRESS CODE
02580   MORSUB MCW   +ALCRT1,IR13  RETURN ADDRESS FROM ADCOMP
02590          B      ADCOMP        CALL ADCOMP

```



02600	ALCRT1	MCW	+ALCRT2,IR13	RETURN ADDRESS FROM SECOND CALL
02610		MCW	IR4,IR5	SAVE FIRST ADDRESS
02620		B	ADCOMP	CALL ADCOMP
02630	ALCRT2	MRIDR	0+X4,0+X12	MOVE UPPER BOUND TO DYNAMIC D.V.
02640		SBR	IR12	NEXT MULTIPLIER LOCATION
02650		MCW	IR5,IR4	RESTORE FIRST ADDRESS
02660		SW	0-4+X12	PUNCTUATION TO STOP ARITHMETIC
02670		BS	3+X4,0-1+X12	SUBTRACT LOWER BOUND
02680		BA	=1B1,0-1+X12	FINISH MULTIPLIER
02690		BIM	0-1+X12,SIZE	UPDATE ARRAY SIZE COMPUTATION
02700		BIM	0-1+X12,CONPRT	UPDATE CONSTANT PART COMPUTATION
02710		BA	3+X4,CONPRT	ADD IN LOWER BOUND
02720		BS	=1B1,IR10	
02730		BCE	DVDNE,IR10,00	
02740		B	MORSUB	
02750	DVDNE	MRIDR	LNGLTH-3,0+X12	ELEMENT LENGTH FACTOR
02760		SBR	IR12	
02770		MRIDR	0+X11,0+X12	BASE ARRAY ADDRESS
02780		BIM	LNGLTH,CONPRT	LENGTH FACTOR
02790		BS	CONPRT,3+X12	FINISH CONSTANT PART
02800		BA	SIZE,3+X11	BUMP STACK TOP LOCATION BY ARRAY SIZE
02810		C	3+X11,DYNEND	CHECK FOR DYNAMIC STORAGE ALLOC OVERFLOW
02820		BL	DYNOFL	
02830		B	FETCH	
02840	LNGLTH	DCW	=4B0	
02850	SIZE	DCW	=4B0	
02860	CONPRT	DCW	=4B0	
02870*	*****			*****
02880*				*
02890*	LDA	-	LOAD ADDRESS TO THE STACK, INCLUDING THE	*
02900*			DATATYPE CODE. IF A CHARACTER STRING ALSO LOAD	*
02910*			THE MAX STRING LENGTH.	*
02920*				*
02930*	*****			*****
02940	LDA	MLWDR	IR4,4+X2	ADDRESS TO STACK
02950		LCA	DTYPE,0+X2	DATA TYPE
02960		BCE	LDLNG,DTYPE,04	IF A CHARACTER STRING LOAD LENGTH

02970	UPSTCK	BA	=1B9,IR2	STACK POINTER
02980		B	FETCH	ELSE FETCH NEXT INSTRUCTION
02990	LDLNG	BCE	LITLNG,STYPE,70	IF A LITERAL COMPUTE LENGTH
03000		MRIDR	0+X3,5+X2	ELSE EXTRACT FROM INSTRUCTION
03010		SAR	IR3	BUMP SEQUENCE COUNTER
03020		B	UPSTCK	
03030	LITLNG	BS	IR5	
03040		MRIN	0+X4,0	
03050		SAR	IR5	
03060		BS	IR4	
03070		MRIDR	IR5-1,5+X2	
03080		B	UPSTCK	
03090	*****			
03100	*			
03110	LD	- LOAD AN OPERAND TO THE STACK, INCLUDING DATA TYPE.		*
03120	*			
03130	*****			
03140	LD	MRIDR	0+X4,1+X2	LOAD DATA
03150		BCE	CHKTYP,DTYPE,07	IF UNDETERMINED TYPE CHECK IT
03160		LCA	DTYPE,0+X2	SET TYPE
03170		B	UPSTCK	
03180	CHKTYP	BI	3+X4,SETINT	INTEGER
03190		BI	7+X4,SETFLT	FLOAT
03200		B	TYPERR	
03210	SETINT	LCA	=1B1,0+X2	
03220		B	UPSTCK	
03230	SETFLT	LCA	=1B2,0+X2	
03240		B	UPSTCK	
03250	*****			
03260	*			
03270	STO	-STORES THE ITEM AT THE TOP OF THE STACK AT THE ADDRESS		*
03280		NEXT TO THE TOP. NOTE SPECIAL HANDLING OF CHARACTER		*
03290		STRINGS. POP TWO STACK ELEMENTS UPON COMPLETION.		*
03300	*			
03310	SST	-SAME AS STO EXCEPT ADDRESS IS DISCARDED BUT THE		*
03320		STACK TOP IS RETAINED.		*
03330	*			

```

03340 *****
03350 STO EQU *
03360 SST EQU *
03370 BCE CNVRTN,0-18+X2,04 BYPASS IF STRING OR SUBSTRING
03380 BCE CNVRTN,0-18+X2,34
03390 C 0-9+X2,0-18+X2 IF TYPES NOT EQUAL
03400 BNE SCNV DO CONVERSION
03410 CNVRTN MCW 0-14+X2,IR4 RECEIVING ADDRESS
03420 BCE CHRSTR,0-9+X2,04 SPECIAL HANDLING IF CHARACTER STRING
03430 BCE CSBSTR,0-9+X2,34 CHECK FOR SENDING SUBSTRING
03440 MRIDR 0-8+X2,0+X4 STORE IT
03450 SSTCHK BCE PRMOVE,0-9+X2,34
03460 PNCRTN BCE SAVTOP,0-1+X3,23 CHECK FOR SAVE AND STORE (SST)
03470 BS =1B18,IR2 POP STACK TWO ELEMENTS
03480 MCW +ENDPRG,ENDADR RESTORE STRING WORKING STORAGE POINTER
03490 B FETCH
03500 SAVTOP SI 0-9+X2 PUNCTUATION FOR MOVE
03510 MLRDR 0-1+X2,0-10+X2 DISCARD ADDRESS
03520 CI 0-9+X2,0-18+X2 CLEAR ITEM MARKS
03530 BS =1B9,IR2 AND SAVE STACK TOP ON TOP
03540 B FETCH
03550 SCNV BCE CNVRTN,0-18+X2,07 UNIVERSAL TYPE O.K.
03560 BCE FIXFLT,0-9+X2,01
03570 TMA 0-1+X2,00
03580 B IFIX GO CONVERT
03590 R B CNVERR CONVERSION OVERLOW INSTRUCTION
03600 MCW IR15,0-5+X2 INTEGER RESULT
03610 SI 0-5+X2 TO
03620 LCA =1B1,0-9+X2 STACK
03630 B CNVRTN
03640 FIXFLT MLWDI =4C00000027,0-1+X2
03650 AMA 0-1+X2,70 NORMALIZED LOAD
03660 MLWDI =8B0,0-1+X2 CLEAR EXTRANEIOUS PUNCTUATION
03670 TAM 0-1+X2,00 FLOATING
03680 SI 0-1+X2 RESULT
03690 LCA =1B2,0-9+X2 TO STACK
03700 B CNVRTN

```

03710	CHRSTR	BS	IR5	
03720		MCW	=1C10,IR5	MAX LENGTH FOR UNIVERSAL TYPE
03730		BCE	CHRTRN,0-18+X2,07	CHECK FOR UNIVERSAL TYPE
03740		MRID	0-13+X2,IR5-1	MAX LENGTH OF RECEIVING FIELD
03750	CHRTRN	MCW	0-5+X2,IR9	TRANSMITTING ADDRESS
03760		SW	0+X9	POTENTIAL LEFT MOVE MARKER
03770		BS	IR10	CLEAR
03780		MRIN	0+X9,0+X4	
03790		SBR	IR10	END OF PROPOSED MOVE
03800		EA	IR4,IR5	RECEIVING ADDRESS PLUS MAX LENGTH
03810		C	IR5,IR10	IF MOVE TOO
03820		BH	TMOVE	LONG THEN TRUNCATE IT
03830		BCE	SUBMV,0-18+X2,34	IF SUB STRING RECEIVING BE CAREFUL
03840		MRIDR	0+X9,0+X4	MOVE ENTIRE TRANSMITTING FIELD
03850		B	SSTCHK	
03860	SUBMV	MRID	0+X9,0+X4	MOVE ONLY DATA, NO PUNCTUATION
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	CSBSTR	EQU	*	TRANSMITTING FIELD IS A SUBSTRING
03940		MCW	0-5+X2,IR15	SET
03950		BS	IR9	ITEM
03960		MRID	0-4+X2,IR9-1	MARK
03970		BA	IR9,IR15	ON
03980		BI	NOSET,0-1+X15	RIGHT END
03990		SI	0-1+X15	
04000		B	CHRSTR	
04010	NOSET	SST	=1B0,0-9+X2,70	REMOVE SUBSTRING MARKER
04020		B	CHRSTR	
04030	PRMOVE	CI	0-1+X15	REMOVE ITEM MARK
04040		MCW	0-5+X2,IR9	
04050		CW	0+X9	REMOVE WORK MARK
04060		B	PNCRTN	
04070	*****			

```

04080*
04090*   FLAG- MARK STACK FOR THE LIMIT OF PARAMETER ADDRESSES
04100*   PRIOR TO A PROCEDURE CALL
04110*
04120*****
04130  FLAG   LCA   =1C77,0+X2
04140       B     UPSTCK
04150*****
04160*
04170*   ENTPRO- ESTABLISH DYNAMIC   STORAGE DISPLAY,STORE PARAMETER
04180*   ADDRESSES IN NEW DYNAMIC STORAGE AREA AND THEN PASS
04190*   CONTROL TO THE CALLED PROCEDURE.
04200*
04210*****
04220  ENTPRO MRID  0-8+X2,IR14-3      NEW DISPLAY AND STORAGE AREA
04230       BS    =1B9,IR2
04240       BS    IR13
04250       MRSD  0+X3,IR13          LEVEL NUMBER OF PROCEDURE
04260       SAR   IR3                BUMP SEQUENCE COUNTER
04270       BIM   =4B4,IR13
04280       BA    =1B4,IR13
04290       BA    IR14,IR13          LOCATION FOR PARAMETER ADDRESSES
04300       BS    PRMCNT
04310  FLGCHK BS    =1B9,IR2          SEARCH DOWN
04320       BCE   PRMSTO,0+X2,77     AND
04330       BA    =1B1,PRMCNT        COUNT
04340       B     FLGCHK             PARM5
04350  PRMSTO MCW   IR2,IR9
04360  PRMCHK BCE   PRMDNE,PRMCNT,00 IF DONE GO ELSEWHERE
04370       MRIDR 10+X9,0+X13        MOVE ADDRESS TO DYNAMIC STORAGE
04380       SBR   IR13
04390       BA    =1B9,IR9
04400       BS    =1B1,PRMCNT
04410       B     PRMCHK
04420  PRMDNE MRID  0-8+X2,IR9-3     GLOBAL DISPLAY ADDRESS
04430       BA    =1B4,IR9
04440       MRIDR 0-17+X2,0-22+X2    PACK CALLING PROC DISPLAY ADDRESS WITH

```

```

04450      BS      =1B18,IR2      RETURN ADDRESS
04460      MRIDR   IR14-3,0+X14    1ST DISPLAY ENTRY (STACK TOP VALUE)
04470      SBR     IR12            NEXT DISPLAY LOCATION
04480      MRSD    0-1+X3,IR13
04490  DISPLY MRIDR 0+X9,0+X12      MOVE GLOBALLY ACTIVE STORAGE LEVEL
04500      SAR     IR9             ADDRESSES TO CURRENT DISPLAY
04510      SBR     IR12
04520      BS      =1B1,IR13
04530      BCE     CURLEV,IR13,01
04540      B        DISPLY
04550  CURLEV MRIDR IR14-3,0+X12    CURRENT DISPLAY BASE ADDRESS
04560      B        FETCH
04570  PRMCNT DCW   =1B0
04580*****
04590*
04600*      SWAP - SWAP THE TOP TWO ELEMENTS OF THE STACK
04610*
04620*****
04630  SWAP      SI      0-9+X2,0-18+X2
04640      MLRDR   0-1+X2,HOLD
04650      MLRDR   0-10+X2,0-1+X2
04660      MLRDR   HOLD,0-10+X2
04670      CI      0-9+X2,0-18+X2
04680      B        FETCH
04690  HOLD      DCW     =9
04700*****
04710*
04720*      RETURN - RESTORE DISPLAY ADDRESS FOR CALLING PROCEDURE
04730*      AND SET INSTRUCTION COUNTER WITH RETURN ADDRESS
04740*
04750*****
04760  RETRN      MRID    0-8+X2,IR3-3      RETURN ADDRESS TO INST COUNTER
04770      MRID    0-4+X2,IR14-3          DISPLAY AT POINT OF CALL
04780      BS      =1B9,IR2
04790      B        FETCH
04800*****
04810*

```

```

04820*      JUMPA - JUMP TO THE ADDRESS IN THE STACK; POP THE STACK      *
04830*      POPUP - POP THE STACK      *
04840*      *
04850*      ***** SET INSTRUCTION COUNTER *****
04860      JUMPA MRID 0-8+X2,IR3-3
04870      POPUP BS =1B9,IR2
04880      B      FETCH
04890*      *****
04900*      *
04910*      JUMP - JUMP TO THE ADDRESS FOLLOWING THE OP CODE
04920*      JUMPT - JUMP IF TOP OF STACK IS TRUE; POP THE STACK      *
04930*      JUMPF - JUMP IF TOP OF STACK IS FALSE; POP THE STACK      *
04940*      *
04950*      *****
04960      JUMP MRID 0+X3,IR3-3
04970      B      FETCH
04980      JUMPT BCE JUMPP,0-8+X2,T
04990      BA     =1B4,IR3
05000      E      POPUP
05010      JUMPP MRID 0+X3,IR3-3
05020      B      PCPOP
05030      JUMPF BCE JUMPP,0-8+X2,F
05040      BA     =1B4,IR3
05050      B      POPUP
05060*      *****
05070*      *
05080*      STCKC - TESTS 2ND STACK ITEM AGAINST 1ST ACCORDING TO THE LITERAL *
05090*      CHARACTER FOLLOWING THE OP CODE.      *
05100*      RESULTS IN A BOOLEAN VALUE ON TOP OF THE STACK.      *
05110*      *
05120*      *****
05130      STCKC BCE CHK2ND,0-9+X2,02
05140      MLWDI =4C00000027,0-1+X2      CONVERT TO FLOATING INTEGER
05150      CHK2ND BCE SETCND,0-18+X2,02
05160      MLWDI =4C00000027,0-10+X2      CONVERT TO FLOATING INTEGER
05170      SETCND MRSD 0+X3,COND-1      SET CONDITION TO BE TESTED
05180      SAR      IR3      BUMP INSTRUCTION COUNTER

```

05190		AMA	0-10+X2,70	LOAD
05200		SMA	0-1+X2,00	SUBTRACT
05210	COND	FBA	SETT,00	BRANCH ON FLOATING REGISTER COND
05220	SETF	MCW	:F:,0-17+X2	
05230		SI	0-17+X2	PUNCTUATION FOR POSSIBLE STORE
05240		MCW	=1C03,0-18+X2	SET STACK
05250		B	POPUP	
05260	SETT	MCW	:T:,0-17+X2	SET STACK
05270		SI	0-17+X2	PUNCTUATION FOR POSSIBLE STORE
05280		MCW	=1C03,0-18+X2	
05290		B	POPUP	
05300*****				
05310*				*
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*			PADDED ON THE RIGHT WITH BLANKS.	*
05360*			THE COMPARISON CONDITION IS CONTAINED IN THE CHARACTER	*
05370*			FOLLOWING THE OP CODE.	*
05380*				*
05390*****				
05400	COMPC	BS	IR9	CLEAR
05410		BS	IR10	
05420		BS	IR15	
05430		MCW	0-14+X2,IR13	LEFT ADDRESSES OF STRINGS
05440		BCE	SUBMV1,0-18+X2,34	IF SUBSTRING PUT IN WORKING STORE
05450	MVRT1	MCW	0-5+X2,IR12	TO INDEX REGISTERS
05460		BCE	SUBMV2,0-9+X2,34	IF SUBSTRING PUT IN WORKING STORE
05470	MVRT2	MRIN	0+X13,0	
05480		SAR	IR9	DETERMINE
05490		MRIN	0+X12,0	
05500		SAR	IR10	STRING
05510		BS	IR13,IR9	
05520		BS	IR12,IR10	LENGTHS
05530		SW	IR9-2	
05540		C	IR10,IR9	TEST
05550		CW	IR9-2	



05560		BL	MOVNXT	
05570		BH	MOVTOP	
05580	TSTRNG	BS	IR15	SET PROPER
05590		MRSD	0+X3,IR15	CONDITION
05600		SAR	IR3	
05610		MRSD	CNDTBL+X15,CNDTST	CODE
05620		SW	0+X13	WORK MARK FOR COMARE
05630		MRIN	0+X13,0+X12	POSITION
05640		SAR	IR13	INDEX REGISTERS
05650		SBR	IR12	TO RIGHT END
05660		MCW	+ENDPRG,ENDADR	RESTORE STRING WORKING STORAGE POINTER
05670		C	0-1+X12,0-1+X13	COMPARE
05680	CNDTSTBCT		SETT,00	CONDITIONAL BRANCH
05690		B	SETF	
05700	MOVNXT	EQU	*	
05710		MRIDR	0+X13,(ENDADR-3)	MOVE TO TEMPORARY LOCATION
05720		SBR	IR15	
05730		CI	0-1+X15	
05740		MCW	ENDADR,IR13	
05750		MCW	ENDADR,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	ENDADR,IR12	
05880		MCW	ENDADR,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

05930	SAR	IR9	REMEMBER WORD MARK
05940	BA	IR10,IR13	RIGHT END (PLUS ONE)
05950	BA	ENDADR,IR10	
05960	MLWDR	0-1+X13,0-1+X10	
05970	SI	0-1+X10	RIGHT END ITEM MARK
05980	MCW	ENDADR,IR13	ADDRESS IN WORKING STORE
05990	MCW	IR10,ENDADR	NEXT WORKING STORE LOCATION
06000	CW	0+1+X9	REMOVE WORD MARK
06010	B	MVRT1	
06020	ENDADR D5A	ENDPRG	
06030	SUBMV2 B5	IR10	
06040	MRID	0-4+X2,IR10-1	
06050	SW	0+X12	
06060	SAR	IR9	
06070	BA	IR10,IR12	
06080	BA	ENDADR,IR10	
06090	MLWDR	0-1+X12,0-1+X10	
06100	SI	0-1+X10	RIGHT END ITEM MARK
06110	MCW	ENDADR,IR12	
06120	MCW	IR10,ENDADR	
06130	CW	0+1+X9	
06140	B	MVRT2	
06150	CNDTBLDCW	=8C4042414344464547	
06160	BLNK DC	: :	
06170	*****		
06180	*		
06190	ADD	-REPLACE THE TOP TWO NUMBERS ON THE STACK WITH THEIR SUM	*
06200	SUB	-REPLACE THE TOP TWO NUMBERS ON THE STACK WITH THEIR	*
06210		DIFFERENCE (2ND - 1ST)	
06220	MULT	-REPLACE THE TOP TWO NUMBERS ON THE STACK WITH THEIR PRODUCT	*
06230	DIV	-REPLACE THE TOP TWO NUMBERS ON THE STACK WITH THEIR	*
06240		RATIO ( 2ND / 1ST )	
06250	NEG	-UNARY MINUS OPERATION ON THE ELEMENT ON THE TOP OF THE STACK	
06260	NOTE	THE TYPE CONVERSIONS WHICH MAY TAKE PLACE.	
06270	*		
06280	*****		
06290	ADD	B CNVCHK	

06300		BCE	INTADD,ARI,01	
06310		TMA	0-1+X2,00	STACK TOP TO FRO
06320		AMA	0-10+X2,00	ADD NEXT
06330	SETBCK	FBI	FERR,06	BRANCH ON ANY ERROR
06340		BS	=1B9,IR2	REDUCE STACK
06350		MLWDR	=8B0,0-1+X2	CLEAR EXTRANEIOUS PUNCTUATION
06360		SI	0-1+X2	PROPER PUNCTUATION
06370		TAM	0-1+X2,00	STORE RESULT
06380		LCA	=1B2,0-9+X2	DATA TYPE
06390		B	FETCH	
06400	INTADD	SW	0-17+X2	WORD MARK TO STOP ADDITION
06410		BA	0-5+X2,0-14+X2	BINARY ADD
06420		BS	=1B9,IR2	REDUCE STACK
06430		B	FETCH	
06440	SUB	B	CNVCHK	
06450		BCE	INTSUB,ARI,01	IF INTEGERS BRANCH
06460		TMA	0-10+X2,00	FLOATING
06470		SMA	0-1+X2,00	POINT
06480		B	SETBCK	
06490	INTSUB	SW	0-17+X2	WORD MARK TO STOP ARITHMETIC
06500		BS	0-5+X2,0-14+X2	INTEGER SUBTRACTION
06510		BS	=1B9,IR2	
06520		B	FETCH	
06530	MULT	B	CNVCHK	
06540		BCE	INTMLT,ARI,01	IF INTEGERS BRANCH
06550		TMA	0-1+X2,00	FLOATING
06560		MAM	0-10+X2,00	POINT
06570		B	SETBCK	
06580	INTMLT	BIM	0-5+X2,0-14+X2	
06590		BS	=1B9,IR2	
06600		B	FETCH	
06610	DIV	MRSD	0-9+X2,ARI	
06620		BA	0-18+X2,ARI	SUM OF TYPES FOR LATER TESTING
06630		BCE	NUMER,0-9+X2,02	
06640		MLWDI	=4C00000027,0-1+X2	CONVERT TO UNNORMALIZED FLTNG PT
06650	NUMER	BCE	DODIV,0-18+X2,02	
06660		MLWDI	=4C00000027,0-10+X2	CONVERT TO UNNORMALIZED FLTNG PT

06670	DODIV	AMA	0-1+X2,71	NORMALIZED LOAD TO FR1
06680		AMA	0-10+X2,70	NORMALIZED LOAD TO FRO
06690		DAA	10	RESULT IN FRO
06700		BCE	INTDIV,ARI,02	CHECK FOR INTEGER RESULT
06710		B	SETBCK	
06720	INTDIV	B	IFIX	GO CONVERT
06730	R	B	CNVERR	CONVERSION OVERFLOW INSTRUCTION
06740		MLWD	IR15,0-14+X2	PUT RESULT IN STACK
06750		BS	=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	SBR	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	IFALL INTEGER RETURN
06900		BCE	(CNVTR+1),ARI,02	IF ALL FLTNG PT RETURN
06910		BCE	CNVTOP,0-9+X2,01	DO
06920		MLWDI	=4C00000027,0-10+X2	NECESSARY
06930		B	(CNVTR+1)	CONVERSION
06940	CNVTOP	MLWDI	=4C00000027,0-1+X2	
06950	CNVTR	B	*	
06960	*****			
06970	*			
06980	OR	-REPLACE TOP TWO ELEMENTS WITH LOGICAL SUM		*
06990	AND	-REPLACE TOP TWO ELEMENTS WITH LOGICAL PRODUCT		*
07000	NOT	-LOGICAL NEGATION OF TOP ELEMENT		*
07010	*			
07020	*****			
07030	OR	BS	=1B9,IR2	

07040		BCE	BSTT,1+X2,T	
07050		B	FETCH	ALREADY PROPERLY TRUE OR FALSE
07060	AND	BS	=1B9,IR2	
07070		BCE	BSTF,1+X2,F	
07080		B	FETCH	ALREADY PROPERLY TRUE OR FALSE
07090	NOT	BCE	BSTF,0-8+X2,T	
07100	BSTT	MCW	:T:,0-8+X2	SET TRUE ON STACK
07110		SI	0-8+X2	PUNCTUATION FOR POSSIBLE STORE
07120		B	FETCH	
07130	BSTF	MCW	:F:,0-8+X2	SET FALSE ON STACK
07140		SI	0-8+X2	PUNCTUATION FOR POSSIBLE STORE
07150		B	FETCH	
07160	*****			
07170	*			*
07180	INDXA-COMPUTE ADDRESS ACCORDING TO DOPE VECTOR (ADDRESS			*
07190	INSTACK) AND INDECES ON THE STACK. RESULTING ADDRESS			*
07200	ON TOP OF THE STACK.			*
07210	INDXR-SAME AS INDXA EXCEPT THAT ADDRESSED ITEM IS LOADED			*
07220	TO THE STACK.			*
07230	*			*
07240	*****			
07250	INDXA	EQU	*	
07260	INDXR	EQU	*	
07270		BS	IR13	CLEAR
07280		MRSD	0+X3,IR13	INSERT SUBSCRIPT COUNT
07290	SUBCHK	BCE	INTSBS,0-9+X2,01	CHECK FOR
07300		TMA	0-1+X2,00	SUBSCRIPT CONVERSION
07310		B	IFIX	
07320	R	B	CNVERR	
07330		MCW	IR15,0-5+X2	MOVE INTEGER RESULT TO THE STACK
07340	INTSBS	BS	=1B9,IR2	
07350		BS	=1B1,IR13	
07360		BCE	CMPADD,IR13,00	
07370		B	SUBCHK	
07380	CMPADD	MCW	IR2,IR9	SAVE STACK POINTER
07390		MCW	0-5+X2,IR10	ADDRESS OF DOPE VECTOR
07400		MRSD	0+X3,IR13	SUBSCRIPT COUNT

07410		SAR	IR3	
07420		MRI0	1+X2,IR15-3	FIRST SUBSCRIPT
07430	MORIDX	BS	=1B1,IR13	REDUCE SUBSCRIPT COUNT
07440		BCE	SUBDNE,IR13,00	IF NO MORE THEN FINISH
07450		BA	=1B4,IR10	POINT TO NEXT MULTIPLIER
07460		BA	=1B9,IR2	NEXT SUBSCRIPT
07470		BIM	3+X10,4+X2	SUBSCRIPT * MULTIPLIER
07480		BA	4+X2,IR15	ACCUMLATE 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		MLWD	IR15,0-5+X2	ADDRESS TO STACK
07560		BCE	FETCH,0-2+X3,67	IF INDXA THEN DONE
07570		MRIDR	0+X15,0-8+X2	ELSE LOAD RESULT
07580		B	FETCH	
07590	*****			
07600	*			
07610	CAT	-CONCATENATE TWO STRINGS WHOSE ADDRESSES ARE IN THE TOP		*
07620		TWO POSITIONS OF THE STACK. THE RESULTING STRING IS PLACED		*
07630		IN WORKING STRING STORAGE AND THE TWO ADDRESSES ARE REPLACED		*
07640		BY THE ADDRESS IN WORKING STORAGE.		*
07650	*			
07660	*****			
07670	CAT	MCW	0-14+X2,IR13	ADDRESS TO REGISTER
07680		BCE	CTSUB1,0-18+X2,34	CHECK FOR SUBSTRING
07690		MRIDR	0+X13,(ENDADR-3)	
07700		SBR	IR12	
07710		CI	0-1+X12	CLEAR ITEM MARK
07720	CAT2	MCW	0-5+X2,IR13	
07730		BCE	CTSUB2,0-9+X2,34	CHECK FOR SUBSTRING
07740		MRIDR	0+X13,0+X12	
07750		SBR	IR13	NEXT LOCATION IN WORKING STORAGE
07760	CAT3	CW	0+X12	CLEAR POSSIBLE WORD MARK
07770		MCW	ENDADR,0-14+X2	ADDRESS OF RESULTING STRING

07780		MCW	IR13,ENDADR	NEXT LOCATION IN WORKING STORAGE
07790		BS	=1B9,IR2	POP STACK
07800		B	FETCH	
07810	CTSUB1	BS	IR10	
07820		MRID	0-13+X2,IR10-1	MOVE
07830		BA	IR13,IR10	SUBSTRING
07840		SI	0-1+X10	TO
07850		MRIDR	0+X13,(ENDADR-3)	WORKING
07860		SBR	IR12	STORAGE
07870		CI	0-1+X12	
07880		CI	0-1+X10	
07890		SW	(ENDADR-3)	
07900		SST	=1B0,0-18+X2,70	REMOVE SUB STRING MARK
07910		B	CAT2	
07920	CTSUB2	BS	IR10	
07930		MRID	0-4+X2,IR10	CATENATE
07940		BA	IR13,IR10	SUBSTRING
07950		SI	0-1+X10	TO
07960		MRIDR	0+X13,0+X12	WORKING
07970		SBR	IR13	NEXT LOCATION IN WORKING STORAGE
07980		CI	0-1+X10	STORAGE
07990		B	CAT3	
08000	*****			
08010	*			
08020	SUBSTR - PL/I SUBSTRING OPERATION *			
08030	*			
08040	*****			
08050	SBSTR	SW	0-26+X2	COMPUTE
08060		BA	0-14+X2,0-23+X2	STARTING
08070		BS	=1B1,0-23+X2	POINT OF STRING
08080		C	0-5+X2,:00:	IF NO LENGTH GIVEN
08090		BE	FNDLNG	COMPUTE IT
08100		MRIDI	0-6+X2,0-22+X2	ELSE STORE LENGTH
08110	SBSTRL	BS	=1B18,IR2	POP STACK
08120		LCA	=1C34,0-9+X2	MARK AS SUBSTRING ELEMENT
08130		B	FETCH	
08140	FNDLNG	MCW	0-23+X2,IR13	FIND

```

08150      MRIN  0+X13,0      CURRENT REMAINING
08160      SAR   IR4          LENGTH
08170      BS    IR13,IR4     OF
08180      MRIDI IR4-1,0-22+X2 STRING
08190      B      SBSTRL
08200*****
08210*
08220*      FMT -SETS THE ADDRESS OF THE FORMAT CODE AND INDICATORS
08230*      FOR STRING OR UNIT RECORD AND WHETHER INPUT OR OUTPUT.
08240*      IF STRING I/O THE ADDRESS AND LENGTH ARE EXTRACTED FROM
08250*      THE STACK AND THE APPROPRIATE BUFFER POINTER IS SET TO
08260*      THE BEGINNING OF THE STRING.
08270*
08280*****
08290  FMT      MRID  0+X3,FMTCD1-3      SET FORMAT CODE ADDRESS
08300      SAR   IR3          BUMP SEQUENCE COUNTER
08310      BA    =1B1,FMTCD1      JUMP OVER BEGINNING MARKER
08320      MCW   FMTCD1,FMTCD2      SAVE FOR REPETITION
08330      SST   0+X3,INOROT,70      INPUT OR OUTPUT CODE
08340      SST   0+X3,DEVTYP,07      STRING OR UNIT RECORD
08350      BA    =1B1,IR3          BUMP SEQUENCE COUNTER
08360      BCE   STRSET,DEVTYP,00      IF STRING INITIALIZE
08370      BCE   FETCH,INOROT,70      ELSE INITIALIZE FOR UNIT RECORD
08380      MCW   +PRNTBF,IR8        OUTPUT PRINT BUFFER
08390      MCW   +PRNTBF+132,STREND
08400      B      FETCH
08410  FMTCD1  DCW   =4
08420  FMTCD2  DCW   =4
08430  INOROT  DCW   :0:
08440  DEVTYP  DCW   :0:
08450  STRADD  DCW   =4
08460  LNGSTR  DCW   =4B0
08470  STREND  DCW   =4
08480  STRSET  MRID  0-8+X2,STRADD-3      STRING ADDRESS
08490      MRID  0-4+X2,LNGSTR-1      LENGTH
08500      MCW   STRADD,STREND
08510      BA    LNGSTR,STREND      STRING END ADDR (PLUS ONE)

```



```

08520      BS      =1B9,IR2      POP STACK
08530      BCE      FETCH,INROT,70  IF INPUT STRING ALL DONE
08540      MCW      STRADD,IR8      ELSE SET OUTPUT POINTER
08550      MCW      STRADD,IR9
08560  CLRMOR  MRSDR  BLNK,0+X9      CLEAR OUTPUT
08570      SBR      IR9            STRING
08580      C        IR9,STREND      TO
08590      BH      CLRMOR          BLANKS
08600      SI      0-1+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 PUNCUATION      *
08670*      AND RESET BUFFER POINTER TO PRINTER BUFFER.            *
08680*                                     *
08690*****
08700  PUT      BCE      STRPUT,DEVTYP,00
08710L      :PUT      PRINT,
08720      MCW      : : ,OUTPUT+132      CLEAR PRINT
08730      MCW      OUTPUT+132          BUFFER
08740      MCW      =1C21,OUTPUT        CARRIAGE CONTROL
08750      SI      INPUT+80             RESTORE ITEM MARK ON INPUT BUFFER
08760  PBUFF  MCW      +PRNTBF,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,(FMTC1-3),77      CHECK IF RESET IS REQUIRED
08880      BBE      DATFMT,(FMTC1-3),70      CHECK FOR DATA FORMAT

```

08890		BCE	SPACE,(FMTCD1-3),00	ELSE IT MUST BE CONTROL FORMAT
08900		BCE	PAGE,(FMTCD1-3),01	
08910		BCE	SKIP,(FMTCD1-3),02	
08920		BCE	COLMN,(FMTCD1-3),03	
08930		B	FMTERR	
08940	UPCDPT	BA	=1B1,FMTCD1	
08950		E	EDIT	
08960	SPACE	BA	=1B4,FMTCD1	SPACE THE REQUIRED
08970		BA	(FMTCD1-3),IR8	NUMBER OF COLUMNS
08980		B	UPCDPT	
08990	COLMN	BCE	BGSTR,DEVTYP,00	SET THE BUFFER
09000		MCW	+PRNTBF,IR8	POINTER TO
09010	COLSET	BA	=1B4,FMTCD1	THE PROPER CHAR
09020		BA	(FMTCD1-3),IR8	POSITION TO PROPER CHARACTER
09030		BS	=1B1,IR8	ONE ORIGIN INDEX FOR BUFFER POINTER
09040		B	UPCDPT	
09050	BGSTR	MCW	STRADD,IR8	
09060		B	COLSET	
09070	PAGE	EQU	*	
09080		BCE	FMTERR,DEVTYP,00	IF STRING THEN ERROR
09090L		:SKIP	PRINT,57,	TOP OF PAGE
09100L		:PUT	PRINT,	FLUSH OUT CURRENT BUFFER
09110		BS	IR10	CLEAR FOR BCE TEST
09120		BA	=1B1,FMTCD1	BUMP FORMAT CODE POINTER
09130		B	SKPCLR	
09140	SKIP	BA	=1B1,FMTCD1	
09150		MRID	(FMTCD1-3),IR10-3	
09160		SAR	FMTCD1	
09170		BCE	FMTERR,DEVTYP,00	IF STRING THEN ERROR
09180L		:PUT	PRINT,	FLUSH OUT CURRENT BUFFER
09190		BS	=1B1,IR10	DECREMENT LINE COUNT
09200	SKPCLR	MCW	: : ,OUTPUT+132	
09210		MCW	OUTPUT+132	
09220		MCW	=1C21,OUTPUT	
09230		MCW	+PRNTBF,IR8	SET BUFFER POINTER
09240		SI	INPUT+80	
09250	SKPTST	BCE	EDIT,IR10,00	THE

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

09630	BUFOFL	EQU	*	
09640		:PUT	%PRINT,BFOFL,	
09650		B	ERRLOC	
09660	BFOFL	DCW	:A	****BUFFER OVERFLOW****:
09670	L	DCW	=1C45	
09680	IFORM	BA	=1B1,FMTCD1	BUMP FORMAT CODE POINTER
09690		MRID	(FMTCD1-3),IR10-3	I FIELD LENGTH
09700		SAR	FMTCD1	NEXT FORMAT CODE POINTER
09710		BA	IR8,IR10	BUFFER POINTER PLUS LENGTH
09720		C	IR10,STREND	CHECK
09730		BL	BUFOFL	OVERFLOW
09740		BS	CNVFLD	CONVERT
09750		SW	0-8+X2	MARK FOR MOVE
09760		BBE	IFMNEG,0-8+X2,40	CHECK FOR MINUS SIGN
09770		MLWD	0-5+X2,CNVFLD-2	BINARY
09780	IFMCNV	TMA	CNVFLD,00	INTEGER
09790		BTD	CNVFLD,00	TO DECIMAL
09800		MCW	:0:,0-2+X10	ZERO SUPPRESSION SYMBOL
09810		SW	0+X8	LEFT END OF EDIT CONTROL
09820		MCE	CNVFLD,0-1+X10	MOVE AND EDIT DECIMAL INTEGER
09830		MCW	IR10,IR8	NEXT BUFFER LOCATION
09840		BS	=1B9,IR2	POP STACK
09850		BBE	IFMSGN,1+X2,40	CHECK FOR NEGATIVE SIGN PLACEMENT
09860		B	FETCH	
09870	IFMNEG	SW	CNVFLD-5	
09880		BS	0-5+X2,CNVFLD-2	CONVERT TO POSITIVE
09890		CW	CNVFLD-5	
09900		B	IFMCNV	GO CONVERT TO DECIMAL
09910	IFMSGN	BCE	IFMSST,0-1+X10,15	FIND BLANK
09920		BS	=1B1,IR10	NEXT POSITION TO THE LEFT
09930		B	IFMSGN	
09940	IFMSST	MCW	: -: ,0-1+X10	
09950		B	FETCH	
09960	LFORM	BA	=1B1,FMTCD1	
09970		MRID	(FMTCD1-3),IR10-3	
09980		SAR	FMTCD1	NEXT FORMAT CODE POINTER
09990		BA	IR10,IR8	

10000		C	IR8,STREND	
10010		BL	BUFOFL	
10020		BCE	FOUT,0-8+X2,F	
10030		MCW	:T:,0-1+X8	MOVE TRUE
10040		B	FETCH	
10050	FOUT	MCW	:F:,0-1+X8	MOVE FALSE
10060		B	FETCH	
10070	EFORM	BA	=1B1,FMTCD1	
10080		MRID	(FMTCD1-3),IR10-3	OUTPUT LENGTH
10090		SAR	FMTCD1	NEXT FORMAT CODE
10100		C	IR10,STREND	
10110		BL	BUFOFL	
10120		TMA	0-1+X2,00	OPERAND TO FRO
10130		BS	=1B9,IR2	POP STACK
10140		B	FB/FD	CONVERT TO FLOATING DECIMAL
10150	R	DSA	DECRES	ADDRESS OF FLOATING DECIMAL FIELD
10160		MLWD	IR8,IR5	PREPARE FOR BASIC E FORMAT
10170		BBE	ROUND,DECRES-3,20	NO SIGN IF PLUS
10180		SST	:0:,DECRES-3,60	CLEAR SIGN
10190		MCW	:-:,0+X5	MINUS SIGN
10200	ROUND	LCA	:0:,DECRES-14	PREPARE FOR ROUNDING OVERFLOW
10210		CW	DECRES-13	
10220		A	:5:,DECRES-3	ROUND RESULT
10230		BCE	MARK,DECRES-14,00	TEST NO OVERFLOW
10240		MCW	DECRES-4,DECRES-3	MOVE RIGHT ONE CHARACTER ON OVERFLOW
10250		A	:1:,DECRES	ADJUST EXPONENT
10260	MARK	SW	DECRES-13	RESTORE FIELD MARKER
10270		SI	19+X5	RIGHT END OF OUTPUT FIELD
10280		BSN	EFRM,DECRES	BRANCH IF NEGATIVE EXPONENT
10290		SST	:0:,DECRES,60	STRIP SIGN
10300		C	DECRES,:009:	TEST EXPONENT
10310		BL	EFRM	
10320		MCW	DECRES,IR1	
10330		BA	IR1,IR5	
10340		SI	DECRES-4	RIGHT MOVE STOPPER
10350		BCE	MVPT,DECRES,00	ZERO EXPONENT TEST
10360		MCW	DECRES-14+X1,0+X5	MOVE DIGITS LEFT OF .

```

10370 MVPT MCW :.:1+X5 MOVE IN .
10380 MRID DECRES-13+X1,2+X5 MOVE DIGITS TO THE RIGHT OF .
10390 SBR IR5
10400 NZTST BBE NXTFLD,0-1+X5,77 REPLACE
10410 MCW :.:0+X5 ORDER ZEROES
10420 SBR IR5 WITH ELNKS
10430 B NZTST
10440 EFRM MCW :.:1+X5 MOVE IN DECIMAL PT.
10450 MCW DECRES-4,11+X5 MOVE TEN FRACTION DIGITS
10460 MCW :E.:12+X5 MOVE EXPONENT SYMBOL
10470 MCW :-.:13+X5 EXPONENT
10480 BBE LDIGIT,DECRES,40 SIGN
10490 MCW :+.:13+X5 LOGIC
10500 LDIGIT SST DECRES,16+X5,17 LOW DIGIT BUT NO SIGN
10510 MCW MOVE TWO HIGH DIGITS OF EXP
10520 B NXTFLD
10530 DCW :0:
10540 DCW =11
10550 DECRES DCW =3
10560 NXTFLD MRIN 0+X5,0
10570 SAR IR5
10580 CI 0-1+X5 REMOVE ITEM MARK
10590 BA IR10,IR8 NEXT OUTPUT BUFFER LOCATION
10600 B FETCH
10610*****
10620* *
10630* GET - PROCESS INPUT EITHER FROM CARD BUFFER OR STRING *
10640* ACCORDING TO INFORMATION SET BY FMT. *
10650* *
10660*****
10670 GET BCE GETFM,DEVTYP,00 IF STRING INPUT BYPASS EOF TEST
10680 BCE ENDFMS,ENDF,T TEST FOR CLOSED FILE
10690 GETFM BCE IFMRST,(FMTCD1-3),77 CHECK FOR FORMAT RESET
10700 BBE IDTFMT,(FMTCD1-3),70 CHECK FOR DATA FORMAT
10710 BCE ISPCE,(FMTCD1-3),00 ELSE IT MUST BE CONTROL FORMAT
10720 BCE ISKP,(FMTCD1-3),02
10730 BCE ICOL,(FMTCD1-3),03

```

10740		B	FMTERR	IF NONE OF THE ABOVE THEN ERROR
10750	IFMRST	MCW	FMTCD2,FMTCD1	RESET FORMAT POINTER
10760		B	GET	
10770	NXTIFM	EA	=1B1,FMTCD1	NEXT FORMAT CODE
10780		B	GET	
10790	ISPCE	BA	=1B4,FMTCD1	POINT TO SPACE COUNT
10800		MCW	(FMTCD1-3),IR9	MOVE IT TO INDEX REG.
10810	SPCCNT	C	IR9,:000:	
10820		BE	NXTIFM	TEST FINISH
10830		BCE	STRSPC,DEVTYP,00	TEST FOR STRING INPUT
10840		EA	=1B1,IR6	
10850		C	+INPUT+79,IR6	TEST FOR
10860		BH	GETIPT	NEW RECORD REQUIRED
10870		BS	=1B1,IR9	DECREMENT SPACE COUNT
10880		B	SPCCNT	
10890	STRSPC	BA	IR9,STRADD	SPACE
10900		C	STREND,STRADD	AND
10910		BEH	BUFOFL	TEST FOR STRING OVER FLOW
10920		B	NXTIFM	
10930	ICOL	BCE	STRCOL,DEVTYP,00	TEST FOR STRING
10940		MCW	+INPUT,IR6	
10950		BA	=1B4,FMTCD1	POINT TO COL INDICATOR
10960		EA	(FMTCD1-3),IR6	
10970		BS	=1B1,IR6	ONE ORIGIN INDEX FOR COLUMN POINTER
10980		C	+INPUT+79,IR6	
10990		BH	BUFOFL	CHECK FOR ERROR
11000		B	NXTIFM	
11010	STRCOL	MCW	STREND,STRADD	ESTABLISH
11020		BS	LNGSTR,STRADD	ORIGINAL ADDRESS
11030		BA	=1B5,FMTCD1	POINT TO COL INDICATOR
11040		EA	(FMTCD1-3),STRADD	
11050		BS	=1B1,STRADD	ONE ORIGIN COL INDICATOR
11060		C	STREND,STRADD	TEST
11070		BEH	BUFOFL	FOR ERROR
11080		B	NXTIFM	
11090	ISKP	BCE	FMTERR,DEVTYP,00	IF STRING INPUT THEN ERROR
11100		BA	=1B1,FMTCD1	

11110		MRID	(FMTCD1-3),IR10-3	SKIP COUNT TO INDEX REG
11120		SAR	FMTCD1	NEXT FORMAT CODE POINTER
11130	ISKPTS	BCE	GET,IR10,00	TEST FINISH
11140		B	GETIPT	GET NEXT INPUT RECORD
11150		BS	=1B1,IR10	
11160		B	ISKPTS	GO CHECK FOR SKIP COUNT
11170	GETIPT	SBR	GETRTN+4	SET RETURN ADDRESS
11180		BCE	ENDFMS,ENDF,T	TEST FOR CLOSED FILE
11190L		:GET	READ,	
11200		MCW	+INPUT,IR6	RESET BUFFER POINTER
11210		C	INPUT+3,:1EOF:	END OF FILE TEST
11220		BNE	GETRTN	
11230		MCW	:T:,ENDF	SET END OF FILE FLAG
11240	GETRTN	B	*	
11250	ENDF	DCW	:F:	
11260	ENDFMS	EQU	*	
11270L		:PUT	PRINT,EOFMES,	
11280		B	ERRLOC	
11290	EOFMES	DCW	:A *****END OF FILE ON INPUT UNIT*****:	
11300L		DCW	=1C45	
11310	IDTFMT	EQU	*	
11320		BCE	AFRMI,(FMTCD1-3),10	
11330		BCE	EFRMI,(FMTCD1-3),11	
11340		BCE	IFRMI,(FMTCD1-3),12	
11350		BCE	LFMRI,(FMTCD1-3),13	
11360		B	FMterr	
11370	AFRMI	BA	=1B1,FMTCD1	JUMP OVER CODE TYPE
11380		MRID	(FMTCD1-3),IR10-3	A FIELD LENGTH
11390		SAR	FMTCD1	NEXT FORMAT CODE LOCATION
11400		BBE	AOK,0-9+X2,04	
11410		B	CNVERR	
11420	AOK	SW	0-4+X2	
11430		C	IR10,0-5+X2	
11440		BL	FMterr	
11450		MRID	0-8+X2,IR9-3	ADDRESS OF STRING
11460		BA	IR10,IR9	RIGHT END OF RECEIVING FIELD(PLUS ONE)
11470	AMORE	SW	0+X6	MOVE STOPPER



11480		BA	IR10,IR6	BUFFER POINTER(PLUS ONE)
11490		BCE	ASTRNG,DEVTYP,00	CHECK FOR STRING INPUT
11500		C	IR6,+INPUT+80	CHECK SPLIT RECORD INPUT
11510		BL	ASPLIT	
11520		MLWDR	0-1+X6,0-1+X9	MOVE CHARACTERS
11530		SAR	IR4	CLEAR
11540		SBR	IR5	WORD
11550		BE	GETIPT	TEST FOR NEW RECORD REQUIRED
11560	AFIN	CW	1+X4,1+X5	MARKS
11570		BCE	POPUP,0-9+X2,34	CHECK FOR RECEIVING SUBSTRING
11580		SI	0-1+X9	ELSE SET ITEM MARK ON RIGHT
11590		B	POPUP	AND EXIT
11600	ASPLIT	BS	+INPUT+80,IR6	DETERMINE EXCESS CHARACTERS
11610		BS	IR6,IR9	REDUCE RECEIVING FIELD
11620		MLWDR	(INPEND-3),0-1+X9	MOVE WHAT IS AVAILABLE
11630		SAR	IR4	CLEAR
11640		SBR	IR5	WORD
11650		CW	1+X4,1+X5	MARKS
11660		MCW	IR6,IR10	NEW LENGTH
11670		B	GETIPT	GET NEXT RECORD
11680		BA	IR10,IR9	RESTORE END OF RECEIVING FIELD
11690		B	AMORE	GO FINISH MOVE
11700	INPEND	DSA	INPUT+79	END ADDRESS OF INPUT BUFFER
11710	ASTRNG	BA	STRADD,IR10	
11720		C	STREND,IR10	CHECK LENGTH
11730		BH	BUFOFL	
11740		SW	(STRADD-3)	MOVE STOPPER
11750		MLWDR	0-1+X10,0-1+X9	
11760		SAR	IR4	
11770		SBR	IR5	
11780		MCW	IR10,STRADD	NEXT STRING INPUT LOCATION
11790		B	AFIN	FINISH PUNCTUATION AND EXIT
11800	IFRMI	BA	=1B1,FMTCD1	JUMP OVER CODE TYPE
11810		MRID	(FMTCD1-3),IR10-3	I FIELD LENGTH
11820		SAR	FMTCD1	NEXT FORMAT CODE LOCATION
11830		BCE	IOK,0-9+X2,01	CHECK FOR MISMATCH
11840		B	CNVERR	

11850	IOK	BS	IR9	
11860		MCW	:+::ISGN	
11870		BS	CNVFLD	CLEAR CONVERSION FIELD
11880		BCE	ISTRNG,DEVTP,00	CHECK FOR STRING INPUT
11890	IMORE	BCE	STISGN,0+X6,-	TEST FOR MINUS SIGN
11900		BCE	STISGN,0+X6,+	TEST FOR PLUS SIGN
11910		SST	0+X6,DFLD+X9,17	MOVE NUMERIC BITS ONLY
11920		BA	=1B1,IR9	BUMP
11930	IENDT	BA	=1B1,IR6	COUNTERS
11940		C	IR6,INPEND	TEST FOR
11950		BL	GETIPT	NEXT RECORD
11960		C	IR10,IR9	TEST FIELD END
11970		BL	IMORE	
11980	IDECMV	MCW	DFLD-1+X9,CNVFLD	MOVE DECIMAL DIGITS
11990		SST	ISGN,CNVFLD,60	SET SIGN IN CONVERSION FIELD
12000		DTB	CNVFLD,00	
12010		TAM	CNVFLD,00	
12020		MRID	0-8+X2,IR9-3	ADDRESS OF RECEIVING FIELD
12030		SI	CNVFLD-2	PUNCUATION FOR MOVING
12040		MRID	CNVFLD-5,0+X9	MOVE RESULT
12050		CI	CNVFLD-2	CLEAR ITEM MARK
12060		B	POPUP	POP STACK AND EXIT
12070	DFLD	DCW	=30	
12080	ISGN	DCW	:+::	
12090	STISGN	MRSU	0+X6,ISGN	SET SIGN
12100		BS	=1B1,IR10	ADJUST CHARACTER COUNT
12110		E	IENDT	CHECK FOR RECORD END
12120	ISTRNG	SST	(STRADD-3),DFLD+X9,17	MOVE DIGIT
12130		BCE	ISTSGN,(STRADD-3),+	
12140		BCE	ISTSGN,(STRADD-3),-	
12150	ISTRUP	EA	=1B1,IR9	
12160		EA	=1B1,STRADD	
12170		C	IR10,IR9	IF FIELD END
12180		EH	IDECMV	THEN SET UP FOR CONVERSION
12190		C	STREND,STRADD	CHECK BUFFER
12200		BEH	BUFOFL	OVERFLOW
12210		B	ISTRNG	GET ANOTHER DIGIT

12220	ISTSGN	MRSD	(STRADD-3),ISGN	
12230		B	ISTRUP	
12240	EFRMI	BA	=1B1,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 STRINGINPUT
12320		BS	IR11	
12330	EMORE	MRSD	0+X6,EFHLD+X9	
12340		SAR	IR6	
12350		BA	=1B1,IR9	
12360		C	IR6,INPEND	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	BUFOFL	
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

12590		BCE	EXPON,EFHLD+X9,E	
12600		SST	EFHLD+X9,DFRACT+X11,17	
12610		BA	=1B1,IR11	DECIMAL FRACTION POINTER
12620		A	:001:,DEXP	INCREMENT EXPONENT (DECIMAL)
12630	ECNT1	BA	=1B1,IR9	INPUT CHAR COUNTER
12640		C	IR9,IR10	IF MORE
12650		BH	ECKBLK	THEN GO CHECK IT
12660		B	CFDFB	ELSE CONVERT FD/FB
12670	FSGNST	MRSD	EFHLD+X9,FSGN	
12680		B	ECNT1	
12690	FRACT	BA	=1B1,IR9	BUMP CHAR POINTER
12700		C	IR9,IR10	IF NO MORE
12710		BEL	CFDFB	THEN CONVERT FD/FB
12720		BCE	EXPON,EFHLD+X9,E	ELSE CHECK EXPONENT
12730		BCE	EXPON,EFHLD+X9,15	IF BLANK THEN LOOK FOR EXPONENT
12740		SST	EFHLD+X9,DFRACT+X11,17	ELSE MOVE DIGIT
12750		BA	=1B1,IR11	
12760		B	FRACT	GO LOOK FOR MORE
12770	EXPON	BCE	EXPR,EFHLD+X9,E	
12780		BA	=1B1,IR9	BUMP CHAR POINTER
12790		C	IR9,IR10	IF NO MORE
12800		BL	CFDFB	THEN CONVERT FD/FB
12810		B	EXPON	
12820	EXPR	MCW	:+::,ESGN	DEFAULT SIGN
12830		BA	=1B1,IR9	
12840		BCE	ESGNST,EFHLD+X9,-	CHECK FOR
12850		BCE	ESGNST,EFHLD+X9,+	EXPONENT SIGN
12860		B	EXPNUM	GET EXPONENT
12870	ESGNST	MRSD	EFHLD+X9,FSGN	SET SIGN
12880		BA	=1B1,IR9	
12890	EXPNUM	SW	EFHLD+X9	
12900		BS	DECEXP	
12910		MCW	EFHLD-1+X10,DECEXP	
12920		CW	EFHLD+X9	
12930	CFDFB	SST	ESGN,DECEXP,60	SET SIGN
12940		SST	FSGN,DFRACT+10,60	SET FRACTION SIGN FOR CONVERSION
12950		A	DECEXP,DEXP	ACCUMLATE DEC EXP FOR CONVERSION

12960	B	FD/FB	CALL CONVERSION ROUTINE
12970 R	DSA	DEXP	ADDRESS OF RIGHT END OF 14 CHAR
12980*			FLOATING DECIMAL FIELD
12990 R	B	CNVERR	ERROR INSTRUCTION
13000	MRID	0-8+X2,IR10-3	ADDRESS OF RESULT
13010	MLWDI	=8B0,0-1+X2	CLEAR ANY EXTRANEIOUS PUNCTUATION
13020	TAM	7+X10,00	STORE RESULT
13030	SI	7+X10	ITEM MARK RIGHT
13040	B	POPUP	POP STACK AND FETCH NEXT INSTR
13050	DFRACTDCW	=11	
13060	DEXP	DCW =3	
13070	DECEXP	DCW =3	
13080	ESGN	DCW =1	
13090	FSGN	DCW =1	
13100	LFRMI	BA =1B1,FMTCD1	JUMP OVER CODE TYPE
13110	MRID	=FMTCD1-3),IR10-3	FIELD LENGTH
13120	SAR	FMTCD1	NEXT FORMAT CODE LOCATION
13130	BCE	LOK,0-9+X2,03	CHECK FOR TYPE MISMATCH
13140	B	CNVERR	
13150	LOK	BS IR9	
13160	BCE	LSTR,DEVTYP,00	CHECK FOR STRING INPUT
13170	LRECP	BA IR10,IR6	
13180	C	IR6,+INPUT+80	CHECK
13190	BL	LNWREC	NEW RECORD REQUIREMENT
13200	MRID	0-8+X2,IR9-3	MOVE ADDRESS
13210	MRSDR	0-1+X6,0+X9	
13220	BE	GETIPT	
13230	B	POPUP	
13240	LNWREC	BS +INPUT+80,IR6	DETERMINE EXCESS CHAR
13250	MCW	IR6,IR10	NEW LENGTH
13260	B	GETIPT	
13270	B	LRECP	
13280	LSTR	MCW STRADD,IR9	
13290	BA	IR10,IR9	
13300	C	IR9,STREND	
13310	BL	BUFOFL	CHECK OVERFLOW
13320	MRID	0-8+X2,IR10-3	ADDRESS OF RECEIVING FIELD

```

13330      MRSDR 0-1+X9,0+X10      MOVE INPUT
13340      MCW   IK9,STRADD        NEXT STRING INPUT
13350      B     POPUP             POP STACK AND FETCH
13360*****
13370*
13380*      STOP - PRINT INSTRUCTION COUNT MESSAGE AND EXIT.
13390*
13400*****
13410  STOP  BS   CNVFLD
13420      MCW   INSTCT,CNVFLD-2    BINARY COUNT TO CONVERSION FIELD
13430      TMA   CNVFLD,00          CONVERT IT
13440      BTD   CNVFLD,00          TO DECIMAL
13450      LCA   EWORD,PRTCNT       EDIT WORD
13460      MCE   CNVFLD,PRTCNT      MOVE IT TO PRINT FIELD (EDITED)
13470L     :PUT  PRINT,CNTMES,
13480      B     (164)              EXIT
13490  CNTMESDCW :A   ****INSTRUCTION COUNT = :
13500  PRTCNT DC  =9
13510      DC    :****:
13520 L     DCW  =1C45
13530*****
13540*
13550*      ERROR MESSAGES
13560*
13570*****
13580  ERROR  EQU   *
13590L     :PUT  PRINT,BADOP,
13600      B     ERRLOC
13610  BADOP DCW   :A   ****ILLEGAL OP CODE****:
13620 L     DCW  =1C45
13630  DYNOfL EQU  *
13640L     :PUT  PRINT,STROFL,
13650      B     ERRLOC
13660  STROFLDCW :A   ****DYNAMIC STORAGE EXHAUSTED****:
13670 L     DCW  =1C45
13680  CNVERR EQU  *
13690L     :PUT  PRINT,CVEMS,

```

```

13700      B      ERRLOC
13710      CVEMS DCW :A      ****DATA TYPE CONVERSION ERROR****:
13720 L      DCW   =1C45
13730      FERR  EQU  *
13740L      :PUT  PRINT,FERMES,
13750      B      ERRLOC
13760      FERMESDCW :A      ****FLTNG PT OVFLW OR ZERO DIVIDE****:
13770 L      DCW   =1C45
13780      IDXERR EQU  *
13790L      :PUT  PRINT,IDXMES,
13800      B      ERRLOC
13810      IDXMESDCW :A      ****INDEXED ADDRESS BEYOND DYNAMIC STORAGE****:
13820 L      DCW   =1C45
13830      TYPERR EQU  *
13840L      :PUT  PRINT,TYPMES,
13850      B      ERRLOC
13860      TYPMESDCW :A      ****INCONSISTENT DATA TYPE ERROR****:
13870 L      DCW   =1C45
13880      FMTERR EQU  *
13890L      :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      BTDC     CNVFLD,00      TO
13970      LCA      EWORD,ELOC     DECIMAL
13980      MCE      CNVFLD,ELOC     FOR PRINTING
13990L      :PUT  PRINT,LOCMES,
14000      BCT      HALT,01      IF SENSE SWITCH ONE THEN DUMP REQUEST
14010      B        STOP      ELSE EXIT
14020      HALT     H        DUMP REQUEST
14030      EWORD    DCW      : ,      ,0 :
14040      LOCMESDCW :A      ****ERROR AT RELATIVE LOCATION :
14050      ELOC     DC       =9
14060      DC        :****:

```

```

14070 L      DCW    =1C45
14080  CNVFLD DCW    =11B0
14090      LITORG*
14100  ENDPRG EQU    *
14110      END     START

```

# SYMBOL DEFINITION - CARD REFERENCE INDEX

ADCOMP	01840;	ADD	06290;	AFIN	11560;	AFORM	09390;	AFRMA	09450;
AFRMI	11370;	ALCRT1	02600;	ALCRT2	02630;	ALLOC	02490;	AMORE	11470;
AND	07060;	AOK	11420;	ARI	06850;	ASPLIT	11600;	ASTRNG	11710;
BADOP	13610;	BFOFL	09660;	BGSTR	09050;	BLNK	06160;	BSTF	07130;
BSTT	07100;	BUFOFL	09630;	CAT2	07720;	CAT3	07760;	CAT	07670;
CFDFB	12930;	CHK2ND	05150;	CHKBFF	09470;	CHKTYP	03180;	CHRSTR	03710;
CHRTN	03750;	CLRMOR	08560;	CMPADD	07380;	CNDTBL	06150;	CNDTST	05680;
CNTMES	13490;	CNVCHK	06860;	CNVERR	13680;	CNVFLD	14080;	CNVRTN	03410;
CNVTR	06950;	CNVTR	06940;	COLMN	08990;	COLSET	09010;	COMPC	05400;
COND	05210;	CONPRT	02860;	CSBSTR	03930;	CTSUB1	07810;	CTSUB2	07920;
CURLEV	04550;	CVEMS	13710;	DATFMT	09330;	DECEXP	13070;	DECRES	10550;
DEVTYP	08440;	DEXP	13060;	DFLD	12070;	DFRACT	13050;	DISPLY	04490;
DIV	06610;	DODIV	06670;	DTYPE	02220;	DUMMY	09290;	DVDNE	02750;
DYNAM	02280;	DYNEND	00460;	DYNOFL	13630;	DYNSTR	00450;	ECKBLK	12550;
ECNT1	12630;	ECNV	12480;	EDIT	08870;	EFHLD	12410;	EFORM	10070;
EFRM	10440;	EFRMI	12240;	ELOC	14050;	EMORE	12330;	ENDADR	06020;
ENDF	11250;	ENDFMS	11260;	ENDPRG	14100;	ENTPRO	04220;	EOFMES	11290;
EOK	12290;	ERRLOC	13930;	ERROR	13580;	ESGN	13080;	ESGNST	12870;
ESTRNG	12420;	EWORD	14030;	EXPNUM	12890;	EXPON	12770;	EXPR	12820;
FERMES	13760;	FERR	13730;	FETCH	01170;	FIXFLT	03640;	FLAG	04130;
FLGCHK	04310;	FMT	08290;	FMTCD1	08410;	FMTCD2	08420;	FMTERP	13880;
FMTMES	13910;	FMTRST	09310;	FNDLNG	08140;	FOUT	10050;	FRACT	12690;
FSGN	13090;	FSGNST	12670;	GET	10670;	GETFM	10690;	GETIPT	11170;
GETRTN	11240;	GTALN	09560;	GTSBLN	09610;	HALT	14020;	HOLD	04690;
ICOL	10930;	IDECMV	11980;	IDTFMT	11310;	IDXERR	13780;	IDXMES	13810;
IENDT	11930;	IFIX	00780;	IFMCNV	09780;	IFMNEG	09870;	IFMRST	10750;



IFMSGN	09910;	IFMSST	09940;	IFORM	09680;	IFRMI	11800;	IMORE	11890;
INDIRA	02160;	INDXA	07250;	INDXR	07260;	INOROT	08430;	INPEND	11700;
INPUT	00310;	INSTCT	00410;	INTADD	06400;	INTDIV	06720;	INTMLT	06580;
INTNEG	06810;	INTSbS	07340;	INTSUB	06490;	IOK	11850;	IR10	00610;
IR11	00620;	IR12	00630;	IR13	00640;	IR14	00650;	IR15	00660;
IR1	00520;	IR2	00530;	IR3	00540;	IR4	00550;	IR5	00560;
IR6	00570;	IR7	00580;	IR8	00590;	IR9	00600;	ISGN	12080;
ISKP	11090;	ISKPTS	11130;	ISPCE	10790;	ISTRNG	12120;	ISTRUP	12150;
ISTSIGN	12220;	JUMP	04960;	JUMPA	04860;	JUMPF	05030;	JUMPP	05010;
JUMPT	04980;	LCOMP	01870;	LD	03140;	LDA	02940;	LDIGIT	10500;
LDLNG	02990;	LFORM	09960;	LFRMI	13100;	LITADD	02100;	LITLNG	03030;
LNGSTR	08460;	LNGTH	02840;	LNWREC	13240;	LOCME5	14040;	LODLOC	00440;
LQK	13150;	LRECP	13170;	LSTR	13280;	MARK	10260;	MORIDX	07430;
MORSJB	02580;	MOVNXT	05700;	MOVTOP	05840;	MRBLNK	05770;	MULT	06530;
MVPT	10370;	MVRT1	05450;	MVRT2	05470;	NEG	06770;	NOCOMP	02180;
NOSET	04010;	NOT	07090;	NUMER	06650;	NXTFLD	10560;	NXTIFM	10770;
NZTST	10400;	OR	07030;	OUTPUT	00320;	PAGE	09070;	PBUFF	08760;
PNCRTN	03460;	POPUP	04870;	PRCADD	02150;	PRINT	00330;	PRMCHK	04360;
PRMCNT	04570;	PRMDNE	04420;	PRMOVE	04030;	PRMSTO	04350;	PRNTBF	00350;
PRTCNT	13500;	PUT	08700;	READ	00300;	RETRN	04760;	ROUND	10200;
#RDWR	00290;	#SKP	00340;	SAVTOP	03500;	SBSTR	08050;	SBSTRL	08110;
SCNV	03550;	SETBCK	06330;	SETCND	05170;	SETF	05220;	SETFLT	03230;
SETINT	03210;	SETPNC	05820;	SETT	05260;	SIZE	02850;	SKIP	09140;
SKPCLR	09200;	SKPTST	09250;	SPACE	08960;	SPCCNT	10810;	SST	03360;
SSTCHK	03450;	START	00860;	STCKC	05130;	STISGN	12090;	STKEND	00430;
STKSTR	00420;	STKTP	02380;	STO	03350;	STOP	13410;	STRADD	08450;
STRCOL	11010;	STREND	08470;	STROFL	13660;	STRPUT	08780;	STRSET	08480;
STRSPC	10890;	STYPE	02210;	SUB	06440;	SUBCHK	07290;	SURDNE	07500;
SUBMV1	05910;	SUBMV2	06030;	SUBMV	03860;	SWAP	04630;	TMOVE	03880;
TSTRNG	05580;	TVEC	01250;	TYPERR	13830;	TYPMES	13860;	UPCDPT	08940;
UPSTCK	02970;								

\*\*\*\*INSTRUCTION COUNT = 296,018\*\*\*\*

XII. APPENDIX D

```

00010          PROG  MTXMCP
00020          SEG   03
00030*****
00040*
00050*      IDENTIFICATION:
00060*      -----
00070*
00080*      PROGRAM-ID:  MTXMCP03.
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*      MTXMCP02 PROVIDES THE METAX SYSTEM CONTROL FUNCTIONS AND
00180*      SYSTEM SERVICES.  SEE THE CHAPTER ON THE METAX SYSTEM FOR
00190*      A DETAILED DESCRIPTION.
00200*
00210*****
00220          ADMODE4
00230          ORG    3400
00240*****
00250*
00260*      INDEX REGISTER LOCATION DEFINITIONS
00270*
00280*****
00290  IR1      EQU    4
00300  IR2      EQU    8
00310  IR3      EQU   12
00320  IR4      EQU   16
00330  IR5      EQU   20
00340  IR6      EQU   24
00350  IR7      EQU   28
00360  IR13     EQU   52
00370  IR14     EQU   56

```



00750	MTXPRG	DCW	=4C00120000	LOCATION FOR RESIDENT METAX PROGRAM
00760	MTXCRA	DCW	=4C00126000	LOCATION FOR RESIDENT CONTROL RECORD ANALYZER
00770	START	CAM	60	SET FOUR CHAR ADDRESSING MODE
00780L		:GET	READ,	GET NAMES OF RESIDENT METAX PROGRAMS
00790		MCW	INPUT+15,SAVNME	SAVE FOR LATER USE
00800		SW	SAVNME-7	
00810		LCA	MTXCRA,LODFLD	SET LOCATION FOR CONTROL RECORD ANALYZER
00820		SW	INTNME-7	
00830		SW	PRGNME-7	
00840		MCW	INPUT+7,PRGNME	NAME TO METAX COMMUNICATIONS FIELD
00850		MCW	LDRSEG,75	SEGMENT AND
00860		MCW		NAME OF DISK TO MEMORY LOAD PROGRAM
00870		MCW	+RTNST,167	SET UP RETURN FOR RETURN START
00880		B	(168)	FETCH AND EXECUTE
00890	RTNST	SBR	LDRTN+4	SET UP RETURN START
00900		MCW	+LDEXIT,167	EXIT POINT
00910		B	(LDRTN+1)	RETURN TO LOADER
00920	LDEXIT	MCW	INPUT+15,PRGNME	SET NAME OF RESIDENT METAX PROGRAM
00930		LCA	MTXPRG,LODFLD	SET LOCATION FOR RESIDENT METAX PROG
00940		MCW	+CRACLL,167	EXIT POINT
00950	LDRTN	B	*	FETCH RESIDENT METAX PROGRAM
00960	CRACLL	LCA	=4B0,INSTCT	ZERO INSTRUCTION COUNT
00970		LCA	+ENDPRG+1,STCKF1	COMMUNICATIONS
00980		LCA	+ENDPRG+999,STCKF2	AREA
00990		MCW	MTXCRA,LODFLD	FIELDS
01000		BS	PRGNME	FOR EXECUTING
01010		BS	INTNME	CONTROL RECORD ANALYZER
01020		MCW	INTSEG,75	INTERPRETER SEGMENT
01030		MCW		AND NAME
01040		MCW	+INTRT1,167	RETURN POINT
01050		B	(168)	FETCH AND EXECUTE
01060	INTRT1	LCA	=4B0,INSTCT	ZERO INSTRUCTION COUNT
01070		MCW	:0:,CMPLCD	INITIALIZE COMPLETION CODE
01080		LCA	=3C117776,IR15	SET UP
01090		MRSDR	=4B0,ENDPRG+1	FOR
01100		SBR	IR14	MEMORY
01110		SI	0+X15	CLEAR

01120		SW	0+X15	OPERATION
01130		B	CLEAR	DO IT
01140		MCW	LODFLD,LODSAV	
01150		C	PRGNME,SAVNME	TEST EXECUTION OF RESIDENT METAX PROGRAM
01160		BNE	MFETCH	IF NOT GO GET THE RIGHT ONE
01170		MCW	MTXPRG,LODFLD	ALTER EXECUTION LOCATION TO RESIDENT PROG
01180	GETINT	MCW	INTNME,75	INTERPRETER SEGMENT
01190		MCW		AND NAME
01200		MCW	+INTRT2,167	RETURN POINT
01210		B	(168)	FETCH AND EXECUTE
01220	INTRT2	BCE	FATAL,CMPLCD,F	TEST FOR FATAL ERROR ACTION
01230		BCE	LTODSK,DSKLOD,Y	LOAD COMPILED PROGRAM TO DISK IF REQUESTED
01240		BCE	EFTST,EXCPPG,N	IF NO GO THEN SEARCH FOR END OF FILE
01250	GO	EQU	*	ELSE MOVE AND RELOCATE COMPILED
01260		MCW	LODSAV,LODFLD	PROGRAM FOR EXECUTION
01270		SI	1+X5	
01280		SW	W+X5	
01290		MLWD	GENFLD,IR15	COMPILED PROGRAM LOCATION
01300		MLWD	LODFLD,IR14	LOADING LOCATION
01310	MORPRG	MRWDR	0+X15,0+X14	
01320		SAR	IR15	
01330		SBR	IR14	
01340		MRIDR	0+X15,0+X14	MOVE REMAINDER OF ADDRESS
01350		SAR	IR15	
01360		SBR	IR14	
01370		BA	LODFLD,0-1+X14	RELOCATE THE ADDRESS
01380		CW	0-3+X14	
01390		CW	0-1+X14	
01400		BCE	BLKCN,0-4+X14,00	TEST POSSIBLE BLOCK PSEUDO OP CODE
01410	ENDTST	C	IR15,IR5	
01420		BH	MORPRG	
01430		MLWD	=3C117776,IR15	PREPARE FOR CLEARING REMAINING MEMORY
01440		MRS DR	=480,0+X14	
01450		SBR	IR14	
01460		SI	0+X15	
01470		B	CLEAR	
01480		MCW	SYMF1,IR14	CLEAR

01490		MCW	SYMF2,IR15	
01500		MRSOR	=4B0,0+X14	
01510		SBR	IR14	SYMBOL (OR DYNAMIC STORAGE FOR PLEX)
01520		SW	0-1+X15	
01530		SI	0-1+X15	
01540		B	CLEAR	TABLE AREA
01550		LCA	=4B0,INSTCT	ZERO INSTRUCTION COUNT
01560		B	EXINT	GO LOAD INTERPRETER
01570	BLKCNT	BNP	ENDTST,0-5+X14	MAKE SURE IT IS A
01580		BNP	ENDTST,0-4+X14	BLOCK CODE
01590		MRID	0-3+X14,IR14-2	ADJUST MEMORY POINTER BY SIZE OF BLOCK
01600		B	ENDTST	
01610	MTXSTR	DCW	:MTXSTR:	CORE TO DISK UPDATE PROGRAM NAME
01620	STRSEG	DCW	:00:	
01630	MTXLDR	DCW	:MTXL0D:	METAX SYSTEM LOADER PROGRAM NAME
01640	LDRSEG	DCW	:00:	
01650	MTXINT	DCW	:MTXINT:	MTXCRA INTERPRETER NAME
01660	INTSEG	DCW	:02:	
01670	SAVNME	DCW	=16B0	INITIAL METAX PROGRAM NAMES SAVE LOC
01680	LODSAV	DCW	=4B0	LODFLD SAVE LOCATION
01690	CLEAR	SBR	CLRRTN+4	MEMORY CLEAR SUBROUTINE
01700	MOPE	MRIDW	0-1+X14,0+X14	
01710		SBR	IR14	
01720		CI	0-2+X14	
01730		CW	0-2+X14	
01740		C	IR14,IR15	
01750		BH	MORE	
01760	CLRRTN	B	*	
01770	EXINT	MCW	INTNME,75	SEGMENT AND NAME
01780		MCW		FOR EXECUTING COMPILED PROGRAM
01790		MCW	+EOFTST,167	RETURN POINT
01800		B	(168)	FETCH AND EXECUTE
01810	EOFTST	C	INPUT+3,:1EOF:	TEST END OF FILE
01820		BE	CRACLL	NEXT JOB
01830L		:GET	READ,	
01840		B	EOFTST	SEARCH UNTIL IT IS FOUND
01850	FATAL	EQU	*	

01860L	:	PUT	PRINT,FTLMES,	
01870		B	EOFTST	
01880	FTLMESDCW	:	1 FATAL ERROR(S) ENCOUNTERED, JOB AORTED:	
01890 L	DCW	=	1C45	
01900	LTODSK SBR		167	SET RETURN POINT
01910	MCW		IR5,IR14	AVOID SUPERVISOR USE OF IR5
01920	MCW		STRSEG,75	SEGMENT AND NAME
01930	MCW			OF MEMORY TO DISK PROGRAM
01940	B		(168)	FETCH AND EXECUTE
01950	MFETCH MCW		LDRSEG,75	SEGMENT AND NAME
01960	MCW			OF DISK TO MEMORY PROGRAM
01970	MCW		+FTRTST,167	SET UP RETURN FOR RETURN START
01980	B		(168)	FETCH AND EXECUTE
01990	FTRTST SBR		FTRTN+4	SET UP RETURN START
02000	MCW		+GETINT,167	SET EXIT
02010	FTRTN B		*	
02020			LITORG	
02030	ENDPRG EQU		*	
02040	END		START	

323

# SYMBOL DEFINITION - CARD REFERENCE INDEX

BLKCNT	01570;	CLEAR	01690;	CLRRTN	01760;	CMPLCD	00660;	CPACLL	00960;
DSKLOD	00670;	ENDPRG	02030;	ENDTST	01410;	EOFTST	01810;	EXCPPG	00680;
ExINT	01770;	FATAL	01850;	FTLMES	01880;	FTRTN	02010;	FTPTST	01990;
GENFLD	00560;	GETINT	01180;	GO	01250;	INPUT	00470;	INSTCT	00730;
INTNME	00710;	INTRT1	01060;	INTRT2	01220;	INTSEG	01660;	IR13	00360;



IR14	00370;	IR15	00380;	IR1	00290;	IR2	00300;	IR3	00310;
IR4	00320;	IR5	00330;	IR6	00340;	IR7	00350;	LDEXIT	00920;
LDRSEG	01640;	LDRTN	00950;	LODFLD	00570;	LODSAV	01680;	LTODSK	01900;
MFETCH	01950;	MORE	01700;	MORPRG	01310;	MTXCRA	00760;	MTXINT	01650;
MTXLDR	01630;	MTXPRG	00750;	MTXSTR	01610;	OUTPUT	00480;	PRGNME	00700;
PRINT	00490;	PSTLST	00690;	READ	00460;	RTNST	00890;	#RDWR	00450;
#SKP	00500;	SAVNME	01670;	START	00770;	STCKF1	00580;	STCKF2	00590;
STRSEG	01620;	SYMBOL	00720;	SYMF1	00600;	SYMF2	00630;		

\*\*\*\*INSTRUCTION COUNT = 47,601\*\*\*\*

## XIII. APPENDIX E

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

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

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

Floating point instructions also represent a departure from the two address scheme in that floating point registers are used in a one address fashion.

The internal data representations for the respective pseudo-machines correspond to the host computer with the exceptions that addressing is always on the left and item marks are used as right end delimiters. The respective interpreters make the necessary adjustments for addressing and may insert word marks on the left during execution. However, word marks are never generated for data fields during translation or loading.

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

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

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

The addressing structure of the H-1200 is binary. Address modification may be effected with either indirect addressing or indexing. There are three addressing modes

based on the amount of storage to be addressed and the number of index registers to be used. The mode used in all the assembler programs in the METAX system is the four character or 24 bit mode which allows a 19 bit address and a five bit address modifier. The latter is used to specify one of fifteen index registers or indirect addressing.

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

BA      =1B1,IR13

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

The specification of indexing is exemplified by

MCW    TVEC+3+X7,IR14

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

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

SAR    IR1

and

SBR IR10

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

The B register may also be used for subroutine linkage.

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

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

Then

MRIDI 0+X6,SYMBOL

specifies the data and item mark bits of the first operand

are to be moved from left to right to SYMBOL with the move terminated by the first item mark in the sending field while

MRIN    0+X14,0+X13

does nothing more than position the A and B registers according to the first item mark found in the first operand.

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

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

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

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

The system supervisor (24) under which the METAX system operates utilizes its own communications region. Several fields 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-

ecute 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).